Data Warehouse Service

Best Practices

Issue 46

Date 2024-10-29

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

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.

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Import and Export

1.1 Best Practices for Data Import

Importing Data from OBS in Parallel

- Splitting a data file into multiple files

  Importing a huge amount of data takes a long period of time and consumes many computing resources.

  To improve the performance of importing data from OBS, split a data file into multiple files as evenly as possible before importing it to OBS. The preferred number of split files is an integer multiple of the DN quantity.

- Verifying data files before and after an import

  When importing data from OBS, first import your files to your OBS bucket, and then verify that the bucket contains all the correct files, and only those files.

  After the import is complete, run the **SELECT** statement to verify that the required files have been imported.

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

Using GDS to Import Data

- Data skew causes the query performance to deteriorate. Before importing all the data from a table containing over 10 million records, you are advised to import some of the data and check whether there is data skew and whether the distribution keys need to be changed. Troubleshoot the data skew if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported. For details, see **Checking for Data Skew**.
- To speed up the import, you are advised to split files and use multiple Gauss Data Service (GDS) tools to import data in parallel. An import task can be split into multiple concurrent import tasks. If multiple import tasks use the same GDS, you can specify the **-t** parameter to enable GDS multi-thread concurrent import. To prevent physical I/O and network bottleneck, you are advised to mount GDSs to different physical disks and NICs.

- If the GDS I/O and NICs do not reach their physical bottlenecks, you can enable SMP on GaussDB(DWS) for acceleration. SMP will multiply the pressure on GDSs. Note that SMP adaptation is implemented based on the GaussDB(DWS) CPU pressure rather than the GDS pressure. For more information about SMP, see **SMP Manual Optimization Suggestions**.
- For the proper communication between GDSs and GaussDB(DWS), you are advised to use 10GE networks. 1GE networks cannot bear the high-speed data transmission, and, as a result, cannot ensure proper communication between GDSs and GaussDB(DWS). To maximize the import rate of a single file, ensure that a 10GE network is used and the data disk group I/O rate is greater than the upper limit of the GDS single-core processing capability (about 400 MB/s).
- Similar to the single-table import, ensure that the I/O rate is greater than the maximum network throughput in the concurrent import.
- It is recommended that the ratio of GDS quantity to DN quantity be in the range of 1:3 to 1:6.
- To improve the efficiency of importing data in batches to column-store partitioned tables, the data is buffered before being written into a disk. You can specify the number of buffers and the buffer size by setting **partition_mem_batch** and **partition_max_cache_size**, respectively. Smaller values indicate the slower the batch import to column-store partitioned tables. The larger the values, the higher the memory consumption.

Using INSERT to Insert Multiple Rows

If the COPY statement cannot be used during data import, you can use multi-row inserts to insert data in batches. Multi-row inserts improve performance by batching up a series of inserts.

The following example inserts three rows into a three-column table using a single **INSERT** statement. This is still a small insert, shown simply to illustrate the syntax of a multi-row insert.

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

```
INSERT INTO customer_t1 VALUES
(6885, 'maps', 'Joes'),
(4321, 'tpcds', 'Lily'),
(9527, 'world', 'James');
```

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

Using the COPY Statement to Import Data

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

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

Using a gsql Meta-Command to Import Data

The **\copy** command can be used to import data after you log in to a database through any **gsql** client. Compared with the **COPY** command, the **\copy** command

directly reads or writes local files instead of reading or writing files on the database server.

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

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

NOTE

\copy only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or **COPY** is preferred for data import.

1.2 Migrating Data from OBS Buckets to a GaussDB(DWS) Cluster

Overview

This practice demonstrates how to upload sample data to OBS and import OBS data to the target table on GaussDB(DWS), helping you quickly learn how to import data from OBS to a GaussDB(DWS) cluster.

You can import data in TXT, CSV, ORC, PARQUET, CARBONDATA, or JSON format from OBS to a GaussDB(DWS) cluster for query.

This tutorial uses the CSV format as an example to perform the following operations:

- Generate data files in CSV format.
- Create an OBS bucket in the same region as the GaussDB(DWS) cluster, and upload data files to the OBS bucket.
- Create a foreign table to import data from the OBS bucket to GaussDB(DWS) clusters.
- Start GaussDB(DWS), create a table, and import data from OBS to the table.
- Analyze import errors based on the information in the error table and correct these errors.

Estimated time: 30 minutes

Preparing Source Data Files

- Data file **product_info0.csv**

```
100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
```

- Data file **product_info1.csv**

```
200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
```



```
450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
```

- Data file **product_info2.csv**

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

Step 1 Create a text file, open it using a local editing tool (for example, Visual Studio Code), and copy the sample data to the text file.

Step 2 Choose **Format** > **Encode in UTF-8 without BOM**.

Step 3 Choose **File** > **Save as**.

Step 4 In the displayed dialog box, enter the file name, set the file name extension to .csv, and click **Save**.

----End

Uploading Data to OBS

Step 1 Store the three CSV source data files in the OBS bucket.

1. Log in to the OBS management console.

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

2. Create a bucket.

   For details about how to create an OBS bucket, see *Getting Started* > **Creating a Bucket** in *Object Storage Service*.

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

   NOTICE

   Ensure that the two buckets and the GaussDB(DWS) cluster are in the same region. This practice uses the CN-Hong Kong region as an example.

3. Create a folder.

   For details, see **Creating a Folder** in the *Object Storage Service User Guide*.

   Examples:

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

4. Upload the files.

   For details, see For details, see **Uploading an Object** in the *Object Storage Service Quick Start*.

   Examples:

  - Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:
    ```
    product_info0.csv
    product_info1.csv
    ```
  - Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:
    ```
    product_info2.csv
    ```

Step 2 Grant the OBS bucket read permission for the user who will import data.

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

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

----End

Creating a Foreign Table

Step 1 Connect to the GaussDB(DWS) database.

Step 2 Create a foreign table.

NOTE

- ACCESS_KEY and SECRET_ACCESS_KEY

  These parameters specify the AK and SK used to access OBS by a user. Replace them with the actual AK and SK.

  To obtain an access key, log in to the management console, move the cursor to the username in the upper right corner, click **My Credential**, and click **Access Keys** in the navigation pane on the left. On the **Access Keys** page, you can view the existing access key IDs (AKs). To obtain both the AK and SK, click **Create Access Key** to create and download an access key.
- // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
```



```
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

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

```
CREATE FOREIGN TABLE
```

----End

Importing Data

Step 1 Create a table named **product_info** in the GaussDB(DWS) database to store the data imported from OBS.

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

Step 2 Run **INSERT** to import data from OBS to the target table **product_info** through the foreign table **product_info_ext**.

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

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

```
SELECT * FROM product_info;
```

The following information is displayed at the end of the query result:

```
(20 rows)
```

Step 4 Run **VACUUM FULL** on the **product_info** table.

```
VACUUM FULL product_info;
```

Step 5 Update statistics of the **product_info** table.

```
ANALYZE product_info;
```

----End

Deleting Resources

Step 1 If you have performed queries after importing data, run the following statement to delete the target table:

```
DROP TABLE product_info;
```


If the following output is displayed, the foreign table has been deleted:

```
DROP TABLE
```

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

```
DROP FOREIGN TABLE product_info_ext;
```

If the following output is displayed, the foreign table has been deleted:

```
DROP FOREIGN TABLE
```

----End

1.3 Using GDS to Import Table Data from a Remote Server to a GaussDB(DWS) Cluster

Overview

This practice demonstrates how to use General Data Service (GDS) to import data from a remote server to GaussDB(DWS).

GaussDB(DWS) allows you to import data in TXT, CSV, or FIXED format.

In this tutorial, you will:

- Generate the source data files in CSV format to be used in this tutorial.
- Upload the source data files to a data server.
- Create foreign tables used for importing data from a data server to GaussDB(DWS) through GDS.
- Start GaussDB(DWS), create a table, and import data to the table.
- Analyze import errors based on the information in the error table and correct these errors.

Preparing an ECS as the GDS Server

For details about how to purchase an ECS, see **Purchasing an ECS** in the *Elastic Cloud Server Getting Started*. After the purchase, log in to the ECS by referring to **Logging In to a Linux ECS**.

NOTE

- The ECS OS must be supported by the GDS package.
- The ECS and GaussDB(DWS) are in the same region, VPC, and subnet.
- The ECS security group rule must allow access to the GaussDB(DWS) cluster, that is, the inbound rule of the security group is as follows:
  - **Protocol**: **TCP**
  - **Port**: **5000**
  - **Source**: Select **IP Address** and enter the IP address of the GaussDB(DWS) cluster, for example, **192.168.0.10/32**.
- If the firewall is enabled in the ECS, ensure that the listening port of GDS is enabled on the firewall:

```
iptables  -I INPUT -p tcp -m tcp --dport <gds_port> -j ACCEPT
```


Downloading the GDS Package

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

Step 2 In the navigation tree on the left, choose **Management** > **Client Connections**.

Step 3 Select the GDS client of the corresponding version from the drop-down list of **CLI Client**.

Select a version based on the cluster version and the OS where the client is installed.

Step 4 Click **Download**.

----End

Preparing Source Data Files

- Data file **product_info0.csv**

```
100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
```

- Data file **product_info1.csv**

```
200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
```

- Data file **product_info2.csv**

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

Step 1 Create a text file, open it using a local editing tool (for example, Visual Studio Code), and copy the sample data to the text file.

Step 2 Choose **Format** > **Encode in UTF-8 without BOM**.

Step 3 Choose **File** > **Save as**.

Step 4 In the displayed dialog box, enter the file name, set the file name extension to .csv, and click **Save**.

Step 5 Log in to the GDS server as user **root**.

Step 6 Create the **/input_data** directory for storing the data file.

```
mkdir -p /input_data
```

Step 7 Use MobaXterm to upload source data files to the created directory.

----End

Installing and Starting GDS

Step 1 Log in to the GDS server as user **root** and create the **/opt/bin/dws** directory for storing the GDS package.

```
mkdir -p /opt/bin/dws
```

Step 2 Upload the GDS package to the created directory.

For example, upload the **dws_client_8.1.*x*_redhat_x64.zip** package to the created directory.

Step 3 Go to the directory and decompress the package.

```
cd /opt/bin/dws
unzip dws_client_8.1.x_redhat_x64.zip
```

Step 4 Create a user (**gds_user**) and the user group (**gdsgrp**) to which the user belongs. This user is used to start GDS and must have the permission to read the source data file directory.

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

Step 5 Change the owner of the GDS package and source data file directory to **gds_user** and change the user group to **gdsgrp**.

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

Step 6 Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.*x* or earlier, skip **Step 7** and go to **Step 8**.

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

Step 7 Execute the script on which the environment depends (applicable only to 8.1.*x*).

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

Step 8 Start GDS.

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

Replace the italic parts as required.

- **-d** *dir*: directory for storing data files that contain data to be imported. This practice uses **/input_data/** as an example.
- **-p** *ip:port*: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with GaussDB(DWS). The port number ranges from 1024 to 65535. The default value is **8098**. This practice uses **192.168.0.90:5000** as an example.
- **-H** *address_string*: hosts that are allowed to connect to and use GDS. The value must be in CIDR format. Set this parameter to enable a GaussDB(DWS) cluster to access GDS for data import. Ensure that the network segment covers all hosts in a GaussDB(DWS) cluster.
- **-l** *log_file*: GDS log directory and log file name. This practice uses **/opt/bin/dws/gds/gds_log.txt** as an example.

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

----End

Creating a Foreign Table

Step 1 Use an SQL client to connect to the GaussDB(DWS) database.

Step 2 Create the following foreign table:

CAUTION

LOCATION: Replace it with the actual GDS address and port number.

```
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'gsfs://192.168.0.90:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

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

```
CREATE FOREIGN TABLE
```

----End

Importing Data

Step 1 Run the following statements to create the **product_info** table in GaussDB(DWS) to store imported data:

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
```



```
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
)
WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

Step 2 Import data from source data files to the **product_info** table through the foreign table **product_info_ext**.

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

If the following information is displayed, the data is successfully imported:

```
INSERT 0 20
```

Step 3 Run the **SELECT** statement to view the data imported to GaussDB(DWS).

```
SELECT count(*) FROM product_info;
```

If the following information is displayed, the data has been imported:

```
count
-------
    20
(1 row)
```

Step 4 Run **VACUUM FULL** on the **product_info** table.

```
VACUUM FULL product_info
```

Step 5 Update statistics of the **product_info** table.

```
ANALYZE product_info;
```

----End

Stopping GDS

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

Step 2 Perform the following operations to stop GDS:

1. Query the GDS process ID. The GDS process ID is **128954**.

   ```
   ps -ef|grep gds
   gds_user 128954     1  0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -
   l /opt/bin/gds/gds_log.txt  -D
   gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
   ```

2. Run the **kill** command to stop GDS. **128954** indicates the GDS process ID.

   ```
   kill -9 128954
   ```

----End

Deleting Resources

Step 1 Run the following command to delete the target table **product_info**:

```
DROP TABLE product_info;
```

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

```
DROP TABLE
```


Step 2 Run the following command to delete the foreign table **product_info_ext**:

```
DROP FOREIGN TABLE product_info_ext;
```

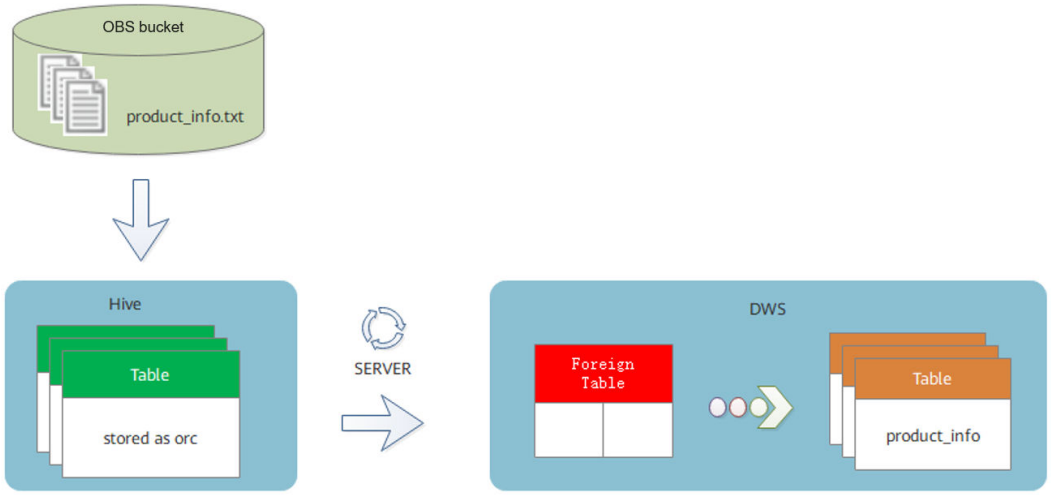
If the following information is displayed, the table has been deleted:

```
DROP FOREIGN TABLE
```

----End

1.4 Importing Table Data from MRS Hive to a GaussDB(DWS) Cluster

In this tutorial, an HDFS foreign table is created to enable GaussDB(DWS) to remotely access or read MRS data sources.



Preparing the Environment

Create a GaussDB(DWS) cluster. Ensure that the MRS and GaussDB(DWS) clusters are in the same region, AZ, and VPC subnet and that the clusters can communicate with each other.

Procedure

This practice takes about 1 hour. The basic process is as follows:

1. Create an MRS cluster deployed with Hive, Spark, and Tez.
2. Upload the local TXT data files to an OBS bucket and import the files to a Hive storage table, and then to an ORC storage table.
3. Create an MRS data source connection.
4. Create a foreign server.
5. Create a foreign table.
6. Import data to a local GaussDB(DWS) table from the foreign table.

Creating an MRS Cluster

Step 1 Log in to the **HUAWEI CLOUD console**, choose **Analytics** > **MapReduce Service** and click **Buy Cluster**. Click the **Custom Config** tab, configure software parameters, and click **Next**.

Table 1-1 Software configuration

Parameter	Value
Region	CN-Hong Kong
Cluster Name	mrs_01
Version	Normal
Cluster Version	MRS 1.9.2 (recommended) **NOTE** • For clusters of version 8.1.1.300 and later, MRS clusters support versions 1.6.*, 1.7.*, 1.8.*, 1.9.*, 2.0.*, 3.0.*, 3.1.*, and later (* indicates a number). • For clusters earlier than version 8.1.1.300, MRS clusters support versions 1.6.*, 1.7.*, 1.8.*, 1.9.*, and 2.0.* (*indicates a number).
Cluster Type	Analysis Cluster
Metadata	Local

Step 2 Configure hardware parameters and click **Next**.

Table 1-2 Hardware configuration

Parameter	Value
Billing Mode	Pay-per-use
AZ	AZ2
VPC	vpc-01
Subnet	subnet-01
Security Group	Auto create
EIP	*10.x.x.x*
Enterprise Project	Default
Master	2
Analysis Core	3
Analysis Task	0

Step 3 When you have completed the advanced settings based on the following table, click **Buy Now** and wait for about 15 minutes. The cluster is successfully created.

Table 1-3 Advanced configuration

Parameter	Value
Tag	test01
Hostname Prefix	(Optional) Prefix for the name of an ECS or BMS in the cluster.
Auto Scaling	Retain the default value.
Bootstrap Action	Retain the default value. MRS 3.x does not support this parameter.
Agency	Retain the default value.
Data Disk Encryption	This function is disabled by default. Retain the default value.
Alarm	Retain the default value.
Rule Name	Retain the default value.
Topic Name	Select a topic.
Kerberos Authentication	This parameter is enabled by default.
Username	admin
Password	This password is used to log in to the cluster management page.
Confirm Password	Enter the password of user **admin** again.
Login Mode	Password
Username	root
Password	This password is used to remotely log in to the ECS.
Confirm Password	Enter the password of user **root** again.
Secure Communications	Select **Enable**.

----End

Preparing the ORC Table Data Source of MRS

Step 1 Create a **product_info.txt** file on the local PC, copy the following data to the file, and save the file to the local PC.

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



```
seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

Step 2 Log in to the OBS console, click **Create Bucket**, configure the following parameters, and click **Create Now**.

Table 1-4 Bucket parameters

Parameter	Value
Region	CN-Hong Kong
Data Redundancy Policy	Single-AZ Storage
Bucket Name	mrs-datasource
Default Storage Class	Standard
Bucket Policy	Private
Default Encryption	Disable
Direct Reading	Disable
Enterprise Project	Default
Tags	N/A

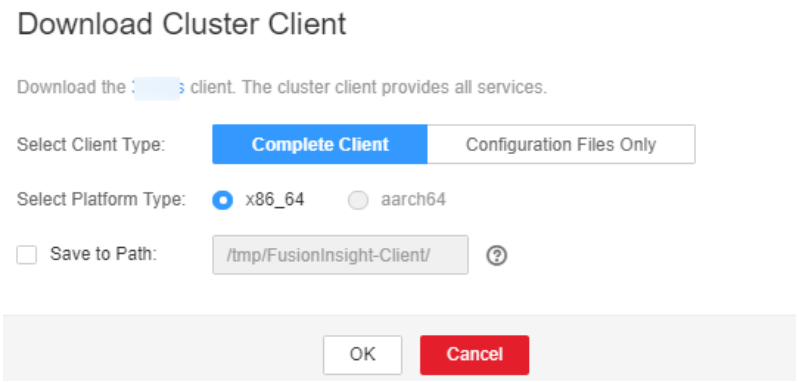
Step 3 After the bucket is created, click the bucket name and choose **Object** > **Upload Object** to upload the **product_info.txt** file to the OBS bucket.

Step 4 Switch back to the MRS console and click the name of the created MRS cluster. On the **Dashboard** page, click the Synchronize button next to **IAM User Sync**. The synchronization takes about 5 minutes.

Step 5 Click **Nodes** and click a master node. On the displayed page, switch to the **EIPs** tab, click **Bind EIP**, select an existing EIP, and click **OK**. If no EIP is available, create one. Record the EIP.

Step 6 Download the client.

1. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.
2. In the **Access MRS Manager** dialog box, click **OK**. You will be redirected to the MRS Manager login page. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
3. Choose **Cluster** > *Name of the desired cluster* > **Dashboard** > **More** > **Download Client**. The **Download Cluster Client** dialog box is displayed.



NOTE

To obtain the client of an earlier version, choose **Services** > **Download Client** and set **Select Client Type** to **Configuration Files Only**.

Step 7 Determine the active master node.

1. Use SSH to log in to the preceding node as user **root**. Run the following command to switch to user **omm**:

   su - omm
2. Run the following command to query the active master node. In the command output, the node whose value of **HAActive** is **active** is the active master node.

   sh ${BIGDATA_HOME}/om-0.0.1/sbin/status-oms.sh

Step 8 Log in to the active master node as user **root** and update the client configuration of the active management node.

cd /opt/client

sh refreshConfig.sh /opt/client *Full_path_of_client_configuration_file_package*

In this tutorial, run the following command:

sh refreshConfig.sh /opt/client /tmp/MRS-client/MRS_Services_Client.tar

Step 9 Switch to user **omm** and go to the directory where the Hive client is located.

su - omm

cd /opt/client

Step 10 Create the **product_info** table whose storage format is TEXTFILE on Hive.

1. Import environment variables to the **/opt/client** directory.

   source bigdata_env

2. Log in to the Hive client.

   beeline

3. Run the following SQL commands in sequence to create a demo database and the **product_info** table:

```
CREATE DATABASE demo;
USE demo;
DROP TABLE product_info;

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

Step 11 Import the **product_info.txt** file to Hive.

1. Switch back to the MRS cluster, click **Files** > **Import Data**.
2. **OBS Path**: Find the **product_info.txt** file in the created OBS bucket and click **Yes**.
3. **HDFS Path**: Select **/user/hive/warehouse/demo.db/product_info/** and click **Yes**.
4. Click **OK** to import the **product_info** table data.

Step 12 Create an ORC table and import data to the table.

1. Run the following SQL commands to create an ORC table:

```
DROP TABLE product_info_orc;

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



```
row format delimited fields terminated by ','
stored as orc;
```

2. Insert data in the **product_info** table into the Hive ORC table **product_info_orc**.

```
INSERT INTO product_info_orc select * from product_info;
```

3. Query whether the data import is successful.

```
SELECT * FROM product_info_orc;
```

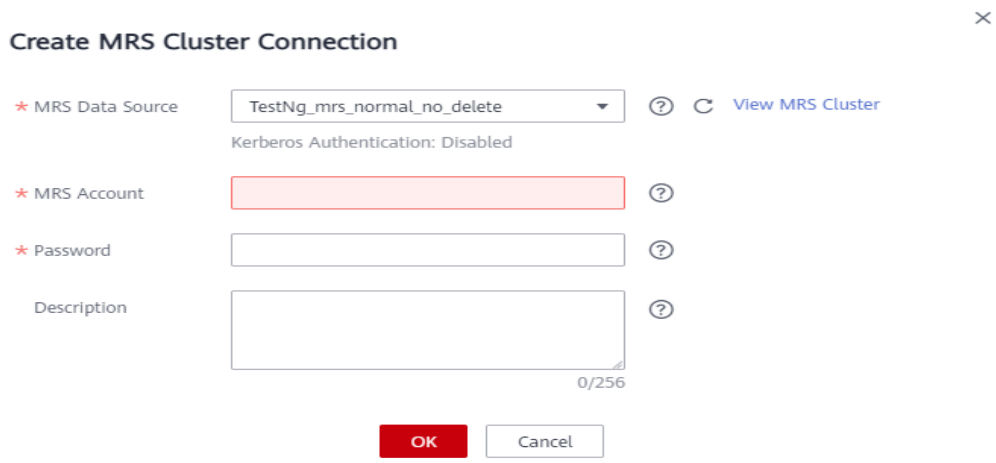
----End

Creating an MRS Cluster Connection

Step 1 Log in to the GaussDB(DWS) console and click the created data warehouse cluster. Ensure that the GaussDB(DWS) and MRS clusters are in the same region, AZ, and VPC subnet.

Step 2 Click the **MRS Data Source** tab and click **Create MRS Cluster Connection**.

Step 3 Select data source **mrs_01** created in the previous step, enter the MRS account name **admin** and its password, and click **OK**.



----End

Creating a Foreign Server

Step 1 Use Data Studio to connect to the created GaussDB(DWS) cluster.

Step 2 Create a user *dbuser* that has the permission for creating databases.

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

Step 3 Switch to user *dbuser*.

```
SET ROLE dbuser PASSWORD 'password';
```

Step 4 Create a database *mydatabase*.

```
CREATE DATABASE mydatabase;
```

Step 5 Perform the following steps to switch to database *mydatabase*:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and select **Refresh** from the shortcut menu. The new database is displayed.

2. Right-click the database name *mydatabase* and select **Connect to DB** from the shortcut menu.
3. Right-click the database name *mydatabase* and select **Open Terminal** from the shortcut menu. The SQL command window for connecting to a database is displayed. Perform the following steps in the window.

Step 6 Grant the permission to create external servers to user dbuser. In 8.1.1 and later versions, you also need to grant the permission to use the public mode.

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
In GRANT ALL ON SCHEMA public TO dbuser; //8.1.1 and later versions, common users do not have permission on the public mode and need to grant permission. In versions earlier than 8.1.1, you do not need to perform this operation.
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. *dbuser* indicates the username of **CREATE SERVER**.

Step 7 Grant user *dbuser* the permission for using foreign tables.

```
ALTER USER dbuser USEFT;
```

Step 8 Switch to the Postgres database and query the foreign server automatically created by the system after the MRS data source is created.

```
SELECT * FROM pg_foreign_server;
```

Information similar to the following is displayed:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                       srvoptions
-----------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                  |       10 |  13673 |         |            |        |
 gsmpp_errorinfo_server                        |       10 |  13678 |         |            |        |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |        |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

Step 9 Switch to database *mydatabase* and switch to user *dbuser*.

```
SET ROLE dbuser PASSWORD 'password';
```

Step 10 Create a foreign server.

The server name, address, and configuration path must be the same as those in **Step 8**.

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA WRAPPER
HDFS_FDW
OPTIONS
(
address '192.168.1.245:9820,192.168.1.218:9820',   //The intranet IP addresses of the active and standby master nodes on the MRS management plane, which can be used to communicate with GaussDB(DWS).
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

Step 11 View the foreign server.

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

The server is successfully created if information similar to the following is displayed:

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



```
+-----------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |        |           |       |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

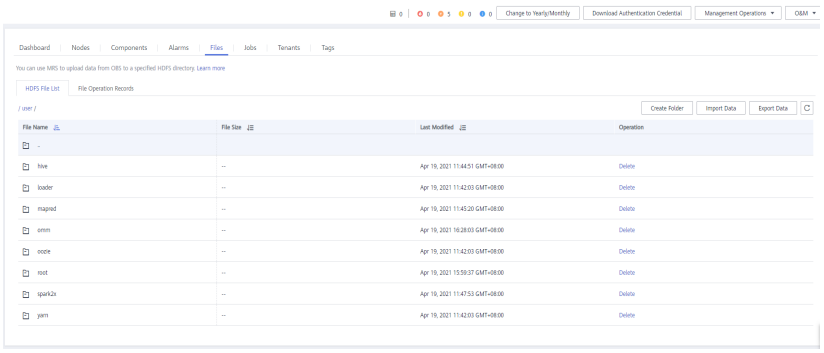
----End

Creating a Foreign Table

Step 1 Obtain the **product_info_orc** file path of Hive.

1. Log in to the MRS console.
2. Choose **Cluster** > **Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.
3. Click the **Files** and click **HDFS File List**.
4. Go to the storage directory of the data to be imported to the GaussDB(DWS) cluster and record the path.

Figure 1-1 Checking the data storage path on MRS



Step 2 Create a foreign table. Set **SERVER** to the name of the external server created in **Step 10** and **foldername** to the path obtained in **Step 1**.

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

CREATE FOREIGN TABLE foreign_product_info
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
encoding 'utf8',
foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

----End

Importing Data

Step 1 Create a local table for data import.

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price               integer       ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Import data to the target table from the foreign table.

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

Step 3 Query the import result.

```
SELECT * FROM product_info;
```

----End

1.5 Enabling Cross-Cluster Access of Hive Metastore Through an External Schema

GaussDB(DWS) 3.0 (with decoupled storage and compute) allows users to access data stored in MRS Hive (including when Hive is connected to HDFS or OBS) by simply creating an external schema. This topic describes how to enable cross-cluster access of the data stored in a Hive metastore.

Preparing the Environment

- You have created a GaussDB(DWS) 3.0 cluster. The MRS and GaussDB(DWS) clusters are in the same region, AZ, and VPC subnet, and can communicate with each other.
- You have obtained the AK and SK for your Huawei Cloud account.

Constraints

- Currently, only the SELECT, INSERT, and INSERT OVERWRITE operations can be performed on tables in the Hive database through external schemas.
- MRS supports two types of data sources. For details, see **Table 1-5**.

Table 1-5 Operations supported by the two types of MRS data sources

Data Source	Table Type	Operation	TEXT	CSV	PARQUET	ORC
HDFS	Non-partitioned table	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
	Partitioned table	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
OBS	Non-partitioned table	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
	Partitioned table	SELECT	x	x	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	x

- Transaction atomicity is no longer guaranteed. If a transaction fails, data consistency cannot be guaranteed. Rollback is not supported.
- GRANT and REVOKE operations cannot be performed on tables created on Hive using external schemas.
- Concurrency support: Concurrent read and write operations on GaussDB(DWS), Hive, and Spark may cause dirty reads. Concurrent operations including INSERT OVERWRITE on the same non-partitioned table or the same partition of the same partitioned table may not guarantee the expected result. Therefore, avoid such operations.
- Hive metastores do not support the federation mechanism.

Procedure

This practice takes approximately 1 hour. The basic procedure is as follows:

1. Create an MRS analysis cluster. (The Hive component must be selected.)
2. Create a table on Hive.

3. Insert data on Hive, or upload a local TXT file to an OBS bucket, then import the file to Hive from the OBS bucket, and import the file from the TXT storage table to the ORC storage table.
4. Create a connection to the MRS data source.
5. Create a foreign server.
6. Create an external schema.
7. Use the external schema to import data to or read data from Hive tables.

Creating an MRS Cluster

Step 1 Log in to the management console, and choose **Analytics** > **MapReduce Service**.

Step 2 Click **Buy Cluster** and select **Custom Config**.

Step 3 Configure software parameters, and click **Next**.

Table 1-6 Software configuration

Parameter	Value
Region	China-Hong Kong
Cluster Name	mrs_01
Version	Normal
Cluster Version	**MRS 3.1.3** (recommended) **NOTE** MRS clusters support 3.0.*, 3.1.*, and later versions (* indicates a number).
Cluster Type	Analysis Cluster
Metadata	Local

Step 4 Configure hardware parameters and click **Next**.

Table 1-7 Hardware configuration

Parameter	Value
Billing Mode	Pay-per-use
AZ	AZ2
VPC	vpc-01
Subnet	subnet-01
Security Group	Auto create
EIP	*10.x.x.x*
Enterprise Project	default

Parameter	Value
Master	2
Analysis Core	3
Analysis Task	0

Step 5 Configure the advanced settings based on the following table, and click **Buy Now**. Cluster creation takes approximately 15 minutes.

Table 1-8 Advanced settings

Parameter	Value
Tag	test01
Hostname Prefix	(Optional) Prefix for the names of ECSs or BMSs in the cluster.
Auto Scaling	Retain the default value.
Bootstrap Action	Retain the default value. MRS 3.x does not support this parameter.
Agency	Retain the default value.
Data Disk Encryption	This function is disabled by default. Retain the default value.
Alarm	Retain the default value.
Rule Name	Retain the default value.
Topic Name	Select a topic.
Kerberos Authentication	This function is enabled by default.
User Name	admin
Password	This password is used for logging in to the cluster management page.
Confirm Password	Enter the password of user **admin** again.
Login Mode	Password
User Name	root
Password	This password is used to remotely log in to an ECS.
Confirm Password	Enter the password of user **root** again.
Agency	In **Advanced Settings**, set **Agency** to the preset agency **MRS_ECS_DEFAULT_AGENCY** of MRS in IAM.

Parameter	Value
Secure Communications	Select **Enable**.

----End

Preparing an ORC Table

Step 1 Create a **product_info.txt** file on the local PC, copy the following data to the file, and save the file to the local PC.

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

Step 2 Log in to OBS Console, click **Create Bucket**, set the following parameters, and click **Create Now**.

Table 1-9 Bucket parameters

Parameter	Value
Region	China-Hong Kong
Data Redundancy Policy	Single-AZ Storage

Parameter	Value
Bucket Name	mrs-datasource
Default Storage Class	Standard
Bucket Policy	Private
Default Encryption	Disable
Direct Reading	Disable
Enterprise Project	default
Tag	-

Step 3 After successful bucket creation, switch back to the MRS console and click the name of the created MRS cluster. On the **Dashboard** page, click the Synchronize button next to **IAM User Sync**. The synchronization takes around 5 minutes.

Step 4 Click **Nodes** and click a master node. On the displayed page, switch to the **EIPs** tab, click **Bind EIP**, select an existing EIP, and click **OK**. If no EIP is available, create one. Record the EIP.

Step 5 (Optional) Connect Hive to OBS.

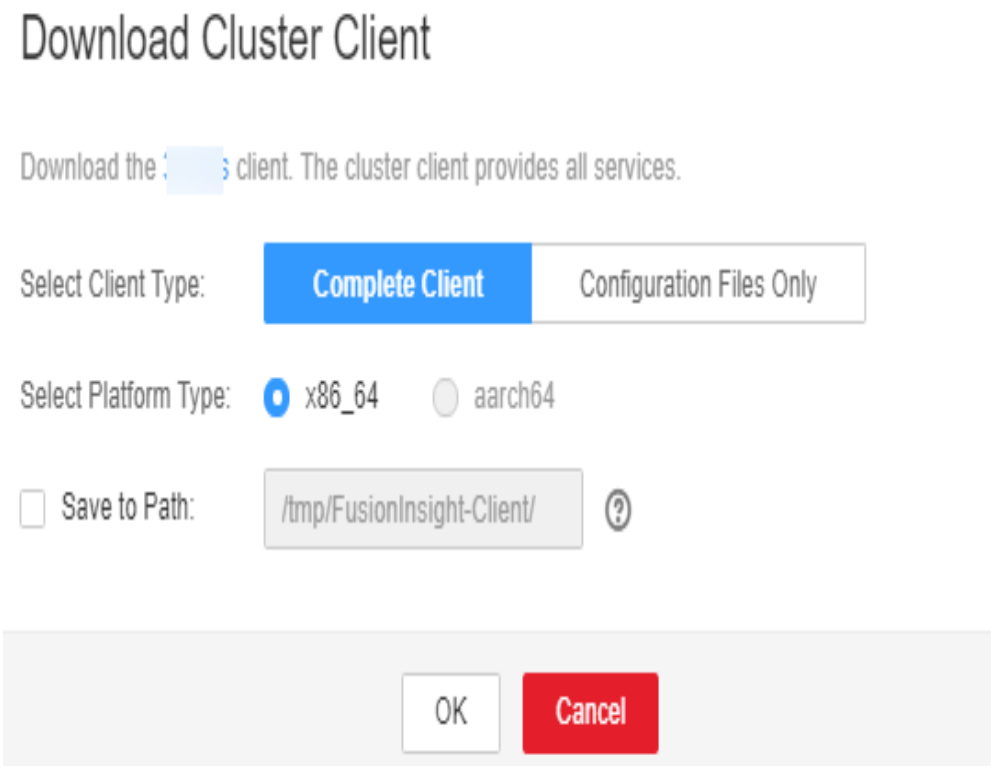
NOTE

Perform this step when Hive interconnects with OBS. Skip this step when Hive interconnects with HDFS.

1. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that an EIP needs to be bound, bind an EIP first.
2. In the **Access MRS Manager** dialog box, click **OK**. You will be redirected to the MRS Manager login page. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
3. Interconnect Hive with OBS by referring to **Interconnecting Hive with OBS**.

Step 6 Download the client.

1. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.
2. In the **Access MRS Manager** dialog box, click **OK**. You will be redirected to the MRS Manager login page. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
3. Choose **Services** > **Download Client**. Set **Client Type** to **Only configuration files** and set **Download To** to **Server**. Click **OK**.



Step 7 Log in to the active master node as user **root** and update the client configuration of the active management node.

cd /opt/client

sh refreshConfig.sh /opt/client *Full_path_of_client_configuration_file_package*

In this example, run the following command:

sh refreshConfig.sh /opt/client /tmp/MRS-client/MRS_Services_Client.tar

Step 8 Switch to user **omm** and go to the directory where the Hive client is located.

su - omm

cd /opt/client

Step 9 Create the **product_info** table whose storage format is TEXTFILE on Hive.

1. Import environment variables to the **/opt/client** directory.

   source bigdata_env

   NOTE

   If **find: 'opt/client/Hudi': Permission denied** is displayed, ignore it. This does not affect subsequent operations.

2. Log in to the Hive client.

   a. If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission for creating Hive tables. . Configure a role with the required permissions. For details, see **Creating a User** in the *MapReduce Service User Guide*. Bind a role to the user. If Kerberos

authentication is not enabled for the current cluster, there is no need to run the following command:

kinit *MRS cluster user*

b. Run the following command to start the Hive client:

beeline

3. Run the following SQL commands in sequence to create a demo database and the **product_info** table:

```
CREATE DATABASE demo;
USE demo;
DROP TABLE product_info;

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

Step 10 Import the **product_info.txt** file to Hive.

- Hive is interconnected with OBS: Go back to OBS Console, click the name of the bucket, choose **Objects** > **Upload Object**, and upload the **product_info.txt** file to the path of the **product_info table** in the OBS bucket.
- Hive is interconnected with HDFS: Import the **product_info.txt** file to the HDFS path **/user/hive/warehouse/demo.db/product_info/**. For details about how to import data to an MRS cluster, see section **Managing Data Files** in the *MapReduce Service User Guide*..

Step 11 Create an ORC table and import data to the table.

1. Run the following SQL commands to create an ORC table:

```
DROP TABLE product_info_orc;

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

2. Insert data in the **product_info** table into the Hive ORC table **product_info_orc**.

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


3. Query whether the data import is successful.

```
SELECT * FROM product_info_orc;
```

----End

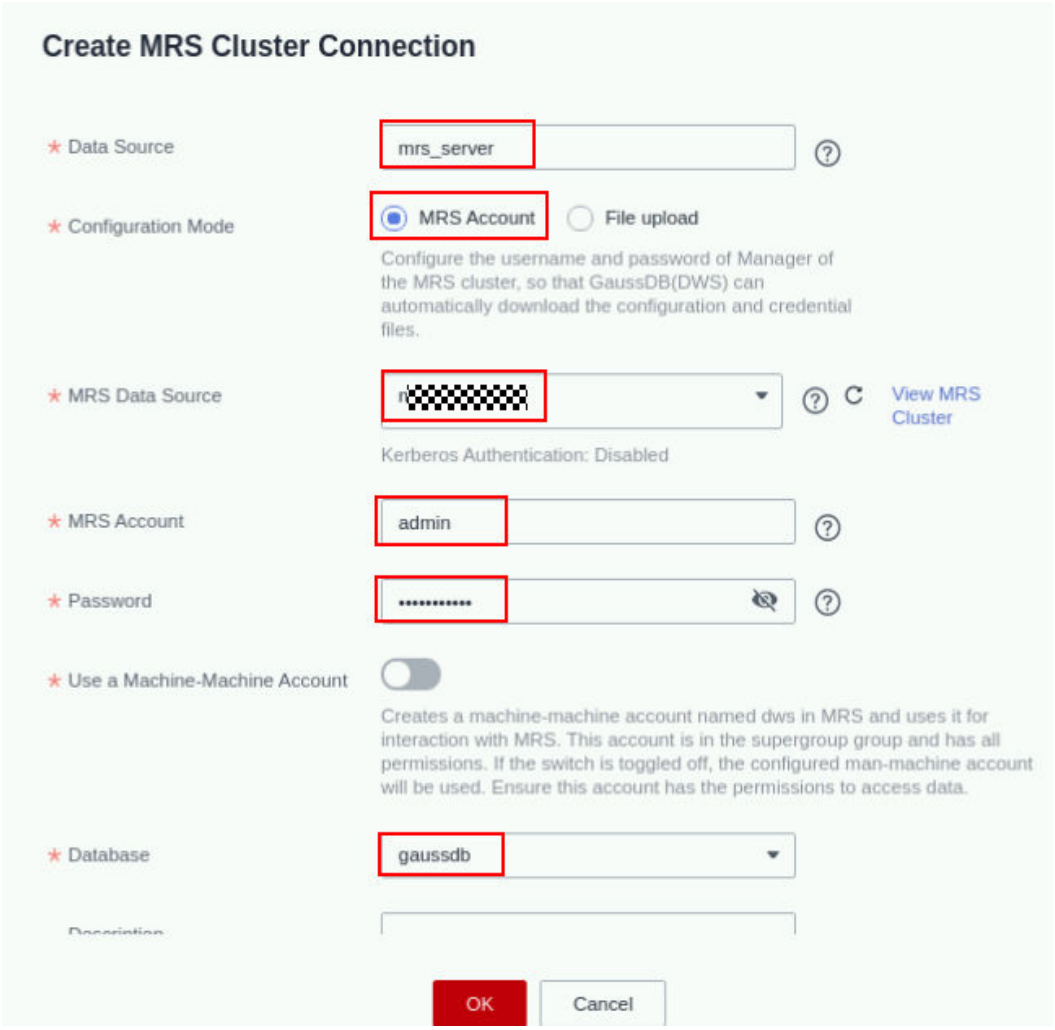
Creating an MRS Cluster Connection

Step 1 Log in to the GaussDB(DWS) console and click the created data warehouse cluster. Ensure that the GaussDB(DWS) and MRS clusters are in the same region, AZ, and VPC subnet.

Step 2 Click the **MRS Data Source** tab and click **Create MRS Cluster Connection**.

Step 3 Set the following parameters and click **OK**.

- **Data Source**: mrs_server
- **Configuration Mode**: **MRS Account**
- **MRS Data Source**: Select the created **mrs_01** cluster.
- **MRS Account**: admin
- **Password**: Enter the password of the **admin** user created for the MRS data source.



----End

Creating a Foreign Server

Perform this step only when Hive is connected to OBS. Skip this step if Hive is connected to HDFS.

Step 1 Use Data Studio to connect to the created GaussDB(DWS) cluster.

Step 2 Run the following statement to create a foreign server. {AK value} and {SK value} are obtained from **Preparing the Environment**.

NOTICE

Hard-coded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.

```
CREATE SERVER obs_servevr FOREIGN DATA WRAPPER DFS_FDW
OPTIONS
(
address'obs.xxx.com:5443', //Address for accessing OBS.
encrypt 'on',
access_key '{AK value}',
secret_access_key '{SK value}',
 type 'obs'
);
```

Step 3 View the foreign server.

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

The server is successfully created if information similar to the following is displayed:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                      srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------
 obs_server |    16476 |  14337 |         |            |
{address=obs.xxx.com:5443,type=obs,encrypt=on,access_key=***,secret_access_key=***}
(1 row)
```

----End

Creating an External Schema

Step 1 Obtain the internal IP address and port number of the Hive metastore service and the name of the Hive database to be accessed.

1. Log in to the MRS console.
2. Choose **Cluster** > **Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.
3. Click **Go to manager** on the O&M Management page and enter the username and password to log in to the FusionInsight management page.
4. Click **Cluster**, **Hive**, **Configuration**, **All Configurations**, **MetaStore**, and **Port** in sequence, and record the value of **hive.metastore.port**.
5. Click **Cluster**, **Hive**, and **Instance** in sequence, and record the MetaStore management IP address of the host whose name contains **master1**.

Step 2 Create an external schema.


```
//When interconnecting Hive with OBS: Set Server to the name of the external server created in Step 2, DATABASE to the database created on Hive, METAADDRESS to the IP address and port number of the Hive metastore service recorded in Step 1, and CONFIGURATION to the default configuration path of the MRS data source.
DROP SCHEMA IF EXISTS ex1;

CREATE EXTERNAL SCHEMA ex1
   WITH SOURCE hive
      DATABASE 'demo'
      SERVER obs_server
      METAADDRESS '***.***.***.***:***'
      CONFIGURATION '/MRS/gaussdb/mrs_server'

//When interconnecting Hive with HDFS: Set Server to mrs_server (name of the data source created in Creating an MRS Cluster Connection), METAADDRESS to the IP address and port number of the Hive metastore service recorded in Step 1, and CONFIGURATION to the default configuration path of the MRS data source.
DROP SCHEMA IF EXISTS ex1;

CREATE EXTERNAL SCHEMA ex1
   WITH SOURCE hive
      DATABASE 'demo'
      SERVER mrs_server
      METAADDRESS '***.***.***.***:***'
      CONFIGURATION '/MRS/gaussdb/mrs_server'
```

Step 3 Check the created external schema.

```
SELECT * FROM pg_namespace WHERE nspname='ex1';
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE nspname = 'ex1');

                  nspid                  | srvname | source | address | database | confpath
|                                          ensoptions   | catalog
-----------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------
               16393                     |    obs_server |  hive | ***.***.***.***:***        |  demo          | ***
|                    |
(1 row)
```

----End

Importing Data

Step 1 Create a local table for data import.

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

Step 2 Import the target table from the Hive table.

```
INSERT INTO product_info SELECT * FROM ex1.product_info_orc;
```

Step 3 Query the import result.

```
SELECT * FROM product_info;
```

----End

Exporting Data

Step 1 Create a local source table.

```
DROP TABLE IF EXISTS product_info_export;
CREATE TABLE product_info_export
(
    product_price               integer       ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
) ;
INSERT INTO product_info_export SELECT * FROM product_info;
```

Step 2 Create a target table on Hive.

```
DROP TABLE product_info_orc_export;

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

Step 3 Import data from the local source table to the Hive table.

```
INSERT INTO ex1.product_info_orc_export SELECT * FROM product_info_export;
```

Step 4 Query the data import result on Hive.

```
SELECT * FROM product_info_orc_export;
```

----End

1.6 Migrating Data Between GaussDB(DWS) Clusters Using Foreign Tables

In the era of big data convergent analysis, GaussDB(DWS) clusters in the same region can communicate with each other. This practice demonstrates how to import data from a remote GaussDB(DWS) cluster to the local GaussDB(DWS) cluster using foreign tables.

The demonstration procedure is as follows: Install the gsql database client on an ECS, connect to GaussDB(DWS) using gsql, and import data from the remote GaussDB(DWS) using a foreign table.

General Procedure

This practice takes about 40 minutes. The basic process is as follows:

1. **Preparations**
2. **Creating an ECS**
3. **Creating a Cluster and Downloading the Tool Package**
4. **Importing Data Sources Using GDS**
5. **Importing Remote GaussDB(DWS) Data Using a Foreign Table**

Preparations

You have registered a Huawei account and enabled Huawei Cloud. The account cannot be in arrears or frozen.

Creating an ECS

For details, see **Purchasing an ECS**. After purchasing an ECS, log in to the ECS by referring to **Logging In to a Linux ECS**.

NOTICE

When creating an ECS, ensure that the ECS and the GaussDB(DWS) clusters to be created are in the same VPC subnet and in the same region and AZ . The ECS OS is the same as that of the gsql client or GDS (CentOS 7.6 is used as an example), and the password is used for login.

Creating a Cluster and Downloading the Tool Package

Step 1 Log in to the Huawei Cloud management console.

Step 2 Choose **Service List** > **Analytics** > **Data Warehouse Service**. On the page that is displayed, click **Create Cluster** in the upper right corner.

Step 3 Configure the parameters according to **Table 1-10**.

Table 1-10 Software configuration

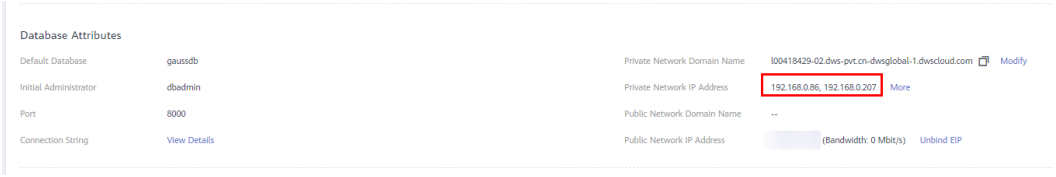
Parameter	Configuration
Region	Select the **CN-Hong Kong** region. **NOTE** • **CN-Hong Kong** is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region. • Ensure that GaussDB(DWS) and the ECS are in the same region, AZ, and VPC subnet.
AZ	AZ2
Resource	Standard data warehouse

Parameter	Configuration
Compute Resource	ECS
Storage Type	Cloud SSD
CPU Architecture	x86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPUs | 128 GB | 2000 GB SSD) **NOTE** If this flavor is sold out, select other AZs or flavors.
Hot Storage	100 GB/node
Nodes	3
Cluster Name	dws-demo01
Administrator Account	dbadmin
Administrator Password	User-defined password
Confirm Password	password
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24) **NOTICE** Ensure that the cluster and the ECS are in the same VPC subnet.
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1 Mbit/s
Advanced Settings	Default

Step 4 Confirm the information, click **Next**, and then click **Submit**.

Step 5 Wait for about 10 minutes. After the cluster is created, click the cluster name to go to the **Basic Information** page. Choose **Network**, click a security group name, and verify that a security group rule has been added. In this example, the client IP address is 192.168.0.*x* (the private network IP address of the ECS where gsql is located is 192.168.0.90). Therefore, you need to add a security group rule in which the IP address is 192.168.0.0/24 and port number is 8000.

Step 6 Return to the **Basic Information** tab of the cluster and record the value of **Private Network IP Address**.



Step 7 Return to the home page of the GaussDB(DWS) console. In the navigation tree on the left, choose **Management** > **Client Connections**, select the appropriate ECS OS (such as **Redhat x86_64** for **CentOS 7.6**), and click **Download** to save the tool package to your local PC. The tool package contains the gsql client and GDS.

Step 8 Repeat **Step 1** to **Step 6** to create a second GaussDB(DWS) cluster and set its name to **dws-demo02**.

----End

Preparing Source Data

Step 1 Create the following three CSV files in the specified directory on the local PC:

- Data file **product_info0.csv**

```
100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
```

- Data file **product_info1.csv**

```
200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
```

- Data file **product_info2.csv**

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

Step 2 Log in to the created ECS as user **root** and run the following command to create a data source file directory:

mkdir -p /input_data

Step 3 Use a file transfer tool to upload the preceding data files to the **/input_data** directory of the ECS.

----End

Importing Data Sources Using GDS

Step 1 Log in to the ECS as user **root** and use a file transfer tool to upload the downloaded tool package in **Step 7** to the **/opt** directory.

Step 2 Decompress the tool package in the **/opt** directory.

```
cd /opt
unzip dws_client_8.1.x_redhat_x64.zip
```

Step 3 Create a GDS user and change the owners of the data source and GDS directories.

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
chown -R gds_user:gdsgrp /opt/gds
chown -R gds_user:gdsgrp /input_data
```

Step 4 Switch to user **gds_user**.

```
su - gds_user
```

Step 5 Import the GDS environment variables.

NOTE

This step is required only for 8.1.*x* or later. For earlier versions, skip this step.

```
cd /opt/gds/bin
source gds_env
```

Step 6 Start GDS.

/opt/gds/bin/gds -d /input_data/ -p *192.168.0.90:5000* **-H** 192.168.0.0/24 **-l /opt/gds/gds_log.txt -D**

- **-d** *dir*: directory for storing data files that contain data to be imported. This practice uses **/input_data/** as an example.
- **-p** *ip:port*: listening IP address and port for GDS. Set this parameter to the private network IP address of the ECS where GDS is installed so that GDS can communicate with GaussDB(DWS). In this example, **192.168.0.90:5000** is used.
- **-H** *address_string*: hosts that are allowed to connect to and use GDS. The value must be in CIDR format. In this example, the network segment of the GaussDB(DWS) private network IP address is used.
- **-l** *log_file*: GDS log directory and log file name. In this example, **/opt/gds/gds_log.txt** is used.
- **-D**: GDS in daemon mode.

Step 7 Connect to the first GaussDB(DWS) cluster using gsql.

1. Run the **exit** command to switch to user **root**, go to the **/opt** directory of the ECS, and import the environment variables of gsql.

   exit

   cd /opt

   source gsql_env.sh

2. Go to the **/opt/bin** directory and connect to the first GaussDB(DWS) cluster using gsql.

   cd /opt/bin

   gsql -d gaussdb -h 192.168.0.8 -p 8000 -U dbadmin -W *password* -r

   - **-d**: name of the connected database. In this example, the default database **gaussdb** is used.
   - **-h**: private network IP address of the connected GaussDB(DWS) database queried in **Step 6**. In this example, **192.168.0.8** is used.
   - **-p**: GaussDB(DWS) port. The value is **8000**.
   - **-U**: database administrator. The value defaults to **dbadmin**.
   - **-W**: administrator password, which is set during cluster creation in **Step 3**. In this example, replace *password* with your actual password.

Step 8 Create a common user **leo** and grant the user the permission for creating foreign tables.

```
CREATE USER leo WITH PASSWORD 'password';
ALTER USER leo USEFT;
```

Step 9 Switch to user **leo** and create a GDS foreign table.

NOTE

Set **LOCATION** to the GDS listening IP address and port number obtained in **Step 6**, for example, **gsfs://192.168.0.90:5000/***.

```
SET ROLE leo PASSWORD 'password';
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'gsfs://192.168.0.90:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```


Step 10 Create a local table.

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

Step 11 Import data from the GDS foreign table and check whether the data is successfully imported.

```
INSERT INTO product_info SELECT * FROM product_info_ext ;
SELECT count(*) FROM product_info;
```

----End

Importing Remote GaussDB(DWS) Data Using a Foreign Table

Step 1 Connect to the second cluster on the ECS by referring to **Step 7**. Change the connection address to the address of the second cluster. In this example, **192.168.0.86** is used.

Step 2 Create a common user **jim** and grant the user the permission for creating foreign tables and servers. The value of **FOREIGN DATA WRAPPER** is **gc_fdws**.

```
CREATE USER jim WITH PASSWORD 'password';
ALTER USER jim USEFT;
GRANT ALL ON FOREIGN DATA WRAPPER gc_fdw TO jim;
```

Step 3 Switch to user **jim** and create a server.

```
SET ROLE jim PASSWORD 'password';
CREATE SERVER server_remote FOREIGN DATA WRAPPER gc_fdw OPTIONS
  (address '192.168.0.8:8000,192.168.0.158:8000' ,
 dbname 'gaussdb',
 username 'leo',
 password 'password'
);
```

- **address**: private network IP addresses and port number of the first cluster obtained in **Step 6**. In this example, **192.168.0.8:8000** and **192.168.0.158:8000** are used.
- **dbname**: database name of the first connected cluster. In this example, **gaussdb** is used.
- **username**: username of the first connected cluster. In this example, **leo** is used.
- **password**: user password

Step 4 Create a foreign table.

NOTICE

The columns and constraints of the foreign table must be consistent with those of the table to be accessed.

```
CREATE FOREIGN TABLE region
(
   product_price               integer      ,
   product_id                  char(30)     ,
   product_time                date         ,
   product_level               char(10)     ,
   product_name                varchar(200)   ,
   product_type1               varchar(20)    ,
   product_type2               char(10)     ,
   product_monthly_sales_cnt   integer      ,
   product_comment_time        date         ,
   product_comment_num         integer      ,
   product_comment_content     varchar(200)
)
SERVER
   server_remote
OPTIONS
(
   schema_name 'leo',
   table_name 'product_info',
   encoding 'utf8'
);
```

- **SERVER**: name of the server created in the previous step. In this example, **server_remote** is used.
- **schema_name**: schema name of the first cluster to be accessed. In this example, **leo** is used.
- **table_name**: table name of the first cluster to be accessed obtained in **Step 10**. In this example, **product_info** is used.
- **encoding**: The value must be the same as that of the first cluster obtained in **Step 9**. In this example, **utf8** is used.

Step 5 View the created server and foreign table.

```
\des+ server_remote
\d+ region
```

Step 6 Create a local table.

NOTICE

The columns and constraints of the table must be consistent with those of the table to be accessed.

```
CREATE TABLE local_region
(
   product_price               integer      not null,
   product_id                  char(30)     not null,
   product_time                date         ,
   product_level               char(10)     ,
   product_name                varchar(200)   ,
   product_type1               varchar(20)    ,
   product_type2               char(10)     ,
   product_monthly_sales_cnt   integer      ,
   product_comment_time        date         ,
   product_comment_num         integer      ,
```



```
    product_comment_content      varchar(200)
)

WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

Step 7 Import data to the local table using the foreign table.

```
INSERT INTO local_region SELECT * FROM region;
SELECT * FROM local_region;
```

Step 8 Query the foreign table without importing data.

```
SELECT * FROM region;
```

----End

1.7 Exporting ORC Data from a GaussDB(DWS) Cluster to an MRS Cluster

GaussDB(DWS) allows you to export ORC data to MRS using an HDFS foreign table. You can specify the export mode and export data format in the foreign table. Data is exported from GaussDB(DWS) in parallel using multiple DNs and stored in HDFS. In this way, the overall export performance is improved.

Preparing the Environment

Create a GaussDB(DWS) cluster. Ensure that the MRS and GaussDB(DWS) clusters are in the same region, AZ, and VPC subnet and that the clusters can communicate with each other.

Creating an MRS Cluster

Step 1 Log in to the **HUAWEI CLOUD console**, choose **Analytics** > **MapReduce Service** and click **Buy Cluster**. Click the **Custom Config** tab, configure software parameters, and click **Next**.

Table 1-11 Software configuration

Parameter	Example Value
Region	CN-Hong Kong
Cluster Name	mrs_01
Cluster Version	MRS 1.9.2 (recommended) **NOTE** • For clusters of version 8.1.1.300 and later, MRS clusters support versions 1.6.*, 1.7.*, 1.8.*, 1.9.*, 2.0.*, 3.0.*, 3.1.*, and later (* indicates a number). • For clusters earlier than version 8.1.1.300, MRS clusters support versions 1.6.*, 1.7.*, 1.8.*, 1.9.*, and 2.0.* (*indicates a number).
Cluster Type	Analysis Cluster

Step 2 Configure hardware parameters and click **Next**.

Table 1-12 Hardware configuration

Parameter	Example Value
Billing Mode	Pay-per-use
AZ	AZ2
VPC	vpc-01
Subnet	subnet-01
Security Group	Auto create
EIP	*10.x.x.x*
Enterprise Project	default
Master	2
Analysis Core	3
Analysis Task	0

Step 3 Configure the advanced settings based on the following table, click **Buy Now**, and wait for about 15 minutes for the cluster creation to complete.

Table 1-13 Advanced settings

Parameter	Example Value
Tag	test01
Hostname Prefix	(Optional) Prefix for the name of an ECS or BMS in the cluster.
Auto Scaling	Retain the default value.
Bootstrap Action	Retain the default value. MRS 3.x does not support this parameter.
Agency	Retain the default value.
Data Disk Encryption	This function is disabled by default. Retain the default value.
Alarm	Retain the default value.
Rule Name	Retain the default value.
Topic Name	Select a topic.
Kerberos Authentication	This parameter is enabled by default.

Parameter	Example Value
User Name	admin
Password	This password is used to log in to the cluster management page.
Confirm Password	Enter the password of user **admin** again.
Login Mode	Password
User Name	root
Password	This password is used to remotely log in to the ECS.
Confirm Password	Enter the password of user **root** again.
Secure Communications	Select **Enable**.

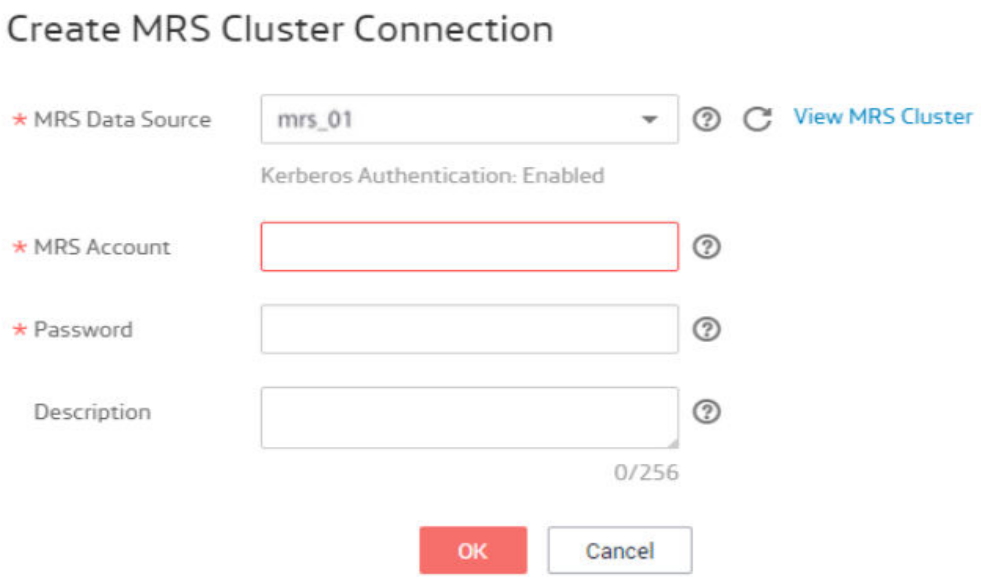
----End

Creating an MRS Cluster Connection

Step 1 Log in to the GaussDB(DWS) console and click the created data warehouse cluster. Ensure that the GaussDB(DWS) and MRS clusters are in the same region, AZ, and VPC subnet.

Step 2 Click the **MRS Data Source** tab and click **Create MRS Cluster Connection**.

Step 3 Select data source **mrs_01** created in the previous step, enter the MRS account name **admin** and its password, and click **OK**.



----End

Creating a Foreign Server

Step 1 Use Data Studio to connect to the created GaussDB(DWS) cluster.

Step 2 Create a user *dbuser* that has the permission for creating databases.

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

Step 3 Switch to user *dbuser*.

```
SET ROLE dbuser PASSWORD 'password';
```

Step 4 Create a database *mydatabase*.

```
CREATE DATABASE mydatabase;
```

Step 5 Perform the following steps to switch to database *mydatabase*:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and choose **Refresh** from the shortcut menu. Then, the new database is displayed.
2. Right-click the database name *mydatabase* and select **Connect to DB** from the shortcut menu.
3. Right-click the database name *mydatabase* and select **Open Terminal** from the shortcut menu. The SQL command window for connecting to a database is displayed. Perform the following steps in the window.

Step 6 Grant the permission to create external servers to user dbuser. In 8.1.1 and later versions, you also need to grant the permission to use the public mode.

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
In GRANT ALL ON SCHEMA public TO dbuser; //8.1.1 and later versions, common users do not have permission on the public mode and need to grant permission. In versions earlier than 8.1.1, you do not need to perform this operation.
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. *dbuser* indicates the username of **CREATE SERVER**.

Step 7 Grant user *dbuser* the permission for using foreign tables.

```
ALTER USER dbuser USEFT;
```

Step 8 Switch to the Postgres database and query the foreign server automatically created by the system after the MRS data source is created.

```
SELECT * FROM pg_foreign_server;
```

Information similar to the following is displayed:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                        srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                   |       10 |  13673 |         |            |
 gsmpp_errorinfo_server                         |       10 |  13678 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

Step 9 Switch to database *mydatabase* and switch to user *dbuser*.

```
SET ROLE dbuser PASSWORD 'password';
```

Step 10 Create a foreign server.

The server name, address, and configuration path must be the same as those in **Step 8**.


```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA WRAPPER
HDFS_FDW
OPTIONS
(
address '192.168.1.245:9820,192.168.1.218:9820',   //The intranet IP addresses of the active and standby
master nodes on the MRS management plane, which can be used to communicate with GaussDB(DWS).
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

Step 11 View the foreign server.

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

The server is successfully created if information similar to the following is displayed:

```
                  srvname                   | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                          srvoptions
----------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |        |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----End

Creating a Foreign Table

Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is **hdfs_server**, the format of the file on HDFS corresponding to the table is ORC, and the data storage path on OBS is **/user/hive/warehouse/product_info_orc/**.

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext;
CREATE FOREIGN TABLE product_info_output_ext
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
foldername '/user/hive/warehouse/product_info_orc/',
   compression 'snappy',
    version '0.12'
) Write Only;
```

Exporting Data

Create an ordinary table **product_info_output**.

```
DROP TABLE product_info_output;
CREATE TABLE product_info_output
(
    product_price                int            ,
    product_id                   char(30)       ,
```



```
    product_time               date          ,
    product_level              char(10)      ,
    product_name                 varchar(200)   ,
    product_type1              varchar(20)    ,
    product_type2              char(10)      ,
    product_monthly_sales_cnt    int           ,
    product_comment_time         date          ,
    product_comment_num          int       ,
    product_comment_content      varchar(200)
)
with (orientation = column,compression=middle)
distribute by hash (product_name);
```

Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

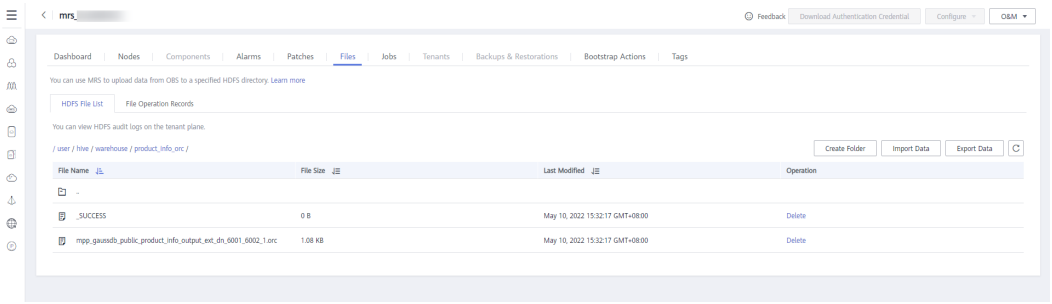
If the following information is displayed, the data is successfully exported:

```
INSERT 0 10
```

Viewing the Export Result

Step 1 Go to the MRS cluster list. Click a cluster name to go to the cluster details page.

Step 2 Click the **Files** tab and click **HDFS File List**. Check the exported ORC file in the **user/hive/warehouse/product_info_orc** directory.



NOTE

ORC data exported from GaussDB(DWS) complies with the following rules:

- Data exported to MRS (HDFS): When data is exported from a DN, the data is stored in HDFS in the segment format. The file is named in the format of **mpp**_*DatabaseName_SchemaName_TableName_NodeName_n*.**orc**.
- You are advised to export data from different clusters or databases to different paths. The maximum size of an ORC file is 128 MB, and the maximum size of a stripe is 64 MB.
- After the export is complete, the **_SUCCESS** file is generated.

----End

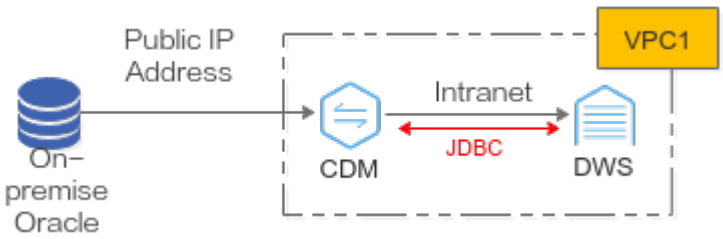
2 Data Migration

2.1 Using CDM to Migrate Oracle Data to a GaussDB(DWS) Cluster

2.1.1 Migration Process

This tutorial demonstrates how to migrate Oracle table data to GaussDB(DWS). **Figure 2-2** and **Table 2-1** show the migration process.

Figure 2-1 Migration scenario



NOTICE

- This practice describes how to migrate data in the **APEX2_DYNAMIC_ADD_REMAIN_TEST** table of user **db_user01** in the Oracle database.
- Network connection: In this practice, the Oracle database is deployed on-premises, so CDM is used to connect Oracle to GaussDB(DWS). CDM connects to Oracle via a public IP address. CDM and GaussDB(DWS) are in the same region and VPC and can communicate with each other. **Ensure that all the network is connected during the migration.**
- This practice is for reference only. The actual migration may be complex due to factors such as the network environment, service complexity, node scale, and data volume. It is better to perform the migration under the guidance of technical personnel.

Figure 2-2 Basic process of migrating data from Oracle to GaussDB(DWS)

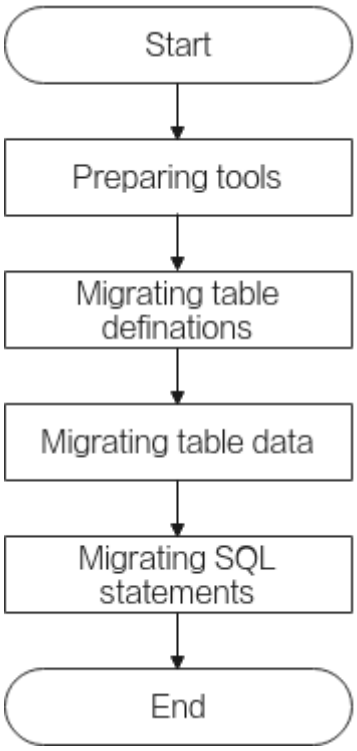


Table 2-1 Basic process of migrating data from Oracle to GaussDB(DWS)

Process	Description
Required Tools	Software tools to be prepared before the migration.
Migrating Table Definition	Use the PL/SQL Developer to migrate table definitions.
Migrating Full Table Data	Use Huawei Cloud Data Migration Service (CDM) to migrate data.
Migrating Service SQL Statements	Use the DSC syntax migration tool to rewrite the syntax so that the Oracle service SQL statements can be adapted to GaussDB(DWS).

2.1.2 Required Tools

The tools required for the migration include PL/SQL Developer, Instant Client, and DSC. For details about how to download the tools, see **Table 2-2**.

Table 2-2 Required tools

Tool	Description	Download Address
PL/SQL Developer	Oracle visual development tool	**PL/SQL Developer download address**

Tool	Description	Download Address
Oracle Instant Client	Oracle client	Instant Client download address
DSC	Syntax migration tool for GaussDB(DWS)	DSC Download Address

2.1.3 Migrating Table Definition

2.1.3.1 Installing the PL/SQL Developer on the Local Host

Procedure

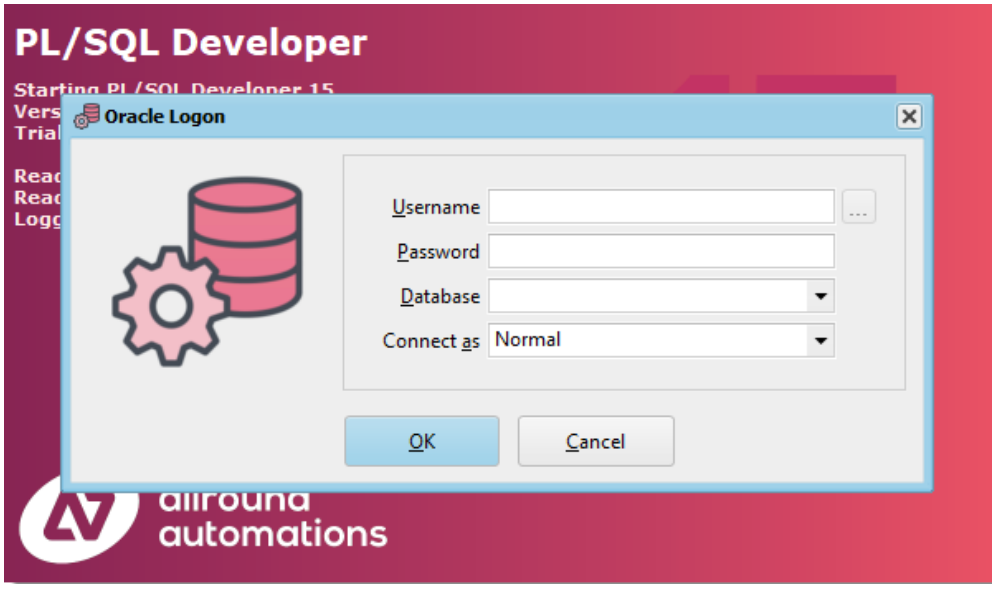
Step 1 Decompress the PL/SQL Developer, Instant Client, and DSC packages.

Step 2 Configure an Oracle home and OCL library for PL/SQL Developer.

NOTE

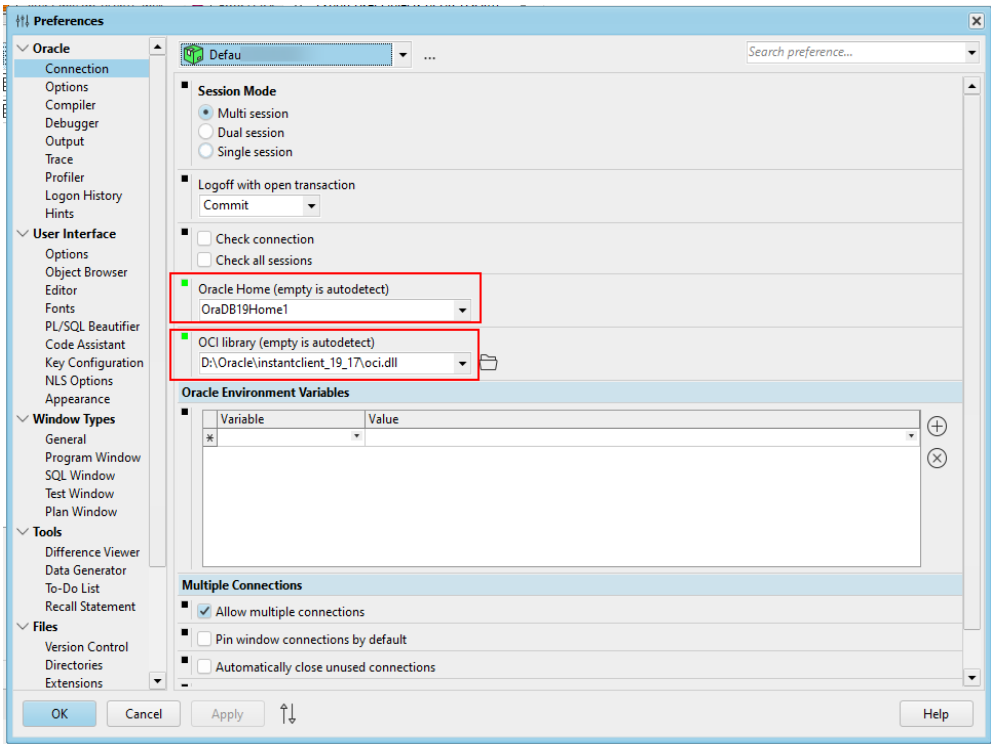
The following uses the PL/SQL Developer Trial Version as an example.

1. On the login page, click Cancel.

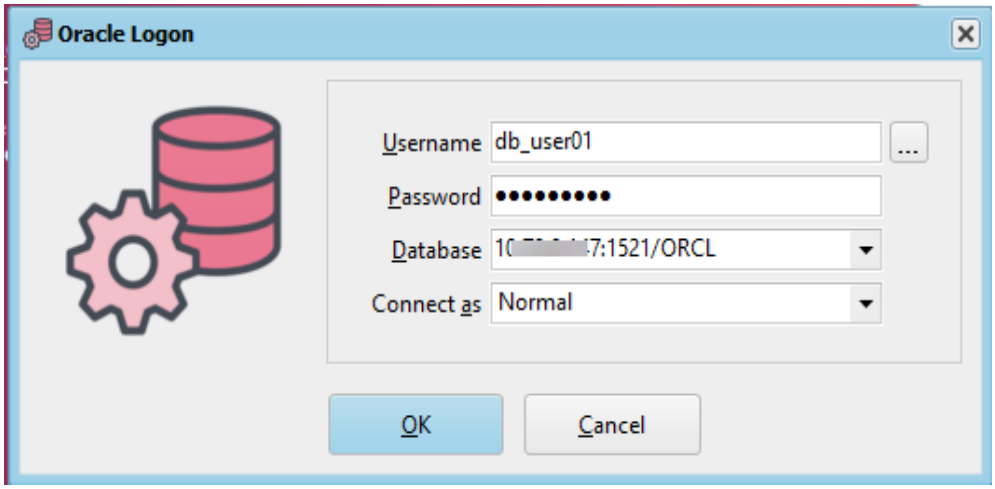


2. Choose **Configure** > **Preferences** > **Connection**, and add the Oracle Home and OCl library configurations.
3. Copy the instantclient path obtained from **Step 1** (for example, **D:\Oracle \instantclient_19_17\oci.dll**) to the home directory of the Oracle database.

   Copy the **oci.dll** file path (for example, **D:\Oracle \instantclient_19_17\oci.dll**) in the instantclient file to the OCI library.



Step 3 Go back to the PL/SQL Developer login page. Enter the username, password, and database address.



Step 4 Click **OK**. If the database is connected, it indicates that the PL/SQL Developer is installed successfully.

----End

2.1.3.2 Migrating Table Definitions and Syntax

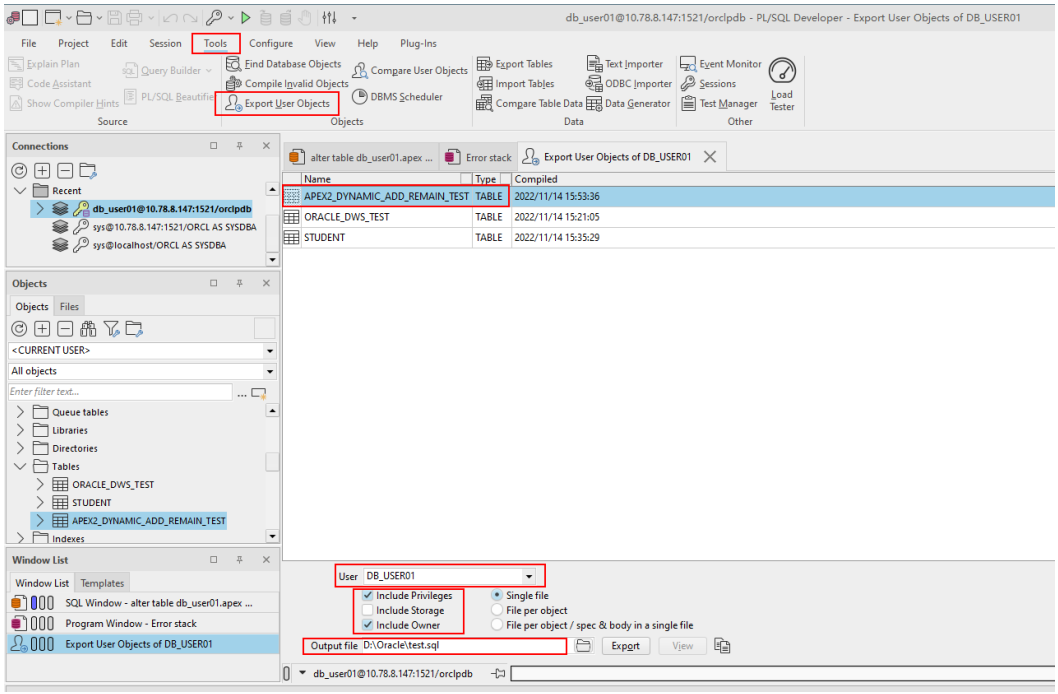
Step 1 Log in to the PL/SQL Developer use an account with the **sysdba** permission. In this example, the account **db_user01** is used.

NOTE

The following uses the PL/SQL Developer Trial Version as an example.

Step 2 On the menu bar, choose **Tools** > **Export User Objects**.

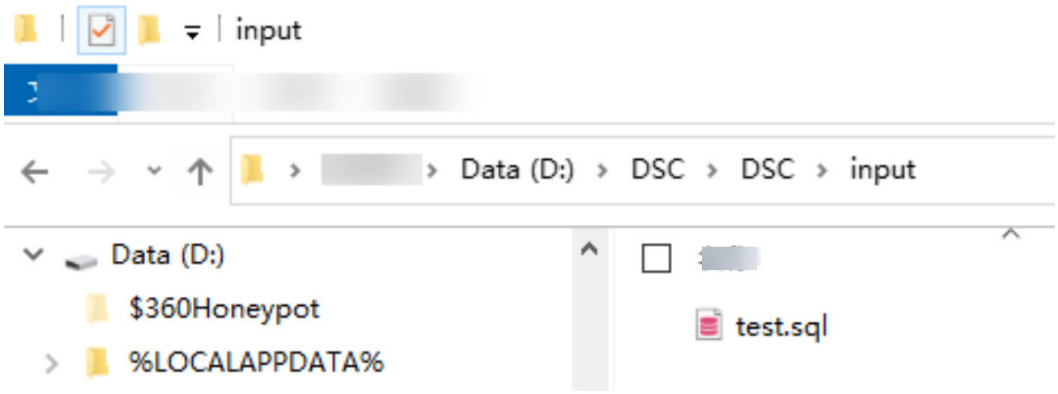
Step 3 Select the logged-in user **db_user01**, select the table object **APEX2_DYNAMIC_ADD_REMAIN_TEST** of the user, select the path to the output file (name the output SQL file as **test**), and click **Export**.



The exported DDL file is as follows:

```
D: > DSC > DSC > output > output > test.sql
1   prompt PL
2   /
3   SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by 16        ) on 2
4    /* SET define off; */
5   /*spool test.log*/
6   \echo
7    \echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
8    \echo ========================================
9    \echo
10   CREATE
11       UNLOGGED TABLE
12           DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
13                id INTEGER NOT NULL
14               ,TIME DATE
15               ,add_users NUMBER
16               ,remain_users NUMBER
17               ,PRIMARY KEY (ID)
18           ) ;
19  \echo Done
20   /*spool off*/
21  SET define
22       ON ;
```

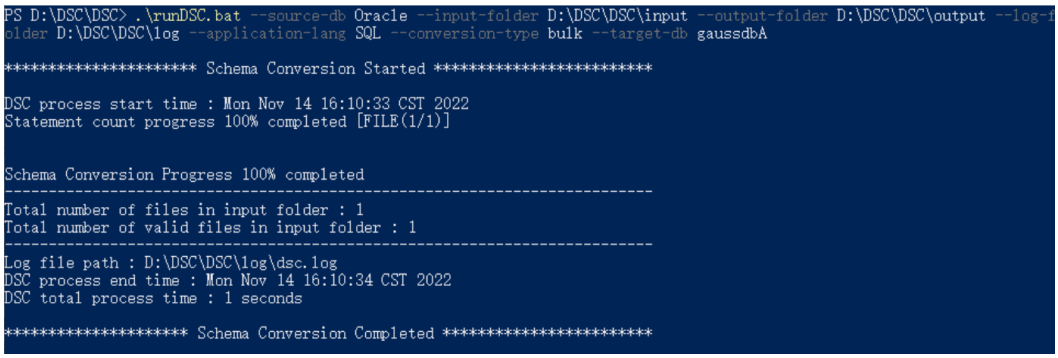
Step 4 Place the exported DDL file in the **input** directory of the decompressed DSC folder.

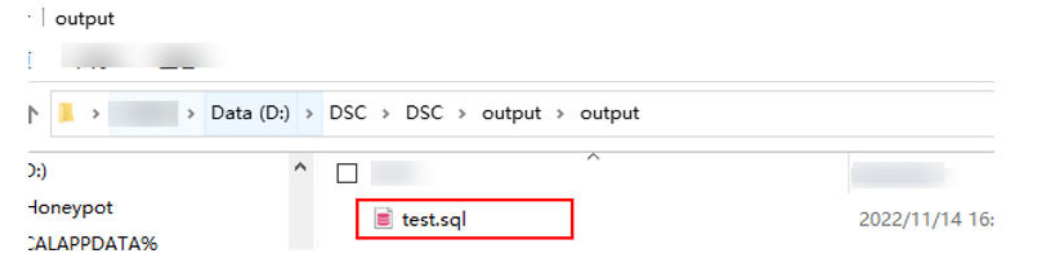


Step 5 In the directory of runDSC.bat, press Shift and right-click. Choose **Open PowerShell window here** and perform the conversion. Replace **D:\DSC\DSC\input**, **D:\DSC\DSC\output**, and **D:\DSC\DSC\log** with the actual DSC paths.

```
.\runDSC.bat --source-db Oracle --input-folder D:\DSC\DSC\input --output-folder D:\DSC\DSC\output --log-folder D:\DSC\DSC\log --application-lang SQL --conversion-type bulk --target-db gaussdbA
```

Step 6 After the conversion is complete, the converted DDL file is automatically generated in the **output** directory of DSC.





Step 7 The table definition structure of GaussDB(DWS) is different from that of Oracle. You need to manually modify the converted table definition.

Comment out **\echo** in the file (if you use gsql to import table definitions, you do not need to do this) and manually change the distribution column of the specified table.

- Before the change:


```
D: > DSC > DSC > output > output > test.sql
prompt PL
/
SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by l[illegible] on 2
 /* SET define off; */
/*spool test.log*/
\echo
 \echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
 \echo ========================================
 \echo
 CREATE
     UNLOGGED TABLE
          DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
               id INTEGER NOT NULL
               ,TIME DATE
               ,add_users NUMBER
               ,remain_users NUMBER
               ,PRIMARY KEY (ID)
          ) ;
\echo Done
 /*spool off*/
SET define
     ON ;
```

- After the change:

```
prompt PL
/
SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by [illegible] on
 /* SET define off; */
/*spool test.log*/
 --\echo
 --\echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
  --\echo ========================================
  --\echo
 CREATE
     UNLOGGED TABLE
          DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
               id INTEGER NOT NULL
               ,TIME DATE
               ,add_users NUMBER
               ,remain_users NUMBER
               ,PRIMARY KEY (ID)
          ) DISTRIBUTE BY HASH (ID);
\echo Done
 /*spool off*/
SET define
     ON ;
```

NOTE

The distribution column in a hash table must meet the following requirements, which are ranked by priority in descending order:

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

Step 8 Create a GaussDB(dws) cluster. For details, see **Creating a Cluster**.

Step 9 Connect to the GaussDB(DWS) cluster as the system administrator **dbadmin**. For details, see **Using the Data Studio GUI Client to Connect to a Cluster**. By default, the first connection is to the default database **gaussdb**.

Step 10 Create a new target database **test**, and then switch to it.

```
CREATE DATABASE test WITH ENCODING 'UTF-8' DBCOMPATIBILITY 'ORA' TEMPLATE template0;
```


Step 11 Create a schema and switch to it. The schema name must be the same as the Oracle user name (**db_user01** in this example).

```
CREATE SCHEMA db_user01;
SET CURRENT_SCHEMA = db_user01;
```

Step 12 Copy the converted DDL statements in **Step 7** to Data Studio for execution.

Step 13 If the **APEX2_DYNAMIC_ADD_REMAIN_TEST** table can be found in the schema in the **test** database of the GaussDB(DWS) cluster, the table definition is migrated.

```
SELECT COUNT(*) FORM db_user01.APEX2_DYNAMIC_ADD_REMAIN_TEST;
```

----End

2.1.4 Migrating Full Table Data

2.1.4.1 Configuring a GaussDB(DWS) Data Source Connection

Step 1 Create a CDM cluster and bind an EIP to the cluster by referring to **Creating a CDM cluster**.

NOTICE

Ensure that the CDM cluster and the GaussDB(DWS) cluster are in the same region and VPC to ensure network connectivity.

Step 2 On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Create Link**.

Step 3 Select **Data Warehouse Service** and click **Next**.

Step 4 Configure the GaussDB(DWS) connection, click **Test**. If the connection is successful, click **Save**.

Table 2-3 GaussDB(DWS) connection information

Parameter	Value
Name	dws
Database Server	Click **Select** and select the GaussDB(DWS) cluster to be connected from the cluster list. **NOTE** The system automatically displays the GaussDB(DWS) clusters in the same region and VPC. If no GaussDB(DWS) cluster is available, manually enter the IP address of the GaussDB(DWS) cluster that has been connected to the network.
Host Port	8000
Database Name	test
User Name	dbadmin
Password	Password of user **dbadmin**

Parameter	Value
Use Agent	No

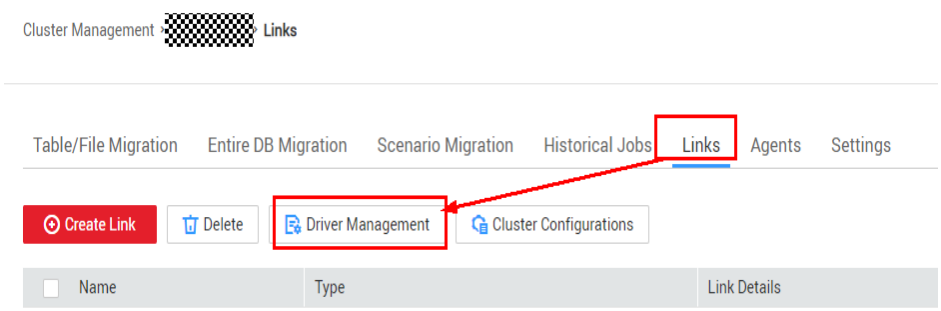
----End

2.1.4.2 Configuring an Oracle Data Source Connection

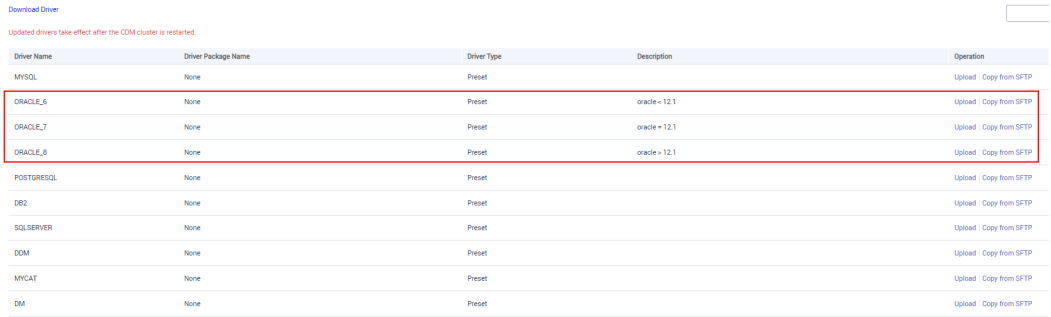
To migrate data from Oracle to GaussDB(DWS), you need to configure an Oracle data source connection first.

Procedure

Step 1 On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Driver Management**.



Step 2 Click **Upload** on the right of ORACLE, select an Oracle driver package (if no driver package is available on the local PC, download it by referring to **Managing Drivers**), and click **Upload**.



Step 3 On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Create Link**.

Step 4 Select Oracle as the connector and click **Next**.

Step 5 Configure the Oracle connection, click **Test**. If the connection is successful, click **Save**.

Table 2-4 Oracle connection information

Parameter	Value
Name	oracle
Database Server	192.168.1.100 (This is an example. Enter the actual public IP address of the Oracle database.)
Host Port	1521
Connection Type	Service Name
Database Name	orcl
User Name	db_user01
Password	-
Use Local API	No
Use Agent	No
Oracle Version	Later than 12.1

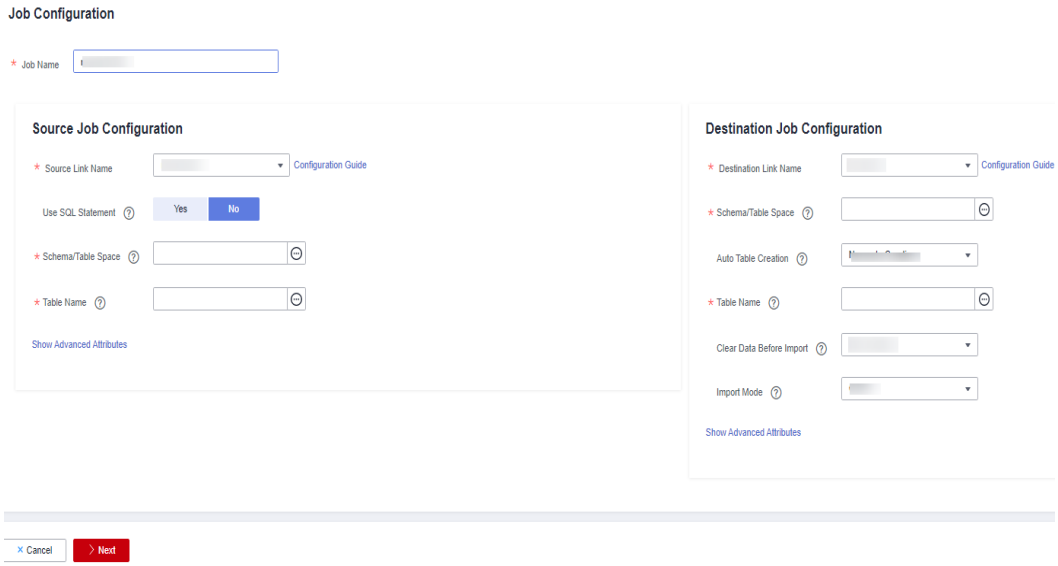
----End

2.1.4.3 Migrating Tables

Procedure

Step 1 On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Table/File Migration** > **Create Job**.

Step 2 Configure jobs at the source end and destination end.



Step 3 Configure source job parameters based on the type of the source database.

Table 2-5 Source job parameters

Parameter	Example Value
Schema/Table Space	db_user01
Use SQL Statement	No
Table Name	APEX2_DYNAMIC_ADD_REMAIN_TEST
WHERE Clause	-
Null in Partition Column	Yes

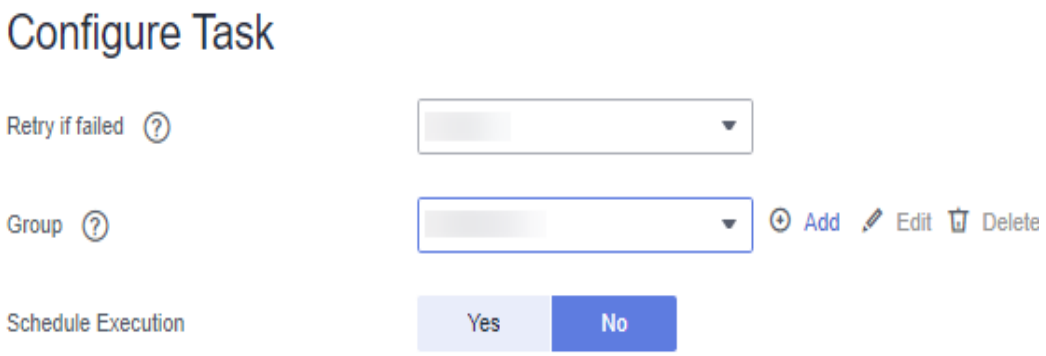
Step 4 Configure the destination job parameters based on the destination cloud service.

Table 2-6 Destination job parameters

1. Parameter	Example Value
Schema/Table Space	db_user01
Auto Table Creation	Non-auto creation
Table Name	apex2_dynamic_add_remain_test
Clear Data Before Import	Clear all data
Import Mode	COPY
Import to Staging Table	No
Prepare for Data Import	-
Complete Statement After Data Import	analyze db_user01. apex2_dynamic_add_remain_test;

Step 5 Mapping between source fields and destination fields.

Step 6 If the task fails to be configured, retry for three times, save the configuration, and run the task.



Step 7 The task is executed, and the data migration is finished.

----End

2.1.4.4 Verification

Step 1 In the **test** database of GaussDB(DWS), run the following SQL statement to query the number of rows in the table **apex2_dynamic_add_remain_test**. If the number of rows is the same as that in the source table, the data is consistent.

```
SELECT COUNT(*) FROM  db_user01.apex2_dynamic_add_remain_test;
```

Step 2 Run the following statement to check the data skewness:

If the data skewness is within 10%, the data distribution is normal. The data migration is complete.

```
SELECT TABLE_SKEWNESS('db_user01.apex2_dynamic_add_remain_test');
```

	table_skewness
1	("dn_6001_6002 ",97,32.119%)
2	("dn_6003_6004 ",105,34.768%)
3	("dn_6005_6006 ",100,33.113%)

----End

2.1.5 Migrating Service SQL Statements

2.1.5.1 Migrating Syntax

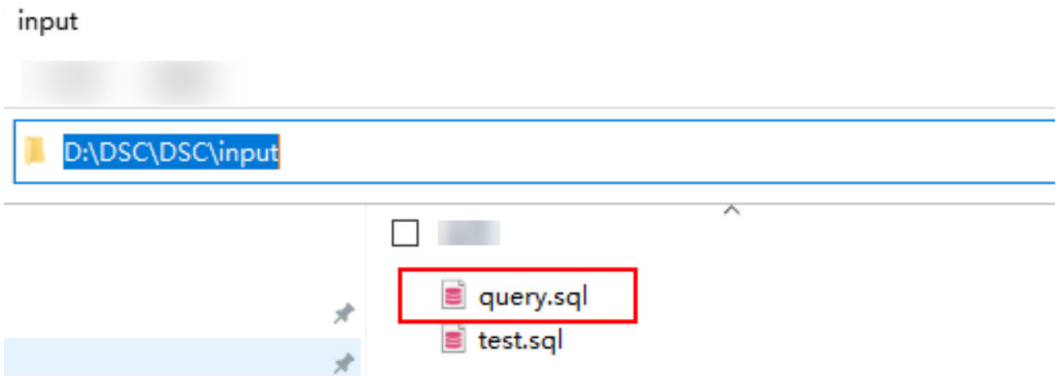
Step 1 Save the following SQL statements in an Oracle database as an query.sql file.

```
-- Generally, the HAVING clause must appear after the GROUP BY clause, but Oracle allows HAVING to appear before or after the GROUP BY clause. Therefore, you need to move the HAVING clause after the GROUP BY clause in the target database.
SELECT
id,
count(*),
sum(remain_users)
FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST
HAVING id <= 5
GROUP BY id;
```



```
UNIQUE keywords are migrated as DISTINCT keywords.
SELECT UNIQUE add_users FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST;

-- In NVL2(expression,value1,value2), if the expression is not Null, NVL2 returns Value1. If the expression is Null, NVL2 returns Value2.
SELECT NVL2(add_users, 1, 2) FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST SHERE rownum <= 2;
```

Step 2 Place the query.sql file obtained in **Step 1** in the **input** directory of the decompressed DSC folder.



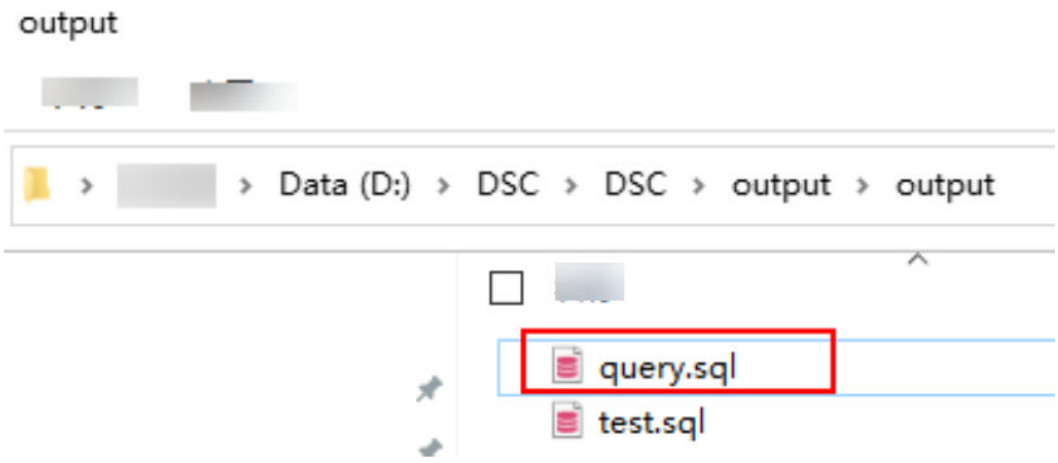
Step 3 In the directory of runDSC.bat, press Shift and right-click. Choose **Open PowerShell window here** and perform the conversion.

Replace **D:\DSC\DSC\input**, **D:\DSC\DSC\output**, and **D:\DSC\DSC\log** with the actual DSC paths.

```
.\runDSC.bat --source-db Oracle --input-folder D:\DSC\DSC\input --output-folder D:\DSC\DSC\output --log-folder D:\DSC\DSC\log --application-lang SQL --conversion-type bulk --target-db gaussdbA
```

```
PS D:\DSC\DSC> .\runDSC.bat --source-db Oracle --input-folder D:\DSC\DSC\input --output-folder D:\DSC\DSC\output --log-folder D:\DSC\DSC\log --application-lang SQL --conversion-type bulk --target-db gaussdbA

********** Schema Conversion Started **********

DSC process start time : Mon Nov 14 16:10:33 CST 2022
Statement count progress 100% completed [FILE(1/1)]


Schema Conversion Progress 100% completed
--------------------------------------------------------------------------------
Total number of files in input folder : 1
Total number of valid files in input folder : 1
--------------------------------------------------------------------------------
Log file path : D:\DSC\DSC\log\dsc.log
DSC process end time : Mon Nov 14 16:10:34 CST 2022
DSC total process time : 1 seconds

********** Schema Conversion Completed **********
```

Step 4 After the conversion is complete, a DML file is generated in the output directory.



----End

2.1.5.2 Verification

Step 1 Execute the SQL statements in the Oracle database before migration.

Step 2 Execute the migrated SQL statements on Data Studio.

Step 3 Compare the execution results. If they are the same, the SQL migration is complete.

----End

2.2 Using CDM to Migrate MySQL Data to a GaussDB(DWS) Cluster

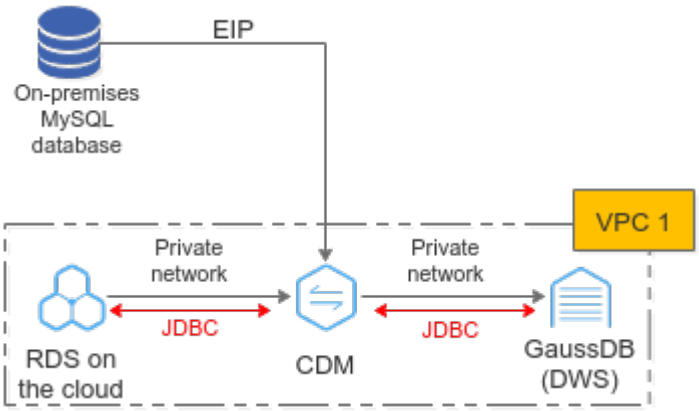
This section describes how to use Cloud Data Migration (CDM) to migrate MySQL data to GaussDB(DWS) clusters in batches.

This section contains the following parts:

1. **Checking Data Before Migration**
2. **Creating a GaussDB(DWS) Cluster**
3. **Creating a CDM cluster**
4. **Creating a Connection**
5. **Creating and Migrating a Job**
6. **Verifying Data Consistency After Migration**

Scenario Description

Figure 2-3 Migration



CDM can migrate an entire cloud/on-premises MySQL database or a single table. The migration of an on-premises MySQL database is used as an example.

- On-premises MySQL data migration:

  CDM accesses the MySQL database through the public IP address. CDM and GaussDB(DWS) are in the same VPC. CDM establishes JDBC connections respectively with MySQL and GaussDB(DWS).

- Cloud RDS MySQL data migration:

  RDS, CDM, and GaussDB(DWS) are in the same VPC. CDM establishes JDBC connections respectively with MySQL and GaussDB(DWS). If cloud RDS and GaussDB(DWS) are not in the same VPC, CDM uses the EIP to access RDS.

Checking Data Before Migration

Step 1 Connect to the MySQL DB instance and check the MySQL database status.

mysql -h *<host>***-P***<port>***-u** <userName>**-p --ssl-ca=***<caDIR>*

Table 2-7 Parameter description

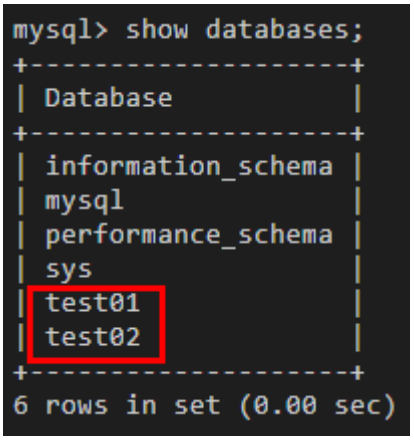
Parameter	Description
<host>	Address for connecting to the MySQL database.
<port>	Database port. By default, the value is **3306**.
<userName>	MySQL administrator account. The default value is **root**.
<caDIR>	Path of the CA certificate. The file must be stored in the path where the command is executed.

Enter the password of the database account as prompted:

```
Enter password:
```


Step 2 Analyze the name and code of the databases to be migrated, and the name and attributes of the tables to be migrated.

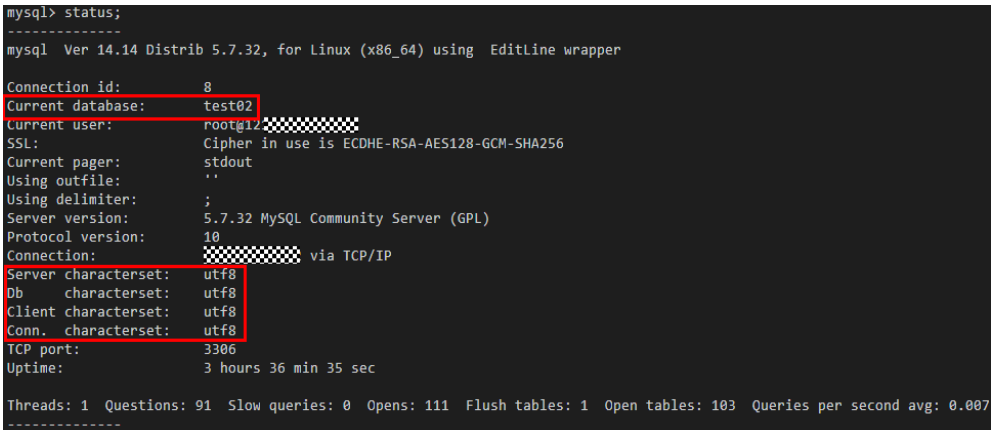
For example, the destination MySQL databases to be migrated are **test01**, **test02**, and the encoding format. The test01 library contains the **orders**, **persons**, and **persons_b** tables and the **persons_beijing view**. The **test02** library contains the **persons_c table**.

1. Query the database name.
   ```
   show databases;
   ```
   

2. Query the database code.
   ```
   use <databasename>;
   status;
   ```

   Figure 2-4 Query database code 1

   Figure 2-5 Query database code 2

3. Query database tables.
   ```
   use <databasename>;
   show full tables;
   ```


NOTICE

- The GaussDB(DWS) database is case-insensitive. If the original MySQL database contains table names that contain both uppercase and lowercase letters or only uppercase letters, for example, **Table01** and **TABLE01**, you need to change the table names to lowercase letters before the migration. Otherwise, GaussDB(DWS) cannot identify the tables after migration.
- You are advised to set the MySQL database to be case-insensitive by modifying **lower_case_table_names** to **1** in **/etc/my.cnf** and restarting the MySQL service.

Figure 2-6 Querying database tables

```
mysql> show full tables;
+------------------+------------+
| Tables_in_test01 | Table_type |
+------------------+------------+
| orders           | BASE TABLE |
| persons          | BASE TABLE |
| persons_b        | BASE TABLE |
| persons_beijing  | VIEW       |
+------------------+------------+
4 rows in set (0.00 sec)
```

Figure 2-7 Querying database tables

```
mysql> show full tables;
+------------------+------------+
| Tables_in_test02 | Table_type |
+------------------+------------+
| persons_c        | BASE TABLE |
+------------------+------------+
1 row in set (0.00 sec)
```

4. Check the attributes of each table for comparison after the migration.

```
use <databasename>;
desc <table name>;
```

Figure 2-8 Viewing table properties

```
mysql> desc persons;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| Id_P      | int(11)      | YES  |     | NULL    |       |
| LastName  | varchar(255) | YES  |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Address   | varchar(255) | YES  |     | NULL    |       |
| City      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

----End

Creating a GaussDB(DWS) Cluster

Step 1 For how to create a cluster, see **Creating a Cluster**. You can select the CN-Hong Kong region

NOTE

Ensure that the GaussDB(DWS) cluster and CDM cluster are in the same region and VPC.

Step 2 Connect to a cluster by referring to **Using the gsql CLI Client to Connect to a Cluster**.

Step 3 Create the target databases **test01** and **test02** in **Checking Data Before Migration** with the same name and database code as the original MySQL database.

```
create database test01 with encoding 'UTF-8' dbcompatibility 'mysql' template template0;
create database test02 with encoding 'UTF-8' dbcompatibility 'mysql' template template0;
```

----End

Creating a CDM cluster

Step 1 Log in to the Huawei Cloud console.

Step 2 Choose **Migration** > **Cloud Data Migration**.

Step 3 Click **Buy CDM Cluster** and set the following parameters:

Table 2-8 CDM cluster parameters

Parameter	Value
Region	Select the CN-Hong Kong region, which is in the same location as GaussDB (DWS).
AZ	AZ1 (If the desired resources are sold out in the current AZ, change the AZ and try again.)
Name	CDM-demo
Instance Type	cdm.large (Select other flavors if the flavor is sold out.)
VPC	demo-vpc, which is in the same location as GaussDB (DWS).
Subnet	subnet-f377(10.1.0.0/24) (example)
Security Group	-
Enterprise Project	default

Step 4 Click **Buy Now**, confirm all the parameters, and click **Submit**.

Step 5 Go back to the **Cluster Management** page. Cluster creation takes about 5 minutes. After the cluster is created, click **Bind EIP** in the **Operation** column of the cluster.

Step 6 Select an available EIP and click **OK**. If no EIP is available, switch to the EIP page to purchase an EIP.

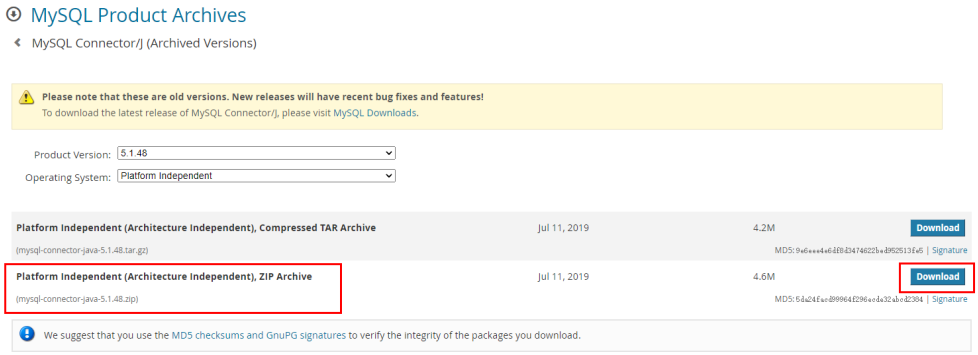
----End

Creating a Connection

Step 1 When creating a MySQL connection for the first time, upload a driver.

1. Access the **MySQL** driver and download the 5.1.48 version.

Figure 2-9 Downloading a driver



2. Download the package to the local host and decompress it to obtain **mysql-connector-java-*xxx*.jar**.
3. On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Driver Management**.
4. Click **Upload** on the right of MySQL, select mysql-connector-java-xxx.jar, and click **Upload**.

Step 2 Create a MySQL connection.

1. On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Create Link**.
2. Select **MySQL** and click **Next**. (If the RDS is deployed on the cloud, select RDS for MySQL.)
3. Enter the connection information according to **Table 2-9**, and click **Test**. If the test is successful, click **Save**.

NOTE

If the test fails, check whether CDM connects to the MySQL database using the public IP address. If the public IP address is used, bind the public IP address by referring to **Step 5**.

Table 2-9 MySQL connection information

Parameter	Value
Name	MySQL

Parameter	Value
Database Server	192.168.1.100 (This is an example, enter the actual public IP address of the on-premises MySQL database. Ensure that the whitelist access permission has been enabled on the MySQL server.)
Port	3306
Database Name	test01
User	root
Password	Password of the user **root**.
Use Local API	No
Use Agent	No

Step 3 Create a GaussDB(DWS) link.

1. On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links** > **Create Link**.
2. Select **Data Warehouse Service** and click **Next**.
3. Enter the connection information according to **Table 2-10**, and click **Test**. If the test is successful, click **Save**.

   Table 2-10 GaussDB(DWS) connection information

Parameter	Value
Name	DWS-test01
Database Server	Click **Select** and select the GaussDB(DWS) cluster to be connected from the cluster list. **NOTE** The system automatically displays the GaussDB(DWS) clusters in the same region and VPC. If no GaussDB(DWS) cluster is available, manually enter the IP address of the GaussDB(DWS) cluster that has been connected to the network.
Port	8000
Database Name	test01 (Ensure that the corresponding database has been manually created on GaussDB(DWS) by referring to **Step 3**.)
Username	dbadmin
Password	Password of user **dbadmin**
Use Agent	No

4. Repeat **Step 3.1** to **Step 3.3** to create the **DWS-test02** link.

----End

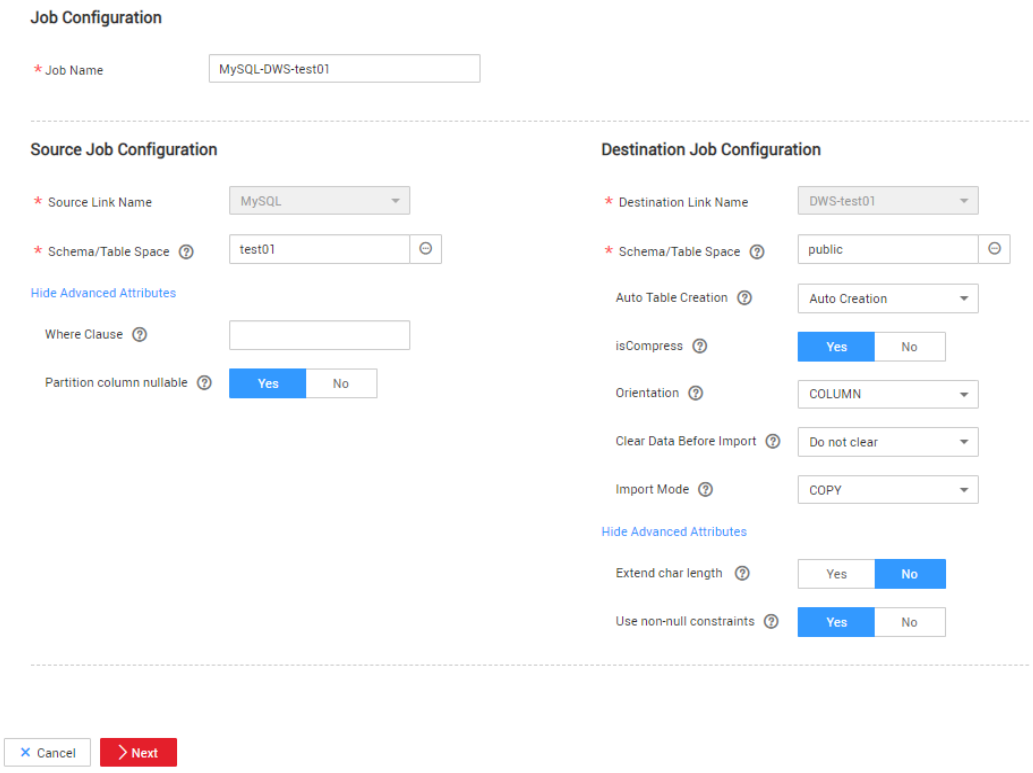
Creating and Migrating a Job

Step 1 On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Entire DB Migration** > **Create Job**.

Step 2 Set the following parameters and click **Next**.

- Job Name: MySQL-DWS-test01
- Source Job Configuration:
  - Source Link Name: MySQL
- Destination Job Configuration:
  - Destination Link Name: DWS-test01
  - Automatic Table Creation: The table is created when it does not exist.
  - isCompress: Yes
  - Orientation: COLUMN
  - Retain the default value for other settings.

Figure 2-10 Configuring a Job

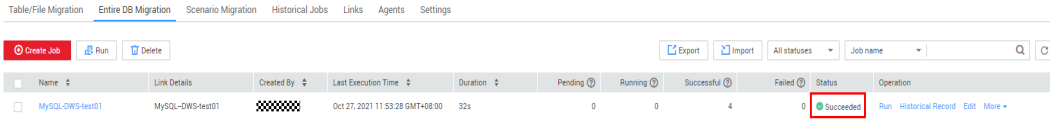


Step 3 Select all tables, click ≫, and click **Next**.

Step 4 Retain the default settings and click **Save and Run**.

Step 5 Check the job running status. If the status is **Succeeded**, the migration is successful.

Figure 2-11 Viewing the job running status



Step 6 Repeat **Step 1** to **Step 5** to migrate all tables in the **test02** database.

NOTICE

When creating a job, select **test02** for the GaussDB(DWS) database of the target source.

----End

Verifying Data Consistency After Migration

Step 1 Use gsql to connect to the **test01** cluster of GaussDB(DWS).

gsql -d *test01* **-h** *IP address of the host* **-p 8000 -U dbadmin -W** *Database user password* **-r;**

Step 2 Query the tables in the **test01** database.

```
select * from  pg_tables where schemaname= 'public';
```

Figure 2-12 Query the tables in the **test01** database.

```
test01=> select * from  pg_tables where schemaname= 'public';
 schemaname |    tablename    | tableowner | tablespace | hasindexes | hasrules | hastriggers | tablecreator |           created            |        last_ddl_time
------------+-----------------+------------+------------+------------+----------+-------------+--------------+------------------------------+-----------------------------
-----
 public     | persons         | dbadmin    |            | f          | f        | f           | dbadmin      | 2021-10-27 03:43:25.306998+00 | 2021-10-27 03:43:25.30699
8+00
 public     | persons_beijing | dbadmin    |            | f          | f        | f           | dbadmin      | 2021-10-27 03:43:25.298073+00 | 2021-10-27 03:43:25.29807
3+00
 public     | orders          | dbadmin    |            | f          | f        | f           | dbadmin      | 2021-10-27 03:43:25.228591+00 | 2021-10-27 03:43:25.22859
1+00
 public     | persons_b       | dbadmin    |            | f          | f        | f           | dbadmin      | 2021-10-27 03:43:25.295822+00 | 2021-10-27 03:43:25.29582
2+00
(4 rows)
```

Step 3 Check whether the data in each table is complete and whether the columns are complete.

```
select count(*) from table name;
\d+ table name;
```

Figure 2-13 Querying table fields

```
test01=> select count(*) from persons;
 count
-------
     5
(1 row)
```

Figure 2-14 Querying table data

```
test01=> \d+ persons;
                                Table "public.persons"
  Column   |          Type          | Modifiers | Storage  | Stats target | Description
-----------+------------------------+-----------+----------+--------------+-------------
 Id_P      | integer                |           | plain    |              |
 LastName  | character varying(255) |           | extended |              |
 firstname | character varying(255) |           | extended |              |
 address   | character varying(255) |           | extended |              |
 city      | character varying(255) |           | extended |              |
Has OIDs: no
Distribute By: HASH(Id_P)
Location Nodes: ALL DATANODES
Options: orientation=column, compression=high, colversion=2.0, enable_delta=false
```

Step 4 Perform sampling check to verify table data.

```
select * from persons where city = 'Beijing' order by id_p;
```

Figure 2-15 Verifying table data

```
test01=> select * from persons where city = 'Beijing' order by "Id_P";
 Id_P | LastName | firstname |    address     |  city
------+----------+-----------+----------------+---------
    1 | Gates    | Bill      | Xuanwumen 10   | Beijing
    4 | Carter   | Thomas    | Changan Street | Beijing
    5 | Carter   | William   | Xuanwumen 10   | Beijing
(3 rows)
```

Step 5 Repeat **Step 2** to **Step 4** to check whether the data in other databases and tables is correct.

----End

2.3 Using DRS to Synchronize MySQL Table Data to a GaussDB(DWS) Cluster in Real Time

This practice demonstrates how to use Data Replication Service (DRS) to synchronize MySQL data to GaussDB(DWS) in real time. For details about DRS, see **What Is DRS?**

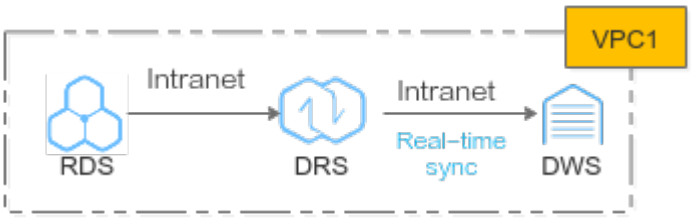
This practice takes about 60 minutes. The process is as follows:

1. **Preparations**
2. **Step 1: Prepare a MySQL Source Table**
3. **Step 2: Create a GaussDB(DWS) Cluster**
4. **Step 3: Create a DRS Synchronization Task**
5. **Step 4: Verify Data Synchronization**

Scenario Description

In big data analysis scenarios, MySQL serves as an OLTP database. After MySQL is connected to the GaussDB(DWS) data warehouse for OLAP analysis, data written by MySQL in real time needs to be synchronized to the GaussDB(DWS) data warehouse in real time. DRS is used to perform the synchronization.

Figure 2-16 DRS real-time synchronization



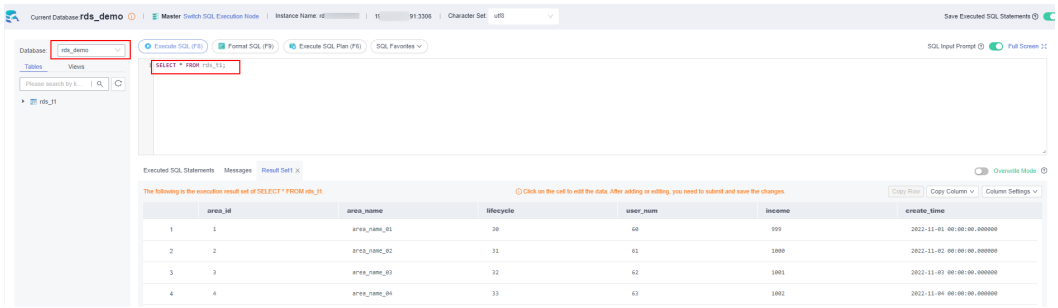
Preparations

- You have registered a Huawei account and enabled Huawei Cloud services.. Before using GaussDB(DWS), check the account status. The account cannot be in arrears or frozen.
- The MySQL source table to be migrated has been prepared. In this practice, a Huawei Cloud RDS MySQL database is used as the source data. If your MySQL database is offline, ensure that the network connection is normal.

Step 1: Prepare a MySQL Source Table

Step 1 You have purchased an RDS MySQL DB engine (this practice use MySQL 8.0.x as an example). For details, see **Buy a DB Instance**.

Step 2 The source database **rds_demo** with the **utf8mb4** character set has been created, and there is the table **rds_t1** with data in the database.



----End

Step 2: Create a GaussDB(DWS) Cluster

Step 1 **Creating a Cluster**. To ensure network connectivity, the GaussDB(DWS) cluster and RDS must be in the same region.

Step 2 Log in to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters**, locate the row that contains the target cluster, and click **Login** in the **Operation** column.

NOTE

This practice uses version 8.1.3.x as an example. 8.1.2 and earlier versions do not support this login mode. You can use Data Studio to connect to a cluster. For details, see **Using Data Studio to Connect to a Cluster**.

Step 3 After logging in to the GaussDB(DWS) database, create the database **rds_demo** for synchronization.


```
CREATE DATABASE rds_demo WITH ENCODING 'UTF-8' DBCOMPATIBILITY 'mysql' TEMPLATE template0;
```

Step 4 Switch to the **rds_demo** database and create a schema named **rds_demo**.

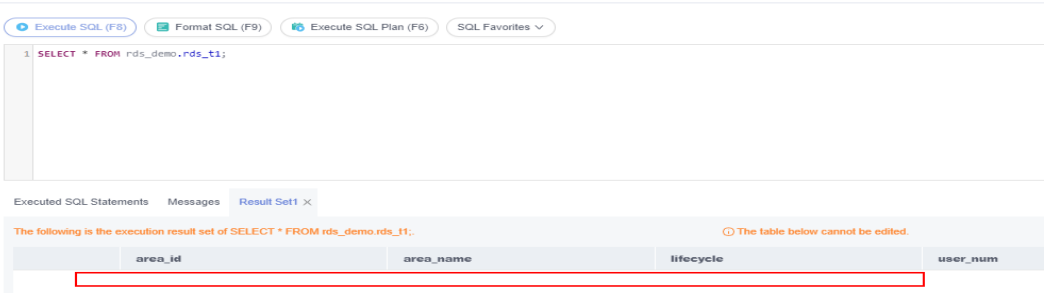
```
CREATE SCHEMA rds_demo;
```

Step 5 Create a table named **rds_t1** in the schema **rds_demo**.

```
CREATE TABLE rds_demo.rds_t1 (
  area_id varchar(256) NOT NULL,
  area_name varchar(256) DEFAULT NULL,
  lifecycle varchar(256) DEFAULT NULL,
  user_num int DEFAULT NULL,
  income  bigint DEFAULT NULL,
  create_time timestamp DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (area_id)
)distribute by hash(area_id);
COMMENT on column rds_demo.rds_t1.area_id is 'Region Code';
COMMENT on column rds_demo.rds_t1.area_name is 'Region Name';
COMMENT on column rds_demo.rds_t1.lifecycle is 'Life Cycle';
 COMMENT on column rds_demo.rds_t1.user_num is 'Subscribers in Each Life Cycle';
 COMMENT on column rds_demo.rds_t1.income is 'Region Income';
COMMENT on column rds_demo.rds_t1.create_time is 'Creation Time';
```

Step 6 Query table data. Currently, the table is empty.

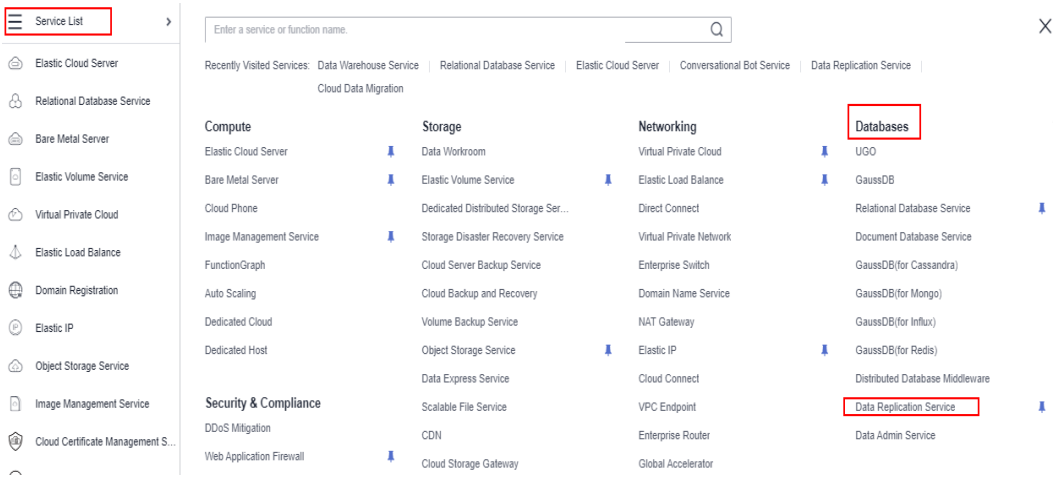
```
SELECT * FROM rds_demo.rds_t1;
```



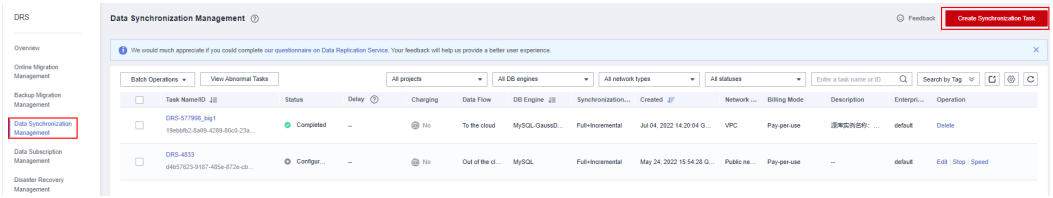
----End

Step 3: Create a DRS Synchronization Task

Step 1 Choose **Service List** > **Databases** > **Data Replication Service** to switch to the DRS console.



Step 2 Choose **Data Synchronization Management** on the left and click **Create Synchronization Task** in the upper right corner.



Step 3 Configure basic parameters. For details, see **Table 2-11**.

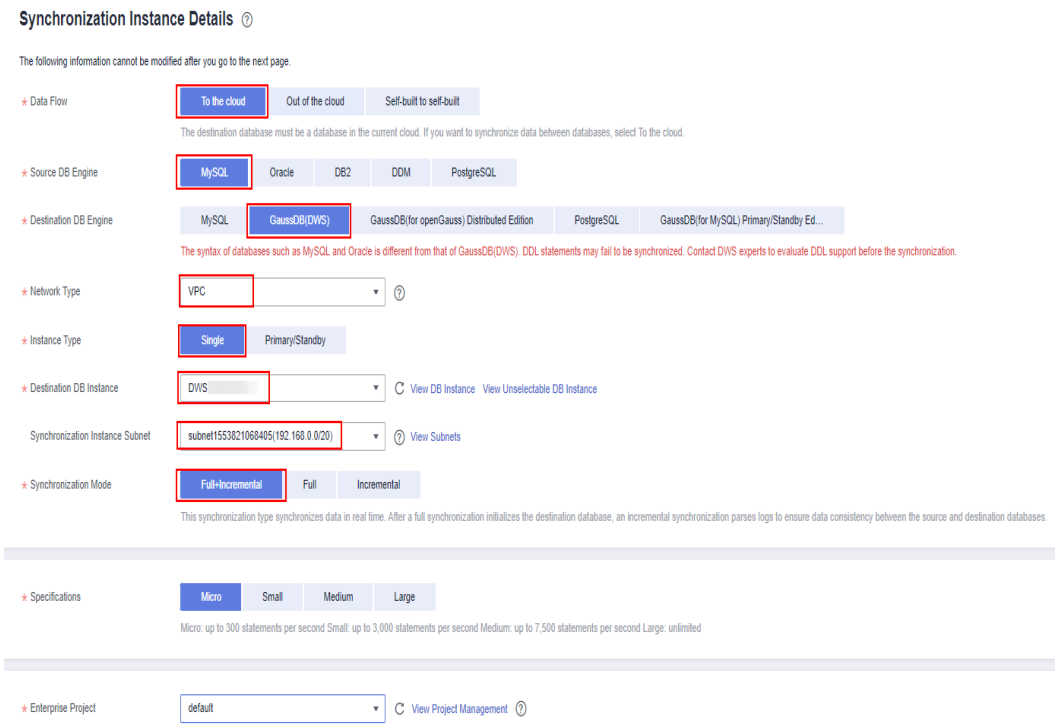
Table 2-11 Basic parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong. Ensure that RDS and GaussDB(DWS) are in the same region.
Project	CN-Hong Kong
Task Name	DRS-DWS
Description	-

Step 4 Configure the following parameters. For details, see **Table 2-12**.

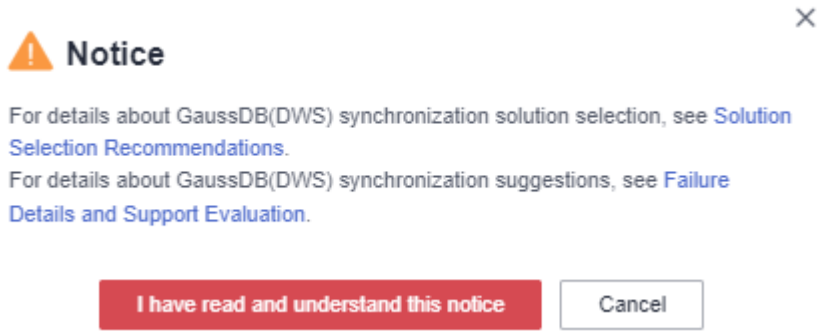
Table 2-12 Synchronized instance parameters

Parameter	Value
Data Flow	To the cloud
Source DB Engine	MySQL
Destination DB engine	GaussDB(DWS)
Network Type	In this practice, select **VPC**. If the MySQL database is offline, select **Public Network**.
Instance Type	Single
Destination DB Instance	Select the cluster created in **Step 2: Create a GaussDB(DWS) Cluster**.
Synchronization Instance Subnet	Select the subnet where the GaussDB(DWS) cluster resides. In this practice, RDS and GaussDB(DWS) are in the same VPC and subnet.
Synchronous Mode	Full+Incremental
Specifications	In this practice, select **Micro**. This option is selected based on the data volume and synchronization rate.



Step 5 Click **Next** and click **I have read and understand this notice**.

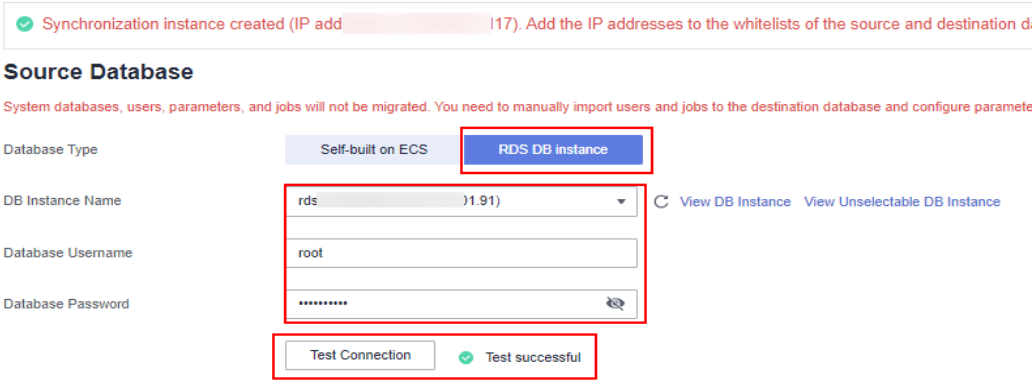
Wait for about 5 to 10 minutes for the synchronization to complete.



Step 6 After the synchronization succeeds, enter the source database information and click **Test Connection**.

Table 2-13 Source database information

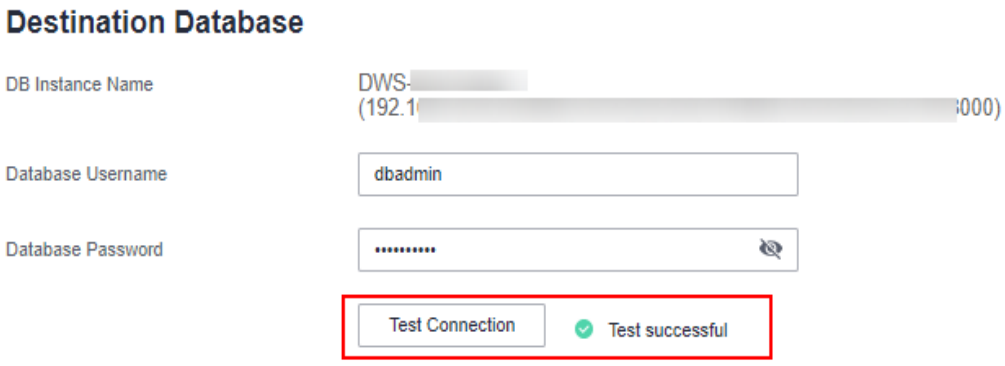
Parameter	Value
Database Type	RDS DB Instance
DB Instance Name	Select the created RDS DB instance.
Database Username	root
Database Password	****



Step 7 Enter the destination database information and click **Test Connection**. The connection test is successful.

Table 2-14 Destination database information

Parameter	Value
Database Username	dbadmin
Database Password	****



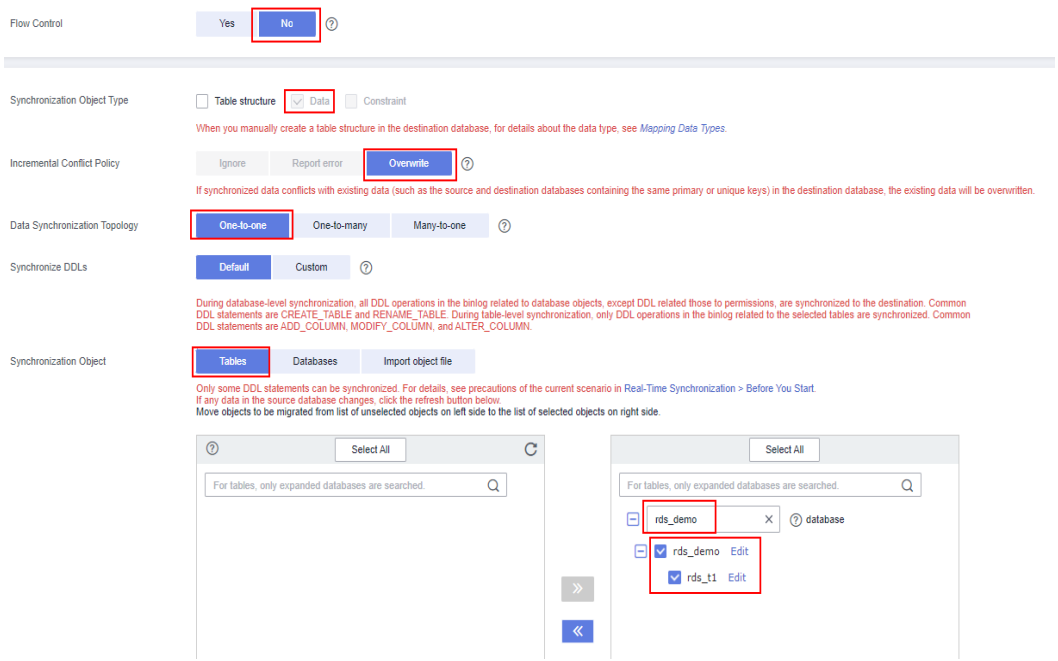
Step 8 Click **Next**, and then click **Agree**.

Step 9 Set the synchronization policy. For details, see **Table 2-15**.

Table 2-15 Synchronization policy

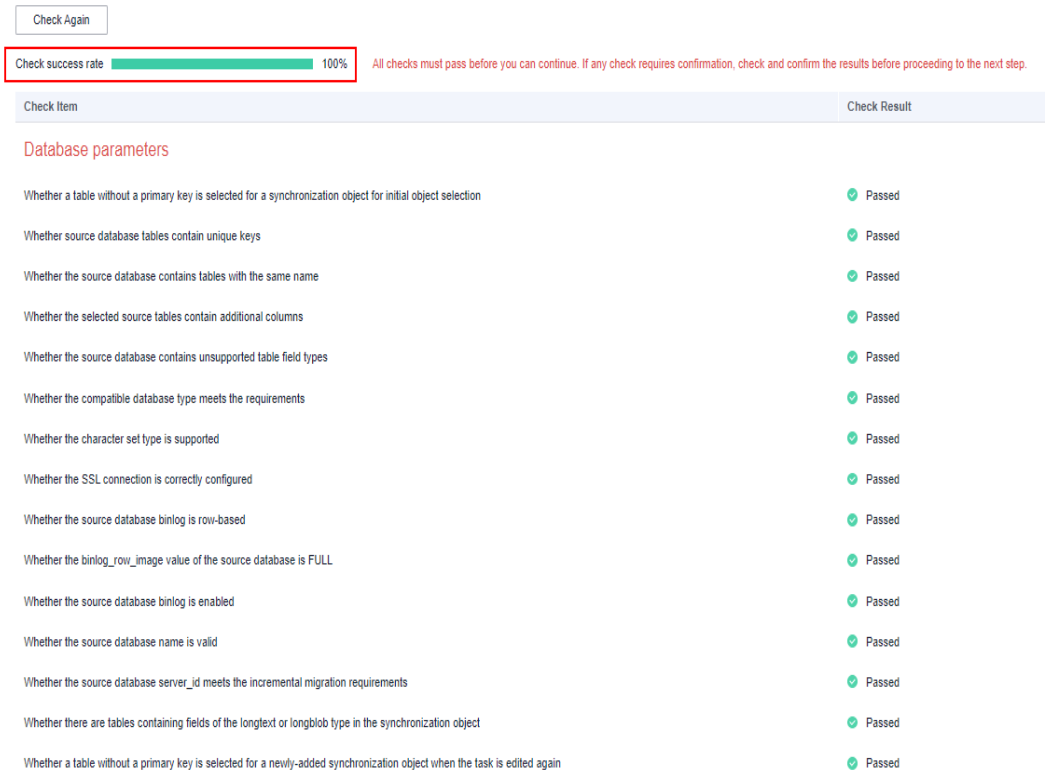
Parameter	Value
Flow Control	No

Parameter	Value
Synchronization Object Type	Data
Incremental Conflict Policy	Overwrite
Data Synchronization Topology	One-to-one
Synchronize DDLs	Default
Synchronization Object	Tables Select the table to be synchronized from the source database. In this practice, select **rds_t1** under **rds_demo**. Enter the name of the GaussDB(DWS) database that data is synchronized to: **rds_demo**

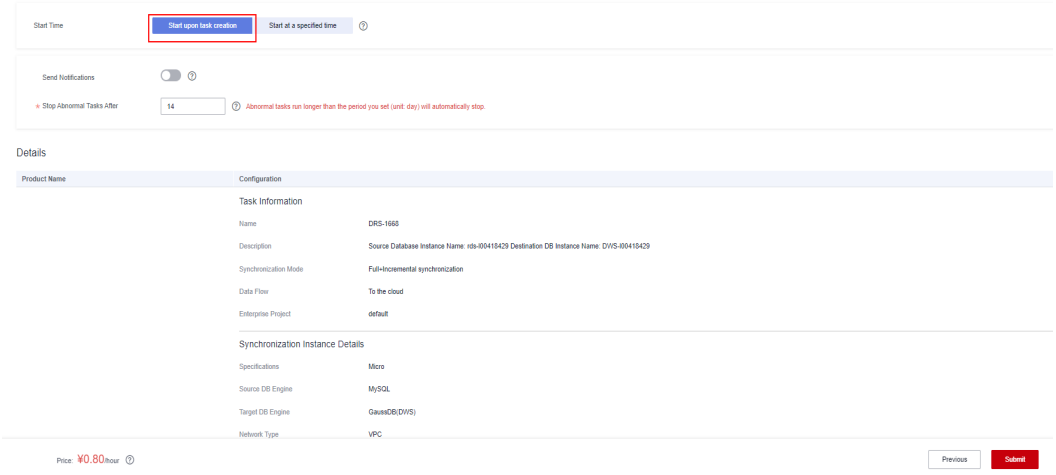


Step 10 Click **Next**, confirm the information, and click **Next**.

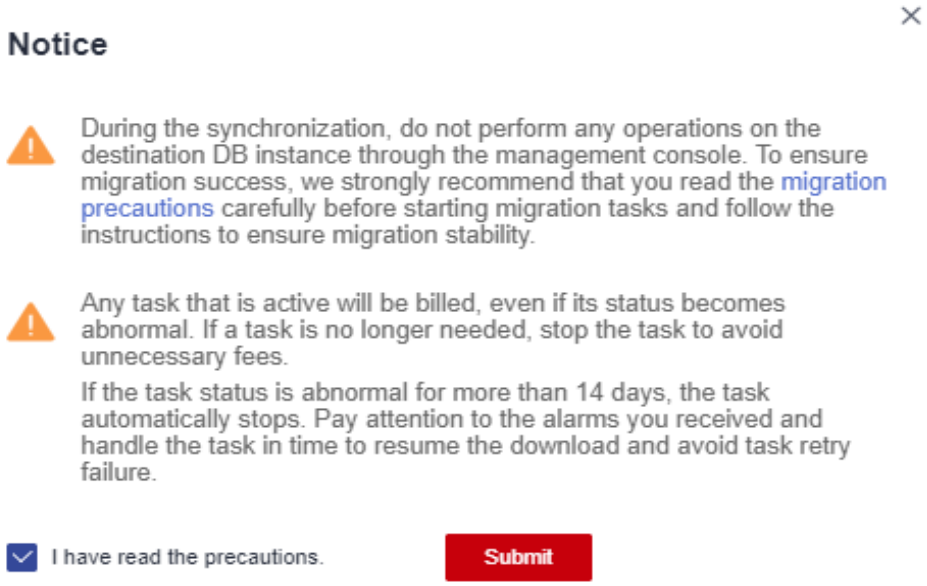
Wait until the database parameter check is successful. If the check fails, click **Check Again**.



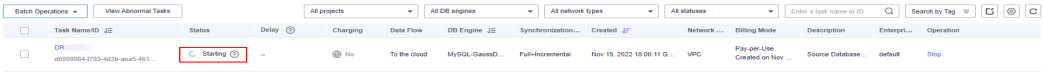
Step 11 Click **Next**, select **Start upon task creation**, verify other information, and click **Submit** in the lower right corner.



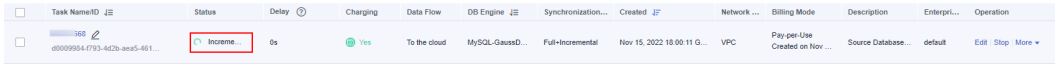
Step 12 In the dialog box that is displayed, confirm the information, select **I have read and understand this notice**, and click **Start Task**.



Go back to the **Data Synchronization Management** page and wait for about 5 to 10 minutes. The synchronization is started successfully.



Wait for about 5 minutes and continue with **Step 4: Verify Data Synchronization**.

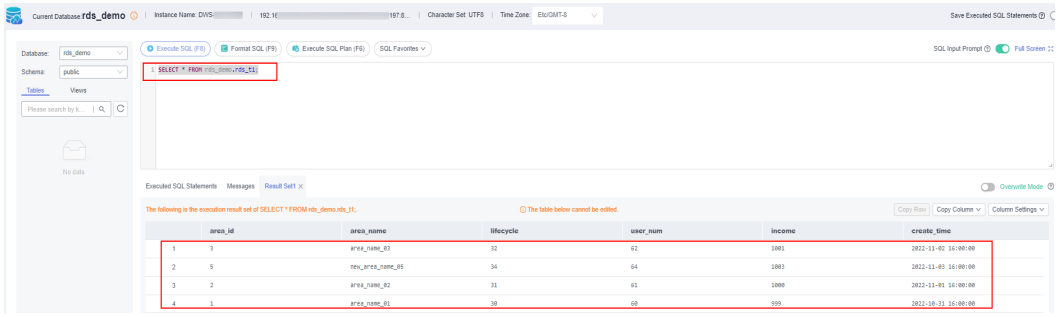


----End

Step 4: Verify Data Synchronization

Step 1 Log in to GaussDB(DWS) console again, and run the following statement to query the table data again. If the result is shown as follows, the full data synchronization is successful.

```
SELECT * FROM rds_demo.rds_t1;
```



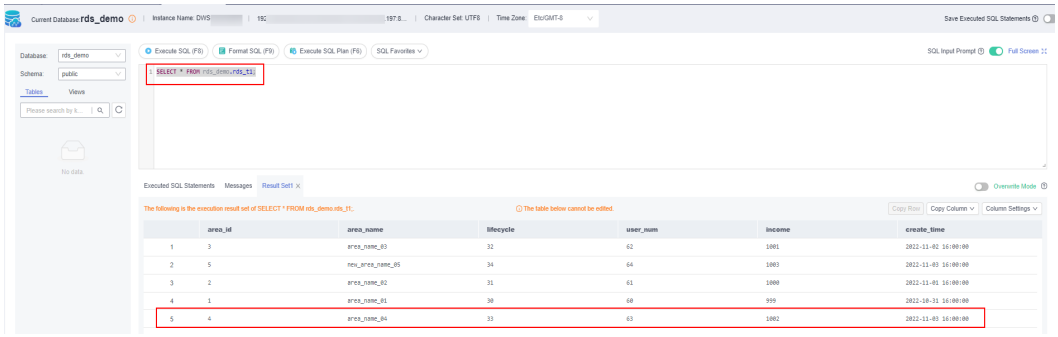
Step 2 Switch to the RDS console, log in to the RDS database, and insert new data into the table **rds_t1**.

```
INSERT INTO rds_t1 VALUES ('5','new_area_name_05',34,64,1003,'2022-11-04');
```


Step 3 Switch back to the GaussDB(DWS) database and run the following statement to query table data:

A row of data is added to the query result, indicating that the data in the MySQL database has been synchronized to GaussDB(DWS) in real time.

```
SELECT * FROM rds_demo.rds_t1;
```



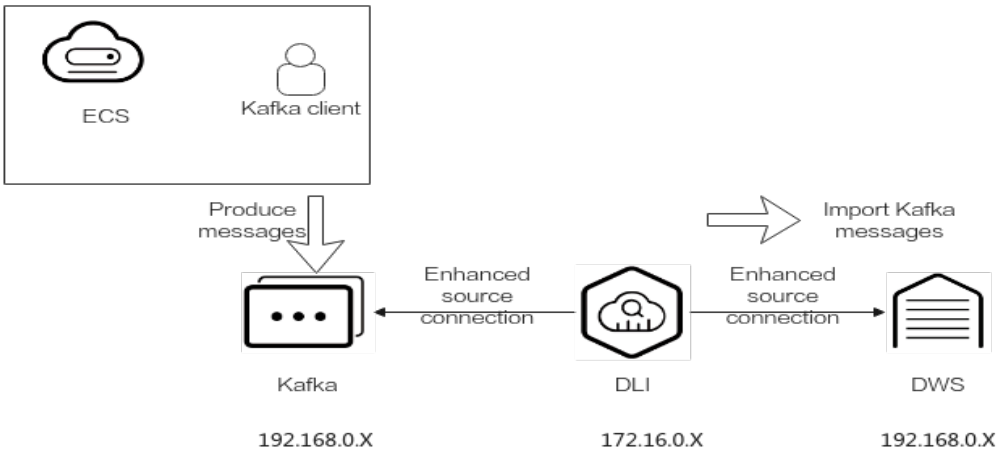
----End

2.4 Using a Flink Job of DLI to Synchronize Kafka Data to a GaussDB(DWS) Cluster in Real Time

This practice demonstrates how to use DLI Flink jobs to synchronize consumption data from Kafka to GaussDB(DWS) in real time. The demonstration process includes writing and updating existing data in real time.

- For details, see **What Is Data Lake Insight?**
- For details about Kafka, see **What Is DMS for Kafka?**

Figure 2-17 Importing Kafka data to GaussDB(DWS) in real time



This practice takes about 90 minutes. The cloud services used in this practice include **Virtual Private Cloud (VPC) and subnets**, **Elastic Load Balance (ELB)**, **Elastic Cloud Server (ECS)**, **Object Storage Service (OBS)**, **Distributed Message Service (DMS) for Kafka**, **Data Lake Insight (DLI)**, and **Data Warehouse Service (DWS)**. The basic process is as follows:

1. **Preparations**
2. **Step 1: Creating a Kafka Instance**
3. **Step 2: Creating a GaussDB(DWS) Cluster and Target Table**
4. **Step 3: Creating a DLI Queue**
5. **Step 4: Creating an Enhanced Datasource Connection for Kafka and GaussDB(DWS)**
6. **Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink**
7. **Step 6: Creating and Editing a DLI Flink Job**
8. **Step 7: Creating and Modifying Messages on the Kafka Client**

Scenario Description

Assume that the sample data of the data source Kafka is a user information table, as shown in **Table 2-16**, which contains the **id**, **name**, and **age** fields. The **id** field is unique and fixed, which is shared by multiple service systems. Generally, the **id** field does not need to be modified. Only the **name** and **age** fields need to be modified.

Use Kafka to generate the following three groups of data and use DLI Flink jobs to synchronize the data to GaussDB(DWS): Change the users whose IDs are **2** and **3** to **jim** and **tom**, and use DLI Flink jobs to update data and synchronize the data to GaussDB(DWS).

Table 2-16 Sample data

id	name	age
1	lily	16
2	lucy > jim	17
3	lilei > tom	15

Constraints

- Ensure that VPC, ECS, OBS, Kafka, DLI, and GaussDB(DWS) are in the same region, for example, China-Hong Kong.
- Ensure that Kafka, DLI, and GaussDB(DWS) can communicate with each other. In this practice, Kafka and GaussDB(DWS) are created in the same region and VPC, and the security groups of Kafka and GaussDB(DWS) allow the network segment of the DLI queues.
- To ensure that the link between DLI and GaussDB(DWS) is stable, bind the ELB service to the created GaussDB(DWS) cluster.

Preparations

- You have registered a Huawei account and enabled Huawei Cloud services.. Before using GaussDB(DWS), check the account status. The account cannot be in arrears or frozen.

- You have created a VPC and subnet. For details, see **Creating a VPC**.

Step 1: Creating a Kafka Instance

Step 1 Log in to the Huawei Cloud management console and choose **Middleware** > **Distributed Message Service (for Kafka)** from the service list. The Kafka management console is displayed.

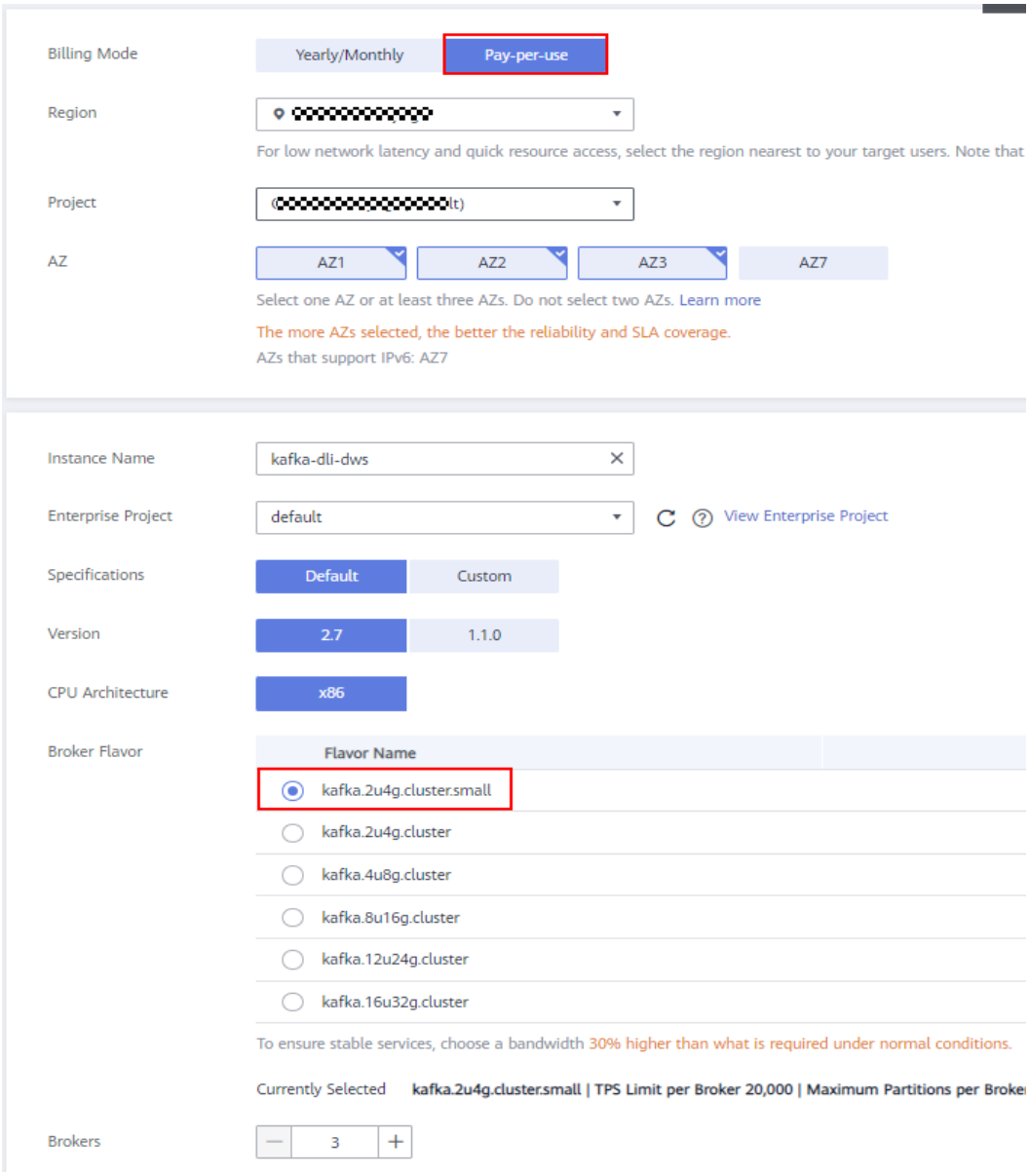
Step 2 Click **DMS for Kafka** on the left and click **Buy Instance** in the upper right corner.

Step 3 Set the following parameters. Retain the default values for other parameters that are not described in the table.

Table 2-17 Kafka instance parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong
Project	Default
AZ	AZ 1 (If not available, select another AZ.)
Instance Name	kafka-dli-dws
Enterprise Project	default
Specifications	Default
Version	2.7
CPU Architecture	x86
Broker Flavor	kafka.2u4g.cluster.small (For reference only. Select the smallest flavor.)
Brokers	3
VPC	Select a created VPC. If no VPC is available, create one.
Security Group	Select a created security group. If no security group is available, create one.
Other parameters	Retain the default value.

Figure 2-18 Creating a Kafka instance



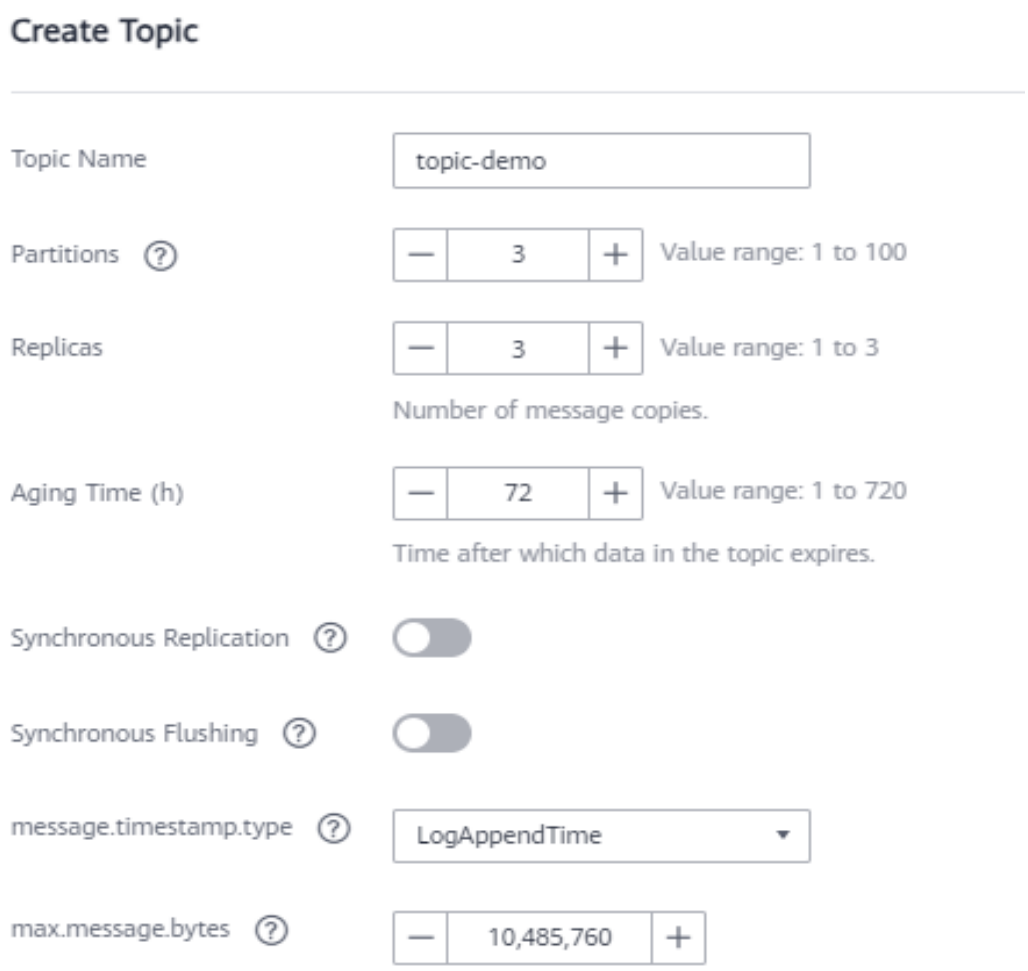
Step 4 Click **Buy** and complete the payment. Wait until the creation is successful.

Step 5 In the Kafka instance list, click the name of the created Kafka instance. The **Basic Information** page is displayed.

Step 6 Choose **Topics** on the left and click **Create Topic**.

Set **Topic Name** to **topic-demo** and retain the default values for other parameters.

Figure 2-19 Creating a topic



Step 7 Click **OK**. In the topic list, you can see that **topic-demo** is successfully created.

Step 8 Choose **Consumer Groups** on the left and click **Create Consumer Group**.

Step 9 Enter **kafka01** for **Consumer Group Name** and click **OK**.

----End

Step 2: Creating a GaussDB(DWS) Cluster and Target Table

Step 1 Create a dedicated load balancer, set **Network Type** to **IPv4 private network**. Set Region and VPC to the same values as those of the Kafka instance. In this example, set Region to China-Hong Kong.

Step 2 Creating a Cluster. To ensure network connectivity, the region and VPC of the GaussDB(DWS) cluster must be the same as those of the Kafka instance. In this practice, the region and VPC are China-Hong Kong. The VPC must be the same as that created for Kafka.

Step 3 Log in to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters**, locate the row that contains the target cluster, and click **Login** in the **Operation** column.

> **NOTE**
>
> This practice uses version 8.1.3.x as an example. 8.1.2 and earlier versions do not support this login mode. You can use Data Studio to connect to a cluster. For details, see **Using Data Studio to Connect to a Cluster**.

Step 4 After the login is successful, the SQL editor is displayed.

Step 5 Copy the following SQL statement. In the SQL window, click Execute SQL to create the target table **user_dws**.

```
CREATE TABLE user_dws (
id int,
name varchar(50),
age int,
PRIMARY KEY (id)
);
```

----End

Step 3: Creating a DLI Queue

Step 1 Log in to the Huawei Cloud management console and choose **Analytics** > **Data Lake Insight** from the service list. The DLI management console is displayed.

Step 2 In the navigation pane on the left, choose **Resources** > **Resource Pool**.

Step 3 Click **Buy Resource Pool** in the upper right corner, set the following parameters, and retain the default values for other parameters that are not described in the table.

Table 2-18 DLI queue parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong
Name	dli_dws
Specifications	Standard
CIDR Block	172.16.0.0/18. It must be in a different network segment from Kafka and GaussDB(DWS). For example, if Kafka and GaussDB(DWS) are in the **192.168.x.x** network segment, select **172.16.x.x** for DLI.

Step 4 Click **Buy** and click **Submit**.

After the resource pool is created, go to the next step.

Step 5 On the elastic resource pool page, locate the row that contains the created resource pool, click **Add Queue** in the **Operation** column, and set the following parameters. Retain the default values for other parameters that are not described in the table.

Table 2-19 Adding a queue

Parameter	Value
Name	dli_dws
Type	General purpose queue

Step 6 Click **Next** and click **OK**. The queue is created.

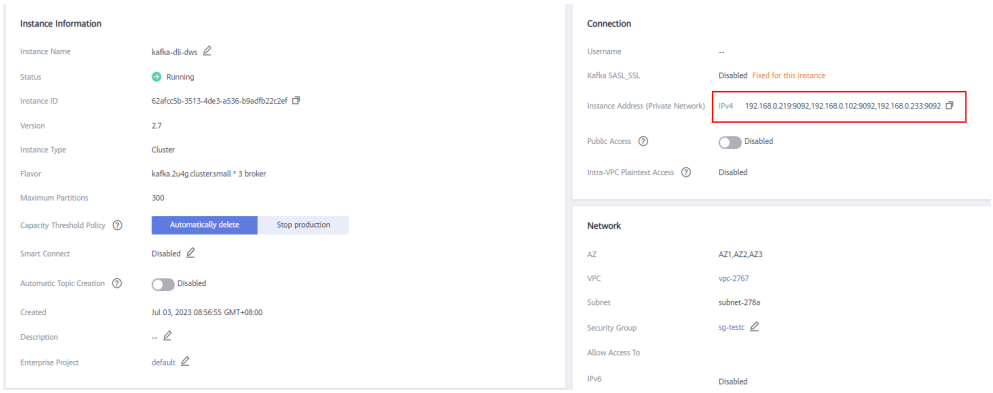
----End

Step 4: Creating an Enhanced Datasource Connection for Kafka and GaussDB(DWS)

Step 1 In the security group of Kafka, allow the network segment where the DLI queue is located.

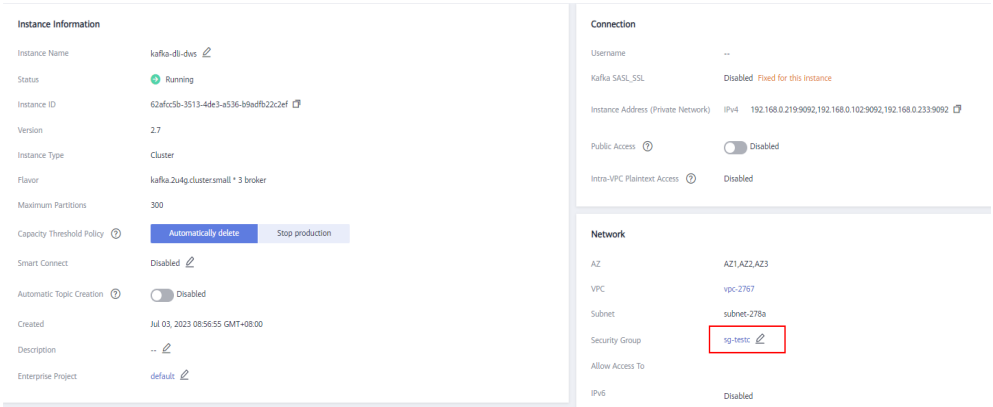
1. Return to the Kafka console and click the Kafka instance name to go to the **Basic Information** page. View the value of **Instance Address (Private Network)** in connection information and record the address for future use.

   Figure 2-20 Kafka private network address

2. Click the security group name.

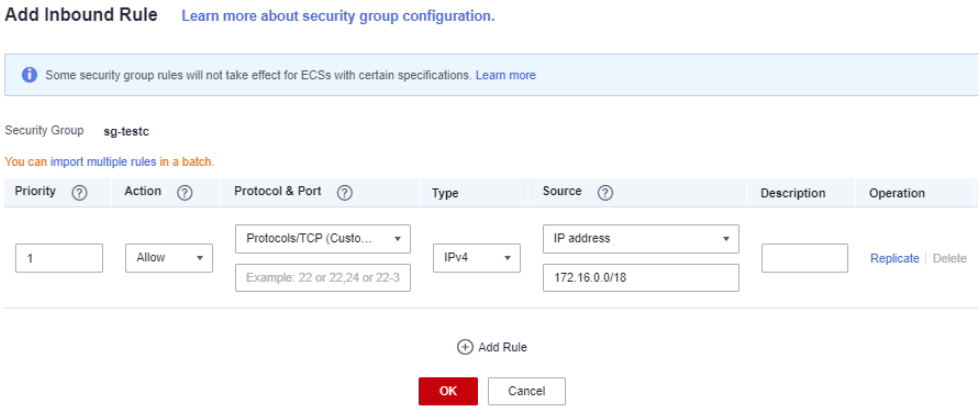
   Figure 2-21 Kafka security group

3. Choose **Inbound Rules** > **Add Rule**, as shown in the following figure. Add the network segment of the DLI queue. In this example, the network segment is

172.16.0.0/18. Ensure that the network segment is the same as that entered during **Step 3: Creating a DLI Queue**.

Figure 2-22 Adding rules to the Kafka security group



4. Click **OK**.

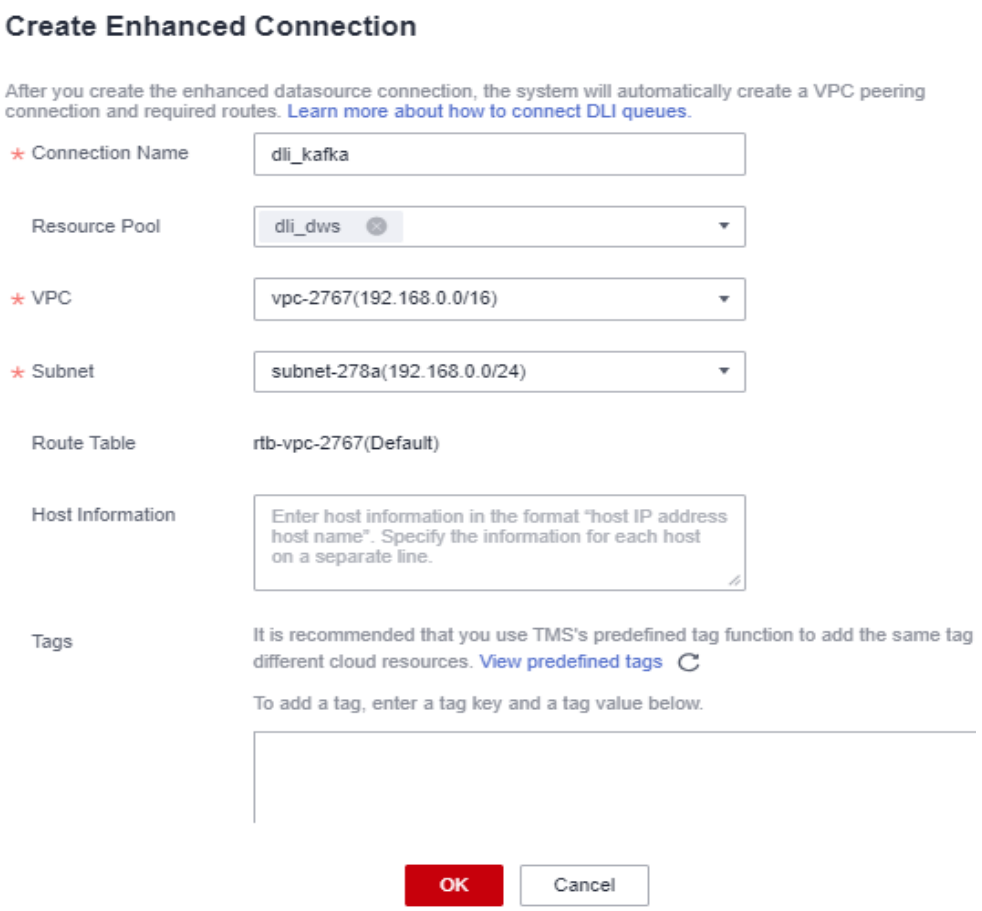
Step 2 Return to the DLI management console, click **Datasource Connections** on the left, select **Enhanced**, and click **Create**.

Step 3 Set the following parameters. Retain the default values for other parameters that are not described in the table.

Table 2-20 Connection from DLI to Kafka

Parameter	Value
Connection Name	dli_kafka
Resource Pool	Select the created DLI queue **dli_dws**.
VPC	Select the VPC of Kafka.
Subnet	Select the subnet where Kafka is located.
Other parameters	Retain the default value.

Figure 2-23 Creating an enhanced connection

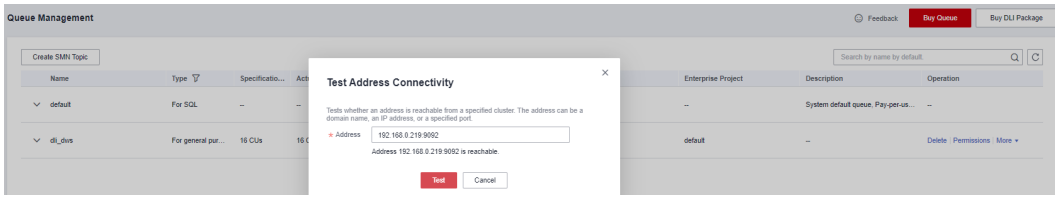


Step 4 Click **OK**. Wait until the Kafka connection is successfully created.

Step 5 Choose **Resources** > **Queue Management** on the left, and choose **More** > **Test Address Connectivity** on the right of **dli_dws**.

Step 6 In the address box, enter the private IP address and port number of the Kafka instance obtained in **Step 1.1**. (There are three Kafka addresses. Enter only one of them.)

Figure 2-24 Testing Kafka connectivity

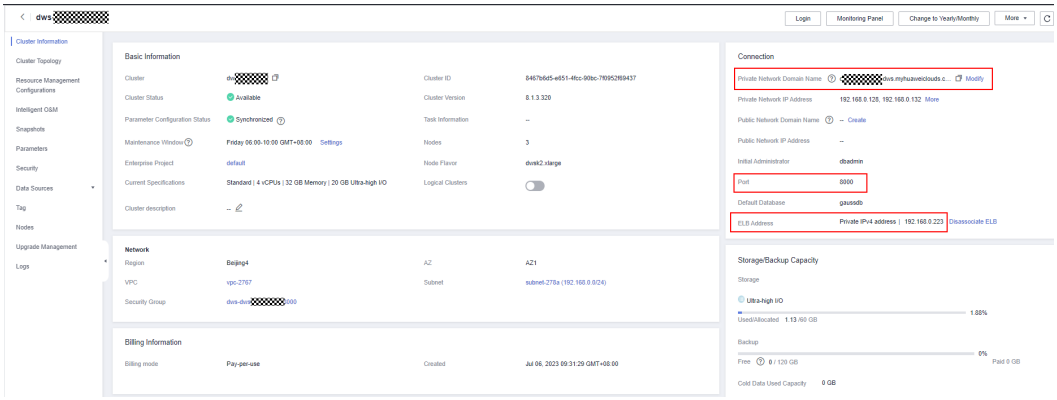


Step 7 Click **Test** to verify that DLI is successfully connected to Kafka.

Step 8 Log in to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters** on the left, and click the cluster name to go to the details page.

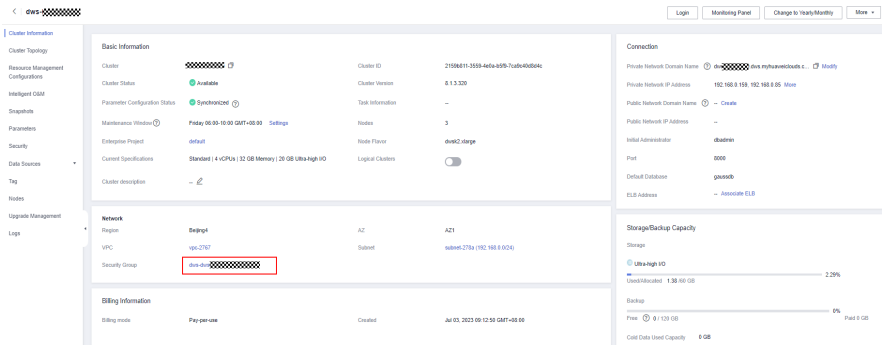
Step 9 Record the private network domain name, port number, and ELB address of the GaussDB(DWS) cluster for future use.

Figure 2-25 Private domain name and ELB address



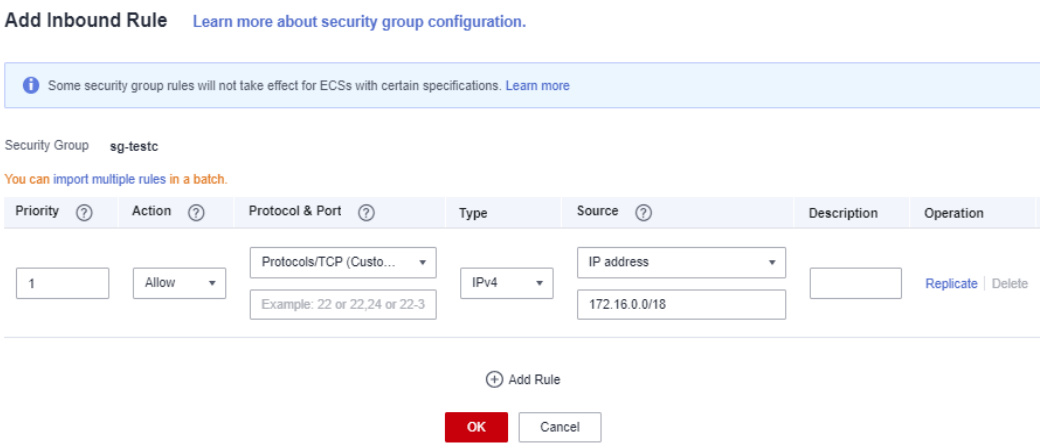
Step 10 Click the security group name.

Figure 2-26 GaussDB(DWS) security group



Step 11 Choose **Inbound Rules** > **Add Rule**, as shown in the following figure. Add the network segment of the DLI queue. In this example, the network segment is **172.16.0.0/18**. Ensure that the network segment is the same as that entered during **Step 3: Creating a DLI Queue**.

Figure 2-27 Adding a rule to the GaussDB(DWS) security group

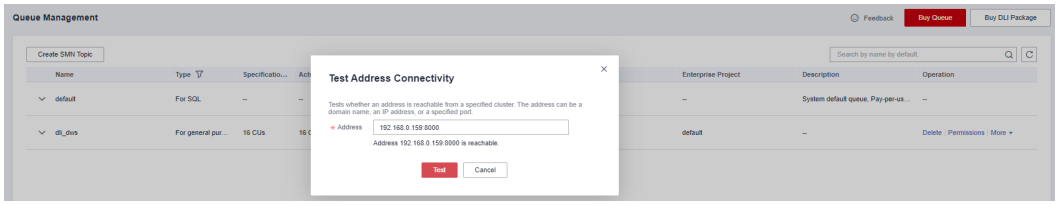


Step 12 Click **OK**.

Step 13 Switch to the DLI console, choose **Resources** > **Queue Management** on the left, and click **More** > **Test Address Connectivity** on the right of **dli_dws**.

Step 14 In the address box, enter the ELB address and port number of the GaussDB(DWS) cluster obtained in **Step 9**.

Figure 2-28 Testing GaussDB(DWS) connectivity



Step 15 Click **Test** to verify that DLI is successfully connected to GaussDB(DWS).

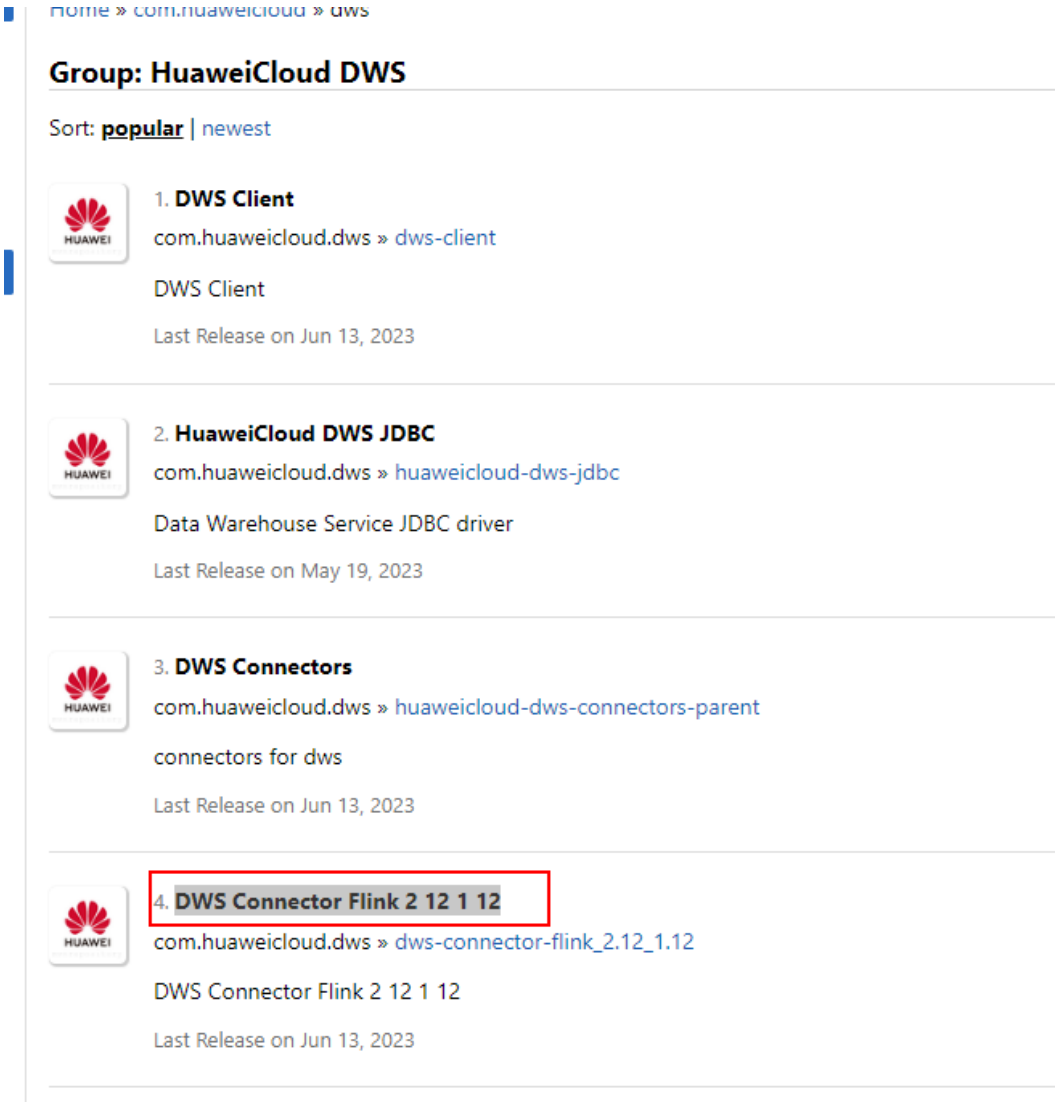
----End

Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink

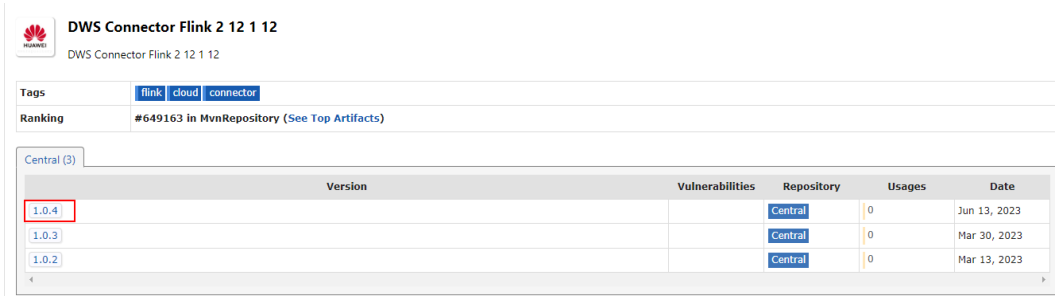
dws-connector-flink is a tool for interconnecting with Flink based on GaussDB(DWS) JDBC APIs. During DLI job configuration, this tool and its dependencies are stored in the Flink class loading directory to improve the capability of importing Flink jobs to GaussDB(DWS).

Step 1 Go to **https://mvnrepository.com/artifact/com.huaweicloud.dws** using a browser.

Step 2 In the software list, select the latest version of GaussDB(DWS) Connectors Flink. In this practice, select **DWS Connector Flink 2 12 1 12**.

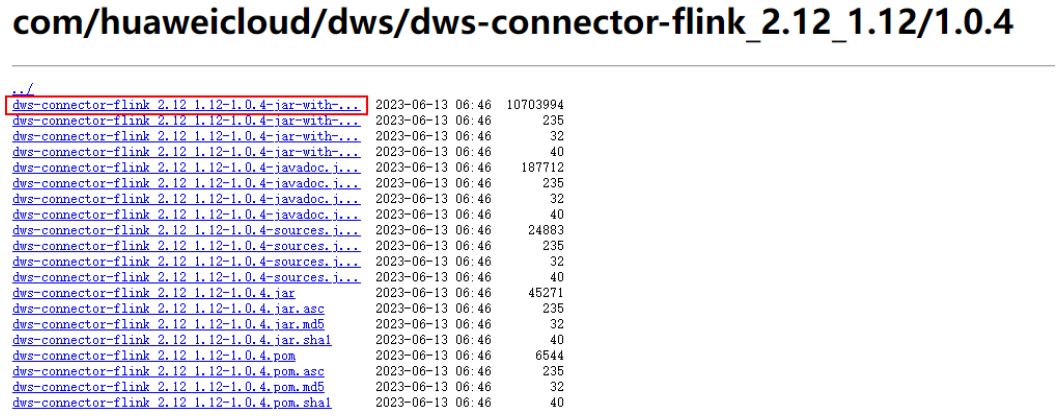


Step 3 Click the 1.0.4 branch. (Click the newest branch in actual scenarios).



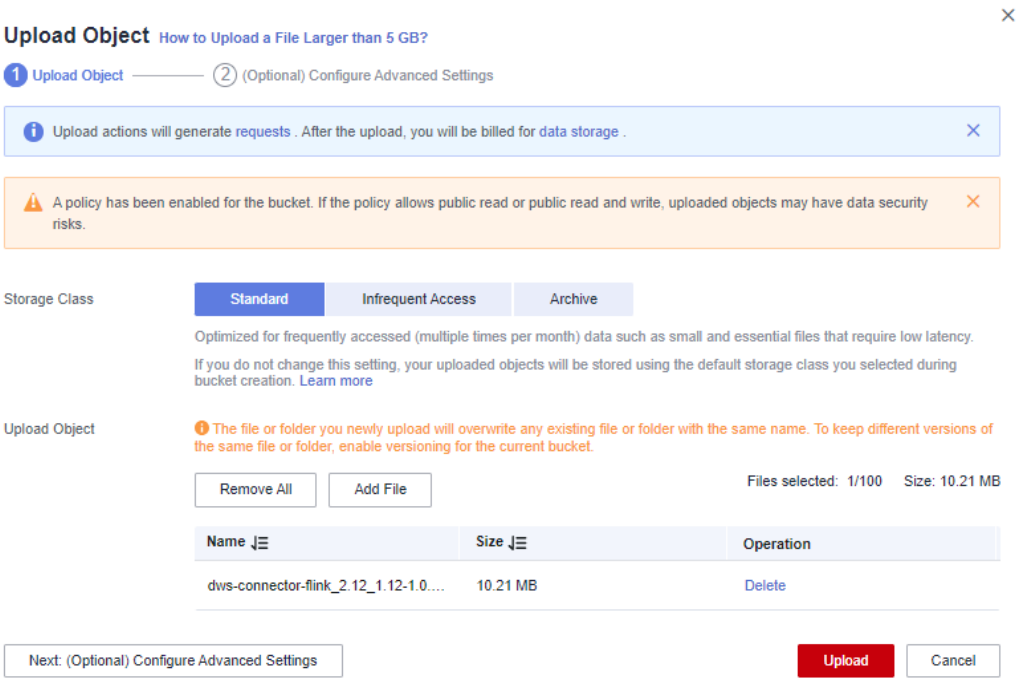
Step 4 Click **View ALL**.

Step 5 Click **dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar** to download it to the local host.



Step 6 Create an OBS bucket. In this practice, set the bucket name to **obs-flink-dws** and upload the file to the OBS bucket. Ensure that the bucket is in the same region as DLI, which in this practice is China-Hong Kong.

Figure 2-29 Uploading the JAR package to the OBS bucket



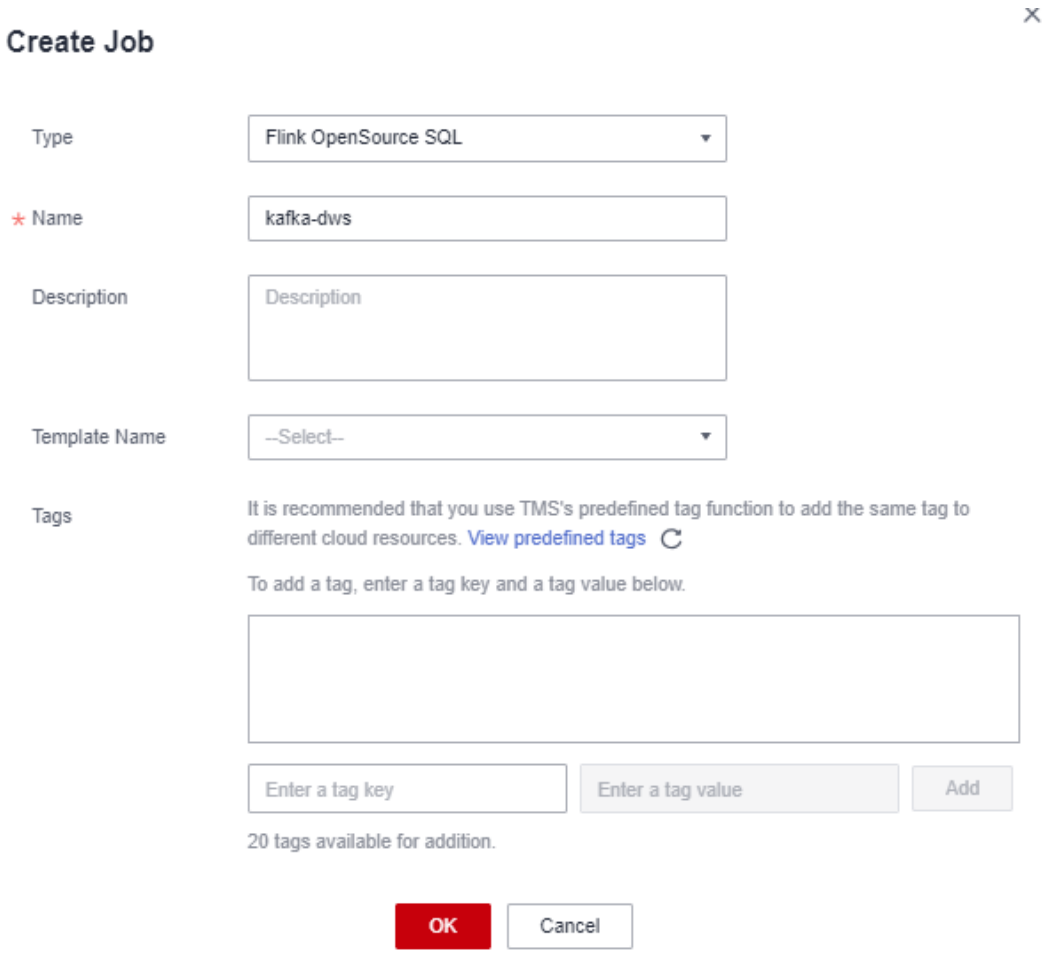
----End

Step 6: Creating and Editing a DLI Flink Job

Step 1 Return to the DLI management console, choose **Job Management** > **Flink Jobs** on the left, and click **Create Job** in the upper right corner.

Step 2 Set **Type** to **Flink OpenSource SQL** and **Name** to **kafka-dws**.

Figure 2-30 Creating a job



Step 3 Click **OK**. The page for editing the job is displayed.

Step 4 Set the following parameters on the right of the page. Retain the default values for other parameters that are not described in the table.

Table 2-21 Flink job parameters

Parameter	Value
Queue	dli_dws
Flink Version	1.12

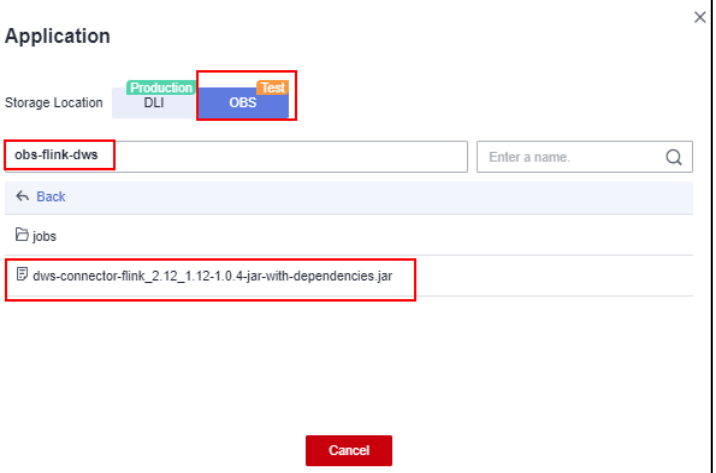
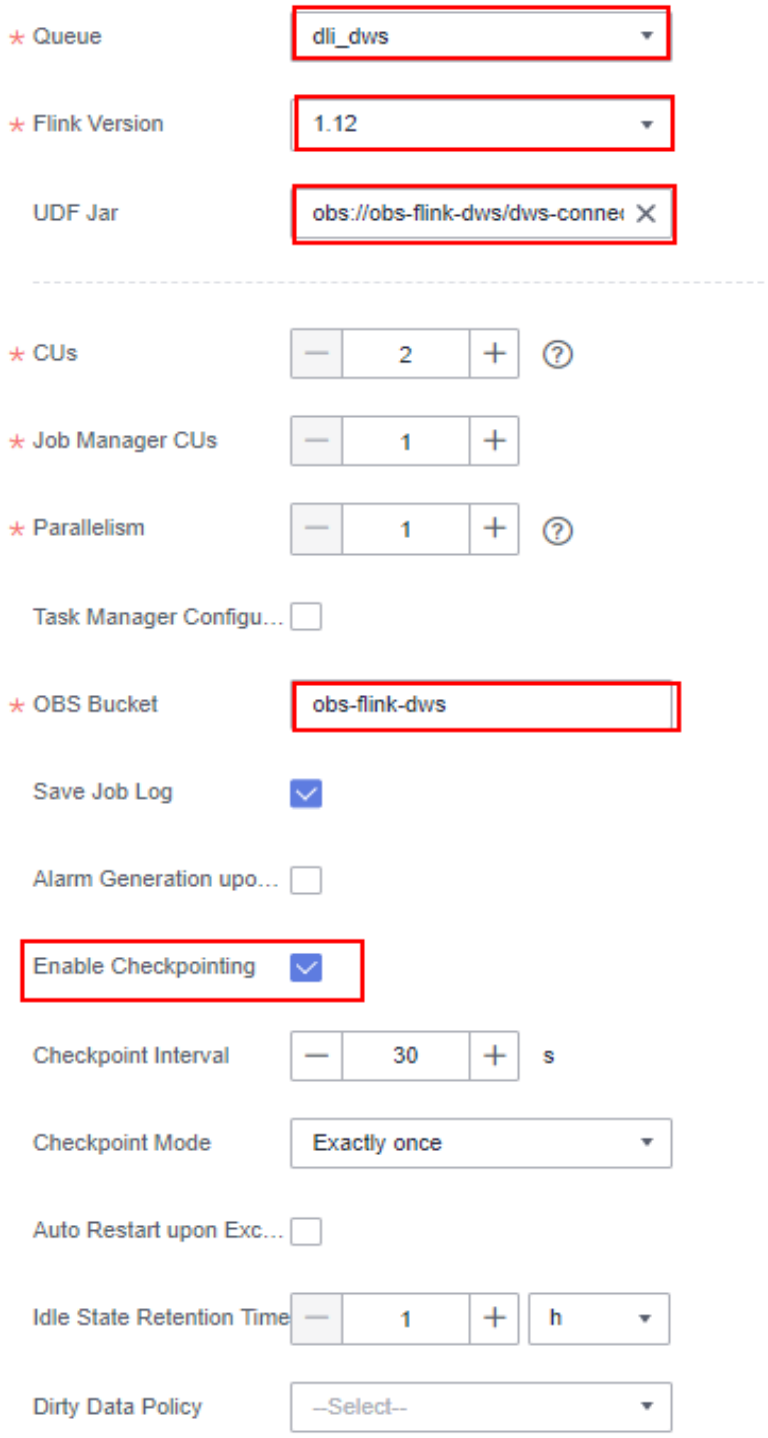
Parameter	Value
UDF Jar	Select the JAR file in the OBS bucket created in **Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink**. 
OBS Bucket	Select the bucket created in **Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink**.
Enable Checkpointing	Check the box.
Other parameters	Retain the default value.

Figure 2-31 Editing a job



Step 5 Copy the following SQL code to the SQL code window on the left.

Obtain the private IP address and port number of the Kafka instance from **Step 1.1**, and obtain the private domain name from **Step 9**.

```
CREATE TABLE user_kafka (
  id string,
  name string,
  age int
) WITH (
```

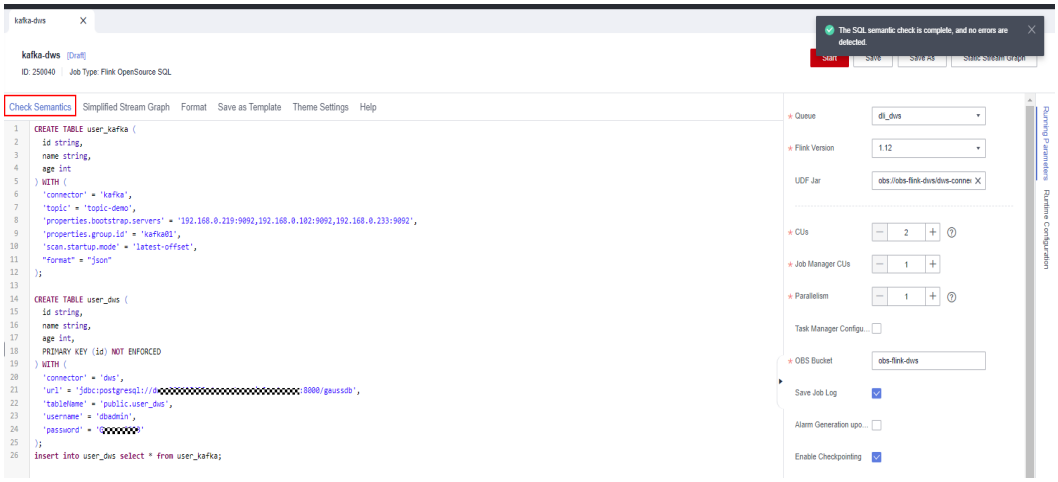


```
  'connector' = 'kafka',
  'topic' = 'topic-demo',
'properties.bootstrap.servers' ='Private IP address and port number of the Kafka instance',
  'properties.group.id' = 'kafka01',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE user_dws (
  id string,
  name string,
  age int,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'dws',
'url'='jdbc:postgresql://GaussDB(DWS) private network domain name:8000/gaussdb',
  'tableName' = 'public.user_dws',
  'username' = 'dbadmin',
'password' ='Password of database user dbdamin'
);

INSERT INTO user_dws select * from user_kafka;
```

Step 6 Click **Check Semantics** and wait until the verification is successful.

If the verification fails, check whether the SQL input has syntax errors.

Figure 2-32 SQL statement of a job



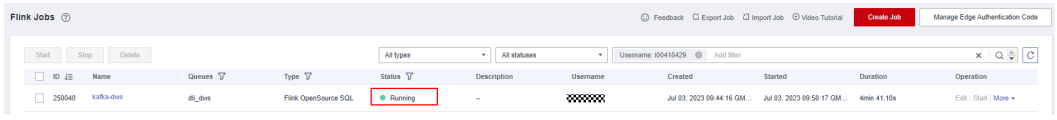
Step 7 Click **Save**.

Step 8 Return to the DLI console home page and choose **Job Management** > **Flink Jobs** on the left.

Step 9 Click **Start** on the right of the job name **kafka-dws** and click **Start Now**.

Wait for about 1 minute and refresh the page. If the status is **Running**, the job is successfully executed.

Figure 2-33 Job execution status

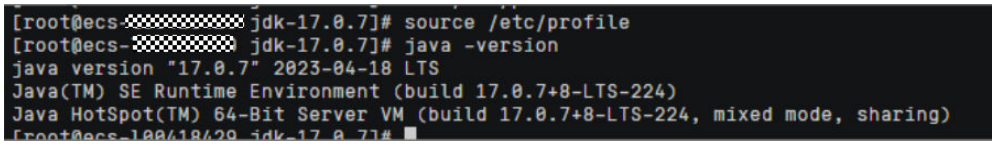


----End

Step 7: Creating and Modifying Messages on the Kafka Client

Step 1 Create an ECS by referring to the ECS document. Ensure that the region and VPC of the ECS are the same as those of Kafka.

Step 2 Install JDK.

1. Log in to the ECS, go to the **/usr/local** directory, and download the JDK package.
   ```
   cd /usr/local
   wget https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz
   ```
2. Decompress the downloaded JDK package.
   ```
   tar -zxvf jdk-17_linux-x64_bin.tar.gz
   ```
3. Run the following command to open the **/etc/profile** file:
   ```
   vim /etc/profile
   ```
4. Press **i** to enter editing mode and add the following content to the end of the **/etc/profile** file:
   ```
   export JAVA_HOME=/usr/local/jdk-17.0.7 #JDK installation directory
   export JRE_HOME=${JAVA_HOME}/jre
   export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:${JAVA_HOME}/test:${JAVA_HOME}/lib/gsjdbc4.jar:${JAVA_HOME}/lib/dt.jar:${JAVA_HOME}/lib/tools.jar:$CLASSPATH
   export JAVA_PATH=${JAVA_HOME}/bin:${JRE_HOME}/bin
   export PATH=$PATH:${JAVA_PATH}
   ```
   
5. Press **Esc** and enter **:wq!** to save the settings and exit.
6. Run the following command for the environment variables to take effect:
   ```
   source /etc/profile
   ```
7. Run the following command. If the following information is displayed, the JDK is successfully installed:
   ```
   java -version
   ```
   

Step 3 Install the Kafka client.

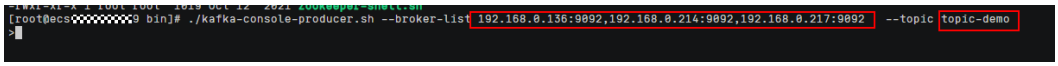
1. Go to the **/opt** directory and run the following command to obtain the Kafka client software package.
   ```
   cd /opt
   wget https://archive.apache.org/dist/kafka/2.7.2/kafka_2.12-2.7.2.tgz
   ```
2. Decompress the downloaded software package.
   ```
   tar -zxf kafka_2.12-2.7.2.tgz
   ```
3. Go to the Kafka client directory.
   ```
   cd /opt/kafka_2.12-2.7.2/bin
   ```

Step 4 Run the following command to connect to Kafka: {*Connection address*} indicates the internal network connection address of Kafka. For details about how to obtain the address, see **Step 1.1**. *topic* indicates the name of the Kafka topic created in **Step 6**.

```
./kafka-console-producer.sh --broker-list {connection address} --topic {Topic name}
```

The following is an example:

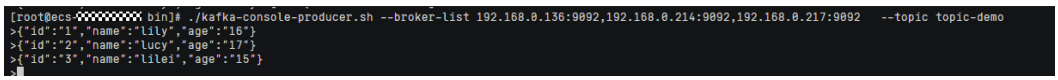
```
./kafka-console-producer.sh --broker-list
192.168.0.136:9092,192.168.0.214:9092,192.168.0.217:9092 --topic topic-demo
```

If **>** is displayed and no other error message is displayed, the connection is successful.

Step 5 In the window of the connected Kafka client, copy the following content (one line at a time) based on the data planned in the **Scenario Description** and press **Enter** to produce messages:

```
{"id":"1","name":"lily","age":"16"}
{"id":"2","name":"lucy","age":"17"}
{"id":"3","name":"lilei","age":"15"}
```



Step 6 Return to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters** on the left, and click **Log In** on the right of the GaussDB(DWS) cluster. The SQL page is displayed.

Step 7 Run the following SQL statement to verify that data is successfully imported to the database in real time:

```
SELECT * FROM user_dws ORDER BY id;
```

	id	name	age
1	1	lily	16
2	2	lucy	17
3	3	lilei	15

Step 8 Go back to the client window for connecting to Kafka on the ECS, copy the following content (one line at a time), and press **Enter** to produce messages.

```
{"id":"2","name":"jim","age":"17"}
{"id":"3","name":"tom","age":"15"}
```

Step 9 Go back to the opened SQL window of GaussDB(DWS) and run the following SQL statement. It is found that the names whose IDs are **2** and **3** have been changed to **jim** and **tom**.

The scenario description is as expected. End of this practice.

```
SELECT * FROM user_dws ORDER BY id;
```

	id	name	age
1	1	lily	16
2	2	jim	17
3	3	tom	15

----End

2.5 Using a Flink Job of DLI to Synchronize MySQL Data to a GaussDB(DWS) Cluster in Real Time

This practice demonstrates how to use a Flink job of DLI to synchronize MySQL data to GaussDB(DWS) in real time.

For details, see **What Is Data Lake Insight?**

This exercise lasts for approximately 60 minutes and involves utilizing various cloud services such as Virtual Private Cloud (VPC) and Subnet, Relational

Database Service (RDS), Data Lake Insight (DLI), Object Storage Service (OBS), and GaussDB(DWS). The following is an outline of the exercise.

1. **Preparations**
2. **Step 1: Preparing MySQL Data**
3. **Step 2: Creating a GaussDB(DWS) Cluster**
4. **Step 3: Creating a DLI Queue**
5. **Step 4: Creating an Enhanced Datasource Connection**
6. **Step 5: Creating a DLI Flink Job**
7. **Step 6: Verifying Data Synchronization**
8. **More Information**

Preparations

- You have registered a Huawei ID and enabled Huawei Cloud services.. The account cannot be in arrears or frozen.
- You have created a VPC and subnet. For details, see **Creating a VPC**.

Step 1: Preparing MySQL Data

Step 1 Buy an RDS instance and set the parameters listed in **Table 2-22** (retain the default values for other parameters). For details, see **Relational Database Service**.

Table 2-22 RDS parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong
DB Instance Name	rds-demo
DB Engine	MySQL
DB Engine Version	5.7 or later
Database Port	3306

Step 2 Connect to the RDS instance and create an instance named **mys_data**.

```
CREATE DATABASE mys_data;
```

Step 3 Switch to the new database **mys_data** and run the following command to create the **mys_orders** table:

```
CREATE TABLE mys_data.mys_order
    ( order_id      VARCHAR(12),
      order_channel VARCHAR(32),
      order_time    DATETIME,
      cust_code     VARCHAR(6),
      pay_amount    DOUBLE,
      real_pay      DOUBLE,
      PRIMARY KEY (order_id) );
```


Step 4 insert data to the table.

```
INSERT INTO mys_data.mys_order VALUES ('202306270001', 'webShop', TIMESTAMP('2023-06-27 10:00:00'), 'CUST1', 1000, 1000);
INSERT INTO mys_data.mys_order VALUES ('202306270002', 'webShop', TIMESTAMP('2023-06-27 11:00:00'), 'CUST2', 5000, 5000);
```

Step 5 Check whether the data is inserted.

```
SELECT * FROM mys_data.mys_order;
```

----End

Step 2: Creating a GaussDB(DWS) Cluster

Step 1 **Creating a Cluster**. To ensure network connectivity, select the same region and VPC as those of the RDS instance. In this practice, select China-Hong Kong. The VPC must be the same as that created for RDS.

Step 2 Log in to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters**, locate the row that contains the target cluster, and click **Login** in the **Operation** column. The login information is as follows:

- Cluster: the created GaussDB(DWS) cluster.
- Database: gaussdb
- Data source name: dws-demo-01
- Username: dbadmin
- Password: password set when the GaussDB(DWS) cluster is created

Step 3 Select **Remember Password**, click **Test Connection**, and wait until the connection is successful.

Step 4 Copy the following SQL statements. In the SQL window, click **Execute SQL** to create a schema named **dws_data**.

```
CREATE SCHEMA dws_data;
```

Step 5 Create the **dws_order** table in the new schema.

```
CREATE TABLE dws_data.dws_order
      ( order_id      VARCHAR(12),
        order_channel VARCHAR(32),
        order_time    TIMESTAMP,
        cust_code     VARCHAR(6),
        pay_amount    DOUBLE PRECISION,
        real_pay      DOUBLE PRECISION );
```

Step 6 Query data. The current table is empty.

```
SELECT * FROM dws_data.dws_order;
```

----End

Step 3: Creating a DLI Queue

Step 1 Log in to the Huawei Cloud console and choose **Analytics** > **Data Lake Insight** from the service list. The DLI console is displayed.

Step 2 In the navigation pane on the left, choose **Resources** > **Resource Pool**.

Step 3 Click **Buy Resource Pool** in the upper right corner, set the following parameters, and retain the default values for other parameters that are not described in the table.

Table 2-23 DLI elastic resource pool

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong
Name	dli_dws
Specifications	Standard
CIDR Block	172.16.0.0/18, which must be in a different network segment from MySQL and GaussDB(DWS). For example, if MySQL and GaussDB(DWS) are in the **192.168.x.x** network segment, select **172.16.x.x** for DLI.

Step 4 Click **Buy** and click **Submit**.

After the resource pool is created, go to the next step.

Step 5 On the elastic resource pool page, locate the row that contains the created resource pool, click **Add Queue** in the **Operation** column, and set the following parameters. Retain the default values for other parameters that are not described in the table.

Table 2-24 Adding a queue

Parameter	Value
Name	dli_dws
Type	General purpose queue

Step 6 Click **Next** and click **OK**. The queue is created.

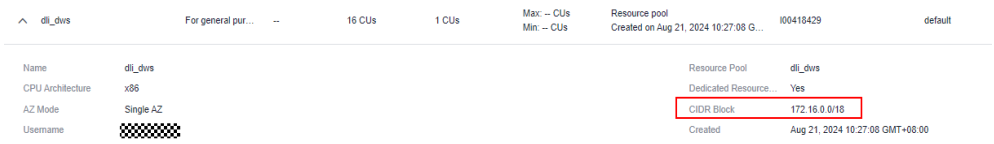
----End

Step 4: Creating an Enhanced Datasource Connection

Step 1 In the security group of RDS, allow the network segment where the DLI queue is located.

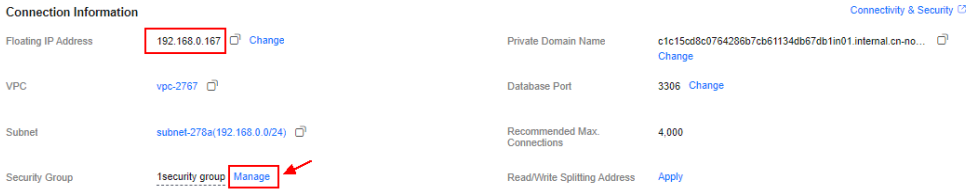
1. In the navigation pane on the left, choose **Resources** > **Queue Management** and record the network segment of **dli_dws**.

Figure 2-34 DLI queue network segment



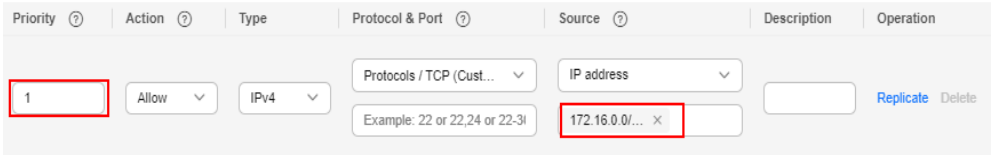
2. Go to the RDS console, choose **Instance Management** in the navigation pane, and click the name of the created RDS instance.
3. Record the value of **Private IP Address** in the **Connection Information** area, which will be used in the subsequent connectivity test.
4. Click **Manage** next to the security group in **Connection Information**.

Figure 2-35 RDS security group



5. In the security group list that is displayed, click the security group name to go to the security group configuration page.
6. Choose **Inbound Rules** > **Add Rule**, as shown in the following figure. Add the network segment of the DLI queue. In this example, the network segment is **172.16.0.0/18**. Ensure that the network segment is the same as that entered in **Step 3: Creating a DLI Queue**.

Figure 2-36 Adding a rule to the RDS security group



7. Click **OK**.

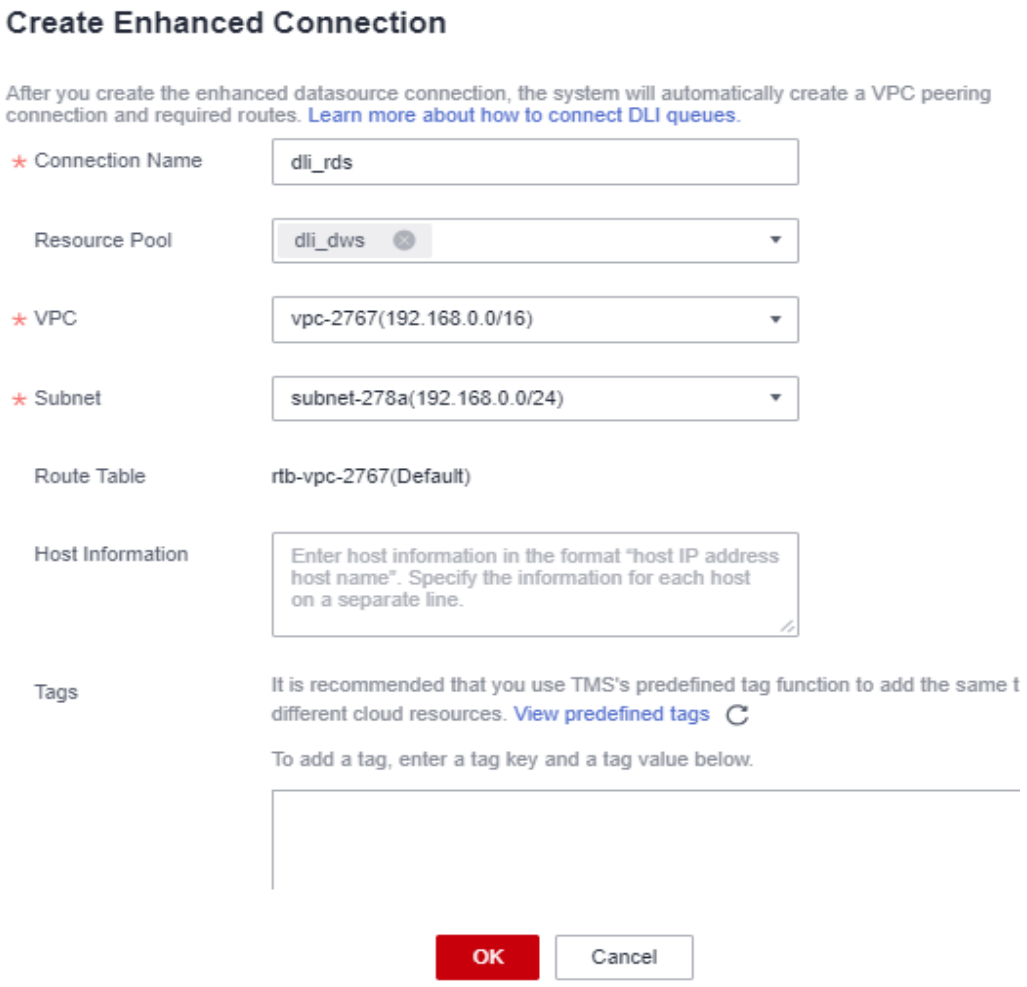
Step 2 Return to the DLI console, click **Datasource Connections** on the left, select **Enhanced**, and click **Create**.

Step 3 Set the following parameters. Retain the default values for other parameters that are not described in the table.

Table 2-25 Connection from DLI to RDS

Parameter	Value
Connection Name	dli_rds
Resource Pool	Select the created DLI elastic resource pool.
VPC	Select the VPC where RDS is located.
Subnet	Select the subnet where RDS is located.
Other parameters	Retain the default values.

Figure 2-37 Creating a datasource connection

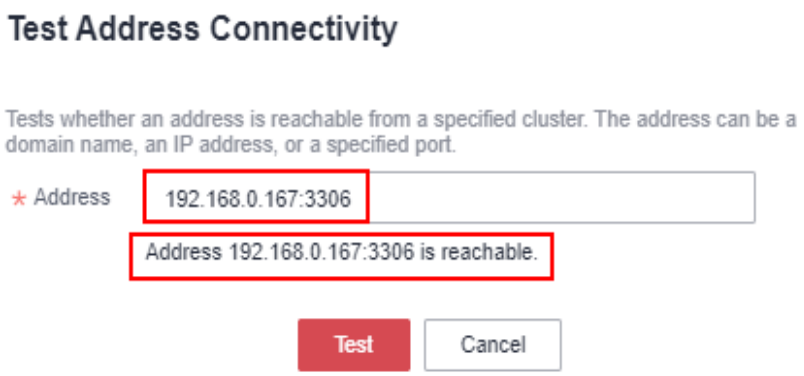


Step 4 Click **OK**. Wait until the RDS connection is created.

Step 5 Test the connectivity between DLI and RDS.

1. Choose **Resources** > **Queue Management** on the left, and choose **More** > **Test Address Connectivity** on the right of **dli_dws**.
2. Enter the private IP address of RDS recorded in **Step 1.3** and port **3306** in the address box.
3. Click **Test** to verify that DLI is successfully connected to RDS.

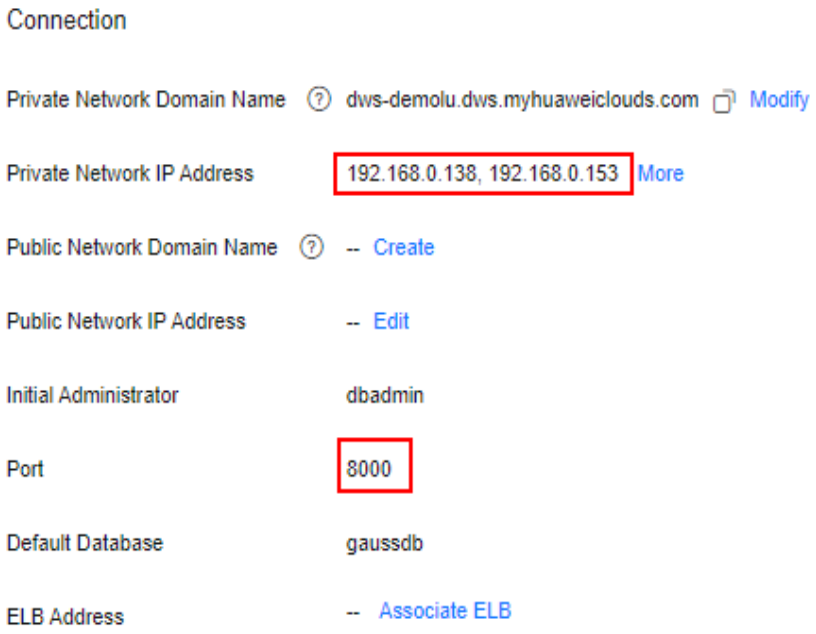
Figure 2-38 Testing the connection between RDS and DLI



Step 6 Test the connectivity between DLI and GaussDB(DWS).

1. Log in to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters** on the left, and click the cluster name to go to the details page.
2. As shown in the following figure, record the private IP address and port number of the GaussDB(DWS) cluster for future use.

Figure 2-39 GaussDB(DWS) internal IP address



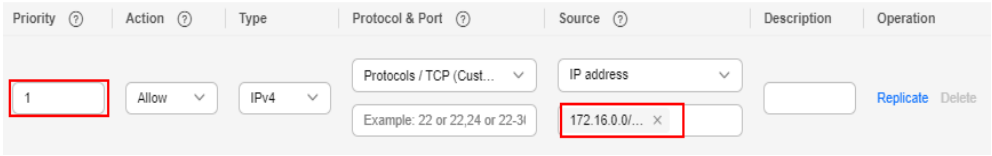
3. Click the security group name.

Figure 2-40 GaussDB(DWS) security group

4. Choose **Inbound Rules** > **Add Rule**, as shown in the following figure. Add the network segment of the DLI queue. In this example, the network segment is

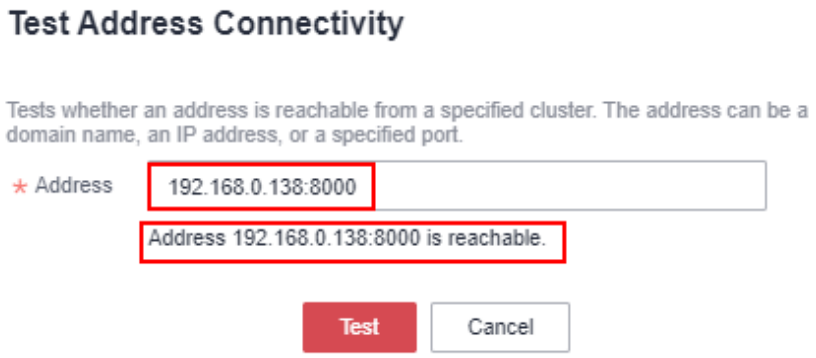
172.16.0.0/18. Ensure that the network segment is the same as that entered in 4.

Figure 2-41 Adding a rule to the GaussDB(DWS) security group



5. Click **OK**.
6. Switch to the DLI console, choose **Resources** > **Queue Management** on the left, and click **More** > **Test Address Connectivity** on the right of **dli_dws**.
7. In the address box, enter the private IP address and port number of the GaussDB(DWS) cluster.
8. Click **Test** to verify that DLI is successfully connected to GaussDB(DWS).

Figure 2-42 Testing GaussDB(DWS) connectivity



----End

Step 5: Creating a DLI Flink Job

Step 1 Log in to the OBS console and create an OBS bucket to store Flink jobs. For details, see the **OBS User Guide**.

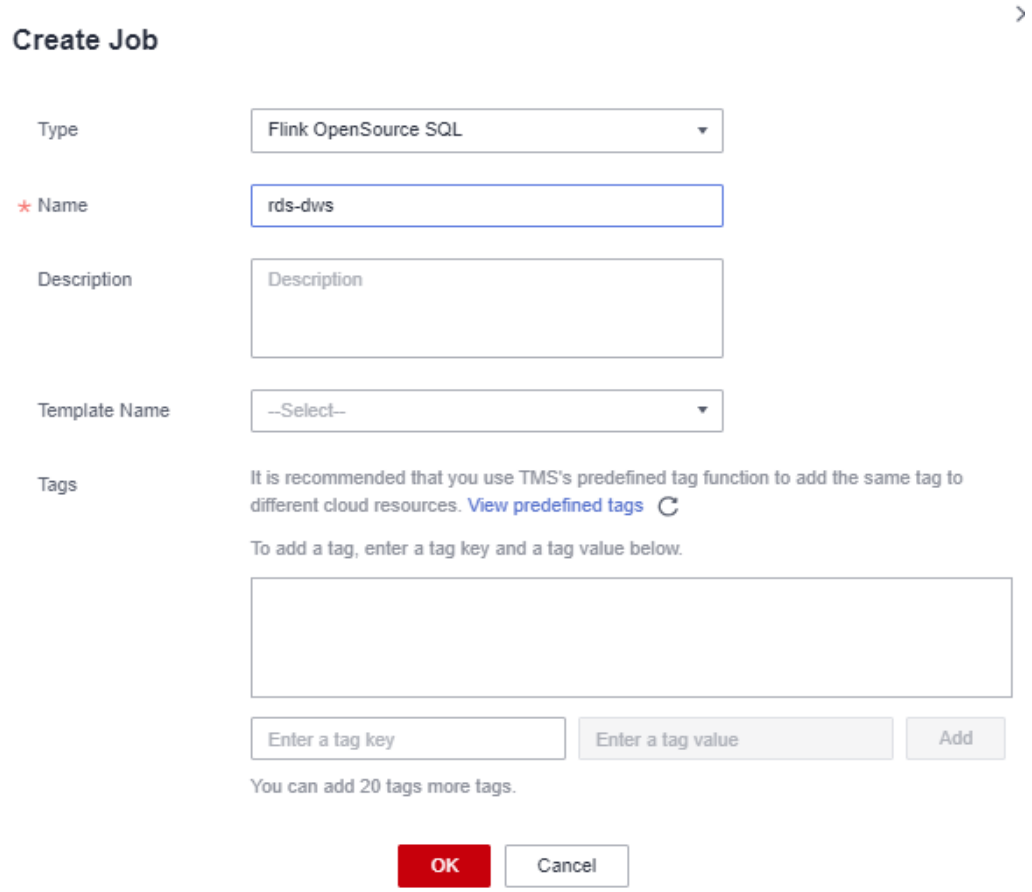
Set key parameters as follows and retain the default values for other parameters.

- **Region**: **CN-Hong Kong**
- **Bucket Name**: **dli-obs01** (If a conflict occurs, the bucket name can be increased from 02 to 03.)
- **Bucket Policy**: **Private**

Step 2 Return to the DLI console, choose **Job Management** > **Flink Jobs** on the left, and click **Create Job** in the upper right corner.

Step 3 Set **Type** to **Flink OpenSource SQL** and **Name** to **rds-dws**.

Figure 2-43 Creating a job



Step 4 Click **OK**. The page for editing the job is displayed.

Step 5 Set the following key parameters on the right of the page. Retain the default values for other parameters that are not described.

- **Queue**: Select **dli_dws** obtained in **4**.
- **Flink Version**: Select version 1.15 or later. (The actual version is subject to the GUI.)
- **OBS Bucket**: Select the bucket created in **Step 1** and click **Authorize**.
- (Optional) Select **Save Job Log**.

Step 6 Copy the following SQL code to the SQL code window on the left.

For how to obtain the internal IP address of the RDS database, see **Step 1.3**. For details about how to obtain the internal IP address of the GaussDB(DWS) cluster, see **Step 6.2**. Change the password of user **root** of the RDS database and the password of user **dbadmin** of GaussDB(DWS).

```
CREATE TABLE
  mys_order (
    order_id STRING,
    order_channel STRING,
    order_time TIMESTAMP,
    cust_code STRING,
    pay_amount DOUBLE,
    real_pay DOUBLE,
    PRIMARY KEY (order_id) NOT ENFORCED
  )
```



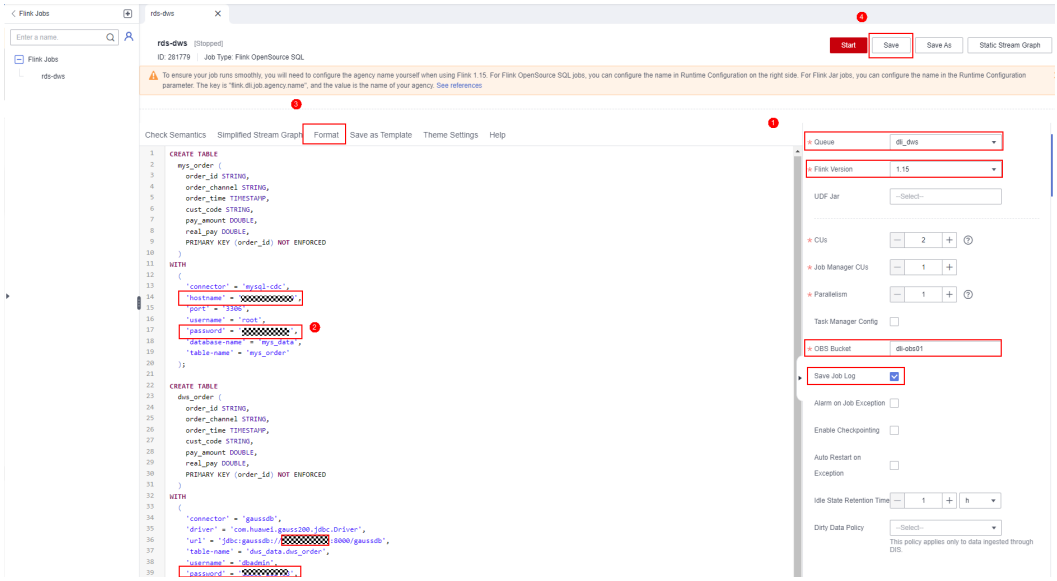
```
WITH
  (
    'connector' = 'mysql-cdc',
    'hostname' = 'Private IP address of the RDS DB instance',
    'port' = '3306',
    'username' = 'root',
    'password' = 'Password of user root of the RDS DB instance',
    'database-name' = 'mys_data',
    'table-name' = 'mys_order'
  );

CREATE TABLE
  dws_order (
    order_id STRING,
    order_channel STRING,
    order_time TIMESTAMP,
    cust_code STRING,
    pay_amount DOUBLE,
    real_pay DOUBLE,
    PRIMARY KEY (order_id) NOT ENFORCED
  )
WITH
  (
    'connector' = 'gaussdb',
    'driver' = 'com.huawei.gauss200.jdbc.Driver',
    'url' = 'jdbc:gaussdb://GaussDB(DWS) cluster private IP address:8000/gaussdb',
    'table-name' = 'dws_data.dws_order',
    'username' = 'dbadmin',
    'password' = 'Password of GaussDB(DWS) user dbadmin',
    'write.mode' = 'insert'
  );

INSERT INTO
  dws_order
SELECT
  *
FROM
  mys_order;
```

Step 7 Click **Format** and click **Save**.

NOTICE

Click **Format** to format the SQL code. Otherwise, new null characters may be introduced during code copy and paste, causing job execution failures.

Figure 2-44 Flink job parameters

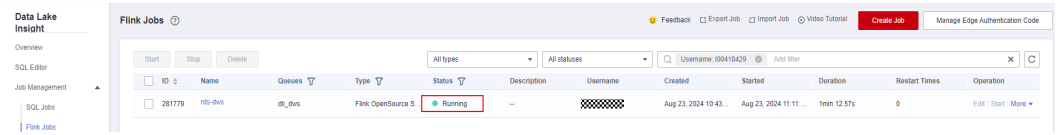


Step 8 Return to the DLI console home page and choose **Job Management** > **Flink Jobs** on the left.

Step 9 Click **Start** on the right of the job name **rds-dws** and click **Start Now**.

Wait for about 1 minute and refresh the page. If the status is **Running**, the job is executed.

Figure 2-45 Running succeeded



----End

Step 6: Verifying Data Synchronization

Step 1 Go back to the SQL window of the GaussDB(DWS) database. If the connection times out, perform the following operations to log in again:

1. Go to the GaussDB(DWS) console.
2. In the navigation pane on the left, choose **Dedicated Clusters** > **Clusters**, and click **Log In** on the right of **dws-demo**.

Step 2 Check whether two rows of data in the MySQL table have been synchronized to GaussDB(DWS).

```
SELECT * FROM dws_data.dws_order;
```

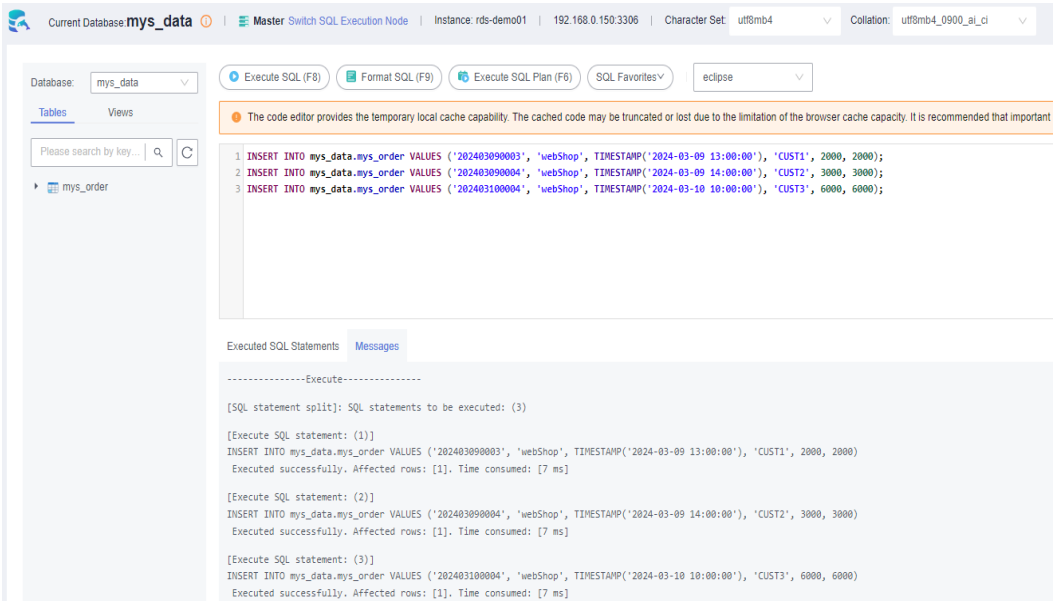
Figure 2-46 Query result

order_id	order_channel	order_time	cust_code	pay_amount	real_pay
202306270001	webShop	2023-06-27 10:00:00	CUST1	1000	1000
202306270002	webShop	2023-06-27 11:00:00	CUST2	5000	5000

Step 3 Switch to the RDS for MySQL page and run the following statements to insert three new data records:

```
INSERT INTO mys_data.mys_order VALUES ('202403090003', 'webShop', TIMESTAMP('2024-03-09 13:00:00'), 'CUST1', 2000, 2000);
INSERT INTO mys_data.mys_order VALUES ('202403090004', 'webShop', TIMESTAMP('2024-03-09 14:00:00'), 'CUST2', 3000, 3000);
INSERT INTO mys_data.mys_order VALUES ('202403100004', 'webShop', TIMESTAMP('2024-03-10 10:00:00'), 'CUST3', 6000, 6000);
```

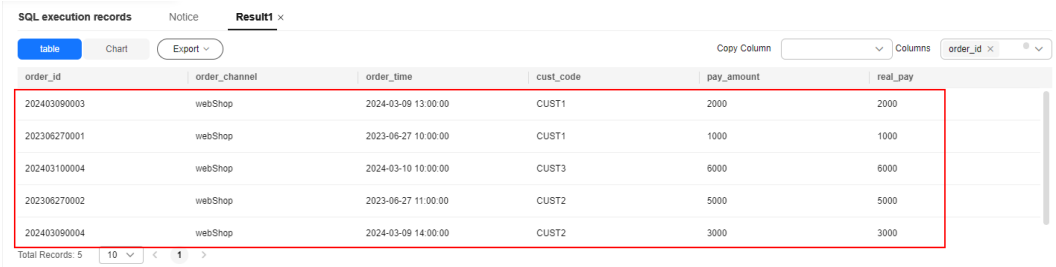
Figure 2-47 New MySQL data



Step 4 Go back to the SQL window of GaussDB(DWS) and run the following SQL statement again. The returned result shows that the MySQL data has been synchronized to GaussDB(DWS) in real time.

```
SELECT * FROM dws_data.dws_order;
```

Figure 2-48 Real-time data synchronization



----End

More Information

Storing authentication information for a data source directly in the job script for Flink cross-source development can result in password exposure. To enhance security, use DLI's datasource authentication function instead of specifying MySQL and GaussDB(DWS) usernames and passwords directly in job scripts.

> **NOTE**
>
> Currently, only Flink 1.12 supports this function. Pay attention to the document changes on the official website.

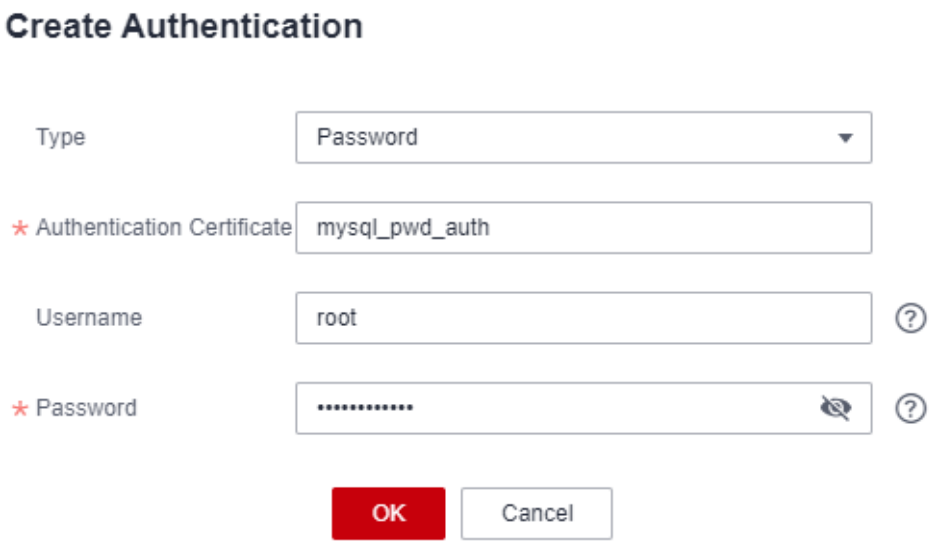
Step 1 Log in to the DLI console, click **Datasource Connections**, and click **Datasource Authentication**.

Step 2 Click **Create**.

Step 3 Create the password authentication for the **root** user of the MySQL database.

1. Set the following parameters:
   - **Type**: **Password**
   - **Authentication Certificate**: **mysql_pwd_auth**
   - **Username**: **root**
   - **Password**: password of user **root**

   Figure 2-49 MySQL password authentication

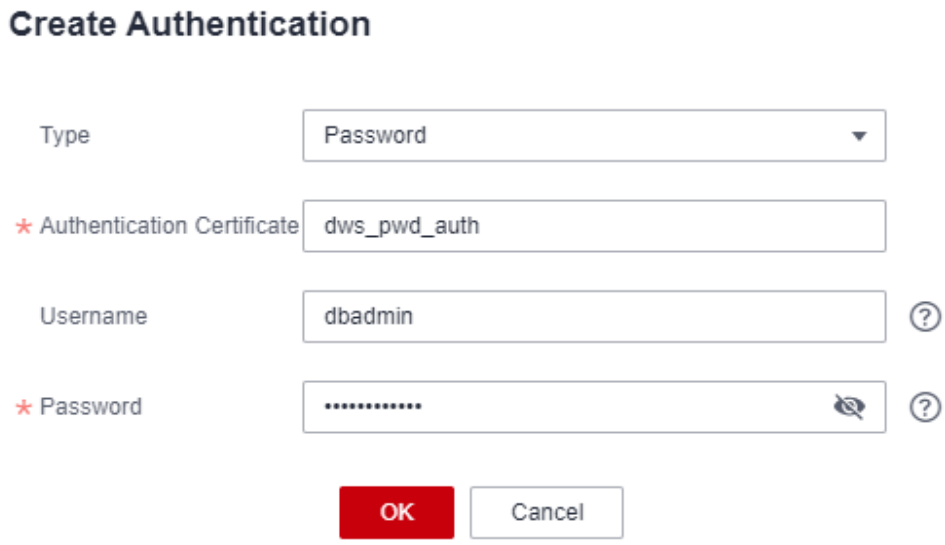


2. Click **OK**.

Step 4 Create password authentication for the **dbadmin** user of GaussDB(DWS).

1. Set the following parameters:
   - **Type**: **Password**
   - **Authentication Certificate**: **dws_pwd_auth**
   - **Username**: **dbadmin**
   - **Password**: password of user **dbadmin**

Figure 2-50 GaussDB(DWS) password authentication



2. Click **OK**.

Step 5 On the DLI console, choose **Job Management** > **Flink Jobs**. Locate the row that contains the job created in **Step 5: Creating a DLI Flink Job**, and choose **More** > **Stop** to stop the job.

Step 6 After the job is stopped, you can edit the job name.

Step 7 Replace the SQL script with the latest one.

Replace the private IP addresses of RDS and GaussDB(DWS).

```
CREATE TABLE mys_order (
  order_id STRING,
  order_channel STRING,
  order_time TIMESTAMP,
  cust_code STRING,
  pay_amount DOUBLE,
  real_pay DOUBLE,
  PRIMARY KEY (order_id) NOT ENFORCED )
WITH (
  'connector' = 'mysql-cdc',
  'hostname' = 'Private IP address of RDS',
  'port' = '3306',
  'pwd_auth_name' = 'mysql_pwd_auth',
  'database-name' = 'mys_data',
  'table-name' = 'mys_order' );

CREATE TABLE dws_order (
    order_id STRING,
    order_channel STRING,
    order_time TIMESTAMP,
    cust_code STRING,
    pay_amount DOUBLE,
    real_pay DOUBLE,
    PRIMARY KEY (order_id) NOT ENFORCED )
WITH (
    'connector' = 'gaussdb',
    'driver' = 'com.huawei.gauss200.jdbc.Driver',
    'url' = 'jdbc:gaussdb://GaussDB(DWS) private IP address:8000/gaussdb',
    'table-name' = 'dws_data.dws_order',
    'pwd_auth_name' = 'dws_pwd_auth',
    'write.mode' = 'insert' );

INSERT INTO dws_order SELECT * FROM mys_order;
```


Step 8 Click **Format** and click **Save**.

Step 9 Restart the job and verify data synchronization by referring to **Step 6: Verifying Data Synchronization**.

----End

2.6 Migrating Data Between GaussDB(DWS) Clusters Using GDS

This practice demonstrates how to migrate 15 million rows of data between two GaussDB(DWS) clusters within minutes based on the high concurrency of GDS import and export.

NOTE

- This function is supported only by clusters of version 8.1.2 or later.
- GDS is a high-concurrency import and export tool developed by GaussDB(DWS). For more information, visit **GDS Usage Guide**.
- This section describes only the operation practice. For details about GDS interconnection and syntax description, see **GDS-based Cross-Cluster Interconnection**.

This practice takes about 90 minutes. The cloud services used in this practice are GaussDB(DWS), Elastic Cloud Server (ECS), and Virtual Private Cloud (VPC). The basic process is as follows:

1. **Prerequisites**
2. **Step 1: Creating Two GaussDB(DWS) Clusters**
3. **Step 2: Preparing Source Data**
4. **Step 3: Installing and Starting the GDS Server**
5. **Step 4: Implementing Data Interconnection Across GaussDB(DWS) Clusters**

Supported Regions

Table 2-26 describes the regions where OBS data has been uploaded.

Table 2-26 Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1
CN North-Beijing2	dws-demo-cn-north-2
CN North-Beijing4	dws-demo-cn-north-4
CN North-Ulanqab1	dws-demo-cn-north-9
CN East-Shanghai1	dws-demo-cn-east-3
CN East-Shanghai2	dws-demo-cn-east-2

Region	OBS Bucket
CN South-Guangzhou	dws-demo-cn-south-1
CN South-Guangzhou-InvitationOnly	dws-demo-cn-south-4
CN-Hong Kong	dws-demo-ap-southeast-1
AP-Singapore	dws-demo-ap-southeast-3
AP-Bangkok	dws-demo-ap-southeast-2
LA-Santiago	dws-demo-la-south-2
AF-Johannesburg	dws-demo-af-south-1
LA-Mexico City1	dws-demo-na-mexico-1
LA-Mexico City2	dws-demo-la-north-2
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

Constraints

In this practice, two sets of GaussDB(DWS) and ECS services are deployed in the same region and VPC to ensure network connectivity.

Prerequisites

- You have obtained the AK and SK of the account.
- You have created a VPC and subnet. For details, see **Creating a VPC**.

Step 1: Creating Two GaussDB(DWS) Clusters

Create two GaussDB(DWS) clusters. For details, see **Creating a Cluster**. You are advised to create the clusters in the CN-Hong Kong region. Name the two clusters **dws-demo01** and **dws-demo02**.

Step 2: Preparing Source Data

Step 1 On the cluster management page of the GaussDB(DWS) console, locate the row that contains the **dws-demo01** cluster and click **Login** in the **Operation** column.

NOTE

This practice uses version 8.1.3.x as an example. 8.1.2 and earlier versions do not support this login mode. You can use Data Studio to connect to a cluster. For details, see **Using Data Studio to Connect to a Cluster**.

Step 2 After the login is successful, the SQL editor is displayed.

Step 3 Copy the following SQL statements to the SQL window and click **Execute SQL** to create the test TPC-H table **ORDERS**.


```
CREATE TABLE ORDERS
 (
 O_ORDERKEY BIGINT NOT NULL ,
 O_CUSTKEY BIGINT NOT NULL ,
 O_ORDERSTATUS CHAR(1) NOT NULL ,
 O_TOTALPRICE DECIMAL(15,2) NOT NULL ,
 O_ORDERDATE DATE NOT NULL ,
 O_ORDERPRIORITY CHAR(15) NOT NULL ,
 O_CLERK CHAR(15) NOT NULL ,
 O_SHIPPRIORITY BIGINT NOT NULL ,
 O_COMMENT VARCHAR(79) NOT NULL)
 with (orientation = column)
 distribute by hash(O_ORDERKEY)
 PARTITION BY RANGE(O_ORDERDATE)
 (
 PARTITION O_ORDERDATE_1 VALUES LESS THAN('1993-01-01 00:00:00'),
 PARTITION O_ORDERDATE_2 VALUES LESS THAN('1994-01-01 00:00:00'),
 PARTITION O_ORDERDATE_3 VALUES LESS THAN('1995-01-01 00:00:00'),
 PARTITION O_ORDERDATE_4 VALUES LESS THAN('1996-01-01 00:00:00'),
 PARTITION O_ORDERDATE_5 VALUES LESS THAN('1997-01-01 00:00:00'),
 PARTITION O_ORDERDATE_6 VALUES LESS THAN('1998-01-01 00:00:00'),
 PARTITION O_ORDERDATE_7 VALUES LESS THAN('1999-01-01 00:00:00')
 );
```

Step 4 Run the SQL statements below to create an OBS foreign table.

Replace AK and SK with the actual AK and SK of the account. <obs_bucket_name> is obtained from **Supported Regions**.

NOTE

Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE ORDERS01
 (
LIKE orders
 )
 SERVER gsmpp_server
 OPTIONS (
 ENCODING 'utf8',
 LOCATION 'obs://<obs_bucket_name>/tpch/orders.tbl',
 FORMAT 'text',
 DELIMITER '|',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
 CHUNKSIZE '64',
 IGNORE_EXTRA_DATA 'on'
 );
```

Step 5 Run the SQL statement below to import data from the OBS foreign table to the source GaussDB(DWS) cluster. The import takes about 2 minutes.

NOTE

If an import error occurs, the AK and SK values of the foreign table are incorrect. In this case, run **DROP FOREIGN TABLE order01** to delete the foreign table, create a foreign table again, and run the following statement to import data again.

```
INSERT INTO orders SELECT * FROM orders01;
```

Step 6 Repeat the preceding steps to log in to the destination cluster **dws-demo02** and run the following SQL statements to create the target table **orders**.

```
CREATE TABLE ORDERS
 (
 O_ORDERKEY BIGINT NOT NULL ,
 O_CUSTKEY BIGINT NOT NULL ,
 O_ORDERSTATUS CHAR(1) NOT NULL ,
```



```
O_TOTALPRICE DECIMAL(15,2) NOT NULL ,
O_ORDERDATE DATE NOT NULL ,
O_ORDERPRIORITY CHAR(15) NOT NULL ,
O_CLERK CHAR(15) NOT NULL ,
O_SHIPPRIORITY BIGINT NOT NULL ,
O_COMMENT VARCHAR(79) NOT NULL)
with (orientation = column)
distribute by hash(O_ORDERKEY)
PARTITION BY RANGE(O_ORDERDATE)
(
PARTITION O_ORDERDATE_1 VALUES LESS THAN('1993-01-01 00:00:00'),
PARTITION O_ORDERDATE_2 VALUES LESS THAN('1994-01-01 00:00:00'),
PARTITION O_ORDERDATE_3 VALUES LESS THAN('1995-01-01 00:00:00'),
PARTITION O_ORDERDATE_4 VALUES LESS THAN('1996-01-01 00:00:00'),
PARTITION O_ORDERDATE_5 VALUES LESS THAN('1997-01-01 00:00:00'),
PARTITION O_ORDERDATE_6 VALUES LESS THAN('1998-01-01 00:00:00'),
PARTITION O_ORDERDATE_7 VALUES LESS THAN('1999-01-01 00:00:00')
);
```

----End

Step 3: Installing and Starting the GDS Server

Step 1 Create an ECS by referring to **Purchasing an ECS**. Note that the ECS and GaussDB(DWS) instances must be created in the same region and VPC. In this example, the CentOS 7.6 version is selected as the ECS image.

Step 2 Download the GDS package.

1. Log in to the GaussDB(DWS) console.
2. In the navigation tree on the left, choose **Management** > **Client Connections**.
3. Select the GDS client of the target version from the drop-down list of **CLI Client**.

   Select a version based on the cluster version and the OS where the client is installed.
4. Click **Download**.

Step 3 Use the SFTP tool to upload the downloaded client (for example, **dws_client_8.2.x_redhat_x64.zip**) to the **/opt directory** of the ECS.

Step 4 Log in to the ECS as the **root** user and run the following commands to go to the **/opt** directory and decompress the client package.

```
cd /opt
unzip dws_client_8.2.x_redhat_x64.zip
```

Step 5 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

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

Step 6 Change the owner of the GDS package directory and source data file directory to the GDS user.

```
chown -R gds_user:gdsgrp /opt/gds/bin
chown -R gds_user:gdsgrp /opt
```

Step 7 Switch to user **gds**.

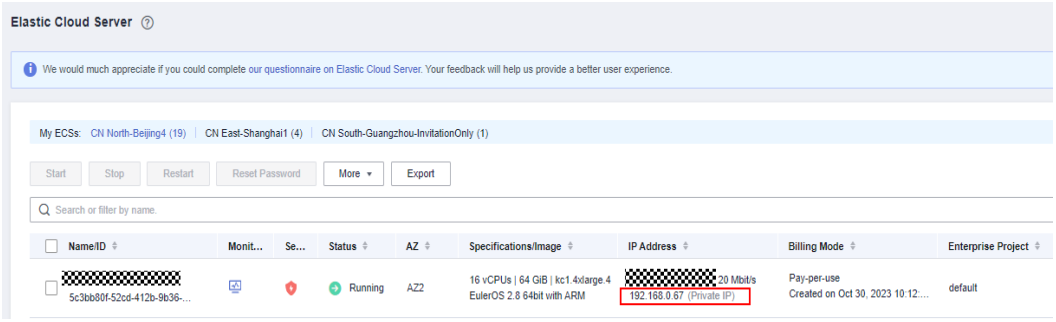
```
su - gds_user
```

Step 8 Run the following commands to go to the **gds** directory and execute environment variables.


```
cd /opt/gds/bin
source gds_env
```

Step 9 Run the following command to start GDS. You can view the private IP address of the ECS on the ECS console.

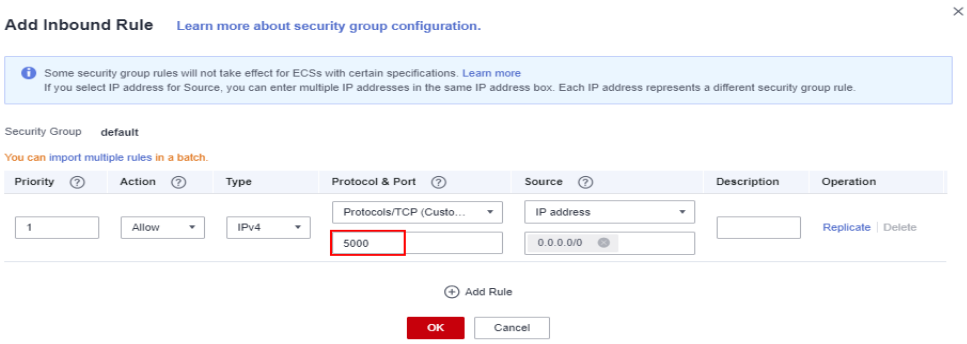
/opt/gds/bin/gds -d /opt -p *Private IP address of the ECS*:**5000 -H 0.0.0.0/0 -l /opt/gds/bin/gds_log.txt -D -t 2**



Step 10 Enable the network port between the ECS and GaussDB(DWS).

The GDS server (ECS in this practice) needs to communicate with GaussDB(DWS). The default security group of the ECS does not allow inbound traffic from GDS port 5000 and GaussDB(DWS) port 8000. Perform the following steps:

1. Return to the ECS console and click the ECS name to go to the ECS details page.
2. Click the **Security Groups** tab and click **Manage Rule**.
3. Choose **Inbound Rules** and click **Add Rule**. Set **Priority** to **1**, set **Protocol & Port** to **5000**, and click **OK**.



4. Repeat the preceding steps to add an inbound rule of 8000.

----End

Step 4: Implementing Data Interconnection Across GaussDB(DWS) Clusters

Step 1 Create a server.

1. Obtain the private IP address of the source GaussDB(DWS) cluster. Specifically, go to the GaussDB(DWS) console, choose **Dedicated Clusters** > **Clusters**, and click the source cluster name **dws-demo01**.
2. Go to the cluster details page and record the private network IP address.

3. Switch back to the GaussDB(DWS) console and click **Log In** in the **Operation** column of the destination cluster **dws-demo02**. The SQL window is displayed.

   Run the commands below to create a server.

   In the commands, *Private network IP address of the source GaussDB(DWS) cluster* is obtained in the previous step, *Private IP address of the ECS* is obtained from the ECS console, and *Login password of user dbadmin* is set when the GaussDB(DWS) cluster is created.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
 (
 address 'Private network IP address of the source GaussDB(DWS) cluster:8000',
 dbname 'gaussdb',
 username 'dbadmin',
 password 'Login password of user dbadmin',
 syncsrv 'gsfs://Private IP address of the ECS:5000'
 )
 ;
```

Step 2 Create a foreign table for interconnection.

In the SQL window of the destination cluster **dws-demo02**, run the following statements to create a foreign table for interconnection:

```
CREATE FOREIGN TABLE ft_orders
 (
 O_ORDERKEY BIGINT ,
 O_CUSTKEY BIGINT ,
 O_ORDERSTATUS CHAR(1) ,
 O_TOTALPRICE DECIMAL(15,2) ,
 O_ORDERDATE DATE ,
 O_ORDERPRIORITY CHAR(15) ,
 O_CLERK CHAR(15) ,
 O_SHIPPRIORITY BIGINT ,
 O_COMMENT VARCHAR(79)

 )
 SERVER server_remote
 OPTIONS
 (
 schema_name 'public',
 table_name 'orders',
 encoding 'SQL_ASCII'
 );
```


Step 3 Import all table data.

In the SQL window, run the SQL statement below to import full data from the **ft_orders** foreign table: Wait for about 1 minute.

```
INSERT INTO orders SELECT * FROM ft_orders;
```

Run the following SQL statement to verify that 15 million rows of data are successfully imported.

```
SELECT count(*) FROM orders;
```

Step 4 Import data based on filter criteria.

```
INSERT INTO orders SELECT * FROM ft_orders WHERE o_orderkey < '10000000';
```

----End

3 Data Analytics

3.1 Using GaussDB(DWS) to Query Vehicle Routes at Traffic Checkpoints in Seconds

This practice shows you how to analyze passing vehicles at checkpoints. In this practice, 890 million data records from checkpoints are loaded to a single database table on GaussDB(DWS) for accurate and fuzzy query, demonstrating the ability of GaussDB(DWS) to perform high-performance query for historical data.

NOTE

The sample data has been uploaded to the **traffic-data** folder in an OBS bucket, and all Huawei Cloud accounts have been granted the read-only permission for accessing the OBS bucket.

General Procedure

This practice takes about 40 minutes. The basic process is as follows:

1. **Making Preparations**
2. **Step 1: Creating a Cluster**
3. **Step 2: Using Data Studio to Connect to a Cluster**
4. **Step 3: Importing Sample Data**
5. **Step 4: Performing Vehicle Analysis**

Supported Regions

Table 3-1 describes the regions where OBS data has been uploaded.

Table 3-1 Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1

Region	OBS Bucket
CN North-Beijing2	dws-demo-cn-north-2
CN North-Beijing4	dws-demo-cn-north-4
CN North-Ulanqab1	dws-demo-cn-north-9
CN East-Shanghai1	dws-demo-cn-east-3
CN East-Shanghai2	dws-demo-cn-east-2
CN South-Guangzhou	dws-demo-cn-south-1
CN South-Guangzhou-InvitationOnly	dws-demo-cn-south-4
CN-Hong Kong	dws-demo-ap-southeast-1
AP-Singapore	dws-demo-ap-southeast-3
AP-Bangkok	dws-demo-ap-southeast-2
LA-Santiago	dws-demo-la-south-2
AF-Johannesburg	dws-demo-af-south-1
LA-Mexico City1	dws-demo-na-mexico-1
LA-Mexico City2	dws-demo-la-north-2
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

Making Preparations

- You have registered a GaussDB(DWS) account and checked the account status before using GaussDB(DWS). The account cannot be in arrears or frozen.
- You have obtained the AK and SK of the account.

Step 1: Creating a Cluster

Step 1 Log in to the management console.

Step 2 Click **Service List** and choose **Analytics** > **GaussDB(DWS)**.

Step 3 Choose **Dedicated Clusters** > **Clusters**. On the displayed page, click **Create GaussDB(DWS) Cluster** in the upper right corner.

Step 4 Configure the parameters according to **Table 3-2**.

Table 3-2 Basic configurations

Parameter	Configuration
Region	Select **CN North-Beijing4** or **CN-Hong KongEU-Dublin**. **NOTE** **CN-Hong Kong** is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region.
AZ	AZ2
Resource	Standard Warehouse
Compute Resource	ECS
Storage type	Cloud SSD
CPU Architecture	X86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPUs | 128 GB | 2000 GB SSD) **NOTE** If this flavor is sold out, select other AZs or flavors.
Hot Storage	100 GB/node
Nodes	3

Step 5 Verify that the information is correct and click **Next: Configure Network**. Configure the network by referring to **Table 3-3**.

Table 3-3 Configuring the network

Parameter	Configuration
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1Mbit/s
ELB	Do not use

Step 6 Click **Next: Configure Advanced Settings** to access advanced configurations. **Table 3-4** lists the required parameters.

Table 3-4 Configuring advanced settings

Parameter	Configuration
Cluster Name	dws-demo
Cluster Version	Use the recommended version.
Administrator Account	dbadmin
Administrator Password	N/A
Confirm Password	N/A
Database Port	8000
Enterprise Project	Default
Advanced Settings	Default

Step 7 Click **Next: Confirm**, confirm the settings, and click **Buy Now**.

Step 8 Wait about 6 minutes. After the cluster is created, click next to the cluster name. On the displayed cluster information page, record the value of **Public Network Address**, for example, **dws-demov.dws.huaweicloud.com**.

Figure 3-1 Cluster information

----End

Step 2: Using Data Studio to Connect to a Cluster

Step 1 Ensure that JDK 1.8.0 or later has been installed on the client host. Choose **PC > Properties > Advanced System Settings > Environment Variables** and set

JAVA_HOME (for example, **C:\Program Files\Java\jdk1.8.0_191**).
Add **;%JAVA_HOME%\bin** to the variable **path**.

Step 2 On the GaussDB(DWS) console, choose **Management** > **Client Connections** and download the Data Studio client.

Step 3 Decompress the downloaded Data Studio software package, go to the decompressed directory, and double-click **Data Studio.exe** to start the client.

Step 4 On the Data Studio main menu, choose **File > New Connection**. In the dialog box that is displayed, configure the connection based on **Table 3-5**.

Table 3-5 Data Studio software configuration

Parameter	Configuration
Database Type	GaussDB(DWS)
Connection Name	dws-demo
Host	dws-demov.dws.huaweicloud.com The value of this parameter must be the same as the value of **Public Network Address** queried in **Step 1: Creating a Cluster**.
Host Port	8000
Database Name	gaussdb
User Name	dbadmin
Password	N/A
Enable SSL	Disable

Step 5 Click **OK**.

----End

Step 3: Importing Sample Data

After connecting to the cluster using the SQL client tool, perform the following operations on the SQL client tool to import the sample data from traffic checkpoints and perform data queries.

Step 1 Create a database **traffic**.

```
CREATE DATABASE traffic encoding 'utf8' template template0;
```

Step 2 Perform the following steps to switch to the new database:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and choose **Refresh** from the shortcut menu. Then, the new database is displayed.
2. Right-click the name of the new database **traffic** and choose **Connect to DB** from the shortcut menu.

3. Right-click the name of the new database **traffic** and choose **Open Terminal** from the shortcut menu. The SQL command window for connecting to the specified database is displayed. Perform the following steps in the window.

Step 3 Execute the following statements to create a database table for storing vehicle information from traffic checkpoints:

```
CREATE SCHEMA traffic_data;
SET current_schema= traffic_data;
DROP TABLE if exists GCJL;
CREATE TABLE GCJL
(
    kkbh   VARCHAR(20),
    hphm   VARCHAR(20),
    gcsj   DATE ,
    cplx   VARCHAR(8),
    cllx   VARCHAR(8),
    csys   VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by hash(hphm);
```

Step 4 Create a foreign table, which is used to identify and associate the source data on OBS.

NOTICE

- *<obs_bucket_name>* indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see **Supported Regions**. GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- , and replace *<Access_Key_Id>* and *<Secret_Access_Key>* with the value obtained in **Making Preparations**.
- Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
- If the message "ERROR: schema "*xxx*" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
CREATE SCHEMA tpchobs;
SET current_schema = 'tpchobs';
DROP FOREIGN table if exists GCJL_OBS;
CREATE FOREIGN TABLE GCJL_OBS
(
    like traffic_data.GCJL
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/traffic-data/gcxx',
    format 'text',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
```

Step 5 Import data from a foreign table to a database table.

```
INSERT INTO traffic_data.GCJL SELECT * FROM tpchobs.GCJL_OBS;
```


It takes some time to import data.

----End

Step 4: Performing Vehicle Analysis

1. **Execute ANALYZE.**

   This statement collects statistics related to ordinary tables in databases. The statistics are saved to the system catalog **PG_STATISTIC**. When you run the planner, the statistics help you develop an efficient query execution plan.

   Execute the following statement to generate the table statistics:

   ```
   ANALYZE;
   ```

2. **Querying the data volume of the data table**

   Execute the following statement to query the number of loaded data records:

   ```
   SET current_schema= traffic_data;
   SELECT count(*) FROM traffic_data.gcjl;
   ```

3. **Accurate vehicle query**

   Run the following statements to query the driving route of a vehicle by the license plate number and time segment. GaussDB(DWS) responds to the request in seconds.

   ```
   SET current_schema= traffic_data;
   SELECT hphm, kkbh, gcsj
   FROM traffic_data.gcjl
   where hphm =  'YD38641'
   and gcsj between '2016-01-06' and '2016-01-07'
   order by gcsj desc;
   ```

4. **Fuzzy vehicle query**

   Run the following statements to query the driving route of a vehicle by the license plate number and time segment. GaussDB(DWS) responds to the request in seconds.

   ```
   SET current_schema= traffic_data;
   SELECT hphm, kkbh, gcsj
   FROM traffic_data.gcjl
   where hphm like  'YA23F%'
   and kkbh in('508', '1125', '2120')
   and gcsj between '2016-01-01' and '2016-01-07'
   order by hphm,gcsj desc;
   ```

3.2 Using GaussDB(DWS) to Analyze the Supply Chain Requirements of a Company

This practice describes how to load the sample data set from OBS to a data warehouse cluster and perform data queries. This example comprises multi-table analysis and theme analysis in the data analysis scenario.

NOTE

In this example, a standard TPC-H-1x data set of 1 GB size has been generated on GaussDB(DWS), and has been uploaded to the **tpch** folder of an OBS bucket. All HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket. Users can easily import the data set using their accounts.

General Procedure

This practice takes about 60 minutes. The process is as follows:

1. **Making Preparations**
2. **Step 1: Importing Sample Data**
3. **Step 2: Performing Multi-Table Analysis and Theme Analysis**

Supported Regions

Table 3-6 describes the regions where OBS data has been uploaded.

Table 3-6 Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1
CN North-Beijing2	dws-demo-cn-north-2
CN North-Beijing4	dws-demo-cn-north-4
CN North-Ulanqab1	dws-demo-cn-north-9
CN East-Shanghai1	dws-demo-cn-east-3
CN East-Shanghai2	dws-demo-cn-east-2
CN South-Guangzhou	dws-demo-cn-south-1
CN South-Guangzhou-InvitationOnly	dws-demo-cn-south-4
CN-Hong Kong	dws-demo-ap-southeast-1
AP-Singapore	dws-demo-ap-southeast-3
AP-Bangkok	dws-demo-ap-southeast-2
LA-Santiago	dws-demo-la-south-2
AF-Johannesburg	dws-demo-af-south-1
LA-Mexico City1	dws-demo-na-mexico-1
LA-Mexico City2	dws-demo-la-north-2
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

Scenario Description

Understand the basic functions of GaussDB(DWS) and how to import data. Analyze the order data of a company and its suppliers as follows:

1. Analyze the revenue brought by suppliers in a region to the company. The statistics can be used to determine whether a local allocation center needs to be established in a given region.
2. Analyze the relationship between parts and suppliers to obtain the number of suppliers for parts based on the specified contribution conditions. The information can be used to determine whether suppliers are sufficient for large order quantities when the task is urgent.
3. Analyze the revenue loss of small orders. You can query the average annual revenue loss if there are no small orders. Filter out small orders that are lower than 20% of the average supply volume, and calculate the total amount of those small orders to figure out the average annual revenue loss.

Making Preparations

- You have registered a GaussDB(DWS) account and checked the account status before using GaussDB(DWS). The account cannot be in arrears or frozen.
- You have obtained the AK and SK of the account.
- A cluster has been created and connected using Data Studio. For details, see **Using GaussDB(DWS) to Query Vehicle Routes at Traffic Checkpoints in Seconds**.

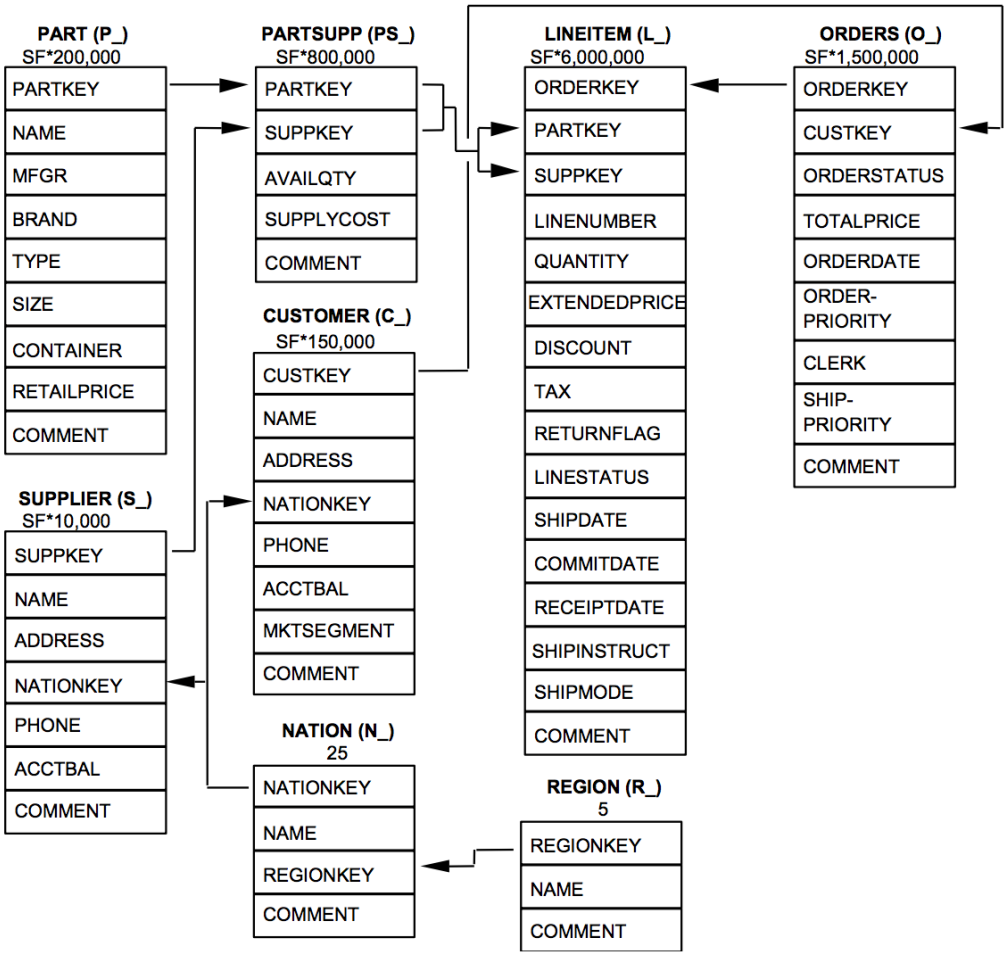
Step 1: Importing Sample Data

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the TPC-H sample data and perform data queries.

Step 1 Create a database table.

The TPC-H sample data consists of eight database tables whose associations are shown in **Figure 3-2**.

Figure 3-2 TPC-H data tables



Execute the following statements to create tables in the **gaussdb** database.

```
CREATE SCHEMA tpch;
SET current_schema = tpch;

DROP TABLE if exists region;
CREATE TABLE REGION
(
        R_REGIONKEY  INT NOT NULL ,
        R_NAME       CHAR(25) NOT NULL ,
        R_COMMENT    VARCHAR(152)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by replication;

DROP TABLE if exists nation;
CREATE TABLE NATION
(
        N_NATIONKEY  INT NOT NULL,
        N_NAME       CHAR(25) NOT NULL,
        N_REGIONKEY  INT NOT NULL,
        N_COMMENT    VARCHAR(152)
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by replication;

DROP TABLE if exists supplier;
CREATE TABLE SUPPLIER
```



```
(
        S_SUPPKEY     BIGINT NOT NULL,
        S_NAME        CHAR(25) NOT NULL,
        S_ADDRESS     VARCHAR(40) NOT NULL,
        S_NATIONKEY   INT NOT NULL,
        S_PHONE       CHAR(15) NOT NULL,
        S_ACCTBAL     DECIMAL(15,2) NOT NULL,
        S_COMMENT     VARCHAR(101) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(S_SUPPKEY);

DROP TABLE if exists customer;
CREATE TABLE CUSTOMER
(
        C_CUSTKEY     BIGINT NOT NULL,
        C_NAME        VARCHAR(25) NOT NULL,
        C_ADDRESS     VARCHAR(40) NOT NULL,
        C_NATIONKEY   INT NOT NULL,
        C_PHONE       CHAR(15) NOT NULL,
        C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
        C_MKTSEGMENT  CHAR(10) NOT NULL,
        C_COMMENT     VARCHAR(117) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(C_CUSTKEY);

DROP TABLE if exists part;
CREATE TABLE PART
(
        P_PARTKEY     BIGINT NOT NULL,
        P_NAME        VARCHAR(55) NOT NULL,
        P_MFGR        CHAR(25) NOT NULL,
        P_BRAND       CHAR(10) NOT NULL,
        P_TYPE        VARCHAR(25) NOT NULL,
        P_SIZE        BIGINT NOT NULL,
        P_CONTAINER   CHAR(10) NOT NULL,
        P_RETAILPRICE DECIMAL(15,2) NOT NULL,
        P_COMMENT     VARCHAR(23) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(P_PARTKEY);

DROP TABLE if exists partsupp;
CREATE TABLE PARTSUPP
(
        PS_PARTKEY     BIGINT NOT NULL,
        PS_SUPPKEY     BIGINT NOT NULL,
        PS_AVAILQTY    BIGINT NOT NULL,
        PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
        PS_COMMENT     VARCHAR(199) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(PS_PARTKEY);

DROP TABLE if exists orders;
CREATE TABLE ORDERS
(
        O_ORDERKEY       BIGINT NOT NULL,
        O_CUSTKEY        BIGINT NOT NULL,
        O_ORDERSTATUS    CHAR(1) NOT NULL,
        O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
        O_ORDERDATE      DATE NOT NULL ,
        O_ORDERPRIORITY  CHAR(15) NOT NULL,
        O_CLERK          CHAR(15) NOT NULL ,
        O_SHIPPRIORITY   BIGINT NOT NULL,
        O_COMMENT        VARCHAR(79) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
```



```
distribute by hash(O_ORDERKEY);

DROP TABLE if exists lineitem;
CREATE TABLE LINEITEM
(
        L_ORDERKEY    BIGINT NOT NULL,
        L_PARTKEY     BIGINT NOT NULL,
        L_SUPPKEY     BIGINT NOT NULL,
        L_LINENUMBER  BIGINT NOT NULL,
        L_QUANTITY    DECIMAL(15,2) NOT NULL,
        L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
        L_DISCOUNT    DECIMAL(15,2) NOT NULL,
        L_TAX         DECIMAL(15,2) NOT NULL,
        L_RETURNFLAG  CHAR(1) NOT NULL,
        L_LINESTATUS  CHAR(1) NOT NULL,
        L_SHIPDATE    DATE NOT NULL,
        L_COMMITDATE  DATE NOT NULL ,
        L_RECEIPTDATE DATE NOT NULL,
        L_SHIPINSTRUCT CHAR(25) NOT NULL,
        L_SHIPMODE     CHAR(10) NOT NULL,
        L_COMMENT      VARCHAR(44) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(L_ORDERKEY);
```

Step 2 Create a foreign table, which is used to identify and associate the source data on OBS.

NOTICE

- *<obs_bucket_name>* indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see **Supported Regions**. GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- , and replace *<Access_Key_Id>* and *<Secret_Access_Key>* with the value obtained in **Making Preparations**.
- Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
- If the message "ERROR: schema "*xxx*" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
CREATE SCHEMA tpchobs;
SET current_schema='tpchobs';
DROP FOREIGN table if exists region;
CREATE FOREIGN TABLE REGION
(
        like tpch.region
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/region.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists nation;
```



```
CREATE FOREIGN TABLE NATION
(
      like tpch.nation
)
SERVER gsmpp_server
OPTIONS (
       encoding 'utf8',
       location 'obs://<obs_bucket_name>/tpch/nation.tbl',
       format 'text',
       delimiter '|',
       access_key '<Access_Key_Id>',
       secret_access_key '<Secret_Access_Key>',
       chunksize '64',
       IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists supplier;
CREATE FOREIGN TABLE SUPPLIER
(
      like tpch.supplier
)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/supplier.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists customer;
CREATE FOREIGN TABLE CUSTOMER
(
      like tpch.customer
)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/customer.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists part;
CREATE FOREIGN TABLE PART
(
      like tpch.part

)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/part.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists partsupp;
CREATE FOREIGN TABLE PARTSUPP
(
```



```
    like tpch.partsupp
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/tpch/partsupp.tbl',
    format 'text',
    delimiter '|',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists orders;
CREATE FOREIGN TABLE ORDERS
(
    like tpch.orders
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/tpch/orders.tbl',
    format 'text',
    delimiter '|',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists lineitem;
CREATE FOREIGN TABLE LINEITEM
(
    like tpch.lineitem
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/tpch/lineitem.tbl',
    format 'text',
    delimiter '|',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
```

Step 3 Copy and execute the following statements to import the foreign table data to the corresponding database table.

Run the **insert** command to import the data in the OBS foreign table to the GaussDB(DWS) database table. The database kernel concurrently imports the OBS data at a high speed to GaussDB(DWS).

```
INSERT INTO tpch.lineitem SELECT * FROM tpchobs.lineitem;
INSERT INTO tpch.part SELECT * FROM tpchobs.part;
INSERT INTO tpch.partsupp SELECT * FROM tpchobs.partsupp;
INSERT INTO tpch.customer SELECT * FROM tpchobs.customer;
INSERT INTO tpch.supplier SELECT * FROM tpchobs.supplier;
INSERT INTO tpch.nation SELECT * FROM tpchobs.nation;
INSERT INTO tpch.region SELECT * FROM tpchobs.region;
INSERT INTO tpch.orders SELECT * FROM tpchobs.orders;
```

It takes 10 minutes to import data.

----End

Step 2: Performing Multi-Table Analysis and Theme Analysis

The following uses standard TPC-H query as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying revenue of a supplier in a region (TPCH-Q5)**

  By executing the TPCH-Q5 query statement, you can query the revenue statistics of a spare parts supplier in a region. The revenue is calculated based on **sum(l_extendedprice * (1 - l_discount))**. The statistics can be used to determine whether a local allocation center needs to be established in a given region.

  Copy and execute the following TPCH-Q5 statement for query. This statement features multi-table join query with **GROUP BY**, **ORDER BY**, and **AGGREGATE**.

```
SET current_schema='tpch';
SELECT
n_name,
sum(l_extendedprice * (1 - l_discount)) as revenue
FROM
customer,
orders,
lineitem,
supplier,
nation,
region
where
c_custkey = o_custkey
and l_orderkey = o_orderkey
and l_suppkey = s_suppkey
and c_nationkey = s_nationkey
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'ASIA'
and o_orderdate >= '1994-01-01'::date
and o_orderdate < '1994-01-01'::date + interval '1 year'
group by
n_name
order by
revenue desc;
```

- **Querying relationships between spare parts and suppliers (TPCH-Q16)**

  By executing the TPCH-Q16 query statement, you can obtain the number of suppliers that can supply spare parts with the specified contribution conditions. This information can be used to determine whether there are sufficient suppliers when the order quantity is large and the task is urgent.

  Copy and execute the following TPCH-Q16 statement for query. The statement features multi-table connection operations with group by, sort by, aggregate, deduplicate, and NOT IN subquery.

```
SET current_schema='tpch';
SELECT
p_brand,
p_type,
p_size,
count(distinct ps_suppkey) as supplier_cnt
FROM
```



```
partsupp,
part
where
p_partkey = ps_partkey
and p_brand <> 'Brand#45'
and p_type not like 'MEDIUM POLISHED%'
and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
and ps_suppkey not in (
        select
        s_suppkey
        from
        supplier
        where
        s_comment like '%Customer%Complaints%'
)
group by
p_brand,
p_type,
p_size
order by
supplier_cnt desc,
p_brand,
p_type,
p_size
limit 100;
```

- **Querying revenue loss of small orders (TPCH-Q17)**

  You can query the average annual revenue loss if there are no small orders. Filter out small orders that are lower than the 20% of the average supply volume, and calculate the total amount of those small orders to figure out the average annual revenue loss.

  Copy and execute the following TPCH-Q17 statement for query. The statement features multi-table connection operations with aggregate and aggregate subquery.

```
SET current_schema='tpch';
SELECT
sum(l_extendedprice) / 7.0 as avg_yearly
FROM
lineitem,
part
where
p_partkey = l_partkey
and p_brand = 'Brand#23'
and p_container = 'MED BOX'
and l_quantity < (
        select 0.2 * avg(l_quantity)
        from lineitem
        where l_partkey = p_partkey
);
```

3.3 Using GaussDB(DWS) to Analyze the Operational Status of a Retail Department Store

Background

In this practice, the daily business data of each retail store is loaded from OBS to the corresponding table in the data warehouse cluster for summarizing and querying KPIs. This data includes store turnover, customer flow, monthly sales ranking, monthly customer flow conversion rate, monthly price-rent ratio, and sales per unit area. This example demonstrates the multidimensional query and analysis of GaussDB(DWS) in the retail scenario.

NOTE

The sample data has been uploaded to the **retail-data** folder in an OBS bucket, and all HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket.

General Procedure

This practice takes about 60 minutes. The process is as follows:

1. **Preparations**
2. **Step 1: Importing Sample Data from the Retail Department Store**
3. **Step 2: Performing Operations Status Analysis**

Supported Regions

Table 3-7 describes the regions where OBS data has been uploaded.

Table 3-7 Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1
CN North-Beijing2	dws-demo-cn-north-2
CN North-Beijing4	dws-demo-cn-north-4
CN North-Ulanqab1	dws-demo-cn-north-9
CN East-Shanghai1	dws-demo-cn-east-3
CN East-Shanghai2	dws-demo-cn-east-2
CN South-Guangzhou	dws-demo-cn-south-1
CN South-Guangzhou-InvitationOnly	dws-demo-cn-south-4
CN-Hong Kong	dws-demo-ap-southeast-1
AP-Singapore	dws-demo-ap-southeast-3
AP-Bangkok	dws-demo-ap-southeast-2
LA-Santiago	dws-demo-la-south-2
AF-Johannesburg	dws-demo-af-south-1
LA-Mexico City1	dws-demo-na-mexico-1
LA-Mexico City2	dws-demo-la-north-2
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

Preparations

- You have registered a GaussDB(DWS) account, and the account is not in arrears or frozen.
- You have obtained the AK and SK of the account.
- A cluster has been created and connected using Data Studio. For details, see **Step 1: Creating a Cluster** and **Step 2: Using Data Studio to Connect to a Cluster**.

Step 1: Importing Sample Data from the Retail Department Store

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the sample data from retail department stores and perform queries.

Step 1 Execute the following statement to create the **retail** database:

```
CREATE DATABASE retail encoding 'utf8' template template0;
```

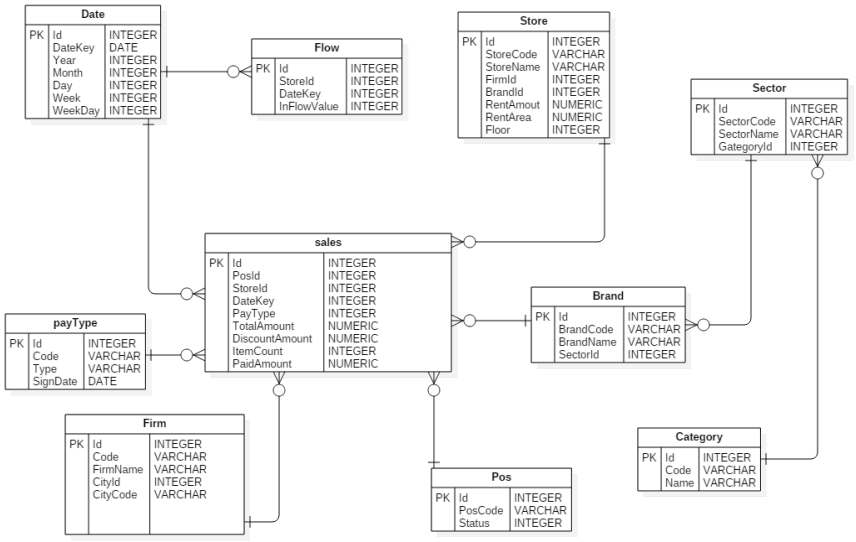
Step 2 Perform the following steps to switch to the new database:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and choose **Refresh** from the shortcut menu. Then, the new database is displayed.
2. Right-click the name of the new database **retail** and choose **Connect to DB** from the shortcut menu.
3. Right-click the name of the new database **retail** and choose **Open Terminal** from the shortcut menu. The SQL command window for connecting to the specified database is displayed. Perform the following steps in the window.

Step 3 Create a database table.

The sample data consists of 10 database tables whose associations are shown in **Figure 3-3**.

Figure 3-3 Sample data tables of retail department stores



Copy and execute the following statements to switch to create a database table of retail department store information.

```
CREATE SCHEMA retail_data;
SET current_schema='retail_data';

DROP TABLE IF EXISTS STORE;
CREATE TABLE STORE (
	ID INT,
	STORECODE VARCHAR(10),
	STORENAME VARCHAR(100),
	FIRMID INT,
	FLOOR INT,
	BRANDID INT,
	RENTAMOUNT NUMERIC(18,2),
	RENTAREA NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS POS;
CREATE TABLE POS(
	ID INT,
	POSCODE VARCHAR(20),
	STATUS INT,
	MODIFICATIONDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS BRAND;
CREATE TABLE BRAND (
	ID INT,
	BRANDCODE VARCHAR(10),
	BRANDNAME VARCHAR(100),
	SECTORID INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SECTOR;
CREATE TABLE SECTOR(
	ID INT,
	SECTORCODE VARCHAR(10),
	SECTORNAME VARCHAR(20),
	CATEGORYID INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS CATEGORY;
CREATE TABLE CATEGORY(
	ID INT,
	CODE VARCHAR(10),
	NAME VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS FIRM;
CREATE TABLE FIRM(
	ID INT,
	CODE VARCHAR(4),
	NAME VARCHAR(40),
	CITYID INT,
	CITYNAME VARCHAR(10),
	CITYCODE VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS DATE;
CREATE TABLE DATE(
	ID INT,
	DATEKEY DATE,
	YEAR INT,
```



```
    MONTH INT,
    DAY INT,
    WEEK INT,
    WEEKDAY INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS PAYTYPE;
CREATE TABLE PAYTYPE(
    ID INT,
    CODE VARCHAR(10),
    TYPE VARCHAR(10),
    SIGNDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SALES;
CREATE TABLE SALES(
     ID INT,
     POSID INT,
     STOREID INT,
     DATEKEY INT,
     PAYTYPE INT,
     TOTALAMOUNT NUMERIC(18,2),
     DISCOUNTAMOUNT NUMERIC(18,2),
     ITEMCOUNT INT,
     PAIDAMOUNT NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);

DROP TABLE IF EXISTS FLOW;
CREATE TABLE FLOW (
     ID INT,
     STOREID INT,
     DATEKEY INT,
     INFLOWVALUE INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);
```

Step 4 Create a foreign table, which is used to identify and associate the source data on OBS.

NOTICE

- *<obs_bucket_name>* indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see **Supported Regions**. GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- , and replace *<Access_Key_Id>* and *<Secret_Access_Key>* with the value obtained in **Preparations**.
- Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
- If the message "ERROR: schema "*xxx*" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
CREATE SCHEMA retail_obs_data;
SET current_schema='retail_obs_data';
DROP FOREIGN table if exists SALES_OBS;
CREATE FOREIGN TABLE SALES_OBS
(
    like retail_data.SALES
```



```
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/sales',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);

DROP FOREIGN table if exists FLOW_OBS;
CREATE FOREIGN TABLE FLOW_OBS
(
        like retail_data.flow
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/flow',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);

DROP FOREIGN table if exists BRAND_OBS;
CREATE FOREIGN TABLE BRAND_OBS
(
        like retail_data.brand
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/brand',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);


DROP FOREIGN table if exists CATEGORY_OBS;
CREATE FOREIGN TABLE CATEGORY_OBS
(
      like retail_data.category
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/category',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);
```



```
DROP FOREIGN table if exists DATE_OBS;
CREATE FOREIGN TABLE DATE_OBS
(
    like retail_data.date
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/date',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

DROP FOREIGN table if exists FIRM_OBS;
CREATE FOREIGN TABLE FIRM_OBS
(
    like retail_data.firm
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/firm',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);


DROP FOREIGN table if exists PAYTYPE_OBS;
CREATE FOREIGN TABLE PAYTYPE_OBS
(
    like retail_data.paytype
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/paytype',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);


DROP FOREIGN table if exists POS_OBS;
CREATE FOREIGN TABLE POS_OBS
(
    like retail_data.pos
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/pos',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
```



```
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);

DROP FOREIGN table if exists SECTOR_OBS;
CREATE FOREIGN TABLE SECTOR_OBS
(
        like retail_data.sector
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/sector',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);


DROP FOREIGN table if exists STORE_OBS;
CREATE FOREIGN TABLE STORE_OBS
(
        like retail_data.store
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/store',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);
```

Step 5 Copy and execute the following statements to import the foreign table data to the cluster:

```
INSERT INTO retail_data.store SELECT * FROM retail_obs_data.STORE_OBS;
INSERT INTO retail_data.sector SELECT * FROM retail_obs_data.SECTOR_OBS;
INSERT INTO retail_data.paytype SELECT * FROM retail_obs_data.PAYTYPE_OBS;
INSERT INTO retail_data.firm SELECT * FROM retail_obs_data.FIRM_OBS;
INSERT INTO retail_data.flow SELECT * FROM retail_obs_data.FLOW_OBS;
INSERT INTO retail_data.category SELECT * FROM retail_obs_data.CATEGORY_OBS;
INSERT INTO retail_data.date SELECT * FROM retail_obs_data.DATE_OBS;
INSERT INTO retail_data.pos SELECT * FROM retail_obs_data.POS_OBS;
INSERT INTO retail_data.brand SELECT * FROM retail_obs_data.BRAND_OBS;
INSERT INTO retail_data.sales SELECT * FROM retail_obs_data.SALES_OBS;
```

It takes some time to import data.

Step 6 Copy and execute the following statement to create the **v_sales_flow_details** view:

```
SET current_schema='retail_data';
CREATE VIEW v_sales_flow_details AS
SELECT
FIRM.ID FIRMID, FIRM.NAME FIRNAME, FIRM. CITYCODE,
CATEGORY.ID CATEGORYID, CATEGORY.NAME CATEGORYNAME,
SECTOR.ID SECTORID, SECTOR.SECTORNAME,
BRAND.ID BRANDID, BRAND.BRANDNAME,
STORE.ID STOREID, STORE.STORENAME, STORE.RENTAMOUNT, STORE.RENTAREA,
DATE.DATEKEY, SALES.TOTALAMOUNT, DISCOUNTAMOUNT, ITEMCOUNT, PAIDAMOUNT, INFLOWVALUE
```



```
FROM SALES
INNER JOIN STORE ON SALES.STOREID = STORE.ID
INNER JOIN FIRM ON STORE.FIRMID = FIRM.ID
INNER JOIN BRAND ON STORE.BRANDID = BRAND.ID
INNER JOIN SECTOR ON BRAND.SECTORID = SECTOR.ID
INNER JOIN CATEGORY ON SECTOR.CATEGORYID = CATEGORY.ID
INNER JOIN DATE ON SALES.DATEKEY = DATE.ID
INNER JOIN FLOW ON FLOW.DATEKEY = DATE.ID AND FLOW.STOREID = STORE.ID;
```

----End

Step 2: Performing Operations Status Analysis

The following uses standard query of retail information from department stores as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying the monthly sales revenue of each store**

  Copy and execute the following statements to query the total revenue of each store in a certain month:

  ```
  SET current_schema='retail_data';
  SELECT DATE_TRUNC('month',datekey)
  AT TIME ZONE 'UTC' AS __timestamp,
  SUM(paidamount)
  AS sum__paidamount
  FROM v_sales_flow_details
  GROUP BY DATE_TRUNC('month',datekey) AT TIME ZONE 'UTC'
  ORDER BY SUM(paidamount) DESC;
  ```

- **Querying the sales revenue and price-rent ratio of each store**

  Copy and execute the following statement to query the sales revenue and price-rent ratio of each store:

  ```
  SET current_schema='retail_data';
  SELECT firname AS firname,
  storename AS storename,
  SUM(paidamount)
  AS sum__paidamount,
  AVG(RENTAMOUNT)/SUM(PAIDAMOUNT)
  AS rentamount_sales_rate
  FROM v_sales_flow_details
  GROUP BY firname, storename
  ORDER BY SUM(paidamount) DESC;
  ```

- **Analyzing the sales revenue of each city**

  Copy and execute the following statement to analyze and query the sales revenue of all provinces:

  ```
  SET current_schema='retail_data';
  SELECT citycode AS citycode,
  SUM(paidamount)
  AS sum__paidamount
  FROM v_sales_flow_details
  GROUP BY citycode
  ORDER BY SUM(paidamount) DESC;
  ```

- **Analyzing and comparing the price-rent ratio and customer flow conversion rate of each store**


```
SET current_schema='retail_data';
SELECT brandname AS brandname,
firname AS firname,
SUM(PAIDAMOUNT)/AVG(RENTAREA) AS sales_rentarea_rate,
SUM(ITEMCOUNT)/SUM(INFLOWVALUE) AS poscount_flow_rate,
AVG(RENTAMOUNT)/SUM(PAIDAMOUNT) AS rentamount_sales_rate
FROM v_sales_flow_details
GROUP BY brandname,  firname
ORDER BY sales_rentarea_rate DESC;
```

- **Analyzing brands in the retail industry**

```
SET current_schema='retail_data';
SELECT categoryname AS categoryname,
brandname AS brandname,
SUM(paidamount) AS sum__paidamount
FROM v_sales_flow_details
GROUP BY categoryname,
brandname
ORDER BY sum__paidamount DESC;
```

- **Querying daily sales information of each brand**

```
SET current_schema='retail_data';
SELECT brandname AS brandname,
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC' AS __timestamp,
SUM(paidamount) AS sum__paidamount
FROM v_sales_flow_details
WHERE datekey >= '2016-01-01 00:00:00'
AND datekey <= '2016-01-30 00:00:00'
GROUP BY brandname,
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC'
ORDER BY sum__paidamount ASC
LIMIT 50000;
```


4 Decoupled Storage and Compute

4.1 DWS 3.0 Decoupled Storage and Compute Usage Suggestions and Performance Optimization

Scenarios

The newly released DWS 3.0 version provides resource pooling, massive storage, and the MPP architecture with decoupled compute and storage. This enables high elasticity, real-time data import and sharing, and lake warehouse integration.

For more information about decoupled compute and storage, see **What Is GaussDB(DWS)?**.

This document describes the performance optimization and precautions of the decoupled storage-compute version.

Purchasing a Cluster

- **EVS disk space**

  In version 9.1.0.x, only column-store user data is stored on OBS, while other data remains on local disks. Consequently, even in a storage-compute decoupled architecture, additional EVS disks still need to be configured, and not just minimally sized disks. The storage-compute decoupling capability will continue to evolve in future versions.

  Table 4-1 Storage of different table types

Table Type	Storage Location	Scenario
Row-store table, temporary table, and column-store index	Local, no compression	Point query, real-time small-batch import, and frequent update.

Table Type	Storage Location	Scenario
Column-store table 2.0	Local, compressed	Batch import, query, real-time small-batch import, point query, and update.
Column-store table 3.0	OBS, compressed	Batch import, query, and low-frequency batch update.

EVS storage: row-store and column-store metadata (min/max), indexes, Delta, WAL, OBS data cache, and temporary files (sort/hash) in computing. The size can be specified when you purchase a GaussDB(DWS) cluster.

Formula for calculating the required total EVS storage size::

(2 copies x (Row-store table size + Index size + Delta table size) + OBS hot data cache size)/0.8 (reserved)

NOTE

When the total EVS storage size surpasses 90%, the cluster transitions to read-only mode. The reserved 10% space is allocated for storing WALs and temporary files.

- OBS hot data: 1. The size of hot data is clearly known. 2. If you do not know the data volume, you can select 30% of the total OBS data volume.
- Size of a column-store index = Size of original uncompressed data x Index column width x 3 (Bloat rate) /Total column width
- The column-store data is assessed using a 3x compression ratio. Assuming a table contains 20 columns, with two serving as the primary key, the index constitutes 30% of the data prior to compression, equivalent to the data post-compression.
- Delta size: Max (10 GB, size of the compressed table/10) of a table (or partition)
- Row-store indexes are evaluated based on the rate of 30%.

Recommended EVS disk space: Utilizing the aforementioned algorithm can be complex. It is advisable to configure the total EVS disk space to match the total data volume after compression, assuming a compression ratio of 5x. If the table lacks an index (with EVS primarily serving as cache), set the total EVS disk space to 50% or 30% of the total data volume (excluding archived data) and subsequently increase the disk cache size (refer to the following section for details).

Minimum capacity

- Performance-sensitive customers: Ensure that each primary/standby DN has a minimum disk capacity of 500 GB to achieve a throughput of 350 MB/s per disk. For instance, if two primary and two standby DNs are deployed on an ECS, at least four 500 GB disks must be mounted to the ECS.
- Cost-sensitive customers: At least 200 GB (160 MB/s per disk) is mounted to each primary/standby DN.

- **OBS configuration**

  OBS must be deployed in three AZs and support parallel file systems.

OBS performance

If you need to adjust the OBS performance specifications, contact technical support. In public cloud scenarios, OBS metrics do not need to be adjusted for a cluster with six or fewer DNs.

- **CPU configuration**

  In a production environment, it is advised that each node has a minimum of 16 vCPUs. Configurations of 4 vCPUs or 8 vCPUs are typically used only for experimental or testing purposes.

  - Migration scenario: consistent with the migration objects.
  - New deployment scenario: Calculate the number of required CPU cores by dividing the total data volume by 100 GB. Note that the number of CPU cores required can vary based on the specific use case. If the computational load is high, it is recommended to increase the number of CPU cores accordingly.

Table Design Optimization

Table creation statements

By default, DWS creates row-store tables. In OLAP analysis scenarios, you need to explicitly set ORIENTATION to column-store when creating tables.

For details, see **CREATE TABLE** in *SQL Syntax Reference*.

```
CREATE TABLE public.t1(id integer) WITH (ORIENTATION =COLUMN);
```

Table design optimization

In a storage-compute decoupling architecture, data is stored in the Object Storage Service (OBS). To optimize performance, it's crucial to use filtering methods to avoid unnecessary remote Compute Unit (CU) data read overhead. This approach significantly enhances performance.

Filtering methods

GaussDB(DWS) is compatible with the PostgreSQL ecosystem, utilizing both row storage with B-tree indexes similar to PostgreSQL, and self-developed column storage with its own indexing system. When creating a table, it's important to select an appropriate storage mode, distribution column, partition key, and index to ensure that data can be quickly accessed during SQL execution, thereby reducing I/O consumption. The following figure shows the process from initiating an SQL statement to obtaining data. You can understand the function of each technical approach for better performance optimization.

1. When the SQL statement is executed, the partition table is optimized using the Partition Column to pinpoint the specific partition.
2. The Distribute Column is used in a distributed hash table to quickly identify the data shard where the data resides. In a storage-compute coupled architecture, the data shard is located on a DN, while in a storage-compute decoupled architecture, it's located on a bucket.
3. In row-store mode, B-tree is used to quickly locate the data page. In column-store mode, the min-max index is used to quickly locate the CU data block that may contain relevant data. This index is particularly effective when filtering on the Partition Key (PCK) column.

4. The system automatically maintains the min-max index for all columns in the column-store mode. There's no need for manual index definition. The min-max index serves as a coarse filter. However, CU data blocks that meet the min-max condition may not necessarily contain data rows that meet the filter condition. If a bitmap column is defined, the bitmap index can be used to quickly locate the row number of the data that meets the filter condition within the CU. For ordered CUs, binary search can also be employed to quickly find the row number.
5. Column-store also supports B-tree and GIN indexes, which can be used to quickly locate the CU and row number of the data that meets the conditions. However, the maintenance cost of these indexes is high. Unless there are high performance requirements for point queries, it is recommended to use bitmap indexes instead of B-tree or GIN indexes.

Optimization methods

The following uses a table creation statement to describe the existing optimization methods of GaussDB(DWS). For details, see **CREATE TABLE** in *SQL Syntax Reference*.

```
create table t1(c1 int, c2 text, c3 varchar(15), c4 numeric, c5 numeric(16,
 primary key(c1, c6),
 partial cluster key(c5))
 with(orientation=column,enable_hstore_opt=true,secondary_part_column='c7'
distribute by hash(c1)
partition by range(c6)
 (partition p1 valeus less than('1999-10-01 00:00:00'),
  partition p2 values less than('2000-10-01 00:00:00')
 );
```

Table 4-2 Optimization methods

No.	Optimization Item	Suggestion	Example SQL	Modifiable After Creation
1	String type	• The string type exhibits slower performance compared to the fixed-length type. It is not recommended for scenarios where the fixed-length type is suitable. • When the specified length is less than 16, performance can be improved by up to double. However, this optimization does not provide benefits if the specified length exceeds 16.	-	Yes. Modification rewrites existing data.

No.	Optimization Item	Suggestion	Example SQL	Modifiable After Creation
2	Numeric type	Specify precision for the numeric type, which doubles the performance. Do not use numeric types without precision.	--	Yes. Modification rewrites existing data.
3	Partition by Column	• Define partition tables. Partition keys enable pruning, and partition-wise joins are supported for equality and range query scenarios. • Define less than 1000 partitions with a maximum of 2 partition columns.	`SELECT * FROM t1 WHERE t1.c1='p1';`	No. If modification is needed, create a table again.
4	secondary_part_column	• Define this field. It is applicable only to column-store tables and equality queries. • Specify a level-2 partition on the most commonly used equivalent filter.	`SELECT *  FROM t1 WHERE t1.c1='p1';`	No. If modification is needed, create a table again.
5	Distribute by Column	Define this field. It is applicable to the join field frequently used for GROUP BY or multi-table join operations. Local joins reduce data shuffling and are suitable for equivalent queries.	`SELECT * FROM t1 join t2 on t1.c3 = t2.c1;`	No. If modification is needed, create a table again.
6	Bitmap_columns	Create an adaptive bitmap index (for cardinality ≤ 32) or bloom filter (for cardinality > 32) based on repeated values in the CU. This is applicable to equivalent query scenarios for VARCHAR or TEXT columns. It is recommended to define the columns involved in the WHERE condition.	`SELECT * FROM t1 WHERE t1.c4 = 'hello';`	Yes. Modification does not rewrite existing data. Only the new data is affected.

No.	Optimization Item	Suggestion	Example SQL	Modifiable After Creation
7	Min-max index	• The min-max index does not need to be user-defined and is applicable to equality and range query scenarios. • The min-max filtering effect depends on the data order. Specifying the PCK column enhances the filtering effect.	SELECT * FROM t1 WHERE c3 > 100 and c3 < 200;	The PCK can be modified. Modification does not rewrite existing data. Only the new data is affected.
8	Primary key (btree index)	• Importing upsert data to the database heavily relies on the primary key, which must be defined by users and is applicable to equality and range query scenarios. • It is preferable to use fixed-length type columns when service requirements allow. Place columns with more distinct values at the front whenever possible.	SELECT * FROM t1 WHERE c3 >100 and c3 < 200;	Yes. After modification, the index will be recreated.
	GIN index	• Define this parameter. It is suitable for multi-condition equality queries. Avoid using columns with more than 1 million distinct values. • It is recommended when the data volume after filtering is less than 1000. If the data volume remains large after filtering, it is not recommended.	SELECT * FROM t1 WHERE c1 = 200 and c2 = 105;	Yes. After modification, the index will be recreated.
9	Orientation=column/row	Specify whether a table is stored in row or column mode. Row-store tables are uncompressed, ideal for point queries and frequent updates. Column-store tables are compressed and are best suited for analysis scenarios.	-	No. If modification is needed, create a table again.

Disk Cache

DWS caches frequently accessed data on local EVS disks to minimize direct reads from OBS and enhance data query performance. The disk cache is exclusive to the DN compute nodes and is not present on the CNs.

Cache size

The default cache size (**disk_cache_max_size**) of the cluster is 1/3 of the EVS capacity.

The EVS capacity is divided into two parts: 1/3 x 2 replicas are used to store local persistent data (such as column-store indexes, row-store tables, and local column-store tables), and the other 1/3 is reserved for cache. GaussDB(DWS) indexes differ from Redshift indexes. Redshift indexes serve solely as optimizer prompts and do not contain actual index data. In contrast, GaussDB(DWS) indexes resemble Oracle indexes and store index data.

If no index is created for a column-store table, increase the cache size by adjusting the value of **disk_cache_max_size** on the GaussDB(DWS) management console.

Cache status

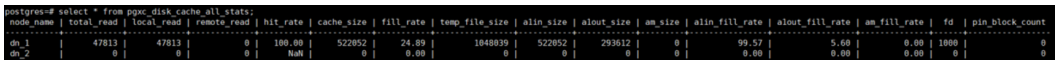
When a user queries data, the system first checks if the data exists in the local disk cache. If the data is not present, the system reads it from OBS and caches it to the local disk for future access. This caching mechanism can significantly enhance the speed of OBS data queries.

By default, the disk cache uses two disks in active/standby mode as the cache media. You can query the following parameters to view related information:

- Use the **disk_cache_base_paths** parameter to view, add, or delete cache disk paths.
- Use the **disk_cache_max_size** parameter to view and adjust the disk cache size.

Use the **pgxc_disk_cache_all_stats** view to view the current cache hit ratio and the disk usage of each DN.

Figure 4-1 Query result from pgxc_disk_cache_all_stats



Cache dual-write

Enabling **Cache Dual-Write** can enhance the performance of the first data query. Specifically, when data is written to the remote OBS, it is also written to the local disk cache. This improves read efficiency significantly during the initial data access. You can use the **disk_cache_dual_write_option** to configure whether to enable cache dual-write. The options are as follows:

- **none**: Disable cache dual-write.
- **hstore_only** (default value): Enable cache dual-write only for the Hstore opt table during delta merge.
- **all**: Enable cache dual-write for both common v3 tables and hstore opt tables.

Clearing the Cache

Use the **pgxc_clear_disk_cache()** function to clear all disk caches.

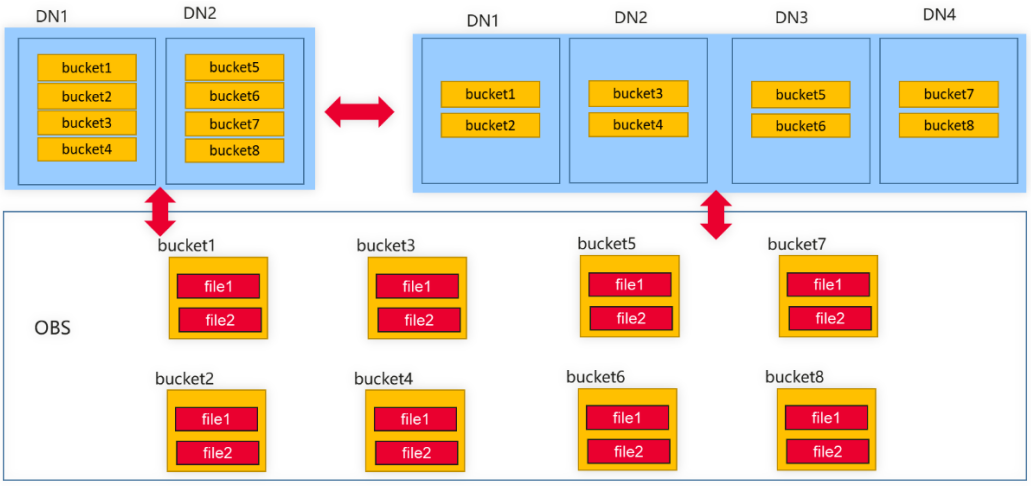
Insertion Performance

Bucket storage

Bucket storage is a method of data sharding that, similar to partitioning technology, groups data with the same attribute values together. This approach facilitates the adjustment of the mapping between storage and computing, enabling a separation of storage and computing resources, and allowing for elastic scaling and on-demand allocation of computing resources.

For instance, if there are eight buckets and two DNs, each DN would be responsible for four buckets. Conversely, if there are four DNs, each DN would be responsible for two buckets.

Figure 4-2 Bucket storage



Import optimization

Data needs to be saved to the database in batches, with asynchronous I/O.

Batching: This technique is employed to avoid small Compute Units (CUs) and enhance subsequent query performance.

Asynchronous I/O: Following the decoupling of storage and compute, the latency for writing data to OBS is approximately ten times higher than writing data to EVS. Asynchronous I/O optimizes read and write performance.

- For partitioned tables, 2.0 tables require only partition batching. 3.0 tables, in comparison, require bucket batching (equivalent to level-2 partitions), which may consume more memory and disk space.
- Only hash-distributed tables necessitate bucket batching.

Batching overhead and suggestions

Overhead

```
Number of partitions
Number of buckets on each node: #Nb
```


RowGroup size before compression: #Nr
Maximum size of a single bucket: #Mb = max (partition_max_cache_size/partition_men_batch, 16M) = 16M (default configuration)

Single-concurrency batching consumption: #Np * #Nb * #Nr
Single-concurrency batching memory consumption: **partition_max_cache_size**. The default value is 2 GB.
Single-concurrency batching disk consumption: #Np * #Nb * #Nr * 1.2 (bloat rate) - Memory consumption

Assume that data is copied at a time, 1000 partitions are involved, #Nb ≈ 10, the size of a single record is 1 KB, and the total batching size is 10,000 rows.
Single-concurrency batching consumption: 1000 * 10 * 1K * 10000 * 1.2 = 120 GB

Suggestions

1. Application layer optimization: The key factor is the number of partitions. It is recommended to use a single partition for importing data into the database. If the consumed space of a single concurrent batching operation is reduced from 120 GB to 120 MB, the memory can be directly utilized for batching.
2. Database Kernel Optimization: Modify the **min_batch_rows** parameter to adjust the batch size. You can execute the SET statement to apply the changes for the current session or modify the configuration file to make the changes effective immediately.

5 Data Development

5.1 Cutting Costs by Switching Between Cold and Hot Data Storage in GaussDB(DWS)

Scenarios

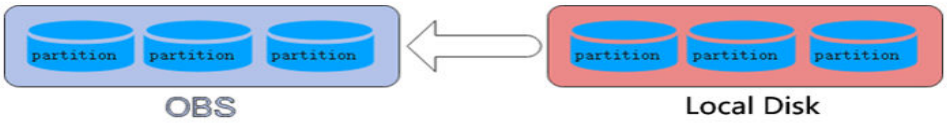
In massive big data scenarios, with the growing of data, data storage and consumption increase rapidly. The need for data may vary in different time periods, therefore, data is managed in a hierarchical manner, improving data analysis performance and reducing service costs. In some data usage scenarios, data can be classified into hot data and cold data by accessing frequency.

Hot and cold data is classified based on the data access frequency and update frequency.

- Hot data: Data that is frequently accessed and updated and requires fast response.
- Cold data: Data that cannot be updated or is seldom accessed and does not require fast response

You can define cold and hot management tables to switch cold data that meets the specified rules to OBS for storage. Cold and hot data can be automatically determined and migrated by partition.

Figure 5-1 Hot and cold data management



When data is inserted to GaussDB(DWS) column-store tables, the data is first stored in hot partitions. As data accumulates, you can manually or automatically migrate the cold data to OBS for storage. The metadata, description tables, and indexes of the migrated cold data are stored locally to ensure the read performance.

The hot and cold partitions can be switched based on LMT (Last Modify Time) and HPN (Hot Partition Number) policies. LMT indicates that the switchover is performed based on the last update time of the partition, and HPN indicates that the switchover is performed based on the number of reserved hot partitions.

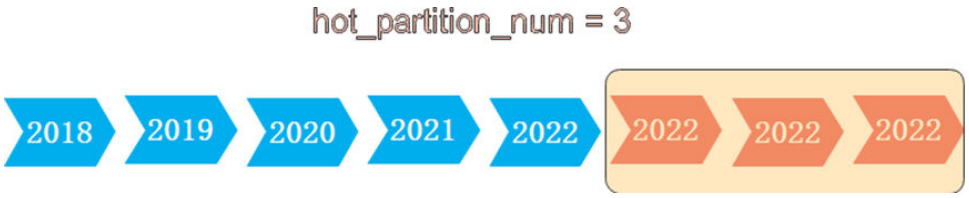
- **LMT**: Switch the hot partition data that is not updated in the last *[day]* days to the OBS tablespace as cold partition data. *[day]* is an integer ranging from 0 to 36500, in days.

  In the following figure, *day* is set to **2**, indicating that the partitions modified in the last two days are retained as the hot partitions, while the rest is retained as the cold partitions. Assume that the current time is April 30. The delete operation is performed on the partition **[4-26]** on April 30, and the insert operation is performed on the partition **[4-27]** on April 29. Therefore, partitions **[4-26]** **[4-27]** **[4-29]** **[4-30]** are retained as hot partitions.

- HPN: indicates the number of hot partitions to be reserved. The partitions are sequenced based on partition sequence IDs. The sequence ID of a partition is a built-in sequence number generated based on the partition boundary values and is not shown. For a range partition, a larger boundary value indicates a larger sequence ID. For a list partition, a larger maximum enumerated value of the partition boundary indicates a larger sequence ID. During the cold and hot switchover, data needs to be migrated to OBS. HPN is an integer ranging from 0 to 1600. If HPN is set to **0**, hot partitions are not reserved. During a cold/hot switchover, all partitions with data are converted to cold partitions and stored on OBS.

  In the following figure, HPN is set to 3, indicating that the last three partitions with data are retained as the hot partitions with the rest as the cold partitions during hot and cold partition switchover.

Constraints

- Supports DML operations on cold and hot tables, such as **INSERT**, **COPY**, **DELETE**, **UPDATE**, and **SELECT**.
- Supports DCL operations such as permission management on cold and hot tables.
- Supports ANALYZE, VACUUM, MERGE INTO, and PARTITION operations on cold and hot tables.
- Supports common column-store partitioned tables to be upgraded to hot and cold data tables.

- Supports upgrade, scale-out, scale-in, and redistribution operations on tables with cold and hot data management enabled.
- 8.3.0 and later versions support mutual conversion between cold and hot partitions. Versions earlier than 8.3.0 support only conversion from hot data to cold data.
- If a table has both cold and hot partitions, the query becomes slow because cold data is stored on OBS and the read/write speed are lower than those of local queries.
- Currently, cold and hot tables support only column-store partitioned tables of version 2.0. Foreign tables do not support cold and hot partitions.
- Only the cold and hot switchover policies can be modified. The tablespace of cold data in cold and hot tables cannot be modified.
- Restrictions on partitioning cold and hot tables:
  - Data in cold partitions cannot be exchanged.
  - **MERGE PARTITION** supports only the merge of hot-hot partitions and cold-cold partitions.
  - Partition operations, such as **ADD**, **MERGE**, and **SPLIT**, cannot be performed on an OBS tablespace.
  - Tablespaces of cold and hot table partitions cannot be specified or modified during table creation.
- Cold and hot data switchover is not performed immediately upon conditions are met. Data switchover is performed only after users manually, or through a scheduler, invoke the switchover command. Currently, the automatic scheduling time is 00:00 every day and can be modified.
- Cold and hot data tables do not support physical fine-grained backup and restoration. Only hot data is backed up during physical backup. Cold data on OBS does not change. The backup and restoration does not support file deletion statements, such as **TRUNCATE TABLE** and **DROP TABLE**.

Procedure

This practice takes about 30 minutes. The basic process is as follows:

1. **Creating a cluster**.
2. **Using the gsql CLI Client to Connect to a Cluster**.
3. **Creating Hot and Cold Tables**.
4. **Hot and Cold Data Switchover**.
5. **Viewing Data Distribution in Hot and Cold Tables**.

Creating a cluster

Step 1 Log in to the Huawei Cloud management console.

Step 2 Choose **Service List** > **Analytics** > **Data Warehouse Service**. On the page that is displayed, click **Create Cluster** in the upper right corner.

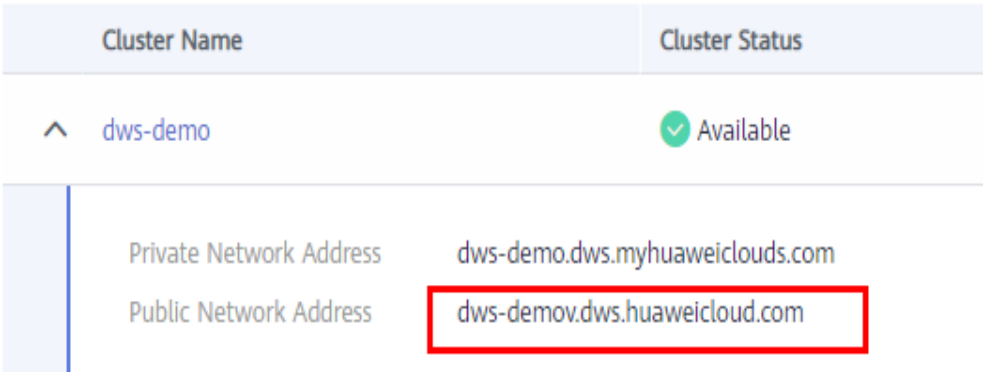
Step 3 Configure the parameters according to **Table 5-1**.

Table 5-1 Software configuration

Parameter	Configuration
Region	Select the **CN-Hong Kong** region. **NOTE** **CN-Hong Kong** is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region.
AZ	AZ2
Product	Standard data warehouse
CPU Architecture	X86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPUs | 128 GB | 2000 GB SSD) **NOTE** If this flavor is sold out, select other AZs or flavors.
Nodes	3
Cluster Name	dws-demo
Administrator Account	dbadmin
Administrator Password	N/A
Confirm Password	N/A
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1Mbit/s
Advanced Settings	Default

Step 4 Confirm the information, click **Next**, and then click **Submit**.

Step 5 Wait about 6 minutes. After the cluster is created, click next to the cluster name. On the displayed cluster information page, record the value of **Public Network Address**, for example, **dws-demov.dws.huaweicloud.com**.



----End

Using the gsql CLI Client to Connect to a Cluster

Step 1 Remotely log in to the Linux server where gsql is to be installed as user **root**, and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.ap-southeast-1.myhuaweicloud.com/dws/download/dws_client_8.1.x_redhat_x64.zip --no-check-certificate
```

Step 2 Decompress the client.

```
cd <Path_for_storing_the_client> unzip dws_client_8.1.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x64.zip*: This is the client tool package name of **RedHat x64**. Replace it with the actual name.

Step 3 Configure the GaussDB(DWS) client.

```
source gsql_env.sh
```

If the following information is displayed, the gsql client is successfully configured:

```
All things done.
```

Step 4 Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

----End

Creating Hot and Cold Tables

Create a column-store cold and hot data management table **lifecycle_table** and set the hot data validity period LMT to 100 days.

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:100')
PARTITION BY RANGE (i)
(
PARTITION P1 VALUES LESS THAN(5),
PARTITION P2 VALUES LESS THAN(10),
PARTITION P3 VALUES LESS THAN(15),
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

Hot and Cold Data Switchover

Switch hot partition data to cold partition data.

- Automatic switchover: The scheduler automatically triggers the switchover at 00:00 every day.

  You can use the **pg_obs_cold_refresh_time(table_name, time)** function to customize the automatic switchover time. For example, set the automatic triggering time to 06:30 every morning.

  ```
  SELECT * FROM pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
  pg_obs_cold_refresh_time
  --------------------------
   SUCCESS
  (1 row)
  ```

- Manual

  Run the **ALTER TABLE** statement to manually switch a single table.

  ```
  ALTER TABLE lifecycle_table refresh storage;
  ALTER TABLE
  ```

  Use the **pg_refresh_storage()** function to switch all hot and cold tables in batches.

  ```
  SELECT pg_catalog.pg_refresh_storage();
   pg_refresh_storage
  --------------------
   (1,0)
  (1 row)
  ```

Convert cold partition data into hot partition data. This function is supported only in 8.3.0 or later.

- Convert all cold partitions to hot partitions.
  ```
  SELECT pg_catalog.reload_cold_partition('lifecycle_table');
  ```
- Convert a specified cold partition to a hot partition:
  ```
  SELECT pg_catalog.reload_cold_partition('lifecycle_table', 'cold_partition_name');
  ```

Viewing Data Distribution in Hot and Cold Tables

- View the data distribution in a single table:
  ```
  SELECT * FROM pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
  schemaname |     tablename     |    nodename    | hotpartition | coldpartition | switchablepartition |
  hotdatasize | colddatasize | switchabledatasize
  ------------+-------------------+----------------+--------------+---------------+---------------------+-------------
  +--------------+--------------------
   public     | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8  |               |                     | 96 KB       | 0
  bytes      | 0 bytes
   public     | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8  |               |                     | 96 KB       | 0
  bytes      | 0 bytes
  ```



```
 public     | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8  |              |                  | 96 KB       | 0
bytes      | 0 bytes
(3 rows)
```

- View data distribution in all hot and cold tables:

```
SELECT * FROM pg_catalog.pg_lifecycle_node_data_distribute();
schemaname |     tablename     |   nodename   | hotpartition | coldpartition | switchablepartition |
hotdatasize | colddatasize | switchabledatasize
------------+-------------------+--------------+--------------+---------------+---------------------+-------------
+--------------+--------------------
 public     | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8  |              |                  |       98304 |
0 |                0
 public     | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8  |              |                  |       98304 |
0 |                0
 public     | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8  |              |                  |       98304 |
0 |                0
(3 rows)
```

5.2 Cutting Partition Maintenance Costs for the E-commerce and IoT Industries by Leveraging GaussDB(DWS)'s Automatic Partition Management Feature

Scenarios

For partition tables whose partition columns are time, the automatic partition management function can be added to automatically create partitions and delete expired partitions, reducing partition table maintenance costs and improving query performance. To facilitate data query and maintenance, the time column is often used as the partition column of a partitioned table that stores time-related data, such as e-commerce order information and real-time IoT data. When the time-related data is imported to a partitioned table, the table should have partitions of the corresponding time ranges. Common partition tables do not automatically create new partitions or delete expired partitions. Therefore, maintenance personnel need to periodically create new partitions and delete expired partitions, leading to increased O&M costs.

GaussDB(DWS) has introduced an automatic partition management feature to address this issue. You can set the table-level parameters **period** and **ttl** to enable the automatic partition management function, which automatically creates partitions and deletes expired partitions, reducing partition table maintenance costs and improving query performance.

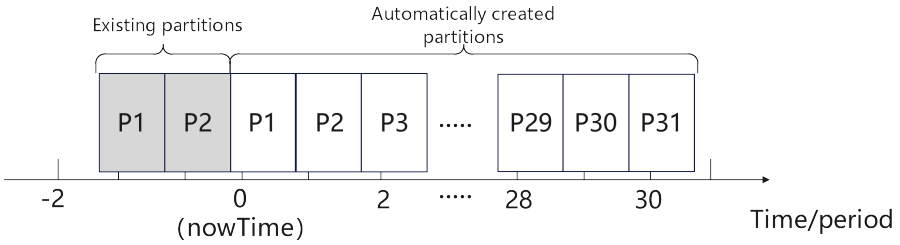
period: interval for automatically creating partitions. The default value is 1 day. The value range is 1 hour ~ 100 years.

ttl: time for automatically eliminate partitions. The value range is 1 hour ~ 100 years. Partition elimination occurs when nowtime - Partition boundary > ttl, resulting in the removal of qualifying partitions.

- Automatic partition creation

  One or more partitions are automatically created at the interval specified by **period** to make the maximum partition boundary time greater than nowTime + 30 x period. As long as there is an automatically created partition, real-time data will not fail to be imported within the next 30 periods.

Figure 5-2 Automatic partition creation



- Automatically deleting expired partitions

  Partitions whose boundary time is earlier than **nowTime-ttl** are considered expired partitions. The automatic partition management function traverses all partitions and deletes expired partitions after each **period**. If all partitions are expired partitions, the system retains one partition and truncates the table.

Constraints

When using the partition management function, ensure that the following requirements are met:

- It cannot be used on midrange servers, acceleration clusters, or stand-alone clusters.
- It can be used in clusters of version 8.1.3 or later.
- It can only be used for row-store range partitioned tables, column-store range partitioned tables, time series tables, and cold and hot tables.
- The partition key must be unique and its type must be timestamp, timestamptz, or date.
- The maxvalue partition is not supported.
- The value of (nowTime - boundaryTime)/period must be less than the maximum number of partitions. **nowTime** indicates the current time, and **boundaryTime** indicates the earliest partition boundary time.
- The values of **period** and **ttl** range from 1 hour to 100 years. In addition, in a database compatible with Teradata or MySQL, if the partition key type is date, the value of period cannot be less than 1day.
- The table-level parameter **ttl** cannot exist independently. You must set **period** in advance or at the same time, and the value of **ttl** must be greater than or equal to that of **period**.
- During online cluster scale-out, partitions cannot be automatically added. Partitions reserved each time partitions are added will ensure that services are not affected.

Creating an ECS

For details, see **Purchasing an ECS**. After purchasing an ECS, log in to the ECS by referring to **Logging In to a Linux ECS**.

NOTICE

When creating an ECS, ensure that the ECS is in the same region, AZ, and VPC subnet as the stream data warehouse. Select the OS used by the gsql client (CentOS 7.6 is used as an example) as the ECS OS, and select using passwords to log in.

Creating a cluster

Step 1 Log in to the Huawei Cloud management console.

Step 2 Choose **Service List** > **Analytics** > **Data Warehouse Service**. On the page that is displayed, click **Create Cluster** in the upper right corner.

Step 3 Configure the parameters according to **Table 5-2**.

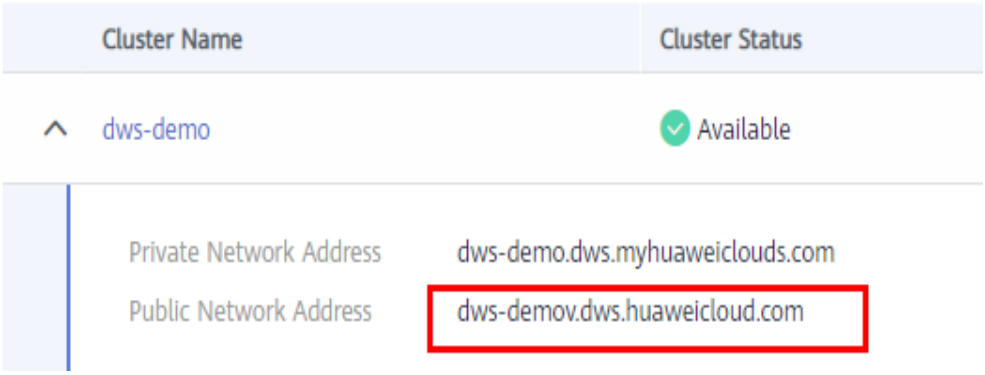
Table 5-2 Software configuration

Parameter	Configuration
Region	Select the **CN-Hong Kong** region. **NOTE** **CN-Hong Kong** is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region.
AZ	AZ2
Product	Standard data warehouse
CPU Architecture	X86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPUs | 128 GB | 2000 GB SSD) **NOTE** If this flavor is sold out, select other AZs or flavors.
Nodes	3
Cluster Name	dws-demo
Administrator Account	dbadmin
Administrator Password	N/A
Confirm Password	N/A
Database Port	8000

Parameter	Configuration
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1Mbit/s
Advanced Settings	Default

Step 4 Confirm the information, click **Next**, and then click **Submit**.

Step 5 Wait about 6 minutes. After the cluster is created, click next to the cluster name. On the displayed cluster information page, record the value of **Public Network Address**, for example, **dws-demov.dws.huaweicloud.com**.



----End

Using the gsql CLI Client to Connect to a Cluster

Step 1 Remotely log in to the Linux server where gsql is to be installed as user **root**, and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.ap-southeast-1.myhuaweicloud.com/dws/download/dws_client_8.1.x_redhat_x64.zip --no-check-certificate
```

Step 2 Decompress the client.

```
cd <Path_for_storing_the_client> unzip dws_client_8.1.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x64.zip*: This is the client tool package name of **RedHat x64**. Replace it with the actual name.

Step 3 Configure the GaussDB(DWS) client.

```
source gsql_env.sh
```

If the following information is displayed, the gsql client is successfully configured:

```
All things done.
```

Step 4 Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

----End

Automatic partition management

The partition management function is bound to the table-level parameters **period** and **ttl**. Automatic partition creation is enabled with the enabling of **period**, and automatic partition deletion is enabled with the enabling of **ttl**. 30 seconds after **period** or **ttl** is set, the automatic partition creation or deletion works for the first time.

You can enable the partition management function in either of the following ways:

- Specify **period** and **ttl** when creating a table.

  This way is applicable when you create a partition management table. There are two syntaxes for creating a partition management table. One specifies partitions, and the other does not.

  If partitions are specified when a partition management table is created, the syntax rules are the same as those for creating a common partitioned table. The only difference is that the syntax specifies the table-level parameters **period** and **ttl**.

  The following example shows how to create a partition management table **CPU1** and specify partitions.

  ```
  CREATE TABLE CPU1(
      id integer,
      IP text,
      time timestamp
  ) with (TTL='7 days',PERIOD='1 day')
  partition by range(time)
  (
      PARTITION P1 VALUES LESS THAN('2023-02-13 16:32:45'),
      PARTITION P2 VALUES LESS THAN('2023-02-15 16:48:12')
  );
  ```

  When creating a partition management table, you can specify only the partition key but not partitions. In this case, two default partitions will be created with **period** as the partition time range. The boundary time of the first default partition is the first hour, day, week, month, or year past the current time. The time unit is selected based on the maximum unit of PERIOD. The boundary time of the second default partition is the boundary time of the first partition plus PERIOD. Assume that the current time is 2023-02-17 16:32:45, and the boundary of the first default partition is described in the following table.

Table 5-3 Description of the period parameter

period	Maximum PERIOD Unit	Boundary of First Default Partition
1hour	Hour	2023-02-17 17:00:00
1day	Day	2023-02-18 00:00:00
1month	Month	2023-03-01 00:00:00
13months	Year	2024-01-01 00:00:00

Run the following command to create the partition management table **CPU2** with no partitions specified:

```
CREATE TABLE CPU2(
    id integer,
    IP text,
    time timestamp
) with (TTL='7 days',PERIOD='1 day')
partition by range(time);
```

- Run the **ALTER TABLE RESET** command to set **period** and **ttl**.

  This method is used to add the partition management function to an ordinary partitioned table that meets the partition management constraints.

  - Run the following command to create an ordinary partition table **CPU3**:

    ```
    CREATE TABLE CPU3(
        id integer,
        IP text,
        time timestamp
    )
    partition by range(time)
    (
        PARTITION P1 VALUES LESS THAN('2023-02-14 16:32:45'),
        PARTITION P2 VALUES LESS THAN('2023-02-15 16:56:12')
    );
    ```

  - To enable the automatic partition creation and deletion functions, run the following command:

    ```
    ALTER TABLE CPU3 SET (PERIOD='1 day',TTL='7 days');
    ```

  - To enable only the automatic partition creation function, run the following command:

    ```
    ALTER TABLE CPU3 SET (PERIOD='1 day');
    ```

  - To enable only the automatic partition deletion function, run the following command (If automatic partition creation is not enabled in advance, the operation will fail):

    ```
    ALTER TABLE CPU3 SET (TTL='7 days');
    ```

  - Modify the **period** and **ttl** parameters to modify the partition management function.

    ```
    ALTER TABLE CPU3 SET (TTL='10 days',PERIOD='2 days');
    ```

- Disabling the partition management function

  You can run the **ALTER TABLE RESET** command to delete the table-level parameters **period** and **ttl** to disable the partition management function.

NOTE

- The **period** cannot be deleted separately with **TTL**.
- The time series table does not support **ALTER TABLE RESET**.

– Run the following command to disable the automatic partition creation and deletion functions:

```
ALTER TABLE CPU1 RESET (PERIOD,TTL);
```

– To disable only the automatic partition deletion, run the following command:

```
ALTER TABLE CPU3 RESET (TTL);
```

– To disable only the automatic partition creation function, run the following command (If the table contains the **ttl** parameter, the operation will fail):

```
ALTER TABLE CPU3 RESET (PERIOD);
```

5.3 Improving Development Efficiency by Leveraging GaussDB(DWS)'s View Decoupling and Rebuilding Function

Base table objects cannot be modified independently due to view and table dependency. To solve this problem, GaussDB(DWS) supports view decoupling and rebuilding. This document describes when and how to use the automatic view rebuilding function.

Scenario

GaussDB(DWS) uses object identifiers (OIDs) to store reference relationships between objects. When a view is defined, the OID of the database object on which the view depends is bound to it. No matter how the view name changes, the dependency does not change. If you modify some columns in the base table, an error will be reported because the columns are strongly bound some objects. If you want to delete a table column or the entire table, you need to use the **cascade** keyword to delete the associated views. After the table column is deleted or the table is re-created, you need to re-create the views of different levels one by one. This increases the workload and deteriorates the usability.

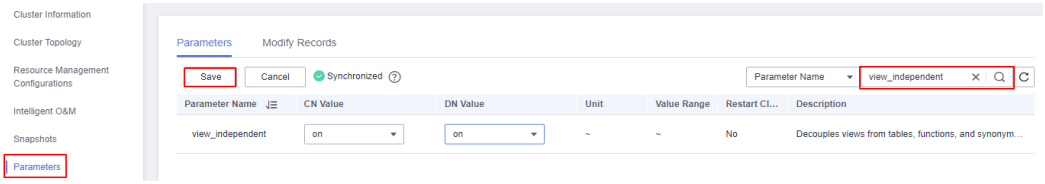
To solve this problem, GaussDB(DWS) 8.1.0 decouples views from their dependent base tables or other database objects (views, synonyms, functions, and table columns), so that these objects can be deleted independently. After the base table is rebuilt, you can run the **ALTER VIEW REBUILD** command to rebuild the dependency. As a development, the version 8.1.1 supports automatic rebuilding. Dependencies can be automatically rebuilt without user awareness. After automatic rebuilding is enabled, lock conflicts may occur. Therefore, you are advised not to enable automatic rebuilding.

Usage

Step 1 Create a cluster on the management console. For details, see **Creating a DWS 2.0 Cluster**.

Step 2 Enable the GUC parameter **view_independent**.

The GUC parameter **view_independent** controls whether to decouple a view from its objects. This parameter is disabled by default. You need to manually enable the parameter. To enable the **view_independent** parameter, log in to the management console and click the cluster name. On the displayed **Cluster Details** page, click the **Parameters** tab, search for **view_independent**, modify the parameter, and save the modification.



Step 3 Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

Step 4 Create a sample table **t1** and insert data into the table.

```
SET current_schema='public';
CREATE TABLE t1 (a int, b int, c char(10)) DISTRIBUTE BY HASH (a);
INSERT INTO t1 VALUES(1,1,'a'),(2,2,'b');
```

Step 5 Create view **v1** that depends on table **t1**, and create view **v11** that depends on view **v1**. Query view **v11**.

```
CREATE VIEW v1 AS SELECT a, b FROM t1;
CREATE VIEW v11 AS SELECT a FROM v1;

SELECT * FROM v11;
 a
---
 1
 2
(2 rows)
```

Step 6 After table **t1** is deleted, an error is reported when you query the view **v11**. However, the views still exist.

GaussDB(DWS) provides the **GS_VIEW_INVALID** view to query all invalid views visible to the user. If the base table, function, or synonym that the view depends on is abnormal, the **validtype** column of the view is displayed as "invalid".

```
DROP TABLE t1;

SELECT * FROM v11;
ERROR:  relation "public.t1" does not exist

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |          definition          | validtype
--------+------------+----------+-----------+------------------------------+-----------
 213563 | public     | v1       | dbadmin   | SELECT a, b FROM public.t1; | invalid
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1;    | invalid
(2 rows)
```

Step 7 After the table **t1** is recreated in a cluster of a version earlier than 8.3.0, the view is automatically recreated. The views are automatically refreshed only when they are used.

```
CREATE TABLE t1 (a int, b int, c char(10)) DISTRIBUTE BY HASH (a);
INSERT INTO t1 VALUES(1,1,'a'),(2,2,'b');
```



```
SELECT * from v1;
 a | b
---+---
 1 | 1
 2 | 2
(2 rows)

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |        definition         | validtype
--------+------------+----------+-----------+---------------------------+-----------
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1; | invalid
(1 row)

SELECT * from v11;
 a
---
 1
 2
(2 rows)

SELECT * FROM gs_view_invalid;
 oid | schemaname | viewname | viewowner | definition | validtype
-----+------------+----------+-----------+------------+-----------
(0 rows)
```

Step 8 After the table t1 is recreated for a cluster of version 8.3.0 or later, the view is not automatically recreated. The view can be automatically refreshed only after the ALTER VIEW REBUILD operation is performed.

```
CREATE TABLE t1 (a int, b int, c char(10)) DISTRIBUTE BY HASH (a);
INSERT INTO t1 VALUES(1,1,'a'),(2,2,'b');

SELECT * from v1;
 a | b
---+---
 1 | 1
 2 | 2
(2 rows)

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |          definition          | validtype
--------+------------+----------+-----------+------------------------------+-----------
 213563 | public     | v1       | dbadmin   | SELECT a, b FROM public.t1; | invalid
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1;    | invalid
(1 row)

ALTER VIEW ONLY v1 REBUILD;

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |        definition         | validtype
--------+------------+----------+-----------+---------------------------+-----------
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1; | invalid
(1 rows)
```

----End

5.4 Best Practices for Using HStore Tables

Working Principles

In GaussDB(DWS), a CU is the smallest unit used to store data in a column-store table. By default, each column in the table stores 60,000 rows of data as a CU. Once generated, the data in a CU cannot be altered. A single CU is generated regardless of whether 1 or 60,000 records are inserted into a column-store table.

Frequent small data insertions hinder effective compression, leading to data expansion, which negatively impacts query performance and disk usage.

CU file data can only be appended, not modified. Deleting data marks it as invalid in the dictionary, while updating marks old data as deleted and writes new data to a new CU. Repeated updates or deletions cause space expansion and inefficient space usage.

The column-store Delta table addresses the issue of small CUs from frequent small data imports but does not resolve lock conflicts from concurrent updates on the same CU. A hybrid data warehouse needs to work with data sources, such as upstream databases or applications. Therefore, in real-time import scenarios, concurrent insert, update, and delete operations are necessary for timely data import and high query efficiency.

HStore tables use additional delta tables. Batch-inserted data is written directly to CUs, maintaining the compression benefits of column-store tables. Updated columns and small batch inserts are serialized, compressed, and periodically merged into primary table CUs.

Use Cases

GaussDB(DWS) uses column storage format for HStore tables to minimize disk usage, enable high-concurrency updates, and enhance query performance. HStore tables are ideal for scenarios that demand real-time data import and query capabilities, as well as the ability to process traditional TP transactions.

To enhance performance, GaussDB(DWS) 8.3.0.100 has optimized HStore tables and kept the old ones for compatibility purposes. The optimized tables are known as **HStore_opt** tables. HStore tables can be replaced by **HStore_opt** tables for better performance, except in scenarios requiring high performance without micro-batch updates.

Creating HStore Tables and Related Views

To create an HStore table, you need to specify the **enable_hstore** table-level parameter.

```
CREATE TABLE test1 (i int,j text) with (orientation = column,enable_hstore=on);
```

To create an **HStore_opt** table, you need to specify the **enable_hstore_opt** table-level parameter.

```
CREATE TABLE test2 (i int,j text) with (orientation = column,enable_hstore_opt=on);
```

Check the number of tuples in the Delta table and the expansion status of the Delta table in the view.

```
SELECT * FROM pgxc_get_hstore_delta_info('tableName');
```

Use functions to perform lightweight cleanup and full cleanup on the Delta table.

- After the lightweight merge accumulates 60,000 I records and deletion information on the CU, the level-4 lock ceases to hinder the addition, deletion, modification, and querying of services. Nevertheless, the space is not freed up for the system.

  ```
  select hstore_light_merge('tableName');
  ```

- Merging all records and truncating the Delta table is necessary to free up space for the system. Nonetheless, holding a level-8 lock will impede services.


```
select hstore_full_merge('tableName');
```

Insert 100 data records into the HStore table in batches. A record whose type is **I** (**n_i_tup** is 1) is generated.

```
CREATE TABLE data(a int primary key, b int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "data_pkey" for table "data"
CREATE TABLE

INSERT INTO data values(generate_series(1,100),1);
INSERT 0 100

CREATE TABLE hs(a int primary key, b int)with(orientation=column, enable_hstore=on);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "hs_pkey" for table "hs"
CREATE TABLE

INSERT INTO hs SELECT * FROM data;
INSERT 0 100

SELECT * FROM pgxc_get_hstore_delta_info('hs');
 node_name |      part_name      | live_tup | n_i_type | n_d_type | n_x_type | n_u_type | n_m_type | data_size
-----------+---------------------+----------+----------+----------+----------+----------+----------+-----------
 dn_1      | non partition table |        1 |        1 |        0 |        0 |        0 |        0 |      8192
(1 row)
```

After **hstore_full_merge** is executed, no tuple exists in the Delta table (the value of **live_tup** is **0**), and the value of **data_size** is **0**.

```
SELECT hstore_full_merge('hs');
 hstore_full_merge
-------------------
                 1
(1 row)

SELECT * FROM pgxc_get_hstore_delta_info('hs');
 node_name |      part_name      | live_tup | n_i_type | n_d_type | n_x_type | n_u_type | n_m_type | data_size
-----------+---------------------+----------+----------+----------+----------+----------+----------+-----------
 dn_1      | non partition table |        0 |        0 |        0 |        0 |        0 |        0 |         0
(1 row)
```

Perform the deletion. The Delta table contains a record whose type is **D** (**n_d_tup** is **1**).

```
DELETE hs where a = 1;
DELETE 1
SELECT * FROM pgxc_get_hstore_delta_info('hs');
 node_name |      part_name      | live_tup | n_i_type | n_d_type | n_x_type | n_u_type | n_m_type | data_size
-----------+---------------------+----------+----------+----------+----------+----------+----------+-----------
 dn_1      | non partition table |        1 |        0 |        1 |        0 |        0 |        0 |      8192
(1 row)
```

Usage Practice

For optimal performance of HStore tables, it is crucial to configure the following parameter settings:

Set **autovacuum_max_workers_hstore** to **3**, **autovacuum_max_workers** to **6**, **autovacuum** to **true**, and **enable_col_index_vacuum** to **on**.

1. **Concurrent update**

   Once a batch of data is inserted into a column-store table, two sessions are initiated. In session 1, a piece of data is deleted, and the transaction is not terminated.

   ```
   CREATE TABLE col(a int , b int)with(orientation=column);
   CREATE TABLE
   ```



```
INSERT INTO col select * from data;
INSERT 0 100

BEGIN;
BEGIN

DELETE col where a = 1;
DELETE 1
```

When session 2 attempts to delete more data, it becomes evident that session 2 can only proceed after session 1 is committed. This scenario imitates the CU lock issue in column storage.

```
BEGIN;
BEGIN
DELETE col where a = 2;
```

Repeat the previous experiment using the HStore table. Session 2 can be executed successfully without any lock wait.

```
BEGIN;
BEGIN
DELETE hs where a = 2;
DELETE 1
```

2. **Compression efficiency**

   Create a data table with 3 million data records.

```
CREATE TABLE data( a int, b bigint, c varchar(10), d varchar(10));

CREATE TABLE
INSERT INTO data values(generate_series(1,100),1,'asdfasdf','gergqer');
INSERT 0 100
INSERT INTO data select * from data;
INSERT 0 100
INSERT INTO data select * from data;
INSERT 0 200

---Insert data cyclically until the data volume reaches 3 million.

INSERT INTO data select * from data;
INSERT 0 1638400
select count(*) from data;
  count
---------
 3276800
(1 row)
```

   Import data to a row-store table in batches and check whether the size is 223 MB.

```
CREATE TABLE row (like data including all);
CREATE TABLE
INSERT INTO row select * from data;
INSERT 0 3276800
select pg_size_pretty(pg_relation_size('row'));
 pg_size_pretty
----------------
 223 MB
(1 row)
```

   Import data to a column-store table in batches and check whether the size is 3.5 MB.

```
CREATE TABLE hs(a int, b bigint, c varchar(10),d varchar(10))with(orientation= column,
enable_hstore=on);
CREATE TABLE
INSERT INTO hs select * from data;
INSERT 0 3276800
select pg_size_pretty(pg_relation_size('hs'));
 pg_size_pretty
----------------
 3568 KB
(1 row)
```


HStore tables have a good compression effect because of their simple table structure and duplicate data. They are usually compressed three to five times more than row-store tables.

3. **Batch query performance**

   It takes approximately four seconds to query the fourth column of the row-store table using the created table.

```
explain analyze select d from data;
explain analye                                                     QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------
  id |          operation           |        A-time        | A-rows  | E-rows  | Peak Memory  | E-memory | A-width | E-width | E-costs
 ----+-----------------------------+---------------------+---------+---------+--------------+----------+---------+---------+----------
   1 | ->  Streaming (type: GATHER) | 4337.881             | 3276800 | 3276800 | 32KB         |          |         |       8 | 61891.00
   2 |    ->  Seq Scan on data      | [1571.995, 1571.995] | 3276800 | 3276800 | [32KB, 32KB] | 1MB      |         |       8 | 61266.00
```

   It takes about 300 milliseconds to query the fourth column of the HStore table.

```
explain analyze select d from hs;
                                                              QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------------------
  id |               operation               |       A-time       | A-rows  | E-rows  |  Peak Memory   | E-memory | A-width | E-width | E-costs
 ----+---------------------------------------+--------------------+---------+---------+----------------+----------+---------+---------+----------
   1 | ->  Row Adapter                       | 335.280            | 3276800 | 3276800 | 24KB           |          |         |       8 | 15561.80
   2 |    ->  Vector Streaming (type: GATHER) | 111.492            | 3276800 | 3276800 | 96KB           |          |         |       8 | 15561.80
   3 |       ->  CStore Scan on hs           | [111.116, 111.116] | 3276800 | 3276800 | [254KB, 254KB] | 1MB      |         |       8 | 14936.80
```

   Only the batch query scenario has been tested, and in this scenario, the stored tables and HStore tables perform better than row-store tables in terms of query performance.

Requirements and Suggestions for Using HStore Tables

- **Configure the parameters.**

  To optimize query performance and compression efficiency, the background thread should perform **MERGE** clearing on HStore tables. Prior to utilizing HStore tables, ensure that the relevant GUC parameters are configured correctly. The suggested parameter values can be found below.

  autovacuum_max_workers_hstore: **3**

  autovacuum_max_workers: **6**

  autovacuum: **true**

  enable_col_index_vacuum: **on**

- **Suggestions on importing data to the database** (The **HStore_opt** table is recommended.)

  Suggestions for importing the **HStore_opt** table to the database:

  a. The performance of importing data using **UPDATE** is poor. You are advised to use **UPSERT** to import data.

b. When using **DELETE** to import data, use index scanning. The **JDBC batch** method is recommended.

c. When using **UPSERT** to import data, enable **enable_hstore_nonconflict_upsert_optimization** when there is no concurrency conflict and disable it in other scenarios. The optimal path is automatically selected.

d. Use **merge into** only when importing over one million data records per DN and there is no concurrent data to prevent duplicate data.

- **Point query suggestion** (The **HStore_opt** table is recommended.)

  Suggestions for querying the **HStore_opt** table:

  a. Create a level-2 partition on columns with evenly distributed distinct values and frequent equivalent filter criteria. Avoid level-2 partitions on columns with skewed or few distinct values.

  b. When dealing with fixed filter criteria columns (excluding level-2 partitions), use the **cbtree** index (up to 5 columns).

  c. When dealing with variable filter criteria columns (excluding level-2 partitions), use the **GIN** index (up to 5 columns).

  d. For all string columns involving equivalent filtering, **bitmap** indexes can be specified during table creation. The number of columns is not limited, but cannot be modified later.

  e. Specify columns that can be filtered by time range as the partition columns.

  f. If point queries return over 100,000 records per DN, index scanning may outperform non-index scanning. Use the GUC parameter enable_seqscan to compare performance.

- **Index-related**

  Indexes occupy extra space and offer limited point query performance improvement. Create primary key or B-tree indexes only for upsert or unique/near-unique point queries.

- **MERGE-related**

  HStore tables rely on background autovacuum for MERGE operations. Ensure data import speed does not exceed MERGE speed to prevent delta table expansion. Control import speed by managing concurrent data import tasks. Delta table space reuse is affected by oldestXmin. Old transactions can delay space reuse, causing expansion.

5.5 Best Practices of GIN Index

A GIN index is a data structure that pairs a key with its posting list. The key indicates a specific value, and the posting list tracks all the locations that this key occurs. For example, 'hello', '14:2 23:4' indicates that **hello** is found at the locations **14:2** and **23:4**. A GIN index efficiently locates tuples with specific keywords, making it ideal for searching elements within multi-valued fields. This section describes how to use GIN indexes to search through array and JSONB types, as well as how to conduct full-text searches.

Using a GIN Index to Search Through the Array Type

Create a GIN index to speed up tag searches.

Step 1 Create a cluster on the management console. For details, see **Creating a DWS 2.0 Cluster**.

Step 2 Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

Step 3 Create the **books** table. The **tags** column stores the tag information of **books** using the array type.

```
CREATE TABLE books (id SERIAL PRIMARY KEY, title VARCHAR(100), tags TEXT[]);
```

Step 4 Insert data.

```
INSERT INTO books (title, tags)
VALUES   ('Book 1', ARRAY['fiction', 'adventure']),
   ('Book 2', ARRAY['science', 'fiction']),
   ('Book 3', ARRAY['romance', 'fantasy']),
   ('Book 4', ARRAY['adventure']);
```

Step 5 Create a GIN index.

```
CREATE INDEX idx_books_tags_gin ON books USING GIN (tags);
```

Step 6 Use the GIN index to perform a search query to find books that contain a specific tag in the **tags** column. Search for books containing the tag "fiction":

```
SELECT * FROM books WHERE tags @> ARRAY['fiction'];
 id | title  |        tags
----+--------+---------------------
  1 | Book 1 | {fiction,adventure}
  2 | Book 2 | {science,fiction}
(2 rows)
```

Step 7 Use the GIN index to search for books that contain both the "fiction" and "adventure" tags.

```
SELECT * FROM books WHERE tags @> ARRAY['fiction', 'adventure'];
 id | title  |        tags
----+--------+---------------------
  1 | Book 1 | {fiction,adventure}
(1 row)
```

----End

Using a GIN Index to Search Through the JSONB Type

When using the JSONB type to store and query JSON data, you can use GIN indexes to improve query performance. GIN indexes are suitable for querying JSONB columns that contain a large number of different key-value pairs.

Step 1 Create the **my_table** table. The **data** column stores information about each person using the JSONB type.

```
CREATE TABLE my_table (id SERIAL PRIMARY KEY, data JSONB);
```

Step 2 Insert data.

```
INSERT INTO my_table (data)
 VALUES ('{"name": "John", "age": 30, "address": {"career": "announcer", "state": "NY"}}'),
```



```
    ('{"name": "Alice", "age": 25, "address": {"career": "architect", "state": "CA"}}'),
    ('{"name": "Bob", "age": 35, "address": {"career": "dentist", "state": "WA"}}');
```

Step 3 Create a GIN index to accelerate the query of JSONB columns.

```
CREATE INDEX my_table_data_gin_index ON my_table USING GIN (data);
```

Step 4 Use the GIN index to perform queries on JSONB columns. For example, search for a person whose occupation is dentist::

```
SELECT * FROM my_table WHERE data @> '{"address": {"career": "dentist"}}';
 id |                                 data
----+-------------------------------------------------------------------------
  3 | {"age": 35, "name": "Bob", "address": {"state": "WA", "career": "dentist"}}
(1 row)
```

Step 5 GIN indexes can also be queried on keys of JSONB columns. For example, search for people who are 30 years old or older:

```
SELECT * FROM my_table WHERE data ->> 'age' >= '30';
 id |                                 data
----+-------------------------------------------------------------------------
  3 | {"age": 35, "name": "Bob", "address": {"state": "WA", "career": "dentist"}}
  1 | {"age": 30, "name": "John", "address": {"state": "NY", "career": "announcer"}}
(2 rows)
```

----End

Using a GIN Index for Full-Text Search

When using GIN indexes for full-text search, you can use the tsvector and tsquery data types and related functions.

NOTE

To build a tsquery object, you need to use the **to_tsquery** function and provide the search criteria and the corresponding text search configuration (english in this case). Other text search functions and operators can also be used for more complex full-text searches, such as **plainto_tsquery** and **ts_rank**. The specific usage depends on your needs.

Step 1 Create an **articles** table in which the **content** column stores the article content.

```
CREATE TABLE articles (id SERIAL PRIMARY KEY,title VARCHAR(100),content TEXT);
```

Step 2 Insert data.

```
INSERT INTO articles (title, content)
 VALUES ('Article 1', 'This is the content of article 1.'),
   ('Article 2', 'Here is the content for article 2.'),
   ('Article 3', 'This article discusses various topics.'),
   ('Article 4', 'The content of the fourth article is different.');
```

Step 3 Creates an auxiliary column **tsvector** for the **content** column that stores the processed text indexes.

```
ALTER TABLE articles ADD COLUMN content_vector tsvector;
```

Step 4 Update the value in the **content_vector** column and convert the text in the **content** column to the tsvector type.

```
UPDATE articles SET content_vector = to_tsvector('english', content);
```

Step 5 Create a GIN index.

```
CREATE INDEX idx_articles_content_gin ON articles USING GIN (content_vector);
```

Step 6 Perform a full-text search, using the tsquery type to specify the search criteria. For example, search for an article that contains the word "content":


```
SELECT * FROM articles WHERE content_vector @@ to_tsquery('english', 'content');
```

----End

5.6 Encrypting and Decrypting Data Columns

Data encryption is widely used in various information systems as a technology to effectively prevent unauthorized access and prevent data leakage. As the core of the information system, the GaussDB(DWS) data warehouse also provides data encryption functions, including transparent encryption and encryption using SQL functions. This section describes SQL function encryption.

NOTE

Currently, GaussDB(DWS) does not support decrypting data encrypted in Oracle, Teradata, and MySQL databases. The encryption and decryption of Oracle, Teradata, and MySQL databases are different from those of GaussDB(DWS). GaussDB(DWS) can only decrypt unencrypted data migrated from Oracle, Teradata, and MySQL databases.

Background

- Hash Functions

  The hash function is also called the digest algorithm. It maps input data of an arbitrary length to an output of fixed length. For example, Hash(data)=result. This process is irreversible. That is, the hash function does not have an inverse function, and data cannot be obtained from the result. In scenarios where plaintext passwords should not be stored (passwords are sensitive) or known by system administrators, hash algorithms should be used to store one-way hash values of passwords.

  In actual use, salt values and iteration are added to prevent same hash values generated by same passwords, hence to prevent rainbow table attacks.

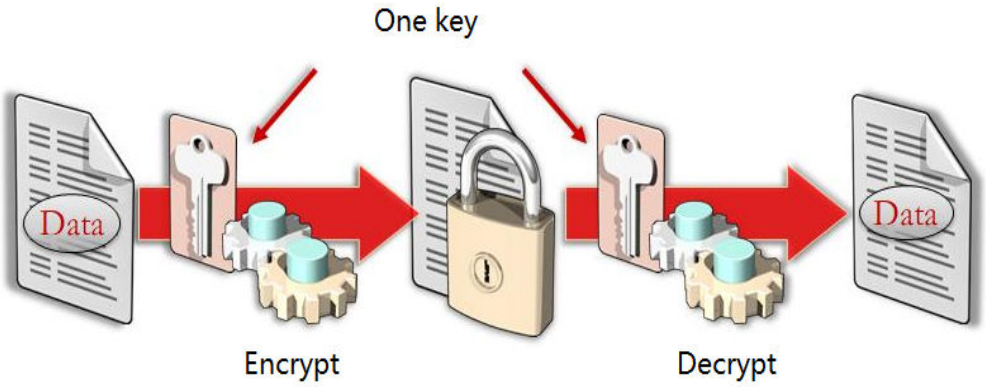
- Symmetric Encryption Algorithms

  Symmetric encryption algorithms use the same key to encrypt and decrypt data. There are two subcategories of symmetric encryption algorithms: block ciphers and stream ciphers.

  Block ciphers break the plaintext into fixed-length groups of bits known as blocks and Each block then gets encrypted as a unit. And if there's not enough data to completely fill a block, "padding" is then used to ensure that the blocks meet the fixed-length requirements. Due to padding, the length of the ciphertext obtained by block ciphers is greater than that of the plaintext.

  In stream ciphers, encryption and decryption parties use same pseudo-random encrypted data stream as keys, and plaintext data is sequentially encrypted by these keys. In practice, data is encrypted one bit at a time using an XOR operation. Stream cyphers do not need to be padded. Therefore the length of the obtained ciphertext is same as the length of the plaintext.

Figure 5-3 Symmetric encryption algorithms



Technical Details

GaussDB(DWS) provides hash functions and symmetric cryptographic algorithms to encrypt and decrypt data columns. Hash functions support sha256, sha384, sha512, and SM3. Symmetric cryptographic algorithms support AES128, AES192, AES256, and SM4.

- Hash Functions
  - md5(string)

    Use MD5 to encrypt string and return a hexadecimal value. MD5 is insecure and is not recommended.
  - gs_hash(hashstr, hashmethod)

    Obtains the digest string of a **hashstr** string based on the algorithm specified by **hashmethod**. **hashmethod** can be **sha256**, **sha384**, **sha512**, or **sm3**.
- Symmetric Encryption Algorithms
  - gs_encrypt(encryptstr, keystr, cryptotype, cryptomode, hashmethod)

    Encrypts an **encryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the encrypted string.
  - gs_decrypt(decryptstr, keystr, cryptotype, cryptomode, hashmethod)

    Decrypts a **decryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the decrypted string. The **keystr** used for decryption must be consistent with that used for encryption.
  - gs_encrypt_aes128(encryptstr, keystr)

    Encrypts **encryptstr** strings using **keystr** as the key and returns encrypted strings. The length of **keystr** ranges from 1 to 16 bytes.
  - gs_decrypt_aes128(decryptstr, keystr)

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

For more information about functions, see **Using Functions for Encryption and Decryption**.

Examples

Step 1 Connect to the database.

For details, see **Using the CLI to Connect to a GaussDB(DWS) Cluster**.

Step 2 Create the table **student** with the columns **id**, **name**, and **score**. Then use hash functions to encrypt and save names, and use symmetric cryptographic algorithms to save scores.

```
CREATE TABLE student (id int, name text, score text, subject text);

INSERT INTO student VALUES (1, gs_hash('alice', 'sha256'), gs_encrypt('95', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('math', '1234'));
INSERT INTO student VALUES (2, gs_hash('bob', 'sha256'), gs_encrypt('92', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('english', '1234'));
INSERT INTO student VALUES (3, gs_hash('peter', 'sha256'), gs_encrypt('98', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('science', '1234'));
```

Step 3 Query the table **student** without using keys. The query result shows that the encrypted data in the name and score columns cannot be viewed even if you have the **SELECT** permission.

```
SELECT * FROM student;
 id |                              name                              |
score                                                 |
                          subject
----+----------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------------------
+-----------
-------------------------------------------------------------------------------
  1 | 2bd806c97f0e00af1a1fc3328fa763a9269723c8db8fac4f93af71db186d6e90 | AAAAAAAAAABAuUC3VQ
+MvPCDAaTUySl1e2gGLr4/ATdCUjTEvova3cb/Ba3ZKqIn1yNVGEFBvJnTq/3sLF4//
Gm8qG7AyfNbbqdW3aYErLVpbE/QWFX9Ig== | aFEWQR2gkj
iu6sfsAad+dHzfFDHePZ6xd44zyekh+qVFlh9FODZ0DoaFAJXctwUsiqaiitTxW8cCSEaNjS/E7Ke1ruY=
  2 | 81b637d8fcd2c6da6359e6963113a1170de795e4b725b84d1e0b4cfd9ec58ce9 | AAAAAAAAAABAuUC3VQ
+MvPCDAaTUySl1taXxAoDqE793hgyCJvC0ESdAX5Mtgdq2LXI1f5ZxraQ73WIJVtIBX8oe3gTDxoXGlHbHht4kzM
4U8dOwr5rjgg== | aFEWQR2gkj
iu6sfsAad+dM8tPTDo/Pds6ZmqdmjGiKxf39+Wzx5NoQ6c8FrzihnRzgc0fycWSu5YGWNOKYWhRsE84Ac=
  3 | 026ad9b14a7453b7488daa0c6acbc258b1506f52c441c7c465474c1a564394ff |
AAAAAAAAAACnyusORPeApqMUgh56ucQu3uso/
Llw5MbPFMkOXuspEzhhnc9vErwOFe6cuGtx8muEyHCX7V5yXs+8FxhNh3n5L3419LDWJJLY2O4merHpSg== |
zomphRfHV4
H32hTtgkio1PyrobVO8N+hN7kAKwtygKP2E7Aaf1vsjmtLHcL88jyeJNe1lxe0fAvodzPJAxAuV3UJN4M=
(3 rows)
```

Step 4 Query the table **student** using keys. The query result shows that the data is decrypted by the function **gs_decrypt** (corresponding to **gs_encrypt**) and can be viewed.

```
SELECT id, gs_decrypt(score, '12345', 'aes128', 'cbc', 'sha256'),gs_decrypt_aes128(subject, '1234') FROM
student;
 id | gs_decrypt | gs_decrypt_aes128
----+------------+-------------------
  1 | 95         | math
  2 | 92         | english
  3 | 98         | science
(3 rows)
```

----End

5.7 Managing Data Permissions Through Views

This section describes how to use views to allow various users to access specific data within the same table, ensuring data permissions management and security.

Scenario

After connecting to a cluster as user **dbadmin**, create an example table **customer**.

```
CREATE TABLE customer (id bigserial NOT NULL, province_id bigint NOT NULL, user_info varchar, primary key (id)) DISTRIBUTE BY HASH(id);
```

Insert test data into the example table **customer**.

```
INSERT INTO customer(province_id,user_info) VALUES (1,'Alice'),(1,'Jack'),(2,'Jack'),(3,'Matu');
INSERT 0 4
```

Query the **customer** table.

```
SELECT * FROM customer;
 id | province_id | user_info
----+-------------+-----------
  3 |           2 | Jack
  1 |           1 | Alice
  2 |           1 | Jack
  4 |           3 | Matu
(4 rows)
```

Requirement: User **u1** can view only the data of province 1 (**province_id** = **1**), and user **u2** can view only the data of province 2 (**province_id** = **2**).

Implementation

You can create a view to meet the requirements in the preceding scenario. The procedure is as follows:

Step 1 After connecting to a cluster as user **dbadmin**, create views **v1** and **v2** for provinces 1 and 2 in **dbadmin** mode.

Run the **CREATE VIEW** statement to create view **v1** for querying the data of province 1.

```
CREATE VIEW v1 AS
   SELECT * FROM customer WHERE province_id=1;
```

Run the **CREATE VIEW** statement to create view **v2** for querying the data of province 2.

```
CREATE VIEW v2 AS
   SELECT * FROM customer WHERE province_id=2;
```

Step 2 Create users **u1** and **u2**.

```
CREATE USER u1 PASSWORD '*********';
CREATE USER u2 PASSWORD '*********';
```

Step 3 Run the **GRANT** statement to grant the data query permission to the target user.

Grant the permission on the target view schema to **u1** and **u2**.

```
GRANT USAGE ON schema dbadmin TO u1,u2;
```

Grant **u1** the permission to query data of province 1 in the **v1** view.


```
GRANT SELECT ON v1 TO u1;
```

Grant **u2** the permission to query data of province 2 in the **v2** view.

```
GRANT SELECT ON v2 TO u2;
```

----End

Verifying the Query Result

- Switch to **u1** to connect to the cluster.

  ```
  SET ROLE u1 PASSWORD '*********';
  ```

  Query the **v1** view. **u1** can query only the **v1** view data.

  ```
  SELECT * FROM dbadmin.v1;
   id | province_id | user_info
  ----+-------------+-----------
    1 |           1 | Alice
    2 |           1 | Jack
  (2 rows)
  ```

  If **u1** attempts to query data in view **v2**, the following error information is displayed:

  ```
  SELECT * FROM dbadmin.v2;
  ERROR:  SELECT permission denied to user "u1" for relation "dbadmin.v2"
  ```

  The result shows that user **u1** can view only the data of province 1 (**province_id** = **1**).

- Use **u2** to connect to the cluster.

  ```
  SET ROLE u2 PASSWORD '*********';
  ```

  Query the **v2** view. **u2** can query only the **v2** view data.

  ```
  SELECT * FROM dbadmin.v2;
   id | province_id | user_info
  ----+-------------+-----------
    3 |           2 | Jack
  (1 row)
  ```

  If **u2** attempts to query data in view **v1**, the following error information is displayed:

  ```
  SELECT * FROM dbadmin.v1;
  ERROR:  SELECT permission denied to user "u2" for relation "dbadmin.v1"
  ```

  The result shows that user **u2** can view only the data of province 2 (**province_id** = **2**).

6 Database Management

6.1 Role-based Access Control (RBAC)

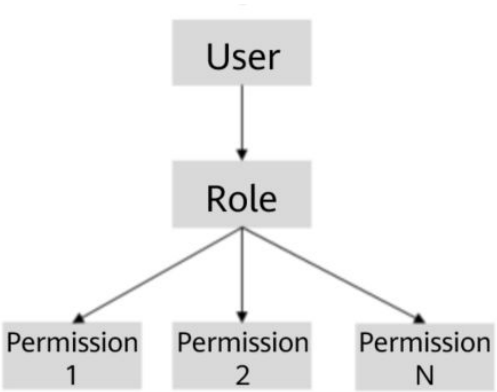
What is RBAC?

- Role-based access control (RBAC) is to grant permissions to roles and let users obtain permissions by associating with roles.
- A role is a set of permissions.
- RBAC greatly simplifies permissions management.

What is the RBAC Model?

Assign appropriate permissions to roles.

Associate users with the roles.



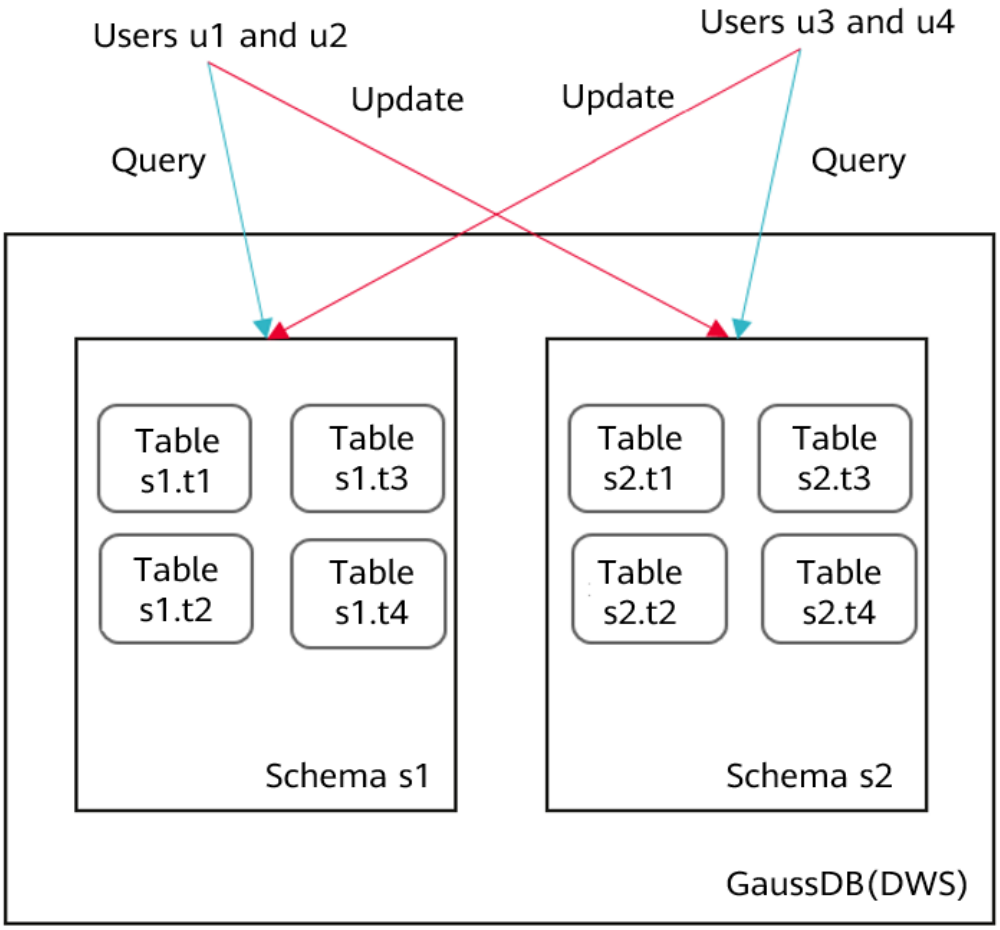
Scenarios

Assume there are two schemas, **s1** and **s2**.

There are two groups of users:

- Users **u1** and **u2** can query all the tables in **s1** and update all the tables in **s2**.

- Users **u3** and **u4** can query all the tables in **s2** and update all the tables in **s1**.



Granting Permissions

Step 1 Connect to the GaussDB(DWS) database as user **dbadmin**.

Step 2 Run the following statements to create schemas **s1** and **s2** and users **u1** to **u4**:

NOTE

Replace *{password}* with the actual password.

```
CREATE SCHEMA s1;
CREATE SCHEMA s2;
CREATE USER u1 PASSWORD '{password}';
CREATE USER u2 PASSWORD '{password}';
CREATE USER u3 PASSWORD '{password}';
CREATE USER u4 PASSWORD '{password}';
```

Step 3 Copy and run the following statements to create the **s1.t1** and **s2.t1** tables:

```
CREATE TABLE s1.t1 (c1 int, c2 int);
CREATE TABLE s2.t1 (c1 int, c2 int);
```

Step 4 Run the following statement to insert data to the tables:

```
INSERT INTO s1.t1 VALUES (1,2);
INSERT INTO s2.t1 VALUES (1,2);
```


Step 5 Run the following statements to create four roles, each having the query or update permission of table **s1** or **s2**:

```
CREATE ROLE rs1_select PASSWORD disable;  -- Permission to query s1
CREATE ROLE rs1_update PASSWORD disable; -- Permission to update s1
CREATE ROLE rs2_select PASSWORD disable; -- Permission to query s2
CREATE ROLE rs2_update PASSWORD disable; -- Permission to update s2
```

Step 6 Run the following statements to grant the access permissions of schemas **s1** and **s2** to the roles:

```
GRANT USAGE ON SCHEMA s1, s2 TO rs1_select, rs1_update,rs2_select, rs2_update;
```

Step 7 Run the following statements to grant specific permissions to the roles:

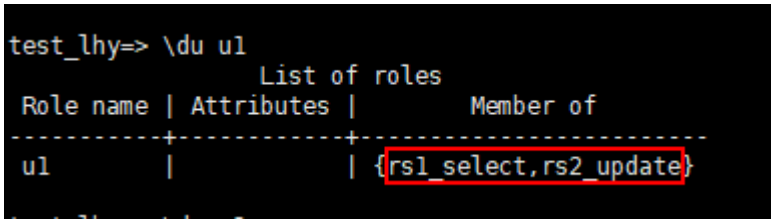
```
GRANT SELECT ON ALL TABLES IN SCHEMA s1 TO rs1_select; -- Grant the query permission on all the tables in s1 to the rs1_select role.
GRANT SELECT,UPDATE ON ALL TABLES IN SCHEMA s1 TO rs1_update;  -- Grant the query and update permissions on all the tables in s1 to the rs1_update role.
GRANT SELECT ON ALL TABLES IN SCHEMA s2 TO rs2_select;  -- Grant the query permission on all the tables in s2 to the rs2_select role.
GRANT SELECT,UPDATE ON ALL TABLES IN SCHEMA s2 TO rs2_update;  -- Grant the query and update permissions on all the tables in s2 to the rs2_update role.
```

Step 8 Run the following statements to grant roles to users:

```
GRANT rs1_select, rs2_update TO u1, u2;  -- Users u1 and u2 have the permissions to query s1 and update s2.
GRANT rs2_select, rs1_update TO u3, u4;  -- Users u3 and u4 have the permissions to query s2 and update s1.
```

Step 9 Run the following statement to view the role bound to a specific user:

```
\du u1;
```

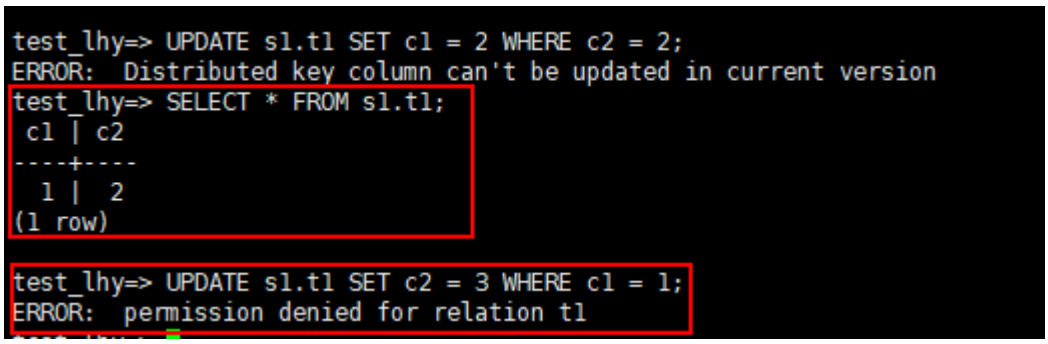


Step 10 Start another session. Connect to the database as user **u1**.

```
gsql -d gaussdb -h GaussDB(DWS)_EIP -U u1 -p 8000 -r -W {password};
```

Step 11 Run the following statements in the new session verify that user **u1** can query but cannot update **s1.t1**:

```
SELECT * FROM s1.t1;
UPDATE s1.t1 SET c2 = 3 WHERE c1 = 1;
```



Step 12 Run the following statements in the new session to verify that user **u1** can update **s2.t1**:

```
SELECT * FROM s2.t1;
UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;
```



```
test_lhy=> SELECT * FROM s2.t1;
 c1 | c2
----+----
  1 |  2
(1 row)

test_lhy=> UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;
UPDATE 1
```

----End

6.2 Configuring Read-Only Permissions

Context

If you need to assign different permissions to employees in your company to access your GaussDB(DWS) resources on Huawei Cloud, IAM is a good choice for fine-grained permissions management. IAM provides identity authentication, fine-grained permissions management, and access control. IAM helps you secure access to your cloud resources. You can use your cloud account to create IAM users, and assign permissions to the users to control their access to specific resources.

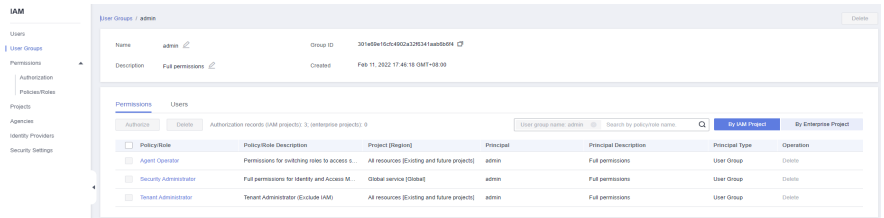
- **Scenario 1**: Allow software developers in your enterprise to use GaussDB(DWS) resources, but do not allow them to delete the resources or perform any high-risk operations. To this end, you can create IAM users for these developers and grant them only the permissions required for using GaussDB(DWS) resources.
- **Scenario 2**: Allow employees to use only GaussDB (DWS) resources, but not the resources of other services. To this end, grant them only the permissions for GaussDB(DWS).

You can use IAM to control cloud resource access and prevents misoperations on cloud resources. This section describes how to configure the read-only permission for an IAM user.

Tutorial 1: Read-Only Operations on IAM Project View

Step 1 **Create a user group and assign permissions** to it.

Use the Huawei Cloud account to log in to the **IAM console**, create a user group, and attach the **DWS ReadOnlyAccess** policy to the group.



Step 2 **Create a user and add it to a user group**.

Create a user on the IAM console and add the user to the group created in **Step 1**.

Step 3 Log in and verify permissions.

Log in to the console by using the user created and verify the user permissions.

- Choose **Service List** > **Data Warehouse Service** to access the GaussDB(DWS) console, and click **Create GaussDB(DWS) Cluster** to create a GaussDB(DWS) cluster. If you cannot create one, **DWS ReadOnlyAccess** has taken effect.
- Choose any other service in **Service List**. If only the **DWS ReadOnlyAccess** policy is added and a message is displayed indicating that you have insufficient permission to access the service, **DWS ReadOnlyAccess** has taken effect.

----End

Tutorial 2: Read-Only Operations in an Enterprise Project

Step 1 Create a user group and assign permissions to it.

Use the Huawei Cloud account to log in to the IAM console, create a user group, and attach the **DWS ReadOnlyAccess** policy to the group.

NOTE

- In the enterprise project view, the system still displays a message indicating that you lack the fine-grained permissions if you perform read-only operations irrelevant to resources. For example, fine-grained permissions related to events and alarms.

Step 2 Configure read-only permissions for events and alarms in the IAM project view.

1. Create the following custom policy **readonly_event_alarm**:

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "dws:alarm*:list*",
            "dws:cluster*:list*",
            "dws:dms*:get*",
            "dws:event*:list*"
         ]
      }
   ]
}
```

2. Log in to the IAM console and create a user group, and assign the newly created policy to the user group.

Step 3 Create a user and add it to a user group.

Create a user on the IAM console and add the user to the group created in **Step 1**.

Step 4 Log in and verify permissions.

Log in to the console by using the user created and verify the user permissions.

- Choose **Service List** > **Data Warehouse Service** to access the GaussDB(DWS) console, and click **Create GaussDB(DWS) Cluster** to create a GaussDB(DWS) cluster. If you cannot create one, **DWS ReadOnlyAccess** has taken effect.
- Choose any other service in **Service List**. If only the **DWS ReadOnlyAccess** policy is added and a message is displayed indicating that you have

insufficient permission to access the service, **DWS ReadOnlyAccess** has taken effect.

----End

6.3 Excellent Practices for SQL Queries

Based on a large number of SQL execution mechanisms and practices, we can optimize SQL statements following certain rules to more quickly execute SQL statements and obtain correct results.

- Replacing **UNION** with **UNION ALL**

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

- **Adding NOT NULL to the join column**

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

- Converting **NOT IN** to **NOT EXISTS**

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

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

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  The generated execution plan is as follows:

  Figure 6-1 NOT EXISTS execution plan

  ```
   id |                operation
  ----+-----------------------------------------
    1 | ->  Streaming (type: GATHER)
    2 |    ->  Hash Right Anti Join (3, 5)
    3 |       ->  Streaming(type: REDISTRIBUTE)
    4 |          ->  Seq Scan on t2
    5 |       ->  Hash
    6 |          ->  Seq Scan on t1

  Predicate Information (identified by plan id)
  ---------------------------------------------
    2 --Hash Right Anti Join (3, 5)
          Hash Cond: (t2.d2 = t1.c1)
  (13 rows)
  ```

- Use **hashagg**.

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

- Replace functions with **CASE** statements

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

- **Do not use functions or expressions for indexes.**

  Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.

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

- **Split complex SQL statements.**

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

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

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

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

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

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

6.4 Excellent Practices for Data Skew Queries

6.4.1 Real-Time Detection of Storage Skew During Data Import

During the import, the system collects statistics on the number of rows imported on each DN. After the import is complete, the system calculates the skew ratio. If the skew ratio exceeds the specified threshold, an alarm is generated immediately. The skew ratio is calculated as follows: Skew ratio = (Maximum number of rows imported on a DN – Minimum number of rows imported on a DN)/Number of imported rows. Currently, data can be imported only by running **INSERT** or **COPY**.

NOTE

enable_stream_operator must be set to **on** so that DNs can return the number of imported rows at a time when a plan is delivered to them. Then, the skew ratio is calculated on the CN based on the returned values.

Usage

1. Set parameters **table_skewness_warning_threshold** (threshold for triggering a table skew alarm) and **table_skewness_warning_rows** (minimum number of rows for triggering a table skew alarm).

- The value of **table_skewness_warning_threshold** ranges from **0** to **1**. The default value is **1**, indicating that the alarm is disabled. Other values indicate that the alarm is enabled.
- The value of **table_skewness_warning_rows** ranges from **0** to **2147483647**. The default value is **100,000**. The alarm is triggered only when the following condition is met: Total number of imported rows > Value of **table_skewness_warning_rows** x Number of DNs involving in the import.

```
show table_skewness_warning_threshold;
set table_skewness_warning_threshold = xxx;
show table_skewness_warning_rows;
set table_skewness_warning_rows = xxx;
```

2. Use **INSERT** or **COPY** to import data.
3. Detect and handle alarms. The alarm information includes the table name, minimum number of rows, maximum number of rows, total number of rows, average number of rows, skew rate, and prompt information about data distribution or parameter modification.

```
WARNING:  Skewness occurs, table name: xxx, min value: xxx, max value: xxx, sum value: xxx, avg value: xxx, skew ratio: xxx
HINT:  Please check data distribution or modify warning threshold
```

6.4.2 Quickly Locating the Tables That Cause Data Skew

Currently, the following skew query APIs are provided: **table_distribution(schemaname text, tablename text)**, **table_distribution()**, and **PGXC_GET_TABLE_SKEWNESS**. You can select one based on service requirements.

Scenario 1: Data Skew Caused by a Full Disk

First, use the **pg_stat_get_last_data_changed_time(oid)** function to query the tables whose data is changed recently. The last change time of a table is recorded only on the CN where **INSERT**, **UPDATE**, and **DELETE** operations are performed. Therefore, you need to query tables that are changed within the last day (the period can be changed in the function).

```
CREATE OR REPLACE FUNCTION get_last_changed_table(OUT schemaname text, OUT relname text)
RETURNS setof record
AS $$
DECLARE
row_data record;
row_name record;
query_str text;
query_str_nodes text;
BEGIN
query_str_nodes := 'SELECT node_name FROM pgxc_node where node_type = ''C''';
FOR row_name IN EXECUTE(query_str_nodes) LOOP
query_str := 'EXECUTE DIRECT ON (' || row_name.node_name || ') ''SELECT b.nspname,a.relname FROM
pg_class a INNER JOIN pg_namespace b on a.relnamespace = b.oid where
pg_stat_get_last_data_changed_time(a.oid) BETWEEN current_timestamp - 1 AND current_timestamp;''';
FOR row_data IN EXECUTE(query_str) LOOP
schemaname = row_data.nspname;
relname = row_data.relname;
return next;
END LOOP;
END LOOP;
return;
END; $$
LANGUAGE plpgsql;
```


Then, execute the **table_distribution(schemaname text, tablename text)** function to query the storage space occupied by the tables on each DN.

```
SELECT table_distribution(schemaname,relname) FROM get_last_changed_table();
```

Scenario 2: Routine Data Skew Inspection

- 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;
  ```
- If the number of tables in the database is no less than 10,000, you are advised to use the **table_distribution()** function instead of the **PGXC_GET_TABLE_SKEWNESS** view because the view takes a longer time (hours) due to the query of the entire database for skew columns. When you use the **table_distribution()** function, you can define the output based on **PGXC_GET_TABLE_SKEWNESS**, optimizing the calculation and reducing the 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;
  ```

Scenario 3: Querying Data Skew of a Table

Run the following SQL statement to query the data skew of a table. Replace **table_name** with the actual table name.

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

The following is an example of the information returned. If the data distribution deviation on each DN is less than 10%, data is evenly distributed. If it is greater than 10%, data skew occurs.

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
------+-----------
11010 | datanode4
10000 | datanode3
12001 | datanode2
 8995 | datanode1
10000 | datanode5
 7999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

6.5 Best Practices for User Management

A GaussDB(DWS) cluster mainly consists of system administrators and common users. This section describes the permissions of system administrators and common users and describes how to create users and query user information.

System Administrator

The user **dbadmin** created when you start a GaussDB(DWS) cluster is a system administrator. It has the highest system permission and can perform all operations, including operations on tablespaces, tables, indexes, schemas, functions, and custom views, as well as query for system catalogs and views.

To create a database administrator, connect to the database as an administrator and run the **CREATE USER** or **ALTER USER** statement with **SYSADMIN** specified.

Examples:

Create user **Jim** as a system administrator.

```
CREATE USER Jim WITH SYSADMIN password '{Password}';
```

Change user **Tom** to a system administrator. **ALTER USER** can be used only for existing users.

```
ALTER USER Tom SYSADMIN;
```

Common User

You can run the **CREATE USER** SQL statement to create a common user. A common user cannot create, modify, delete, or assign tablespaces, and needs to be assigned the permission for accessing tablespaces. A common user has all permissions for its own tables, schemas, functions, and custom views, creates indexes on its own tables, and queries only some system catalogs and views.

The database cluster has one or more named databases. Users are shared within the entire cluster, but their data is not shared.

Common user operations are as follows. Replace **password** with the actual password.

1. Creating a user
   ```
   CREATE USER Tom PASSWORD '{Password}';
   ```
2. Changing a user password

   Change the login password of user **Tom** from **password** to **newpassword**.
   ```
   ALTER USER Tom IDENTIFIED BY 'newpassword' REPLACE '{Password}';
   ```
3. Assigning permissions to a user
   - Add **CREATEDB** when you create a user that has the permission for creating a database.
     ```
     CREATE USER Tom CREATEDB PASSWORD '{Password}';
     ```
   - Add the **CREATEROLE** permission for a user.
     ```
     ALTER USER Tom CREATEROLE;
     ```
4. Revoking user permissions
   ```
   REVOKE ALL PRIVILEGES FROM Tom;
   ```
5. Locking or unlocking a user
   - Lock user **Tom**.
     ```
     ALTER USER Tom ACCOUNT LOCK;
     ```
   - Unlock user **Tom**.
     ```
     ALTER USER Tom ACCOUNT UNLOCK;
     ```
6. Deleting a user


```
DROP USER Tom CASCADE;
```

User Information Query

System views related to users, roles, and permissions include **ALL_USERS**, **PG_USER**, and **PG_ROLES**, and system catalogs include **PG_AUTHID** and **PG_AUTH_MEMBERS**.

- **ALL_USERS** displays all users in the database but does not show the details of them.
- **PG_USER** displays user information, including user IDs, the permission to create databases, and resource pools.
- **PG_ROLES** displays information about database roles.
- **PG_AUTHID** records information about database authentication identifiers (roles), including role permissions to log in or create databases.
- **PG_AUTH_MEMBERS** stores information of roles contained in a role group.

1. You can run **PG_USER** to query all users in the database. User ID (**USESYSID**) and permissions can also be queried.

```
SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelim
it
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+----------+----------+---------------+-------------
---
 Ruby    |       10 | t           | t        | t         | t       | ******** |          |          | default_pool |       0 |
|          |          |             |
 kim     |    21661 | f           | f        | f         | f       | ******** |          |          | default_pool |       0 |
|          |          |             |
 u3      |    22662 | f           | f        | f         | f       | ******** |          |          | default_pool |       0 |
|          |          |             |
 u1      |    22666 | f           | f        | f         | f       | ******** |          |          | default_pool |       0 |
|          |          |             |
 dbadmin |    16396 | f           | f        | f         | f       | ******** |          |          | default_pool |       0
|          |          |             |          |             |
 u5      |    58421 | f           | f        | f         | f       | ******** |          |          | default_pool |       0 |
|          |          |             |
(6 rows)
```

2. **ALL_USERS** displays all users in the database but does not show the details of them.

```
SELECT * FROM all_users;
 username | user_id
----------+---------
 Ruby     |      10
 manager  |   21649
 kim      |   21661
 u3       |   22662
 u1       |   22666
 u2       |   22802
 dbadmin  |   16396
 u5       |   58421
(8 rows)
```

3. **PG_ROLES** stores information about roles that have accessed the database.

```
SELECT * FROM pg_roles;
 rolname | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit | rolpassword | rolvalidbegin | rolv
aliduntil |  rolrespool  | rolparentid | roltabspace | rolconfig |  oid  | roluseft | rolkind | nodegroup |
roltempspace | rolspillspace
---------+----------+------------+---------------+-------------+--------------+-------------+----------------
+---------------+----------------+--------------+-------------+---------------+-----
----------+--------------+-------------+-------------+-----------+-------+----------+---------+-----------
```



```
+--------------+---------------
 Ruby    | t       | t        | t          | t         | t          | t         | t           | t          | t
|          -1 | ********    |            |
        | default_pool |         0 |          |         |   10 | t      | n      |         |          |
 manager | f       | t        | f          | f         | f          | f         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 21649 | f      | n      |         |          |
 kim     | f       | t        | f          | f         | f          | t         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 21661 | f      | n      |         |          |
 u3      | f       | t        | f          | f         | f          | t         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 22662 | f      | n      |         |          |
 u1      | f       | t        | f          | f         | f          | t         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 22666 | f      | n      |         |          |
 u2      | f       | t        | f          | f         | f          | f         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 22802 | f      | n      |         |          |
 dbadmin | f       | t        | f          | f         | f          | t         | f           | f          | t
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 16396 | f      | n      |         |          |
 u5      | f       | t        | f          | f         | f          | t         | f           | f          | f
|          -1 | ********    |            |
        | default_pool |         0 |          |         | 58421 | f      | n      |         |          |
(8 rows)
```

4. To view user properties, query the system catalog **PG_AUTHID**, which stores information about database authorization identifiers (roles). Each cluster, not each database, has only one **PG_AUTHID** system catalog. Only users with system administrator permissions can access the catalog.

```
SELECT * FROM pg_authid;
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit
|

rolpassword
                                        | rolvalidbegin | rolvaliduntil |  rolrespool  | roluseft | rolparentid |
roltabspace | rolkind | rolnodegroup | roltempspace | rolspillspace | rolexcpdata | rolauthinfo
----------+----------+------------+---------------+-------------+--------------+-------------+----------------
+---------------+----------------+--------------
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
------------------+--------------+--------------+-------------+----------+-------------+-------------+---------
+--------------+--------------+--------------+-------------+-------------
 Ruby    | t       | t        | t          | t         | t          | t         | t           | t          | t
|          -1 |
sha256366f1e665be208e6015bc3c5795d13e4dc297a148dca6c60346018c80e5c04c9ba170384ce44609b
31baa741f09a3ea5bedc7dadb906286ca994067c3fbf672dc08c981929e326ca08c005d8df942994e146ed
3302af47000b36e9852b50e39dmd585de11aafebd90ec620b201fc36f07a5ecdficefade3a1456ec0aca9a0
ee01e3bf2971d1dbafd604e596149e2e2928be4060dec2bd8688776588b4cd8c64fd38f1b0beab1603129f
a396556ba8aa4c7d6e137a04623 |              |              | default_pool | t        |          0 |           |
n       |           0 |            |              |             |
 sysadmin | f       | t        | f          | f         | f          | t         | f           | f          | t
|          -1 |
sha256ecaa7f0ca4436143af43074f16cdd825783ad1a5d659fd94f5e2fa5124e7da44045ecf40bda1a9797
5fcf5920dca0c8be375be5c71b51cb1eeeba0851fb3648cfa49f55989f83fd9baf1a9d5853ce19125f4fc29a7
c709c095ed02d00638410dmd556d6e2dcc41594dc7ad8ee909ef81637ecdficefadefd7d9704ee06affef958
1cd6a50a546607f88891198e96a5e84e7e83dccf56c5cd20a500bbc5248e8ea51f0bca70c5a8dcf00953f8b
62c7a181368153abce760 |              |              | default_pool | f        |          0 |           | n
|           |            |           |          |
 Tom     | f       | t        | f          | t         | f          | t         | f           | f          | f
|          -1 |
sha256f43c4f52ac51e297bc4dbdbc751fcf05319c15681dbf5a9c5777d2edce45cb592a948b25457a728e9
9a3e0608592f33b0a4312eba6124936522304ba298caa2002a04578860fecb0286d7c7baec09365eafd049
b2b99f74f21a08864dd7d3f2amd515ee49f0b18ef8e7d0cd27d91ce2fa9decdficefade16bab5f05b6d7c86a
19ae6406cc59c437506c3f6187bfdf3eefc7a7c7033afa076361b255cc8b6ccb6e19d4767effaec654b3308cc
72cebb891d00a4a10362da |              |              | default_pool | f        |          0 |           | n
```



```
|           |           |           |           |
(3 rows)
```

User Resource Query

1. Querying the resource quota and usage of all users

   ```
   SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
   ```

   Example of the resource usage of all users:

   ```
    username | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
   used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
   read_counts | write_counts | read_speed | write_speed
   ----------+-------------+--------------+----------+-----------+------------+-------------+-----------------
   +------------------+------------------+------------------+-------------+--------------+-------------+--------------
   +------------+-------------
   perfadm   |          0 |       17250 |        0 |         0 |         0 |          -1 |             0 |             -1
   |              0 |            -1 |          0 |           0 |          0 |           0 |          0 |           0
   usern     |          0 |       17250 |        0 |        48 |          0 |          -1 |             0 |             -1
   |              0 |            -1 |          0 |           0 |          0 |           0 |          0 |           0
   userg     |         34 |       15525 |    23.53 |        48 |          0 |          -1 |             0 |             -1 |
   814955731 |            -1 |     6111952 |     1145864 |      763994 |      143233 |      42678 |       8001
   userg1    |         34 |       13972 |    23.53 |        48 |          0 |          -1 |             0 |             -1 |
   814972419 |            -1 |     6111952 |     1145864 |      763994 |      143233 |      42710 |       8007
   (4 rows)
   ```

2. Querying the resource quota and usage of a specified user

   ```
   SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
   ```

   Example of the resource usage of user **Tom**:

   ```
   SELECT * FROM GS_WLM_USER_RESOURCE_INFO('Tom');
   userid | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
   used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
   read_counts | write_counts | read_speed | write_speed
   -------+-------------+--------------+----------+-----------+------------+-------------+-----------------
   +------------------+------------------+------------------+-------------+--------------+-------------+--------------
   +------------+-------------
    16523 |         18 |        2831 |        0 |        19 |          0 |          -1 |             0 |             -1
   |              0 |            -1 |          0 |           0 |          0 |           0 |          0 |           0
   (1 row)
   ```

3. Querying the I/O usage of a specified user

   ```
   SELECT * FROM pg_user_iostat('username');
   ```

   Example of the I/O usage of user **Tom**:

   ```
   SELECT * FROM pg_user_iostat('Tom');
   userid | min_curr_iops | max_curr_iops | min_peak_iops | max_peak_iops | io_limits | io_priority
   -------+---------------+---------------+---------------+---------------+-----------+-------------
    16523 |             0 |             0 |             0 |             0 |         0 | None
   (1 row)
   ```

6.6 Viewing Table and Database Information

Querying Table Information

- Querying information about all tables in a database using the **pg_tables** system catalog

  ```
  SELECT * FROM pg_tables;
  ```

- Querying the table structure using **\d+** command of the **gsql** tool.

  Example: Create a table **customer_t1** and insert data into the table.

  ```
  CREATE TABLE customer_t1
  (
      c_customer_sk             integer,
      c_customer_id             char(5),
      c_first_name              char(6),
      c_last_name               char(8)
  ```



```
)
with (orientation = column,compression=middle)
distribute by hash (c_last_name);
INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'map', 'Peter'),
    (4321, 'river', 'Lily'),
    (9527, 'world', 'James');
```

Query the table structure. If no schema is specified when you create a table, the schema of the table defaults to **public**.

```
\d+ customer_t1;
                        Table "public.customer_t1"
   Column     |     Type     | Modifiers | Storage  | Stats target | Description
---------------+--------------+-----------+----------+--------------+-------------
 c_customer_sk | integer      |           | plain    |              |
 c_customer_id | character(5) |           | extended |              |
 c_first_name  | character(6) |           | extended |              |
 c_last_name   | character(8) |           | extended |              |
Has OIDs: no
Distribute By: HASH(c_last_name)
Location Nodes: ALL DATANODES
Options: orientation=column, compression=middle, colversion=2.0, enable_delta=false
```

NOTE

The options may vary in different versions but the difference does not affect services. The options here are for reference only. The actual options are subject to the version.

- Use **pg_get_tabledef** to query the table definition.

```
SELECT * FROM PG_GET_TABLEDEF('customer_t1');
                              pg_get_tabledef
-------------------------------------------------------------------------------------
 SET search_path = tpchobs;                                                         +
 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)
```

- Querying all data in **customer_t1**

```
SELECT * FROM customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
         6885 | map           | Peter        |
         4321 | river         | Lily         |
         9527 | world         | James        |
(3 rows)
```

- Querying all data of a column in **customer_t1** using **SELECT**

```
SELECT c_customer_sk FROM customer_t1;
 c_customer_sk
---------------
         6885
         4321
         9527
(3 rows)
```

- Check whether a table has been analyzed. The time when the table was analyzed will be returned. If nothing is returned, it indicates that the table has not been analyzed.

```
SELECT pg_stat_get_last_analyze_time(oid),relname FROM pg_class where relkind='r';
```

Query the time when the **public** table was analyzed.

```
SELECT pg_stat_get_last_analyze_time(c.oid),c.relname FROM pg_class c LEFT JOIN pg_namespace n
ON c.relnamespace = n.oid WHERE c.relkind='r' AND n.nspname='public';
```



```
 pg_stat_get_last_analyze_time |       relname
-------------------------------+----------------------
 2022-05-17 07:48:26.923782+00 | warehouse_t19
 2022-05-17 07:48:26.964512+00 | emp
 2022-05-17 07:48:27.016709+00 | test_trigger_src_tbl
 2022-05-17 07:48:27.045385+00 | customer
 2022-05-17 07:48:27.062486+00 | warehouse_t1
 2022-05-17 07:48:27.114884+00 | customer_t1
 2022-05-17 07:48:27.172256+00 | product_info_input
 2022-05-17 07:48:27.197014+00 | tt1
 2022-05-17 07:48:27.212906+00 | timezone_test
(9 rows)
```

- Quickly query the column information of a table. If a view in **information_schema** has a large number of objects in the database, it takes a long time to return the result. You can run the following SQL statement to quickly query the column information of one or more tables:

```
SELECT /*+ set (enable_hashjoin off) */T.table_schema AS tableschema,
    T.TABLE_NAME AS tablename,
    T.dtd_identifier AS srcAttrId,
    COLUMN_NAME AS fieldName,
    'N' AS isPrimaryKey,
    nvl ( nvl ( T.character_maximum_length, T.numeric_precision ), 0 ) AS fieldLength,
    T.udt_name AS fieldType
from (
 SELECT  /*+ indexscan(co) indexscan(nco) indexscan(a) indexscan(t) leading((nc c a)) leading((co
nco)) indexscan(bt) indexscan(nt) */
    nc.nspname AS table_schema,
    c.relname AS table_name,
    a.attname AS column_name,
    information_schema._pg_char_max_length(information_schema._pg_truetypid(a.*, t.*),
information_schema._pg_truetypmod(a.*, t.*))::information_schema.cardinal_number AS
character_maximum_length,
    information_schema._pg_numeric_precision(information_schema._pg_truetypid(a.*, t.*),
information_schema._pg_truetypmod(a.*, t.*))::information_schema.cardinal_number AS
numeric_precision,
    COALESCE(bt.typname, t.typname)::information_schema.sql_identifier AS udt_name,
    a.attnum AS dtd_identifier
   FROM pg_attribute a
   LEFT JOIN pg_attrdef ad ON a.attrelid = ad.adrelid AND a.attnum = ad.adnum
   JOIN (pg_class c
   JOIN pg_namespace nc ON c.relnamespace = nc.oid) ON a.attrelid = c.oid
   JOIN (pg_type t
   JOIN pg_namespace nt ON t.typnamespace = nt.oid) ON a.atttypid = t.oid
   LEFT JOIN (pg_type bt
   JOIN pg_namespace nbt ON bt.typnamespace = nbt.oid) ON t.typtype = 'd'::"char" AND
t.typbasetype = bt.oid
   LEFT JOIN (pg_collation co
   JOIN pg_namespace nco ON co.collnamespace = nco.oid) ON a.attcollation = co.oid AND
(nco.nspname <> 'pg_catalog'::name OR co.collname <> 'default'::name)
  WHERE NOT pg_is_other_temp_schema(nc.oid) AND a.attnum > 0 AND NOT a.attisdropped AND
(c.relkind = ANY (ARRAY['r'::"char", 'v'::"char", 'f'::"char"])) AND (pg_has_role(c.relowner,
'USAGE'::text) OR has_column_privilege(c.oid, a.attnum, 'SELECT, INSERT, UPDATE, REFERENCES'::text))

) t
WHERE
    1 = 1
    AND UPPER ( T.TABLE_NAME ) <> 'DIS_USER_DATARIGHT_IF_SPLIT_T'
    AND UPPER ( T.TABLE_NAME ) NOT LIKE'DIS_TMP_%'
    AND UPPER ( T.COLUMN_NAME ) <> '_DISAPP_AUTO_ID_'
    AND (  ( T.TABLE_NAME ),  ( T.table_schema ) ) IN ( ( lower ( 'table_name' )::name, lower
( 'schema_name' )::name ) );
```

  Quickly query the column information of the **customer_t1** table.

```
SELECT /*+ set (enable_hashjoin off) */T.table_schema AS tableschema,
    T.TABLE_NAME AS tablename,
    T.dtd_identifier AS srcAttrId,
    COLUMN_NAME AS fieldName,
    'N' AS isPrimaryKey,
```



```
    nvl ( nvl ( T.character_maximum_length, T.numeric_precision ), 0 ) AS fieldLength,
    T.udt_name AS fieldType
from (
 SELECT  /*+ indexscan(co) indexscan(nco) indexscan(a) indexscan(t) leading((nc c a)) leading((co
nco)) indexscan(bt) indexscan(nt) */
    nc.nspname AS table_schema,
    c.relname AS table_name,
    a.attname AS column_name,
    information_schema._pg_char_max_length(information_schema._pg_truetypid(a.*, t.*),
information_schema._pg_truetypmod(a.*, t.*))::information_schema.cardinal_number AS
character_maximum_length,
    information_schema._pg_numeric_precision(information_schema._pg_truetypid(a.*, t.*),
information_schema._pg_truetypmod(a.*, t.*))::information_schema.cardinal_number AS
numeric_precision,
    COALESCE(bt.typname, t.typname)::information_schema.sql_identifier AS udt_name,
    a.attnum AS dtd_identifier
  FROM pg_attribute a
  LEFT JOIN pg_attrdef ad ON a.attrelid = ad.adrelid AND a.attnum = ad.adnum
  JOIN (pg_class c
  JOIN pg_namespace nc ON c.relnamespace = nc.oid) ON a.attrelid = c.oid
  JOIN (pg_type t
  JOIN pg_namespace nt ON t.typnamespace = nt.oid) ON a.atttypid = t.oid
  LEFT JOIN (pg_type bt
  JOIN pg_namespace nbt ON bt.typnamespace = nbt.oid) ON t.typtype = 'd'::"char" AND
t.typbasetype = bt.oid
  LEFT JOIN (pg_collation co
  JOIN pg_namespace nco ON co.collnamespace = nco.oid) ON a.attcollation = co.oid AND
(nco.nspname <> 'pg_catalog'::name OR co.collname <> 'default'::name)
  WHERE NOT pg_is_other_temp_schema(nc.oid) AND a.attnum > 0 AND NOT a.attisdropped AND
(c.relkind = ANY (ARRAY['r'::"char", 'v'::"char", 'f'::"char"])) AND (pg_has_role(c.relowner,
'USAGE'::text) OR has_column_privilege(c.oid, a.attnum, 'SELECT, INSERT, UPDATE, REFERENCES'::text))

) t
WHERE
    1 = 1
    AND UPPER ( T.TABLE_NAME ) <> 'DIS_USER_DATARIGHT_IF_SPLIT_T'
    AND UPPER ( T.TABLE_NAME ) NOT LIKE'DIS_TMP_%'
    AND UPPER ( T.COLUMN_NAME ) <> '_DISAPP_AUTO_ID_'
    AND (  ( T.TABLE_NAME ),  ( T.table_schema ) ) IN ( ( lower ( 'promotion' )::name, lower
( 'public' )::name ) );
```

- Obtain the table definition by querying audit logs.

  Use the **pgxc_query_audit** function to query audit logs of all CNs. The syntax is as follows:

  ```
  pgxc_query_audit(timestamptz startime,timestamptz endtime)
  ```

  Query the audit records of multiple objects.

  ```
  SET audit_object_name_format TO 'all';
  SELECT object_name,result,operation_type,command_text FROM pgxc_query_audit('2024-05-26
  8:00:00','2024-05-26 22:55:00') where command_text like '%student%';
  ```

Querying the Table Size

- Querying the total size of a table (indexes and data included)

  ```
  SELECT pg_size_pretty(pg_total_relation_size('<schemaname>.<tablename>'));
  ```

  Example:

  First, create an index on **customer_t1**.

  ```
  CREATE INDEX index1 ON customer_t1 USING btree(c_customer_sk);
  ```

  Then, query the size of table **customer_t1** of **public**.

  ```
  SELECT pg_size_pretty(pg_total_relation_size('public.customer_t1'));
   pg_size_pretty
  ----------------
   264 kB
  (1 row)
  ```


- Querying the size of a table (indexes excluded)

```
SELECT pg_size_pretty(pg_relation_size('<schemaname>.<tablename>'));
```

Example: Query the size of table **customer_t1** of **public**.

```
SELECT pg_size_pretty(pg_relation_size('public.customer_t1'));
 pg_size_pretty
----------------
 208 kB
(1 row)
```

- Query all the tables, ranked by their occupied space.

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '")) AS size FROM information_schema.tables
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '") DESC limit xx;
```

Example 1: Query the 15 tables that occupy the most space.

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '")) AS size FROM information_schema.tables
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '") DESC limit 15;
      table_full_name       |  size
----------------------------+---------
 pg_catalog.pg_attribute    | 2048 KB
 pg_catalog.pg_rewrite      | 1888 KB
 pg_catalog.pg_depend       | 1464 KB
 pg_catalog.pg_proc         | 1464 KB
 pg_catalog.pg_class        | 512 KB
 pg_catalog.pg_description | 504 KB
 pg_catalog.pg_collation    | 360 KB
 pg_catalog.pg_statistic    | 352 KB
 pg_catalog.pg_type         | 344 KB
 pg_catalog.pg_operator     | 224 KB
 pg_catalog.pg_amop         | 208 KB
 public.tt1                 | 160 KB
 pg_catalog.pg_amproc       | 120 KB
 pg_catalog.pg_index        | 120 KB
 pg_catalog.pg_constraint   | 112 KB
(15 rows)
```

Example 2: Query the top 20 tables with the largest space usage in the **public** schema.

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '")) AS size FROM information_schema.tables where
table_schema='public'
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '") DESC limit 20;
       table_full_name       |  size
-----------------------------+---------
 public.tt1                  | 160 KB
 public.product_info_input   | 112 KB
 public.customer_t1          | 96 KB
 public.warehouse_t19        | 48 KB
 public.emp                  | 32 KB
 public.customer             | 0 bytes
 public.test_trigger_src_tbl | 0 bytes
 public.warehouse_t1         | 0 bytes
(8 rows)
```

Quickly Querying the Space Occupied by All Tables in the Database

In a large cluster (8.1.3 or later) with a large amount of data (more than 1000 tables), you are advised to use the **pgxc_wlm_table_distribution_skewness** view to query all tables in the database. This view can be used to query the tablespace usage and data skew in the database. The unit of **total_size** and **avg_size** is byte.

```
SELECT *, pg_size_pretty(total_size) as tableSize FROM pgxc_wlm_table_distribution_skewness ORDER BY
total_size desc;
```



```
 schema_name |                table_name                | total_size | avg_size  | max_percent |
min_percent | skew_percent | tablesize
-------------+------------------------------------------+------------+-----------+-------------
+-------------+--------------+-----------
 public      | history_tbs_test_row_1                   |  804347904 | 134057984 |       18.02 |       15.63
|         7.53 | 767 MB
 public      | history_tbs_test_row_3                   |  402096128 |  67016021 |       18.30 |       15.60
|         8.90 | 383 MB
 public      | history_tbs_test_row_2                   |  401743872 |  66957312 |       18.01 |       15.01
|         7.47 | 383 MB
 public      | i_history_tbs_test_1                     |  325263360 |  54210560 |       17.90 |       15.50
|         6.90 | 310 MB
```

The query result shows that the **history_tbs_test_row_1** table occupies the largest space and data skew occurs.

CAUTION

1. The **pgxc_wlm_table_distribution_skewness** view can be queried only when the GUC parameter **use_workload_manager and enable_perm_space** is enabled. In earlier versions, you are advised to use the **table_distribution()** function to query the entire database. If only the size of a table is queried, the **table_distribution(schemaname text, tablename text)** function is recommended.
2. In 8.2.1 and later cluster versions, GaussDB(DWS) supports the **pgxc_wlm_table_distribution_skewness** view, which can be directly used for query.
3. In the 8.1.3 cluster version, you can use the following definition to create a view and then perform query:

```
CREATE OR REPLACE VIEW
pgxc_wlm_table_distribution_skewness AS
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,
(maxsize
- avgsize) * 100 AS skewsize
FROM
pg_catalog.gs_table_distribution()
GROUP
BY schemaname, tablename
)
SELECT
   schemaname AS schema_name,
   tablename AS table_name,
   totalsize AS total_size,
   avgsize::numeric(1000) AS avg_size,
   (
      CASE
         WHEN totalsize = 0 THEN 0.00
         ELSE (maxsize * 100 /
totalsize)::numeric(5, 2)
      END
   ) AS max_percent,
   (
      CASE
         WHEN totalsize = 0 THEN 0.00
         ELSE (minsize * 100 /
totalsize)::numeric(5, 2)
      END
   ) AS min_percent,
   (
      CASE
         WHEN totalsize = 0 THEN 0.00
         ELSE (skewsize /
maxsize)::numeric(5, 2)
      END
   ) AS skew_percent
FROM skew;
```


Querying Database Information

- Querying the database list using the **\l** meta-command of the **gsql** tool.

```
\l
                        List of databases
   Name    | Owner | Encoding  | Collate | Ctype | Access privileges
-----------+-------+-----------+---------+-------+-------------------
 gaussdb   | Ruby  | SQL_ASCII | C       | C     |
 template0 | Ruby  | SQL_ASCII | C       | C     | =c/Ruby          +
           |       |           |         |       | Ruby=CTc/Ruby
 template1 | Ruby  | SQL_ASCII | C       | C     | =c/Ruby          +
           |       |           |         |       | Ruby=CTc/Ruby
(3 rows)
```

  NOTE

  - If the parameters **LC_COLLATE** and **LC_CTYPE** are not specified during database installation, the default values of them are **C**.
  - If **LC_COLLATE** and **LC_CTYPE** are not specified during database creation, the sorting order and character classification of the template database are used by default.

    For details, see **CREATE DATABASE**.

- Querying the database list using the **pg_database** system catalog

```
SELECT datname FROM pg_database;
  datname
-----------
 template1
 template0
 gaussdb
(3 rows)
```

Querying the Database Size

Querying the size of databases

```
select datname,pg_size_pretty(pg_database_size(datname)) from pg_database;
```

Example:

```
select datname,pg_size_pretty(pg_database_size(datname)) from pg_database;
  datname  | pg_size_pretty
-----------+----------------
 template1 | 61 MB
 template0 | 61 MB
 postgres  | 320 MB
(3 rows)
```

Querying the Size of a Table and the Size of the Corresponding Index in a Specified Schema

```
SELECT
    t.tablename,
    indexname,
    c.reltuples AS num_rows,
    pg_size_pretty(pg_relation_size(quote_ident(t.tablename)::text)) AS table_size,
    pg_size_pretty(pg_relation_size(quote_ident(indexrelname)::text)) AS index_size,
    CASE WHEN indisunique THEN 'Y'
       ELSE 'N'
    END AS UNIQUE,
    idx_scan AS number_of_scans,
    idx_tup_read AS tuples_read,
    idx_tup_fetch AS tuples_fetched
FROM pg_tables t
LEFT OUTER JOIN pg_class c ON t.tablename=c.relname
LEFT OUTER JOIN
    ( SELECT c.relname AS ctablename, ipg.relname AS indexname, x.indnatts AS number_of_columns,
```



```
idx_scan, idx_tup_read, idx_tup_fetch, indexrelname, indisunique FROM pg_index x
          JOIN pg_class c ON c.oid = x.indrelid
          JOIN pg_class ipg ON ipg.oid = x.indexrelid
          JOIN pg_stat_all_indexes psai ON x.indexrelid = psai.indexrelid )
    AS foo
    ON t.tablename = foo.ctablename
WHERE t.schemaname='public'
ORDER BY 1,2;
```

6.7 Best Practices of Database SEQUENCE

A sequence is a database object that generates unique integers. A sequence's value automatically adjusts according to certain rules. Typically, sequences serve as primary keys. In GaussDB(DWS), when a sequence is created, a metadata table with the same name is created to record sequence information. For example:

```
CREATE SEQUENCE seq_test;
CREATE SEQUENCE

SELECT * FROM seq_test;
 sequence_name | last_value | start_value | increment_by |      max_value      | min_value | cache_value |
log_cnt | is_cycled | is_called |  uuid
---------------+------------+-------------+--------------+---------------------+-----------+-------------+---------
+-----------+-----------+---------
 seq_test      |         -1 |           1 |            1 | 9223372036854775807 |         1 |           1 |       0 | f         |
f         | 1400050
(1 row)
```

In the preceding information:

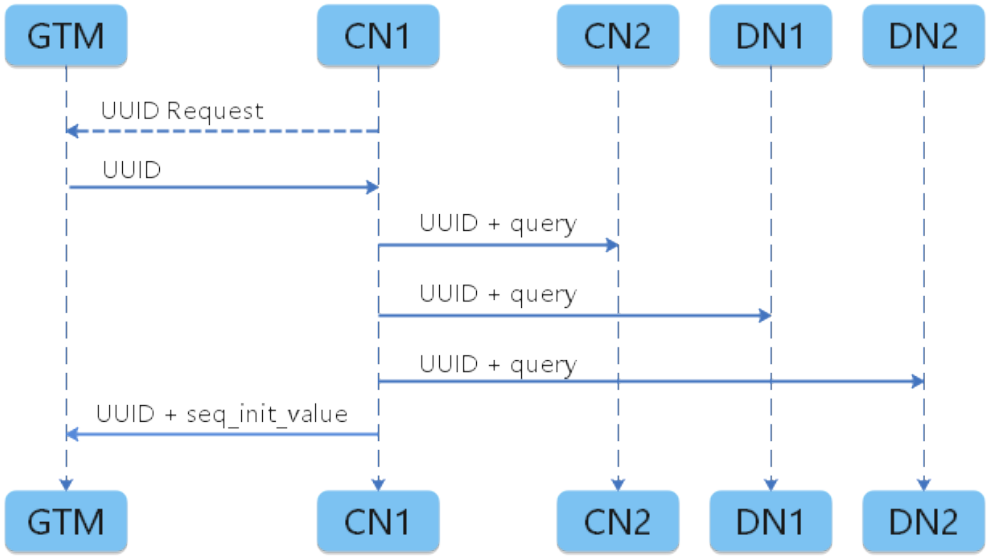
- **sequence_name** indicates the name of the sequence.
- **last_value** is meaningless.
- **start_value** indicates the initial value of the sequence.
- **increment_by** indicates the step of the sequence.
- **max_value** indicates the maximum value of the sequence.
- **min_value** indicates the minimum sequence value.
- **cache_value** determines how many sequence values are preloaded for rapid access to subsequent values. (After this cache is set, the continuity of sequence values cannot be ensured, and unacknowledged sequences may be generated, causing waste of sequences.)
- **log_cnt** indicates the number of sequence values recorded in WAL logs. In GaussDB(DWS), sequences are obtained and managed from GTM. Therefore, **log_cnt** is meaningless.
- **is_cycled** indicates whether to continue the loop after the sequence reaches the minimum or maximum value.
- **is_called** indicates whether the sequence has been called. (It only indicates whether the sequence has been called on the current instance. For example, after the sequence is called on cn1, the value of the filed on cn1 changes to **t**, and the value of the field on cn2 is still **f**.)
- **uuid** indicates the unique ID of the sequence.

Creating a Sequence

In GaussDB(DWS), the Global Transaction Manager (GTM) generates and maintains the global unique information about a transaction, such as the global

transaction ID, transaction snapshot, and sequence. The following figure shows the process of creating a sequence in GaussDB(DWS).

Figure 6-2 Creating a sequence



The specific process is as follows:

1. The CN that receives the SQL command applies for a UUID from the GTM.
2. The GTM returns a UUID.
3. The CN binds the obtained UUID to the sequenceName created by the user.
4. The CN delivers the binding relationship to other nodes, and other nodes create the sequence metadata table synchronously.
5. The CN sends the UUID and startID of the sequence to the GTM for permanent storage.

Therefore, sequence maintenance and request are actually completed on the GTM. **When requesting nextval, each instance obtains a sequence value from the GTM using the sequence's UUID. The number of values requested correlates with the cache size. An instance will only request a new sequence value from the GTM once its cache is depleted. Thus, enlarging the sequence's cache minimizes the communication frequency between the CN/DN and the GTM.**

Two Methods of Creating a Sequence

Method 1: Run the **CREATE SEQUENCE** statement to create a sequence and use nextval to invoke the sequence in the new table.

```
CREATE SEQUENCE seq_test increment by 1 minvalue 1 no maxvalue start with 1;
CREATE SEQUENCE

CREATE TABLE table_1(id int not null default nextval('seq_test'), name text);
CREATE TABLE
```

Method 2: If the serial type is used during table creation, a sequence is automatically created and the default value of the column is set to **nextval**.

```
CREATE TABLE mytable(a int, b serial) distribute by hash(a);
NOTICE:  CREATE TABLE will create implicit sequence "mytable_b_seq" for serial column "mytable.b"
```



```
CREATE TABLE

 \d+ mytable
                                    Table "dbadmin.mytable"
 Column |  Type   |                    Modifiers                     | Storage | Stats target | Description
--------+---------+--------------------------------------------------+---------+--------------+-------------
 a      | integer |                                                  | plain   |              |
 b      | integer | not null default nextval('mytable_b_seq'::regclass) | plain   |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
```

In this example, a sequence named **mytable_b_seq** is automatically created. Technically speaking, the serial type is not an actual data type but rather a method for assigning a unique identifier to a table column. Creating a serial involves generating a linked sequence for that specific column.

It is equivalent to the following statements:

```
CREATE TABLE mytable01(a int, b int) distribute by hash(a);
CREATE TABLE

CREATE SEQUENCE mytable01_b_seq owned by mytable.b;
CREATE SEQUENCE

ALTER SEQUENCE mytable01_b_seq owner to u1; --u1 is the owner of the mytable01 table. The owner does not need to run this statement.
ALTER SEQUENCE

ALTER TABLE mytable01 alter b set default nextval('mytable01_b_seq'), alter b set not null;
ALTER TABLE

\d+ mytable01
                                    Table "dbadmin.mytable01"
 Column |  Type   |                    Modifiers                     | Storage | Stats target | Description
--------+---------+--------------------------------------------------+---------+--------------+-------------
 a      | integer |                                                  | plain   |              |
 b      | integer | not null default nextval('mytable01_b_seq'::regclass) | plain   |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
```

Common Usage of Sequences in Services

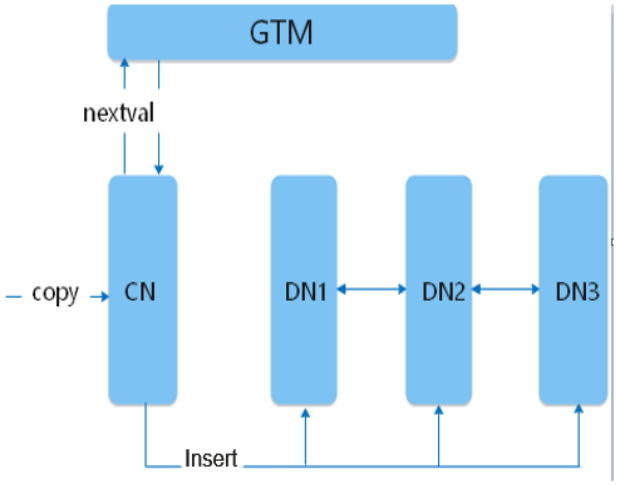
Sequences are commonly used to generate primary keys or unique columns during data import, a frequent practice in data migration scenarios. Different migration tools or service import scenarios use different import methods. Common import methods are classified into **copy** and **insert**. For sequences, the processing in the two scenarios is slightly different.

- **Scenario 1: Insert pushdown**

  ```
  CREATE TABLE test1(a int, b serial) distribute by hash(a);
  NOTICE:  CREATE TABLE will create implicit sequence "test1_b_seq" for serial column "test1.b"
  CREATE TABLE

  CREATE TABLE test2(a int) distribute by hash(a);
  CREATE TABLE

  EXPLAIN VERBOSE INSERT INTO test1(a) SELECT a FROM test2;
                                     QUERY PLAN
  ------------------------------------------------------------------------------------------------
   id |             operation              | E-rows | E-distinct | E-memory | E-width | E-costs
  ----+------------------------------------+--------+------------+----------+---------+---------
    1 | ->  Streaming (type: GATHER)       |      1 |            |          |       4 | 16.34
  ```

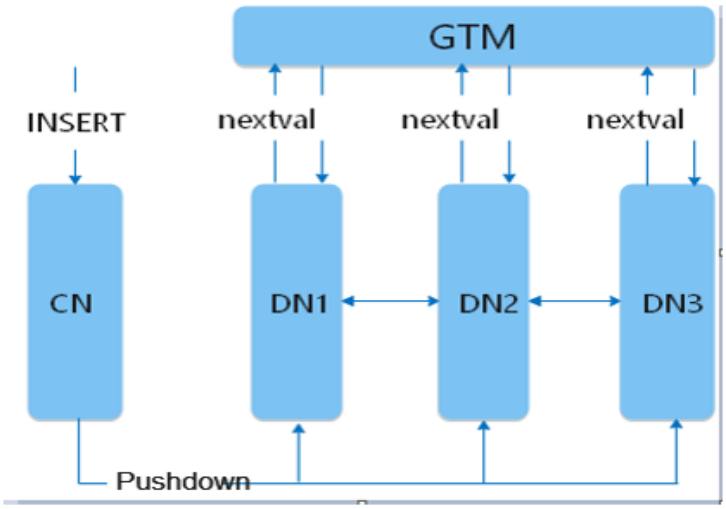
```
 2 |    ->  Insert on dbadmin.test1      |     30 |           |          |      4 | 16.22
 3 |       ->  Seq Scan on dbadmin.test2 |     30 |           | 1MB      |      4 | 14.21

          RunTime Analyze Information
-----------------------------------------------------
       "dbadmin.test2" runtime: 9.586ms, sync stats

    Targetlist Information (identified by plan id)
-----------------------------------------------------------
 1 --Streaming (type: GATHER)
       Node/s: All datanodes
 3 --Seq Scan on dbadmin.test2
       Output: test2.a, nextval('test1_b_seq'::regclass)
       Distribute Key: test2.a

 ====== Query Summary =====
------------------------------
System available mem: 1351680KB
Query Max mem: 1351680KB
Query estimated mem: 1024KB
Parser runtime: 0.076 ms
Planner runtime: 12.666 ms
Unique SQL Id: 831364267
(26 rows)
```

During an INSERT operation, nextval is executed on the DNs. This occurs whether nextval is called with its default value or invoked explicitly. The execution plan confirms that nextval operates at the sequence layer on the DNs. In this scenario, DNs obtain sequence values directly from the GTM and execute the request simultaneously, resulting in a relatively high level of efficiency.



- **Scenario 2: Copy scenario**

  In service development, alongside the INSERT method, the COPY method is also for data import into the database. It allows for the direct copying of file contents or using the CopyManager interface for this purpose. Moreover, the CDM data synchronization tool facilitates batch data import by copying. If the target table to be copied uses the default value **nextval**, the process is as follows.



In the copy process, the CN requests sequence values from the GTM. If the sequence's cache size is too small, the CN must repeatedly connect with the GTM to request nextval, which can lead to a performance bottleneck. **Typical Optimization Scenarios Related to Sequences** describes the service performance in this scenario and provides optimization methods.

Typical Optimization Scenarios Related to Sequences

Service scenarios:

Service scenario: In a service scenario, the CDM data synchronization tool is used to transfer data and import data from the source to the target GaussDB(DWS). Despite changing the CDM concurrency from 1 to 5, the synchronization rate remains unchanged, and there is a significant difference between the import rate and the expected value. Apart from data copying, all other services run smoothly without any performance or resource issues. Thus, it is likely that a bottleneck exists within the service. You are advised to review the job queue specifically for the COPY operation.

```
▒▒▒=# select * from pgxc_thread_wait_status where query_id in (2172986873834471 30,2172986873834757 07,2172986873839563 17,2172986873839623 87,2172986873840266 48) order by query_id;
  node_name   | db_name | thread_name |      query_id       |       tid       |  lwtid  | ptid | tlevel | smpid |     wait_status      |     wait_event
--------------+---------+-------------+---------------------+-----------------+---------+------+--------+-------+----------------------+---------------------
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873834471 30 | 281459614074928 | 2490718 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873834471 30 | 281459123161136 |  211948 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873834471 30 | 281460010731568 | 3913471 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873834471 30 | 281459064444976 | 2974031 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873834471 30 | 281454976845872 |  211939 |      |      0 |     0 | gtm get sequence val |
 cn_5003      | kfpt    |             | 2172986873834757 07 | 281454960064560 |  211941 |      |      0 |     0 | acquire lwlock       | PgStatObjectLock
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873834757 07 | 281459882559536 | 2490701 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873834757 07 | 281460079614000 |  211844 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873834757 07 | 281458634766384 | 3913447 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873834757 07 | 281460012533808 | 2973982 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873839563 17 | 281458383046704 | 3913477 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873839563 17 | 281454926501936 |  211943 |      |      0 |     0 | acquire lwlock       | PgStatObjectLock
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873839563 17 | 281458955372592 |  211952 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873839563 17 | 281459345590320 | 2490722 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873839563 17 | 281458762401840 | 2974037 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873839623 87 | 281459139942448 |  211949 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873839623 87 | 281454943283248 |  211942 |      |      0 |     0 | acquire lwlock       | PgStatObjectLock
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873839623 87 | 281458433390640 | 3913473 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873839623 87 | 281459412715568 | 2490719 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873839623 87 | 281458795964464 | 2974033 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873840266 48 | 281459395934256 | 2490721 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873840266 48 | 281459089598512 |  211951 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873840266 48 | 281458779183152 | 2974036 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873840266 48 | 281454993627184 |  211940 |      |      0 |     0 | acquire lwlock       | PgStatObjectLock
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873840266 48 | 281458399828016 | 3913476 |      |      0 |     0 | wait cmd             |
(25 rows)
```

As shown in the preceding figure, five CDM jobs are executed concurrently. You can see five COPY statements in the active view. Check the waiting view based on **query_id** corresponding to the five COPY statements. Out of the five COPY operations, only one requests a sequence value from the GTM concurrently, while the rests wait for a lightweight lock. As a result, enabling five concurrent jobs does not substantially enhance performance compared to just running a single job.

Causes:

The serial type is used when the target table is created. By default, the cache of the created sequence is 1. As a result, when data is concurrently copied to the database, the CN frequently establishes connections with the GTM, and lightweight lock contention exists between multiple concurrent jobs, resulting in low data synchronization efficiency.

Solutions:

In this scenario, increase the cache value of the sequence to prevent bottlenecks caused by frequent GTM connection establishment. In this service scenario example, about 100,000 data records are synchronized each time. Based on service evaluation, change the cache value to 10,000. (In practice, set a proper cache value based on services to ensure quick access and avoid sequence number waste.)

In cluster versions 8.2.1.100 and later, you can use **ALTER SEQUENCE** to change the cache value.

GaussDB(DWS) clusters of version 8.2.1 or earlier do not allow for the modification of cache values through **ALTER SEQUENCE**. To change the cache value of an existing sequence, follow these steps (the **mytable** table is used as an example):

Step 1 Remove the association between the current sequence and the target table.

```
ALTER SEQUENCE mytable_b_seq owned by none;
ALTER TABLE mytable alter b drop default;
```

Step 2 Record the current sequence value as the start value of the new sequence.

```
SELECT nextval('mytable_b_seq');
```

Delete the sequence.

```
DROP SEQUENCE mytable_b_seq;
```

Step 3 Create a sequence and bind it to the target table. Replace **xxx** with the value of nextval obtained in the previous step.

```
CREATE SEQUENCE mytable_b_seq START with xxx cache 10000 owned by mytable.b;
ALTER SEQUENCE mytable_b_seq owner to u1;--u1 is the owner of the mytable table. The owner does not need to run this statement.
ALTER TABLE mytable alter b set default nextval('mytable_b_seq');
```

----End

7 Performance Tuning

7.1 Optimizing Table Structure Design to Enhance GaussDB(DWS) Query Performance

7.1.1 Before Optimization: Learning Table Structure Design

In this practice, you will learn how to optimize the design of your tables. You will start by creating tables without specifying their storage mode, distribution key, distribution mode, or compression mode. Load test data into these tables and test system performance. Then, follow excellent practices to create the tables again using new storage modes, distribution keys, distribution modes, and compression modes. Load the test data and test performance again. Compare the two test results to find out how table design affects the storage space, and the loading and query performance of the tables.

Before you optimize a table, you need to understand the structure of the table. During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

This section describes how to optimize table performance in GaussDB(DWS) by properly designing the table structure (for example, by selecting the table model, table storage mode, compression level, distribution mode, distribution column, partitioned tables, and local clustering).

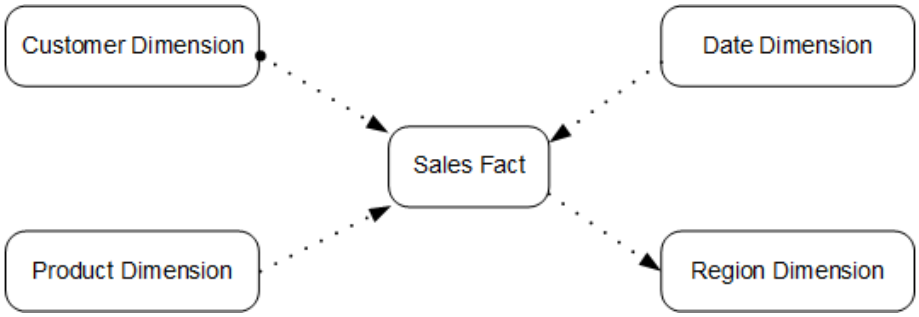
Selecting a Table Model

The most common types of data warehouse table models are star and snowflake models. Consider service and performance requirements when you choose a model for your tables.

- In the **star model**, a central fact table contains the core data for the database and several dimension tables provide descriptive attribute information for the fact table. The primary key of a dimension table associates a foreign key in a fact table, as shown in **Figure 7-1**.

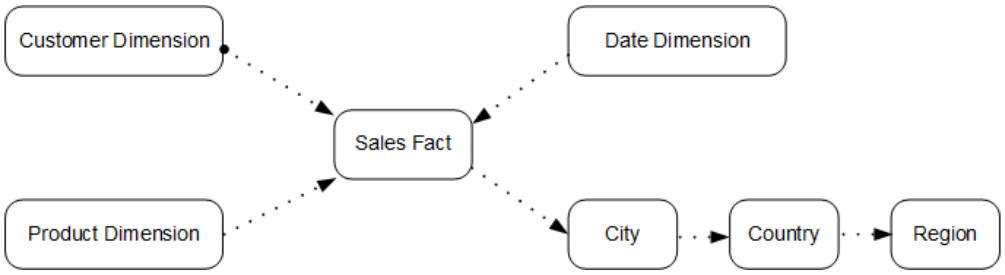
  - All facts must have the same granularity.
  - Different dimensions are not associated.

Figure 7-1 Star model



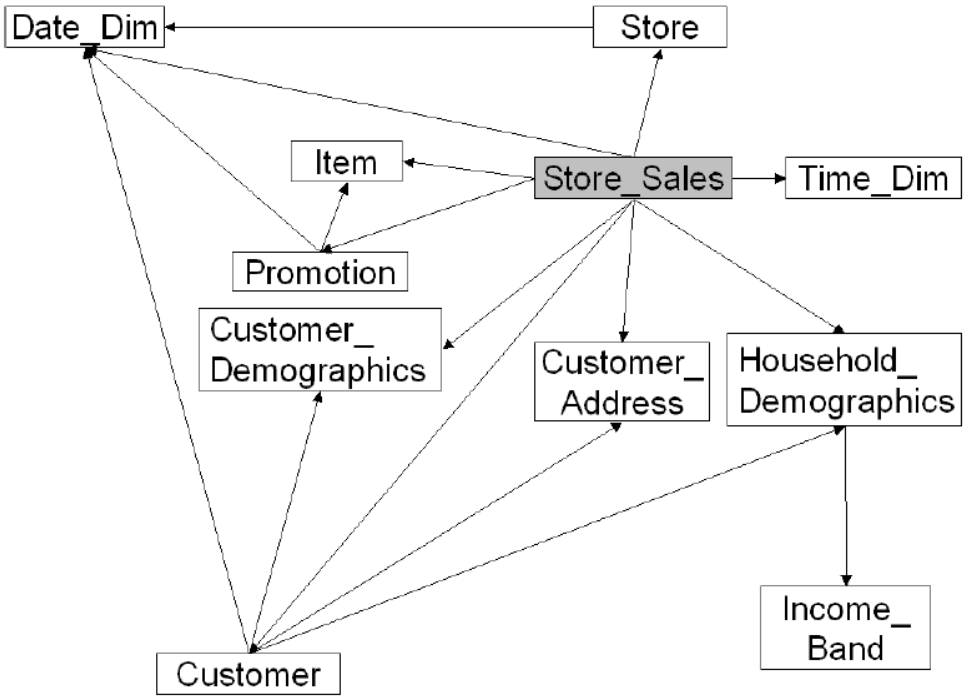
- The **snowflake model** is developed based on the star model. In this model, each dimension can be associated with multiple dimensions and split into tables of different granularities based on the dimension level, as shown in **Figure 7-2**.
  - Dimension tables can be associated as needed, and the data stored in them is reduced.
  - This model has more dimension tables to maintain than the star schema does.

Figure 7-2 Snowflake model



This practice verifies performance using the Store Sales (SS) model of TPC-DS. The model uses the snowflake model. **Figure 7-3** illustrates its structure.

Figure 7-3 TPC-DS store sales ER-Diagram



For details about the **store_sales** fact table and dimension tables in the model, see the official document of TPC-DS at **http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp**.

Selecting a Storage Mode

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

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

Storage Model	Application Scenario
Row storage	Point query (simple index–based query that returns only a few records). Query involving many **INSERT**, **UPDATE**, and **DELETE** operations.
Column storage	Statistical analysis queries. Queries with many groups and joins.

The row/column storage of a table is specified by the **orientation** attribute in the table definition. The value **row** indicates a row-store table and **column** indicates a column-store table. The default value is **row**.

Table Compression

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

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

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

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

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

Selecting a Distribution Mode

GaussDB(DWS) supports the following distribution modes: replication, hash, and Round-robin.

NOTE

Round-robin is supported in cluster 8.1.2 and later.

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

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

Selecting a Distribution Key

If the hash distribution mode is used, a distribution key must be specified for the user table. If a record is inserted, the system performs hash computing based on values in the distribute column and then stores data on the related DN.

Select a hash distribution key based on the following principles:

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

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

   ```
   SELECT
   xc_node_id, count(1)
   FROM tablename
   ```



```
GROUP BY xc_node_id
ORDER BY xc_node_id desc;
```

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

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

Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: The system queries only the concerned partitions rather than the whole table, improving the query efficiency.
2. High availability: If a partition is faulty, data in the other partitions is still available.
3. Easy maintenance: You only need to fix the faulty partition.

The partitioned tables supported by GaussDB(DWS) include range partitioned tables and list partitioned tables. (List partitioned tables are supported only in cluster 8.1.3).

Using Partial Clustering

Partial Cluster Key is the column-based technology. It can minimize or maximize sparse indexes to quickly filter base tables. Partial cluster key can specify multiple columns, but you are advised to specify no more than two columns. Use the following principles to specify columns:

1. The selected columns must be restricted by simple expressions in base tables. Such constraints are usually represented by Col, Op, and Const. Col specifies the column name, Op specifies operators, (including =, >, >=, <=, and <) Const specifies constants.
2. Select columns that are frequently selected (to filter much more undesired data) in simple expressions.
3. List the less frequently selected columns on the top.
4. List the columns of the enumerated type at the top.

Selecting a Data type

You can use data types with the following features to improve efficiency:

1. **Data types that boost execution efficiency**

   Generally, the calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠ and **GROUP BY**) is more efficient than that of strings and floating point numbers. For example, if you need to perform a point query on a column-store table whose **NUMERIC** column is

used as a filter criterion, the query will take over 10 seconds. If you change the data type from **NUMERIC** to **INT**, the query takes only about 1.8 seconds.

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

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

3. **Same data type for a join**

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

Using Indexes

- The purpose of creating indexes is to accelerate queries. Therefore, ensure that indexes can be used in some queries. If an index is not used by any query statement, the index is meaningless. Delete such an index.
- Do not create unnecessary secondary indexes. Useful secondary indexes can accelerate query. However, the space occupied by indexes increases with the number of indexes. Each time an index is added, an additional key-value pair needs to be added when a piece of data is inserted. Therefore, the more indexes, the slower the write speed, and the larger the space usage. In addition, too many indexes affect the optimizer running time, and inappropriate indexes mislead the optimizer. Having more indexes does not necessarily lead to better results.
- Create proper indexes based on service characteristics. In principle, indexes need to be created for columns required in a query to improve performance. Indexes can be created in the following scenarios:
  - For columns with high differentiation, indexes can significantly reduce the number of rows after filtering. For example, you are advised to create an index in the ID card number column, but not in the gender column.
  - If there are multiple query conditions, you can select a combination index. Note that the column of the equivalent condition must be placed before the combination index. For example, if your query is **SELECT * FROM t where c1 = 10 and c2 = 100 and c3 > 10;**, create a composite index **Index cidx (c1, c2, c3)** to optimize scanning.
- When an index column is used as a query condition, do not perform calculation, function, or type conversion on the index column. Otherwise, the optimizer cannot use the index.
- Ensure that the index column contains the query column. Do not always run the **SELECT *** statement to query all columns.
- Indexes are not utilized when **!= or NOT IN** are used in query conditions.
- When LIKE is used, if the condition starts with the wildcard %, the index cannot be used.
- If multiple indexes are available for a query condition but you know which index is the optimal one, you are advised to use the optimizer hint to force the optimizer to use the index. This prevents the optimizer from selecting an incorrect index due to inaccurate statistics or other problems.
- When the IN expression is used as the query condition, the number of matched conditions should not be too large. Otherwise, the execution efficiency is low.

7.1.2 Step 1: Creating an Initial Table and Loading Sample Data

Supported Regions

Table 7-1 describes the regions where OBS data has been uploaded.

Table 7-1 Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1
CN North-Beijing2	dws-demo-cn-north-2
CN North-Beijing4	dws-demo-cn-north-4
CN North-Ulanqab1	dws-demo-cn-north-9
CN East-Shanghai1	dws-demo-cn-east-3
CN East-Shanghai2	dws-demo-cn-east-2
CN South-Guangzhou	dws-demo-cn-south-1
CN South-Guangzhou-InvitationOnly	dws-demo-cn-south-4
CN-Hong Kong	dws-demo-ap-southeast-1
AP-Singapore	dws-demo-ap-southeast-3
AP-Bangkok	dws-demo-ap-southeast-2
LA-Santiago	dws-demo-la-south-2
AF-Johannesburg	dws-demo-af-south-1
LA-Mexico City1	dws-demo-na-mexico-1
LA-Mexico City2	dws-demo-la-north-2
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

Create a group of tables without specifying their storage modes, distribution keys, distribution modes, or compression modes. Load sample data into these tables.

Step 1 (Optional) Create a cluster.

If a cluster is available, skip this step. For how to create a cluster, see **Creating a DWS 2.0 Cluster**.

Furthermore, connect to the cluster and test the connection. For details, see **Methods of Connecting to a Cluster**.

This practice uses an 8-node cluster as an example. You can also use a four-node cluster to perform the test.

Step 2 Create an SS test table **store_sales**.

NOTE

If SS tables already exist in the current database, run the **DROP TABLE** statement to delete these tables first.

For example, delete the **store_sales** table.

```
DROP TABLE store_sales;
```

Do not configure the storage mode, distribution key, distribution mode, or compression mode when you create this table.

Run the **CREATE TABLE** command to create the 11 tables in **Figure 7-3**. This section only provides the syntax for creating the **store_sales** table. To create all tables, copy the syntax in **Appendix: Table Creation Syntax**.

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
) ;
```

Step 3 Load sample data into these tables.

An OBS bucket provides sample data used for this practice. The bucket can be read by all authenticated cloud users. Perform the following operations to load the sample data:

1. Create a foreign table for each table.

   GaussDB(DWS) uses the foreign data wrappers (FDWs) provided by PostgreSQL to import data in parallel. To use FDWs, create FDW tables first (also called foreign tables). This section only provides the syntax for creating the **obs_from_store_sales_001** foreign table corresponding to the **store_sales** table. To create all foreign tables, copy the syntax in **Appendix: Table Creation Syntax**.

NOTE

- Note that *<obs_bucket_name>* in the following statement indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see **Table 7-1**. GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- The columns of the foreign table must be the same as that of the corresponding ordinary table. In this example, **store_sales** and **obs_from_store_sales_001** should have the same columns.
- The foreign table syntax obtains the sample data used for this practice from the OBS bucket. To load other sample data, modify **SERVER gsmpp_server OPTIONS** as needed. For details, see **About Parallel Data Import from OBS**.
- Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE obs_from_store_sales_001
(
    ss_sold_date_sk          integer                       ,
    ss_sold_time_sk          integer                       ,
    ss_item_sk               integer               not null,
    ss_customer_sk           integer                       ,
    ss_cdemo_sk              integer                       ,
    ss_hdemo_sk              integer                       ,
    ss_addr_sk               integer                       ,
    ss_store_sk              integer                       ,
    ss_promo_sk              integer                       ,
    ss_ticket_number         bigint                not null,
    ss_quantity              integer                       ,
    ss_wholesale_cost        decimal(7,2)                  ,
    ss_list_price            decimal(7,2)                  ,
    ss_sales_price           decimal(7,2)                  ,
    ss_ext_discount_amt      decimal(7,2)                  ,
    ss_ext_sales_price       decimal(7,2)                  ,
    ss_ext_wholesale_cost    decimal(7,2)                  ,
    ss_ext_list_price        decimal(7,2)                  ,
    ss_ext_tax               decimal(7,2)                  ,
    ss_coupon_amt            decimal(7,2)                  ,
    ss_net_paid              decimal(7,2)                  ,
    ss_net_paid_inc_tax      decimal(7,2)                  ,
    ss_net_profit            decimal(7,2)
)
-- Configure OBS server information and data format details.
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store_sales',
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
-- If create foreign table failed,record error message
WITH err_obs_from_store_sales_001;
```

2. Set **ACCESS_KEY** and **SECRET_ACCESS_KEY** parameters as needed in the foreign table creation statement, and run this statement in a client tool to create a foreign table.

   For the values of **ACCESS_KEY** and **SECRET_ACCESS_KEY**, see **Creating Access Keys (AK and SK)**.

3. Import data.

Create the **insert.sql** script containing the following statements and execute it:

```
\timing on
\parallel on 4
INSERT INTO store_sales SELECT * FROM obs_from_store_sales_001;
INSERT INTO date_dim SELECT * FROM obs_from_date_dim_001;
INSERT INTO store SELECT * FROM obs_from_store_001;
INSERT INTO item SELECT * FROM obs_from_item_001;
INSERT INTO time_dim SELECT * FROM obs_from_time_dim_001;
INSERT INTO promotion SELECT * FROM obs_from_promotion_001;
INSERT INTO customer_demographics SELECT * from obs_from_customer_demographics_001 ;
INSERT INTO customer_address SELECT * FROM obs_from_customer_address_001 ;
INSERT INTO household_demographics SELECT * FROM obs_from_household_demographics_001;
INSERT INTO customer SELECT * FROM obs_from_customer_001;
INSERT INTO income_band SELECT * FROM obs_from_income_band_001;
\parallel off
```

The returned result is as follows:

```
SET
Timing is on.
SET
Time: 2.831 ms
Parallel is on with scale 4.
Parallel is off.
INSERT 0 402
Time: 1820.909 ms
INSERT 0 73049
Time: 2715.275 ms
INSERT 0 86400
Time: 2377.056 ms
INSERT 0 1000
Time: 4037.155 ms
INSERT 0 204000
Time: 7124.190 ms
INSERT 0 7200
Time: 2227.776 ms
INSERT 0 1920800
Time: 8672.647 ms
INSERT 0 20
Time: 2273.501 ms
INSERT 0 1000000
Time: 11430.991 ms
INSERT 0 1981703
Time: 20270.750 ms
INSERT 0 287997024
Time: 341395.680 ms
total time: 341584  ms
```

4. Calculate the total time spent in creating the 11 tables. The result will be recorded as the loading time in the benchmark table in **Step 1** in the next section.
5. Run the following command to verify that each table is loaded correctly and records lines into the table:

```
SELECT COUNT(*) FROM store_sales;
SELECT COUNT(*) FROM date_dim;
SELECT COUNT(*) FROM store;
SELECT COUNT(*) FROM item;
SELECT COUNT(*) FROM time_dim;
SELECT COUNT(*) FROM promotion;
SELECT COUNT(*) FROM customer_demographics;
SELECT COUNT(*) FROM customer_address;
SELECT COUNT(*) FROM household_demographics;
SELECT COUNT(*) FROM customer;
SELECT COUNT(*) FROM income_band;
```

The number of rows in each SS table is as follows:

Table name	Number of Rows
Store_Sales	287997024
Date_Dim	73049
Store	402
Item	204000
Time_Dim	86400
Promotion	1000
Customer_Demographics	1920800
Customer_Address	1000000
Household_Demographics	7200
Customer	1981703
Income_Band	20

Step 4 Run the **ANALYZE** command to update statistics.

```
ANALYZE;
```

If **ANALYZE** is returned, the execution is successful.

```
ANALYZE
```

The **ANALYZE** statement collects statistics about table content in databases, which will be stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertions and deletions, you are advised to run the **ANALYZE** statement on the table or the entire library to update statistics.

----End

7.1.3 Step 2: Testing System Performance of the Initial Table and Establishing a Baseline

Before and after tuning table structures, test and record the following information to compare differences in system performance:

- Load time
- Storage space occupied by tables
- Query performance

The examples in this practice are based on a dws.d2.xlarge cluster consisting of eight nodes. Because system performance is affected by many factors, clusters of the same flavor may have different results.

Table 7-2 Cluster specifications

Model	dws.d2.xlarge VM
CPU	4*CPU E5-2680 v2 @ 2.80GHZ
Memory	32 GB
Network	1 GB
Disk	1.63 TB
Number of Nodes	8

Record the results using the following benchmark table.

Table 7-3 Recording results

Benchmark	Before	After
Loading time (11 tables)	341584 ms	-
Occupied storage space		
Store_Sales	-	-
Date_Dim	-	-
Store	-	-
Item	-	-
Time_Dim	-	-
Promotion	-	-
Customer_Demographics	-	-
Customer_Address	-	-
Household_Demographics	-	-
Customer	-	-
Income_Band	-	-
Total storage space	-	-
Query execution time		
Query 1	-	-
Query 2	-	-
Query 3	-	-

Benchmark	Before	After
Total execution time	-	-

Perform the following steps to test the system performance before tuning to establish a benchmark:

Step 1 Enter the cumulative load time for all the 11 tables in the benchmarks table in the **Before** column.

Step 2 Record the storage space usage of each table.

Determine how much disk space is used for each table using the **pg_size_pretty** function and record the results in base tables.

```
SELECT T_NAME, PG_SIZE_PRETTY(PG_RELATION_SIZE(t_name)) FROM (VALUES('store_sales'),('date_dim'),
('store'),('item'),('time_dim'),('promotion'),('customer_demographics'),('customer_address'),
('household_demographics'),('customer'),('income_band')) AS names1(t_name);
```

The following information is displayed:

```
        t_name          | pg_size_pretty
------------------------+----------------
 store_sales            | 42 GB
 date_dim               | 11 MB
 store                  | 232 kB
 item                   | 110 MB
 time_dim               | 11 MB
 promotion              | 256 kB
 customer_demographics  | 171 MB
 customer_address       | 170 MB
 household_demographics | 504 kB
 customer               | 441 MB
 income_band            | 88 kB
(11 rows)
```

Step 3 Test query performance.

Run the following queries and record the time spent on each query. The execution durations of the same query can be different, depending on the OS cache during execution. You are advised to perform several rounds of tests and select a group with average values.

```
\timing on
SELECT * FROM (SELECT  COUNT(*)
FROM store_sales
    ,household_demographics
    ,time_dim, store
WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 8
    AND time_dim.t_minute >= 30
    AND household_demographics.hd_dep_count = 5
    AND store.s_store_name = 'ese'
ORDER BY COUNT(*)
 ) LIMIT 100;

SELECT * FROM (SELECT  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
SUM(ss_ext_sales_price) ext_price
 FROM date_dim, store_sales, item,customer,customer_address,store
 WHERE d_date_sk = ss_sold_date_sk
   AND ss_item_sk = i_item_sk
```



```
  AND i_manager_id=8
  AND d_moy=11
  AND d_year=1999
  AND ss_customer_sk = c_customer_sk
  AND c_current_addr_sk = ca_address_sk
  AND substr(ca_zip,1,5) <> substr(s_zip,1,5)
  AND ss_store_sk = s_store_sk
GROUP BY i_brand
     ,i_brand_id
     ,i_manufact_id
     ,i_manufact
ORDER BY ext_price desc
         ,i_brand
         ,i_brand_id
         ,i_manufact_id
         ,i_manufact
) LIMIT 100;

SELECT * FROM (SELECT  s_store_name, s_store_id,
        SUM(CASE WHEN (d_day_name='Sunday') THEN ss_sales_price ELSE null END) sun_sales,
        SUM(CASE WHEN (d_day_name='Monday') THEN ss_sales_price ELSE null END) mon_sales,
        SUM(CASE WHEN (d_day_name='Tuesday') THEN ss_sales_price ELSE  null END) tue_sales,
        SUM(CASE WHEN (d_day_name='Wednesday') THEN ss_sales_price ELSE null END) wed_sales,
        SUM(CASE WHEN (d_day_name='Thursday') THEN ss_sales_price ELSE null END) thu_sales,
        SUM(CASE WHEN (d_day_name='Friday') THEN ss_sales_price ELSE null END) fri_sales,
        SUM(CASE WHEN (d_day_name='Saturday') THEN ss_sales_price ELSE null END) sat_sales
 FROM date_dim, store_sales, store
 WHERE d_date_sk = ss_sold_date_sk AND
       s_store_sk = ss_store_sk AND
       s_gmt_offset = -5 AND
       d_year = 2000
 GROUP BY s_store_name, s_store_id
 ORDER BY s_store_name, s_store_id,sun_sales,mon_sales,tue_sales,wed_sales,thu_sales,fri_sales,sat_sales
  ) LIMIT 100;
```

----End

After the preceding statistics are collected, the benchmark table is as follows:

Benchmark	Before	After
Loading time (11 tables)	341584 ms	-
Occupied storage space		
Store_Sales	42 GB	-
Date_Dim	11 MB	-
Store	232 KB	-
Item	110 MB	-
Time_Dim	11 MB	-
Promotion	256 KB	-
Customer_Demographics	171 MB	-
Customer_Address	170 MB	-
Household_Demographics	504 KB	-

Benchmark	Before	After
Customer	441 MB	-
Income_Band	88 KB	-
Total storage space	42 GB	-
Query execution time		
Query 1	14552.05 ms	-
Query 2	27952.36 ms	-
Query 3	17721.15 ms	-
Total execution time	60225.56 ms	-

7.1.4 Step 3: Optimizing a Table

Selecting a Storage Mode

Sample tables used in this practice are typical multi-column TPC-DS tables where many statistical analysis queries are performed. Therefore, the column storage mode is recommended.

```
WITH (ORIENTATION = column)
```

Selecting a Compression Level

No compression ratio is specified in **Step 1: Creating an Initial Table and Loading Sample Data**, and the low compression ratio is selected by GaussDB(DWS) by default. Specify **COMPRESSION** to **MIDDLE**, and compare the result to that when **COMPRESSION** is set to **LOW**.

The following is an example of selecting a storage mode and the **MIDDLE** compression ratio for a table.

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
```



```
  ss_coupon_amt           decimal(7,2)              ,
  ss_net_paid             decimal(7,2)              ,
  ss_net_paid_inc_tax     decimal(7,2)              ,
  ss_net_profit           decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle);
```

Selecting a Distribution Mode

Based on table sizes provided in **Step 2: Testing System Performance of the Initial Table and Establishing a Baseline**, set the distribution mode as follows.

Table Name	Number of Rows	Distribution Mode
Store_Sales	287997024	Hash
Date_Dim	73049	Replication
Store	402	Replication
Item	204000	Replication
Time_Dim	86400	Replication
Promotion	1000	Replication
Customer_Demographics	1920800	Hash
Customer_Address	1000000	Hash
Household_Demographics	7200	Replication
Customer	1981703	Hash
Income_Band	20	Replication

Selecting a Distribution Key

If your table is distributed using hash, choose a proper distribution key. You are advised to select a distribution key according to **Selecting a Distribution Key**.

Select the primary key of each table as the distribution key of the hash table.

Table Name	Number of Records	Distribution Mode	Distribution Key
Store_Sales	287997024	Hash	ss_item_sk
Date_Dim	73049	Replication	-
Store	402	Replication	-
Item	204000	Replication	-
Time_Dim	86400	Replication	-

Table Name	Number of Records	Distribution Mode	Distribution Key
Promotion	1000	Replication	-
Customer_Demographics	1920800	Hash	cd_demo_sk
Customer_Address	1000000	Hash	ca_address_sk
Household_Demographics	7200	Replication	-
Customer	1981703	Hash	c_customer_sk
Income_Band	20	Replication	-

7.1.5 Step 4: Creating Another Table and Loading Data

After selecting a storage mode, compression level, distribution mode, and distribution key for each table, use these attributes to create tables and reload data. Compare the system performance before and after the table recreation.

Step 1 Delete the tables created before.

```
DROP TABLE store_sales;
DROP TABLE date_dim;
DROP TABLE store;
DROP TABLE item;
DROP TABLE time_dim;
DROP TABLE promotion;
DROP TABLE customer_demographics;
DROP TABLE customer_address;
DROP TABLE household_demographics;
DROP TABLE customer;
DROP TABLE income_band;

DROP FOREIGN TABLE obs_from_store_sales_001;
DROP FOREIGN TABLE obs_from_date_dim_001;
DROP FOREIGN TABLE obs_from_store_001;
DROP FOREIGN TABLE obs_from_item_001;
DROP FOREIGN TABLE obs_from_time_dim_001;
DROP FOREIGN TABLE obs_from_promotion_001;
DROP FOREIGN TABLE obs_from_customer_demographics_001;
DROP FOREIGN TABLE obs_from_customer_address_001;
DROP FOREIGN TABLE obs_from_household_demographics_001;
DROP FOREIGN TABLE obs_from_customer_001;
DROP FOREIGN TABLE obs_from_income_band_001;
```

Step 2 Create tables and specify storage and distribution modes for them.

Only the syntax for recreating the **store_sales** table is provided for simplicity. To recreate all the other tables, copy the syntax in **Appendix: Table Creation Syntax**.

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
```



```
    ss_addr_sk               integer                       ,
    ss_store_sk              integer                       ,
    ss_promo_sk                integer                     ,
    ss_ticket_number           bigint              not null,
    ss_quantity              integer                       ,
    ss_wholesale_cost          decimal(7,2)                ,
    ss_list_price            decimal(7,2)                  ,
    ss_sales_price             decimal(7,2)                ,
    ss_ext_discount_amt        decimal(7,2)                ,
    ss_ext_sales_price         decimal(7,2)                ,
    ss_ext_wholesale_cost      decimal(7,2)                ,
    ss_ext_list_price          decimal(7,2)                ,
    ss_ext_tax               decimal(7,2)                  ,
    ss_coupon_amt              decimal(7,2)                ,
    ss_net_paid              decimal(7,2)                  ,
    ss_net_paid_inc_tax        decimal(7,2)                ,
    ss_net_profit            decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ss_item_sk);
```

Step 3 **Load sample data into these tables.**

Step 4 Record the loading time in the benchmark tables.

Benchmark	Before	After
Loading time (11 tables)	341584 ms	257241 ms
Occupied storage space		
Store_Sales	42 GB	-
Date_Dim	11 MB	-
Store	232 KB	-
Item	110 MB	-
Time_Dim	11 MB	-
Promotion	256 KB	-
Customer_Demographics	171 MB	-
Customer_Address	170 MB	-
Household_Demographics	504 KB	-
Customer	441 MB	-
Income_Band	88 KB	-
Total storage space	42 GB	-
Query execution time		
Query 1	14552.05 ms	-
Query 2	27952.36 ms	-
Query 3	17721.15 ms	-

Benchmark	Before	After
Total execution time	60225.56 ms	-

Step 5 Run the **ANALYZE** command to update statistics.

```
ANALYZE;
```

If **ANALYZE** is returned, the execution is successful.

```
ANALYZE
```

Step 6 Check for data skew.

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

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

xc_node_id corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution key.** In GaussDB(DWS), you can select multiple distribution keys to distribute data evenly.

----End

7.1.6 Step 5: Testing System Performance in the New Table

After recreating the test data set with the selected storage modes, compression levels, distribution modes, and distribution keys, you will retest the system performance.

Step 1 Record the storage space usage of each table.

Determine how much disk space is used for each table using the **pg_size_pretty** function and record the results in base tables.

```
SELECT T_NAME, PG_SIZE_PRETTY(PG_RELATION_SIZE(t_name)) FROM (VALUES('store_sales'),('date_dim'),
('store'),('item'),('time_dim'),('promotion'),('customer_demographics'),('customer_address'),
('household_demographics'),('customer'),('income_band')) AS names1(t_name);
         t_name          | pg_size_pretty
-------------------------+----------------
 store_sales             | 14 GB
 date_dim                | 27 MB
 store                   | 4352 kB
 item                    | 259 MB
 time_dim                | 14 MB
 promotion               | 3200 kB
 customer_demographics   | 11 MB
 customer_address        | 27 MB
 household_demographics | 1280 kB
 customer                | 111 MB
 income_band             | 896 kB
(11 rows)
```

Step 2 Test the query performance and record the performance data in the benchmark table.

Execute the following queries again and record the time spent on each query.


```
\timing on
SELECT * FROM (SELECT  COUNT(*)
FROM store_sales
    ,household_demographics
    ,time_dim, store
WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 8
    AND time_dim.t_minute >= 30
    AND household_demographics.hd_dep_count = 5
    AND store.s_store_name = 'ese'
ORDER BY COUNT(*)
 ) LIMIT 100;

SELECT * FROM (SELECT  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
SUM(ss_ext_sales_price) ext_price
 FROM date_dim, store_sales, item,customer,customer_address,store
 WHERE d_date_sk = ss_sold_date_sk
   AND ss_item_sk = i_item_sk
   AND i_manager_id=8
   AND d_moy=11
   AND d_year=1999
   AND ss_customer_sk = c_customer_sk
   AND c_current_addr_sk = ca_address_sk
   AND substr(ca_zip,1,5) <> substr(s_zip,1,5)
   AND ss_store_sk = s_store_sk
 GROUP BY i_brand
      ,i_brand_id
      ,i_manufact_id
      ,i_manufact
 ORDER BY ext_price desc
         ,i_brand
         ,i_brand_id
         ,i_manufact_id
         ,i_manufact
 ) LIMIT 100;

SELECT * FROM (SELECT  s_store_name, s_store_id,
        SUM(CASE WHEN (d_day_name='Sunday') THEN ss_sales_price ELSE null END) sun_sales,
        SUM(CASE WHEN (d_day_name='Monday') THEN ss_sales_price ELSE null END) mon_sales,
        SUM(CASE WHEN (d_day_name='Tuesday') THEN ss_sales_price ELSE  null END) tue_sales,
        SUM(CASE WHEN (d_day_name='Wednesday') THEN ss_sales_price ELSE null END) wed_sales,
        SUM(CASE WHEN (d_day_name='Thursday') THEN ss_sales_price ELSE null END) thu_sales,
        SUM(CASE WHEN (d_day_name='Friday') THEN ss_sales_price ELSE null END) fri_sales,
        SUM(CASE WHEN (d_day_name='Saturday') THEN ss_sales_price ELSE null END) sat_sales
 FROM date_dim, store_sales, store
 WHERE d_date_sk = ss_sold_date_sk AND
       s_store_sk = ss_store_sk AND
       s_gmt_offset = -5 AND
       d_year = 2000
 GROUP BY s_store_name, s_store_id
 ORDER BY s_store_name, s_store_id,sun_sales,mon_sales,tue_sales,wed_sales,thu_sales,fri_sales,sat_sales
  ) LIMIT 100;
```

The following benchmark table shows the validation results of the cluster used in this tutorial. Your results may vary based on a number of factors, but the relative results should be similar. The execution durations of queries having the same table structure can be different, depending on the OS cache during execution. You are advised to perform several rounds of tests and select a group with average values.

Benchmark	Before	After
Loading time (11 tables)	341584 ms	257241 ms
Occupied storage space		

Benchmark	Before	After
Store_Sales	42 GB	14 GB
Date_Dim	11 MB	27 MB
Store	232 KB	4352 KB
Item	110 MB	259 MB
Time_Dim	11 MB	14 MB
Promotion	256 KB	3200 KB
Customer_Demographics	171 MB	11 MB
Customer_Address	170 MB	27 MB
Household_Demographics	504 KB	1280 KB
Customer	441 MB	111 MB
Income_Band	88 KB	896 KB
Total storage space	42 GB	15 GB
Query execution time		
Query 1	14552.05 ms	1783.353 ms
Query 2	27952.36 ms	14247.803 ms
Query 3	17721.15 ms	11441.659 ms
Total execution time	60225.56 ms	27472.815 ms

Step 3 If you have higher expectations for the performance after the table design, you can run the **EXPLAIN PERFORMANCE** command to view the execution plan for tuning.

For more details about execution plans and query tuning, see **SQL Execution Plan** and **Query Performance Tuning Overview**.

----End

7.1.7 Step 6: Evaluating the Performance of the Optimized Table

Compare the loading time, storage space usage, and query execution time before and after the table tuning.

The following table shows the example results of the cluster used in this tutorial. Your results will be different, but should show similar improvement.

Benchmark	Before	After	Change	Percentage (%)
Loading time (11 tables)	341584 ms	257241 ms	-84343 ms	-24.7%
Occupied storage space			-	-
Store_Sales	42 GB	14 GB	-28 GB	-66.7%
Date_Dim	11 MB	27 MB	16 MB	145.5%
Store	232 KB	4352 KB	4120 KB	1775.9%
Item	110 MB	259 MB	149 MB	1354.5%
Time_Dim	11 MB	14 MB	13 MB	118.2%
Promotion	256 KB	3200 KB	2944 KB	1150%
Customer_Demographics	171 MB	11 MB	-160 MB	-93.6
Customer_Address	170 MB	27 MB	-143 MB	-84.1%
Household_Demographics	504 KB	1280 KB	704 KB	139.7%
Customer	441 MB	111 MB	-330 MB	-74.8%
Income_Band	88 KB	896 KB	808 KB	918.2%
Total storage space	42 GB	15 GB	-27 GB	-64.3%
Query execution time			-	-
Query 1	14552.05 ms	1783.353 ms	-12768.697 ms	-87.7%
Query 2	27952.36 ms	14247.803 ms	-13704.557 ms	-49.0%
Query 3	17721.15 ms	11441.659 ms	-6279.491 ms	-35.4%
Total execution time	60225.56 ms	27472.815 ms	-32752.745 ms	-54.4%

Evaluating the Table After Optimization

- The loading time was reduced by 24.7%.

  The distribution mode has obvious impact on loading data. The hash distribution mode improves the loading efficiency. The replication distribution mode reduces the loading efficiency. When the CPU and I/O are sufficient, the compression level has little impact on the loading efficiency. Typically, the

efficiency of loading a column-store table is higher than that of a row-store table.

- The storage usage space was reduced by 64.3%.

  The compression level, column storage, and hash distribution can save the storage space. A replication table increases the storage usage, but reduces the network overhead. Using the replication mode for small tables is a positive way to use small space for performance.

- The query performance (speed) increased by 54.4%, indicating that the query time decreased by 54.4%.

  The query performance is improved by optimizing storage modes, distribution modes, and distribution keys. In a statistical analysis query on multi-column tables, column storage can improve query performance. In a hash table, I/O resources on each node can be used during I/O read/write, which improves the read/write speed of a table.

  Often, query performance can be improved further by rewriting queries and configuring workload management (WLM). For more information, see **Overview of Query Performance Optimization**.

You can adapt the operations in **Optimizing Table Structure Design to Enhance GaussDB(DWS) Query Performance** to further improve the distribution of tables and the performance of data loading, storage, and query.

Deleting Resources

After this practice is completed, delete the cluster.

To retain the cluster and delete the SS tables, run the following command:

```
DROP TABLE store_sales;
DROP TABLE date_dim;
DROP TABLE store;
DROP TABLE item;
DROP TABLE time_dim;
DROP TABLE promotion;
DROP TABLE customer_demographics;
DROP TABLE customer_address;
DROP TABLE household_demographics;
DROP TABLE customer;
DROP TABLE income_band;
```

7.1.8 Appendix: Table Creation Syntax

This section provides SQL test statements used in this tutorial. You are advised to copy the SQL statements in each section and save them as an .sql file. For example, create a file named **create_table_fir.sql** file and paste the SQL statements in section **Creating an Initial Table** to the file. Executing the file on an SQL client tool is efficient, and the total elapsed time of test cases is easy to calculate. Execute the **.sql** file using **gsql** as follows:

```
gsql -d database_name -h dws_ip -U username -p port_number -W password -f XXX.sql
```

Replace the italic parts in the example with actual values in GaussDB(DWS). For example:

```
gsql -d postgres -h 10.10.0.1 -U dbadmin -p 8000 -W password -f create_table_fir.sql
```

Replace the following information in the example based on the site requirements:

- **postgres**: indicates the name of the database to be connected.
- **10.10.0.1**: cluster connection address.
- **dbadmin**: username of the cluster database. The default administrator is **dbadmin**.
- **8000**: database port set during cluster creation.
- **password**: password set during cluster creation.

Creating an Initial Table

This section contains the table creation syntax used when you create a table for the first time in this tutorial. Tables are created without specifying their storage modes, distribution keys, distribution modes, or compression modes.

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
) ;

CREATE TABLE date_dim
(
    d_date_sk                 integer               not null,
    d_date_id                 char(16)              not null,
    d_date                    date                          ,
    d_month_seq               integer                       ,
    d_week_seq                integer                       ,
    d_quarter_seq             integer                       ,
    d_year                    integer                       ,
    d_dow                     integer                       ,
    d_moy                     integer                       ,
    d_dom                     integer                       ,
    d_qoy                     integer                       ,
    d_fy_year                 integer                       ,
    d_fy_quarter_seq          integer                       ,
    d_fy_week_seq             integer                       ,
    d_day_name                char(9)                       ,
    d_quarter_name            char(6)                       ,
    d_holiday                 char(1)                       ,
    d_weekend                 char(1)                       ,
    d_following_holiday       char(1)                       ,
    d_first_dom               integer                       ,
    d_last_dom                integer                       ,
    d_same_day_ly             integer                       ,
    d_same_day_lq             integer                       ,
```

```
    d_current_day             char(1)                 ,
    d_current_week            char(1)                 ,
    d_current_month           char(1)                 ,
    d_current_quarter         char(1)                 ,
    d_current_year            char(1)
) ;

CREATE TABLE store
(
    s_store_sk                integer               not null,
    s_store_id                char(16)              not null,
    s_rec_start_date          date                  ,
    s_rec_end_date            date                  ,
    s_closed_date_sk          integer               ,
    s_store_name              varchar(50)           ,
    s_number_employees        integer               ,
    s_floor_space             integer               ,
    s_hours                   char(20)              ,
    s_manager                 varchar(40)           ,
    s_market_id               integer               ,
    s_geography_class         varchar(100)          ,
    s_market_desc             varchar(100)          ,
    s_market_manager          varchar(40)           ,
    s_division_id             integer               ,
    s_division_name           varchar(50)           ,
    s_company_id              integer               ,
    s_company_name            varchar(50)           ,
    s_street_number           varchar(10)           ,
    s_street_name             varchar(60)           ,
    s_street_type             char(15)              ,
    s_suite_number            char(10)              ,
    s_city                    varchar(60)           ,
    s_county                  varchar(30)           ,
    s_state                   char(2)               ,
    s_zip                     char(10)              ,
    s_country                 varchar(20)           ,
    s_gmt_offset              decimal(5,2)          ,
    s_tax_precentage          decimal(5,2)
) ;

CREATE TABLE item
(
    i_item_sk                 integer               not null,
    i_item_id                 char(16)              not null,
    i_rec_start_date          date                  ,
    i_rec_end_date            date                  ,
    i_item_desc               varchar(200)          ,
    i_current_price           decimal(7,2)          ,
    i_wholesale_cost          decimal(7,2)          ,
    i_brand_id                integer               ,
    i_brand                   char(50)              ,
    i_class_id                integer               ,
    i_class                   char(50)              ,
    i_category_id             integer               ,
    i_category                char(50)              ,
    i_manufact_id             integer               ,
    i_manufact                char(50)              ,
    i_size                    char(20)              ,
    i_formulation             char(20)              ,
    i_color                   char(20)              ,
    i_units                   char(10)              ,
    i_container               char(10)              ,
    i_manager_id              integer               ,
    i_product_name            char(50)
) ;

CREATE TABLE time_dim
(
    t_time_sk                 integer               not null,
```

```
    t_time_id               char(16)             not null,
    t_time                  integer                      ,
    t_hour                  integer                      ,
    t_minute                integer                      ,
    t_second                integer                      ,
    t_am_pm                 char(2)                      ,
    t_shift                 char(20)                     ,
    t_sub_shift             char(20)                     ,
    t_meal_time             char(20)
) ;

CREATE TABLE promotion
(
    p_promo_sk              integer              not null,
    p_promo_id              char(16)             not null,
    p_start_date_sk         integer                      ,
    p_end_date_sk           integer                      ,
    p_item_sk               integer                      ,
    p_cost                  decimal(15,2)                ,
    p_response_target       integer                      ,
    p_promo_name            char(50)                     ,
    p_channel_dmail         char(1)                      ,
    p_channel_email         char(1)                      ,
    p_channel_catalog       char(1)                      ,
    p_channel_tv            char(1)                      ,
    p_channel_radio         char(1)                      ,
    p_channel_press         char(1)                      ,
    p_channel_event         char(1)                      ,
    p_channel_demo          char(1)                      ,
    p_channel_details       varchar(100)                 ,
    p_purpose               char(15)                     ,
    p_discount_active       char(1)
) ;

CREATE TABLE customer_demographics
(
    cd_demo_sk              integer              not null,
    cd_gender               char(1)                      ,
    cd_marital_status       char(1)                      ,
    cd_education_status     char(20)                     ,
    cd_purchase_estimate    integer                      ,
    cd_credit_rating        char(10)                     ,
    cd_dep_count            integer                      ,
    cd_dep_employed_count   integer                      ,
    cd_dep_college_count    integer
) ;

CREATE TABLE customer_address
(
    ca_address_sk           integer              not null,
    ca_address_id           char(16)             not null,
    ca_street_number        char(10)                     ,
    ca_street_name          varchar(60)                  ,
    ca_street_type          char(15)                     ,
    ca_suite_number         char(10)                     ,
    ca_city                 varchar(60)                  ,
    ca_county               varchar(30)                  ,
    ca_state                char(2)                      ,
    ca_zip                  char(10)                     ,
    ca_country              varchar(20)                  ,
    ca_gmt_offset           decimal(5,2)                 ,
    ca_location_type        char(20)
) ;

CREATE TABLE household_demographics
(
    hd_demo_sk              integer              not null,
    hd_income_band_sk       integer                      ,
    hd_buy_potential        char(15)                     ,
```



```
    hd_dep_count              integer                   ,
    hd_vehicle_count          integer
) ;

CREATE TABLE customer
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                   ,
    c_current_hdemo_sk        integer                   ,
    c_current_addr_sk         integer                   ,
    c_first_shipto_date_sk    integer                   ,
    c_first_sales_date_sk     integer                   ,
    c_salutation              char(10)                  ,
    c_first_name              char(20)                  ,
    c_last_name               char(30)                  ,
    c_preferred_cust_flag     char(1)                   ,
    c_birth_day               integer                   ,
    c_birth_month             integer                   ,
    c_birth_year              integer                   ,
    c_birth_country           varchar(20)               ,
    c_login                   char(13)                  ,
    c_email_address           char(50)                  ,
    c_last_review_date        char(10)
) ;

CREATE TABLE income_band
(
    ib_income_band_sk         integer               not null,
    ib_lower_bound            integer                   ,
    ib_upper_bound            integer
) ;
```

Creating a Another Table After Design Optimization

This section contains the syntax of creating another table after the storage modes, compression levels, distribution modes, and distribution keys are selected in this practice.

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                   ,
    ss_sold_time_sk           integer                   ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                   ,
    ss_cdemo_sk               integer                   ,
    ss_hdemo_sk               integer                   ,
    ss_addr_sk                integer                   ,
    ss_store_sk               integer                   ,
    ss_promo_sk               integer                   ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                   ,
    ss_wholesale_cost         decimal(7,2)              ,
    ss_list_price             decimal(7,2)              ,
    ss_sales_price            decimal(7,2)              ,
    ss_ext_discount_amt       decimal(7,2)              ,
    ss_ext_sales_price        decimal(7,2)              ,
    ss_ext_wholesale_cost     decimal(7,2)              ,
    ss_ext_list_price         decimal(7,2)              ,
    ss_ext_tax                decimal(7,2)              ,
    ss_coupon_amt             decimal(7,2)              ,
    ss_net_paid               decimal(7,2)              ,
    ss_net_paid_inc_tax       decimal(7,2)              ,
    ss_net_profit             decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ss_item_sk);
```



```
CREATE TABLE date_dim
(
    d_date_sk                integer               not null,
    d_date_id                char(16)              not null,
    d_date                   date                          ,
    d_month_seq              integer                       ,
    d_week_seq               integer                       ,
    d_quarter_seq            integer                       ,
    d_year                   integer                       ,
    d_dow                    integer                       ,
    d_moy                    integer                       ,
    d_dom                    integer                       ,
    d_qoy                    integer                       ,
    d_fy_year                integer                       ,
    d_fy_quarter_seq         integer                       ,
    d_fy_week_seq            integer                       ,
    d_day_name               char(9)                       ,
    d_quarter_name           char(6)                       ,
    d_holiday                char(1)                       ,
    d_weekend                char(1)                       ,
    d_following_holiday      char(1)                       ,
    d_first_dom              integer                       ,
    d_last_dom               integer                       ,
    d_same_day_ly            integer                       ,
    d_same_day_lq            integer                       ,
    d_current_day            char(1)                       ,
    d_current_week           char(1)                       ,
    d_current_month          char(1)                       ,
    d_current_quarter        char(1)                       ,
    d_current_year           char(1)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE store
(
    s_store_sk               integer               not null,
    s_store_id               char(16)              not null,
    s_rec_start_date         date                          ,
    s_rec_end_date           date                          ,
    s_closed_date_sk         integer                       ,
    s_store_name             varchar(50)                   ,
    s_number_employees       integer                       ,
    s_floor_space            integer                       ,
    s_hours                  char(20)                      ,
    s_manager                varchar(40)                   ,
    s_market_id              integer                       ,
    s_geography_class        varchar(100)                  ,
    s_market_desc            varchar(100)                  ,
    s_market_manager         varchar(40)                   ,
    s_division_id            integer                       ,
    s_division_name          varchar(50)                   ,
    s_company_id             integer                       ,
    s_company_name           varchar(50)                   ,
    s_street_number          varchar(10)                   ,
    s_street_name            varchar(60)                   ,
    s_street_type            char(15)                      ,
    s_suite_number           char(10)                      ,
    s_city                   varchar(60)                   ,
    s_county                 varchar(30)                   ,
    s_state                  char(2)                       ,
    s_zip                    char(10)                      ,
    s_country                varchar(20)                   ,
    s_gmt_offset             decimal(5,2)                  ,
    s_tax_precentage         decimal(5,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;
```



```
CREATE TABLE item
(
    i_item_sk               integer              not null,
    i_item_id               char(16)             not null,
    i_rec_start_date        date                      ,
    i_rec_end_date          date                      ,
    i_item_desc             varchar(200)              ,
    i_current_price         decimal(7,2)              ,
    i_wholesale_cost        decimal(7,2)              ,
    i_brand_id              integer                   ,
    i_brand                 char(50)                  ,
    i_class_id              integer                   ,
    i_class                 char(50)                  ,
    i_category_id           integer                   ,
    i_category              char(50)                  ,
    i_manufact_id           integer                   ,
    i_manufact              char(50)                  ,
    i_size                  char(20)                  ,
    i_formulation           char(20)                  ,
    i_color                 char(20)                  ,
    i_units                 char(10)                  ,
    i_container             char(10)                  ,
    i_manager_id            integer                   ,
    i_product_name          char(50)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE time_dim
(
    t_time_sk               integer              not null,
    t_time_id               char(16)             not null,
    t_time                  integer                   ,
    t_hour                  integer                   ,
    t_minute                integer                   ,
    t_second                integer                   ,
    t_am_pm                 char(2)                   ,
    t_shift                 char(20)                  ,
    t_sub_shift             char(20)                  ,
    t_meal_time             char(20)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE promotion
(
    p_promo_sk              integer              not null,
    p_promo_id              char(16)             not null,
    p_start_date_sk         integer                   ,
    p_end_date_sk           integer                   ,
    p_item_sk               integer                   ,
    p_cost                  decimal(15,2)             ,
    p_response_target       integer                   ,
    p_promo_name            char(50)                  ,
    p_channel_dmail         char(1)                   ,
    p_channel_email         char(1)                   ,
    p_channel_catalog       char(1)                   ,
    p_channel_tv            char(1)                   ,
    p_channel_radio         char(1)                   ,
    p_channel_press         char(1)                   ,
    p_channel_event         char(1)                   ,
    p_channel_demo          char(1)                   ,
    p_channel_details       varchar(100)              ,
    p_purpose               char(15)                  ,
    p_discount_active       char(1)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;
```



```
CREATE TABLE customer_demographics
(
    cd_demo_sk               integer               not null,
    cd_gender                char(1)                       ,
    cd_marital_status        char(1)                       ,
    cd_education_status      char(20)                      ,
    cd_purchase_estimate     integer                       ,
    cd_credit_rating         char(10)                      ,
    cd_dep_count             integer                       ,
    cd_dep_employed_count    integer                       ,
    cd_dep_college_count     integer
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (cd_demo_sk);

CREATE TABLE customer_address
(
    ca_address_sk            integer               not null,
    ca_address_id            char(16)              not null,
    ca_street_number         char(10)                      ,
    ca_street_name           varchar(60)                   ,
    ca_street_type           char(15)                      ,
    ca_suite_number          char(10)                      ,
    ca_city                  varchar(60)                   ,
    ca_county                varchar(30)                   ,
    ca_state                 char(2)                       ,
    ca_zip                   char(10)                      ,
    ca_country               varchar(20)                   ,
    ca_gmt_offset            decimal(5,2)                  ,
    ca_location_type         char(20)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ca_address_sk);

CREATE TABLE household_demographics
(
    hd_demo_sk               integer               not null,
    hd_income_band_sk        integer                       ,
    hd_buy_potential         char(15)                      ,
    hd_dep_count             integer                       ,
    hd_vehicle_count         integer
)
 WITH (ORIENTATION = column,COMPRESSION=middle)
 DISTRIBUTE BY replication;

 CREATE TABLE customer
(
    c_customer_sk            integer               not null,
    c_customer_id            char(16)              not null,
    c_current_cdemo_sk       integer                       ,
    c_current_hdemo_sk       integer                       ,
    c_current_addr_sk        integer                       ,
    c_first_shipto_date_sk   integer                       ,
    c_first_sales_date_sk    integer                       ,
    c_salutation             char(10)                      ,
    c_first_name             char(20)                      ,
    c_last_name              char(30)                      ,
    c_preferred_cust_flag    char(1)                       ,
    c_birth_day              integer                       ,
    c_birth_month            integer                       ,
    c_birth_year             integer                       ,
    c_birth_country          varchar(20)                   ,
    c_login                  char(13)                      ,
    c_email_address          char(50)                      ,
    c_last_review_date       char(10)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (c_customer_sk);
```



```
CREATE TABLE income_band
(
    ib_income_band_sk         integer               not null,
    ib_lower_bound            integer                       ,
    ib_upper_bound            integer
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;
```

Creating a Foreign Table

This section contains the syntax of foreign tables for obtaining sample data used in this tutorial. The sample data is stored in an OBS bucket accessible to all authenticated cloud users.

NOTE

- Note that *<obs_bucket_name>* in the following statement indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see **Supported Regions**. GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- You can replace **ACCESS_KEY** and **SECRET_ACCESS_KEY** with your own credentials in this example.
- When an OBS foreign table is created, only the mapping relationship is created, and data is not pulled to the GaussDB(DWS) disk.

```
CREATE FOREIGN TABLE obs_from_store_sales_001
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store_sales',
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_store_sales_001;
```



```
CREATE FOREIGN TABLE obs_from_date_dim_001
(
    d_date_sk               integer              not null,
    d_date_id               char(16)             not null,
    d_date                  date                      ,
    d_month_seq               integer                 ,
    d_week_seq              integer                   ,
    d_quarter_seq           integer                   ,
    d_year                  integer                  ,
    d_dow                   integer                   ,
    d_moy                   integer                   ,
    d_dom                   integer                   ,
    d_qoy                   integer                  ,
    d_fy_year               integer                   ,
    d_fy_quarter_seq          integer                 ,
    d_fy_week_seq           integer                   ,
    d_day_name              char(9)                   ,
    d_quarter_name            char(6)                  ,
    d_holiday               char(1)                   ,
    d_weekend               char(1)                    ,
    d_following_holiday       char(1)                   ,
    d_first_dom             integer                   ,
    d_last_dom              integer                   ,
    d_same_day_ly             integer                 ,
    d_same_day_lq             integer                 ,
    d_current_day           char(1)                   ,
    d_current_week            char(1)                  ,
    d_current_month           char(1)                   ,
    d_current_quarter         char(1)                   ,
    d_current_year          char(1)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/date_dim' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_date_dim_001;

CREATE FOREIGN TABLE obs_from_store_001
(
    s_store_sk              integer              not null,
    s_store_id              char(16)             not null,
    s_rec_start_date          date                      ,
    s_rec_end_date            date                      ,
    s_closed_date_sk          integer                   ,
    s_store_name              varchar(50)                ,
    s_number_employees          integer                    ,
    s_floor_space             integer                  ,
    s_hours                 char(20)                  ,
    s_manager                 varchar(40)                ,
    s_market_id               integer                  ,
    s_geography_class           varchar(100)              ,
    s_market_desc             varchar(100)               ,
    s_market_manager            varchar(40)                ,
    s_division_id             integer                  ,
    s_division_name             varchar(50)               ,
    s_company_id              integer                  ,
    s_company_name              varchar(50)                ,
    s_street_number           varchar(10)                ,
    s_street_name             varchar(60)               ,
    s_street_type             char(15)                  ,
    s_suite_number            char(10)                   ,
```



```
    s_city                  varchar(60)                 ,
    s_county                varchar(30)                 ,
    s_state                 char(2)                     ,
    s_zip                   char(10)                    ,
    s_country               varchar(20)                 ,
    s_gmt_offset            decimal(5,2)                ,
    s_tax_precentage        decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_store_001;

CREATE FOREIGN TABLE obs_from_item_001
(
    i_item_sk               integer             not null,
    i_item_id               char(16)            not null,
    i_rec_start_date        date                ,
    i_rec_end_date          date                ,
    i_item_desc             varchar(200)        ,
    i_current_price         decimal(7,2)        ,
    i_wholesale_cost        decimal(7,2)        ,
    i_brand_id              integer             ,
    i_brand                 char(50)            ,
    i_class_id              integer             ,
    i_class                 char(50)            ,
    i_category_id           integer             ,
    i_category              char(50)            ,
    i_manufact_id           integer             ,
    i_manufact              char(50)            ,
    i_size                  char(20)            ,
    i_formulation           char(20)            ,
    i_color                 char(20)            ,
    i_units                 char(10)            ,
    i_container             char(10)            ,
    i_manager_id            integer             ,
    i_product_name          char(50)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/item' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_item_001;

CREATE FOREIGN TABLE obs_from_time_dim_001
(
    t_time_sk               integer             not null,
    t_time_id               char(16)            not null,
    t_time                  integer             ,
    t_hour                  integer             ,
    t_minute                integer             ,
    t_second                integer             ,
```

```
    t_am_pm                 char(2)                 ,
    t_shift                 char(20)                ,
    t_sub_shift             char(20)                ,
    t_meal_time             char(20)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/time_dim' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_time_dim_001;

CREATE FOREIGN TABLE obs_from_promotion_001
(
    p_promo_sk              integer             not null,
    p_promo_id              char(16)            not null,
    p_start_date_sk         integer                     ,
    p_end_date_sk           integer                     ,
    p_item_sk               integer                     ,
    p_cost                  decimal(15,2)               ,
    p_response_target       integer                     ,
    p_promo_name            char(50)                    ,
    p_channel_dmail         char(1)                     ,
    p_channel_email         char(1)                     ,
    p_channel_catalog       char(1)                     ,
    p_channel_tv            char(1)                     ,
    p_channel_radio         char(1)                     ,
    p_channel_press         char(1)                     ,
    p_channel_event         char(1)                     ,
    p_channel_demo          char(1)                     ,
    p_channel_details       varchar(100)                ,
    p_purpose               char(15)                    ,
    p_discount_active       char(1)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/promotion' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_promotion_001;

CREATE FOREIGN TABLE obs_from_customer_demographics_001
(
    cd_demo_sk              integer             not null,
    cd_gender               char(1)                     ,
    cd_marital_status       char(1)                     ,
    cd_education_status     char(20)                    ,
    cd_purchase_estimate    integer                     ,
    cd_credit_rating        char(10)                    ,
    cd_dep_count            integer                     ,
    cd_dep_employed_count   integer                     ,
    cd_dep_college_count    integer
)
SERVER gsmpp_server
OPTIONS (
```



```
LOCATION 'obs://<obs_bucket_name>/tpcds/customer_demographics' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_demographics_001;

CREATE FOREIGN TABLE obs_from_customer_address_001
(
ca_address_sk integer not null,
ca_address_id char(16) not null,
ca_street_number char(10) ,
ca_street_name varchar(60) ,
ca_street_type char(15) ,
ca_suite_number char(10) ,
ca_city varchar(60) ,
ca_county varchar(30) ,
ca_state char(2) ,
ca_zip char(10) ,
ca_country varchar(20) ,
ca_gmt_offset float4 ,
ca_location_type char(20)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/customer_address' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_address_001;

CREATE FOREIGN TABLE obs_from_household_demographics_001
(
    hd_demo_sk              integer             not null,
    hd_income_band_sk       integer                     ,
    hd_buy_potential        char(15)                    ,
    hd_dep_count            integer                     ,
    hd_vehicle_count        integer
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/household_demographics' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_household_demographics_001;

CREATE FOREIGN TABLE obs_from_customer_001
(
    c_customer_sk           integer             not null,
    c_customer_id           char(16)            not null,
    c_current_cdemo_sk      integer                     ,
```

```
    c_current_hdemo_sk        integer                  ,
    c_current_addr_sk         integer                  ,
    c_first_shipto_date_sk    integer                  ,
    c_first_sales_date_sk     integer                  ,
    c_salutation              char(10)                 ,
    c_first_name              char(20)                 ,
    c_last_name               char(30)                 ,
    c_preferred_cust_flag     char(1)                  ,
    c_birth_day               integer                  ,
    c_birth_month             integer                  ,
    c_birth_year              integer                  ,
    c_birth_country           varchar(20)              ,
    c_login                   char(13)                 ,
    c_email_address           char(50)                 ,
    c_last_review_date        char(10)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/customer' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_001;

CREATE FOREIGN TABLE obs_from_income_band_001
(
    ib_income_band_sk         integer              not null,
    ib_lower_bound            integer                  ,
    ib_upper_bound            integer
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/income_band' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_income_band_001;
```

7.2 Analyzing SQL Statements That Are Being Executed to Handle GaussDB(DWS) Performance Issues

During development, developers often encounter problems such as excessive SQL connections, long SQL query time, and SQL query blocking. You can use the **PG_STAT_ACTIVITY** and **PGXC_THREAD_WAIT_STATUS** views to analyze and locate SQL problems. This section describes some common locating methods.

Table 7-4 Some PG_STAT_ACTIVITY fields

Name	Type	Description
usename	name	Name of the user logging in to the backend
client_addr	inet	IP address of the client connected to the backend **null** indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum.
application_name	text	Name of the application connected to the backend
state	text	Overall state of the backend. The value can be: • **active**: The backend is executing queries. • **idle**: The backend is waiting for new client commands. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction (aborted)**: The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a **fast-path** function. • **disabled**: This state is reported if **track_activities** is disabled in this backend. **NOTE** Common users can view only the session status of their own accounts. That is, the state information of other accounts is empty.
waiting	boolean	If the back end is currently waiting for a lock, the value is **t**. Otherwise, the value is **f**. • **t** stands for true. • **f** stands for false.

Name	Type	Description
enqueue	text	Queuing status of a statement. Its value can be: • **waiting in global queue**: The statement is queuing in the global concurrency queue. The number of concurrent statements exceeds the value of **max_active_statements** configured for a single CN. • **waiting in respool queue**: The statement is queuing in the resource pool and the concurrency of simple jobs is limited. The main reason is that the concurrency of simple jobs exceeds the upper limit **max_dop** of the fast track. • **waiting in ccn queue**: The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing. The scenarios are: 1. The available global memory exceeds the upper limit, the job is queuing in the global memory queue. 2. Concurrent requests on the slow lane in the resource pool exceed the upper limit, which is specified by **active_statements**. 3. The slow lane memory of the resource pool exceeds the upper limit, that is, the estimated memory of concurrent jobs in the resource pool exceeds the upper limit specified by **mem_percent**. • Empty or **no waiting queue**: The statement is running.
pid	bigint	ID of the backend thread.

Viewing Connection Information

- Set **track_activities** to **on**.

  ```
  SET track_activities = on;
  ```

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

- You can run the following SQL statements to check the current connection user, connection address, connection application, status, whether to wait for a lock, queuing status, and thread ID.

  ```
  SELECT usename,client_addr,application_name,state,waiting,enqueue,pid FROM PG_STAT_ACTIVITY
  WHERE DATNAME='database name';
  ```

  The following command output is displayed:

  ```
  usename |  client_addr  | application_name | state  | waiting | enqueue |       pid
  ---------+---------------+------------------+--------+---------+---------+-----------------
  ```



```
 leo     | 192.168.0.133 | gsql             | idle   | f       |         | 139666091022080
 dbadmin | 192.168.0.133 | gsql             | active | f       |         | 139666212681472
 joe     | 192.168.0.133 |                  | idle   | f       |         | 139665671489280
(3 rows)
```

- End a session (only the system administrator has the permission).

```
SELECT PG_TERMINATE_BACKEND(pid);
```

Viewing SQL Running Information

- Run the following command to obtain all SQL information that the current user has permission to view (if the current user has administrator or preset role permission, all user query information can be displayed):

```
SELECT usename,state,query FROM PG_STAT_ACTIVITY WHERE DATNAME='database name';
```

  If the value of **state** is **active**, the **query** column indicates the SQL statement that is being executed. In other cases, the **query** column indicates the previous query statement. If the value of **state** is **idle**, the connection is idle and waits for the user to enter a command. The following command output is displayed:

```
 usename | state  |                                query
---------+--------+-----------------------------------------------------------------------------
 leo     | idle   | select * from joe.mytable;
 dbadmin | active | SELECT usename,state,query FROM PG_STAT_ACTIVITY WHERE
DATNAME='gaussdb';
 joe     | idle   | GRANT SELECT ON TABLE mytable to leo;
(3 rows)
```

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

```
SELECT datname,usename,query FROM PG_STAT_ACTIVITY WHERE state != 'idle' ;
```

Viewing Time-Consuming Statements

- Check the SQL statements that take a long time to execute.

```
SELECT current_timestamp - query_start as runtime, datname, usename, query FROM
PG_STAT_ACTIVITY WHERE state != 'idle' order by 1 desc;
```

  Query statements are returned and sorted by execution time length in descending order. The first record is the query statement that takes the longest time to execute.

```
    runtime     | datname  | usename |
query
-----------------+----------+---------
+-----------------------------------------------------------------------------------------------------------------------------
--------------------
 00:04:47.054958 | gaussdb  | leo     | insert into mytable1 select generate_series(1, 10000000);
 00:00:01.72789  | gaussdb  | dbadmin | SELECT current_timestamp - query_start as runtime, datname,
usename, query FROM PG_STAT_ACTIVITY WHERE state != 'idle' order by 1 desc;
(2 rows)
```

- Alternatively, you can set **current_timestamp - query_start** to be greater than a threshold to identify query statements that are executed for a duration longer than this threshold.

```
SELECT query from PG_STAT_ACTIVITY WHERE current_timestamp - query_start > interval '2 days';
```

Querying Blocked Statements

- Run the following command to view blocked query statements:

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

  Run the following statement to end the blocked SQL session:

```
SELECT PG_TERMINATE_BACKEND(pid);
```


NOTE

- In most cases, blocking is caused by internal locks and **waiting=true** is displayed. You can view the blocking in the **pg_stat_activity** view.
- The blocked statements about file write and event schedulers cannot be viewed in the **pg_stat_activity** view.

- View information about the blocked query statements, tables, and schemas.

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid
where w.waiting;
```

The command output includes a session ID, user information, query status, and table or schema that caused the block.

After finding the blocked table or schema information, end the faulty session.

```
SELECT PG_TERMINATE_BACKEND(pid);
```

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

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

If information similar to the following is displayed, the user is attempting to terminate the session, but the session will be reconnected rather than terminated.

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

NOTE

If the **PG_TERMINATE_BACKEND** function is used by the gsql client to terminate the background threads of the session, the client will be reconnected automatically rather than be terminated.

8 Cluster Management

8.1 Binding Different Resource Pools to Two Types of Jobs to Balance Load for GaussDB(DWS)

This practice demonstrates how to use GaussDB(DWS) for resource management, helping enterprises eliminate bottlenecks in concurrent query performance. SQL jobs can run smoothly without affecting each other and consume less resources than before.

This practice takes about 60 minutes. The process is as follows:

1. **Step 1: Creating a Cluster**
2. **Step 2: Connecting to a Cluster and Importing Data**
3. **Step 3: Creating a Resource Pool**
4. **Step 4: Verifying Exception Rules**

Scenarios

When multiple database users execute SQL jobs on GaussDB(DWS) at the same time, the following situations may occur:

1. Some complex SQL statements occupy cluster resources for a long time, affecting the performance of other queries. For example, a group of database users continuously submit complex and time-consuming queries, and another group of users frequently submit short queries. In this case, short queries may have to wait in the resource pool for the time-consuming queries to complete.
2. Some SQL statements occupy too much memory or disk space due to data skew or unoptimized execution plans. As a result, the statements that fail to apply for memory report errors, or the cluster switches to the read-only mode.

To increase the system throughput and improve SQL performance, you can use workload management of GaussDB(DWS). For example, create a resource pool for users who frequently submit complex query jobs, and allocate more resources to this resource pool. The complex jobs submitted by these users can use only the resources of this resource pool. Create another resource pool that occupies less resources and add users who submit short queries to this resource pool. In this way, the two types of jobs can be smoothly executed at the same time.

For example, user A processes online transaction processing (OLTP) and online analytical processing (OLAP) services. The priority of the OLAP service is lower than that of OLTP service. A large number of concurrent complex SQL queries may cause server resource contention, whereas a large number of concurrent simple SQL queries can be quickly processed without being queued. Resources must be properly allocated and managed to ensure both OLAP and OLTP services can run smoothly.

OLAP services are often complex, and do not require high priority or real-time response. OLAP and OLTP services are operated by different users. For example, the database user **budget_config_user** is used for core transaction services, and the database user **report_user** is used for report services. The users are under independent CPU and concurrency management to improve database stability.

Based on the workload survey, routine monitoring, and test and verification of OLAP services, it is found that less than 50 concurrent SQL queries do not cause server resource contention or slow service system response. OLAP users can use 20% CPU resources.

Based on the workload survey, routine monitoring, and test and verification of OLTP services, it is found that less than 100 concurrent SQL queries do not pose continuous pressure onto the system. OLTP users can use 60% of CPU resources.

- Resource configuration for OLAP users (corresponding to **pool_1**): CPU = 20%, memory = 20%, storage = 1,024,000 MB, concurrency = 20.
- Resource configuration for OLTP users (corresponding to **pool_2**): CPU = 60%, memory = 60%, storage = 1,024,000 MB, concurrency = 200.

Set the maximum memory that can be used by a single statement. An error will be reported if the memory usage exceeds the value.

In **Exception Rule**, set **Blocking Time** to 1200s and **Execution Time** to 1800s. A query job will be terminated after being executed for more than 1800 seconds.

Step 1: Creating a Cluster

Create a cluster by referring to **Creating a cluster**.

Step 2: Connecting to a Cluster and Importing Data

Step 1 Use the client to connect to the cluster.

Step 2 Import sample data. For details, see**Importing TPC-H Data**.

Step 3 Run the following statements to create the OLTP user **budget_config_user** and OLAP user **report_user**.

```
CREATE USER budget_config_user PASSWORD 'password';
CREATE USER report_user PASSWORD 'password';
```

Step 4 For test purposes, grant all permissions on all tables in schema **tpch** to both users.

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA tpch to budget_config_user,report_user;
```

Step 5 Check the resource allocation of the two users.

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO where username in ('budget_config_user' , 'report_user');
```



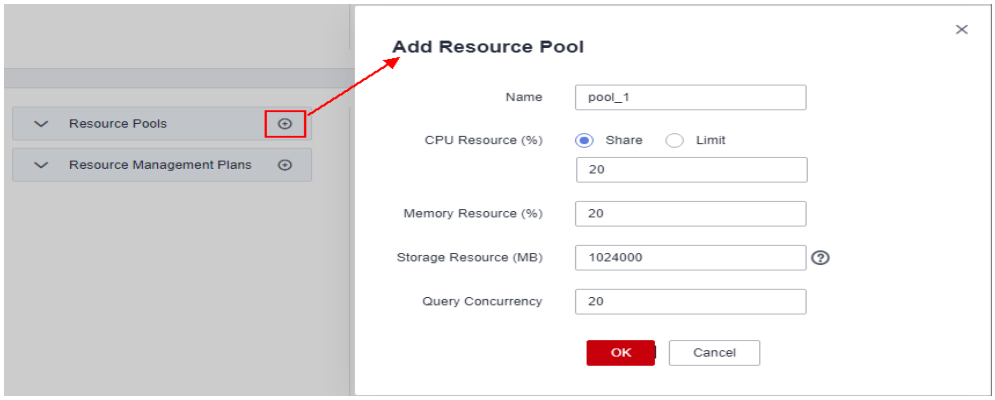
```
tpch=> SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO where username in ('budget_config_user' , 'report_user');
      username     | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space | used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbyte
s | read_counts | write_counts | read_speed | write_speed
-------------------+-------------+--------------+----------+-----------+------------+-------------+-----------------+------------------+------------------+-------------------+-------------+------------
---+-------------+--------------+------------+-------------
 budget_config_user |           0 |        10796 |        0 |         8 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
0 |           0 |            0 |          0 |         0 |
 report_user       |           0 |        10796 |        0 |         8 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
0 |           0 |            0 |          0 |         0 |
(2 rows)
```

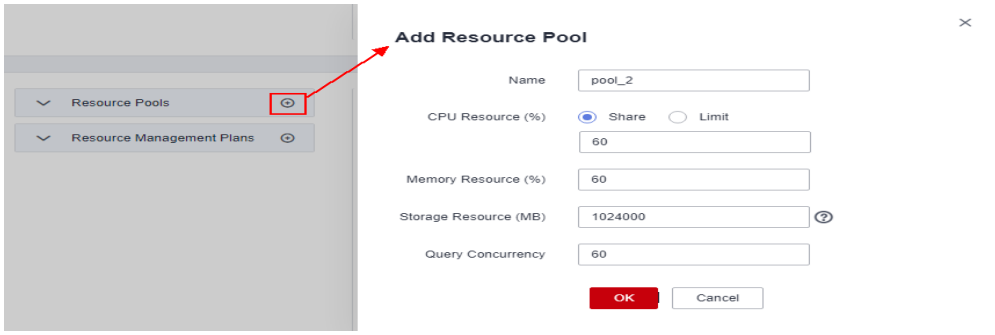
----End

Step 3: Creating a Resource Pool

Step 1 Log in to the GaussDB(DWS) management console, click a cluster name in the cluster list. The **Resource Management Configurations** page is displayed.

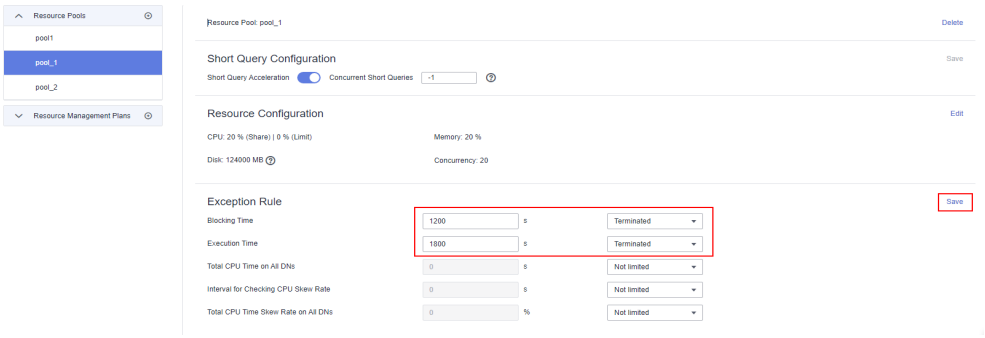
Step 2 Click **Add Workload Queue**. Create the report resource pool **pool_1** and transaction resource pool **pool_2** by referring to **Scenarios**.





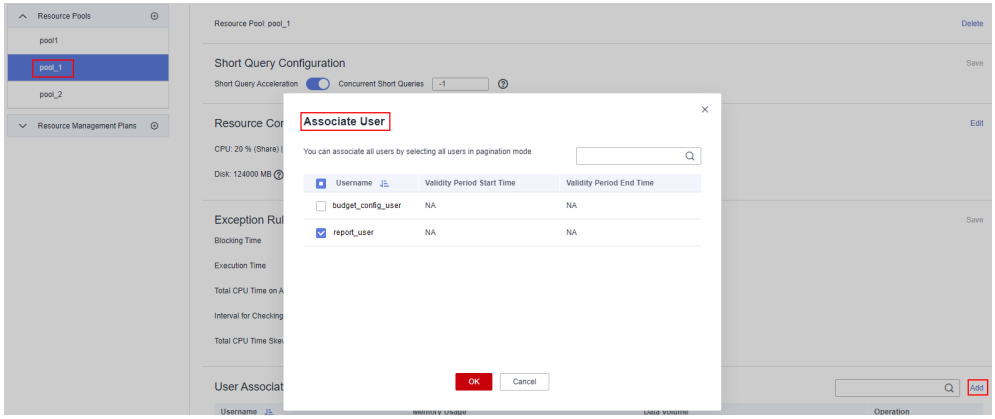
Step 3 Modify the exception rules.

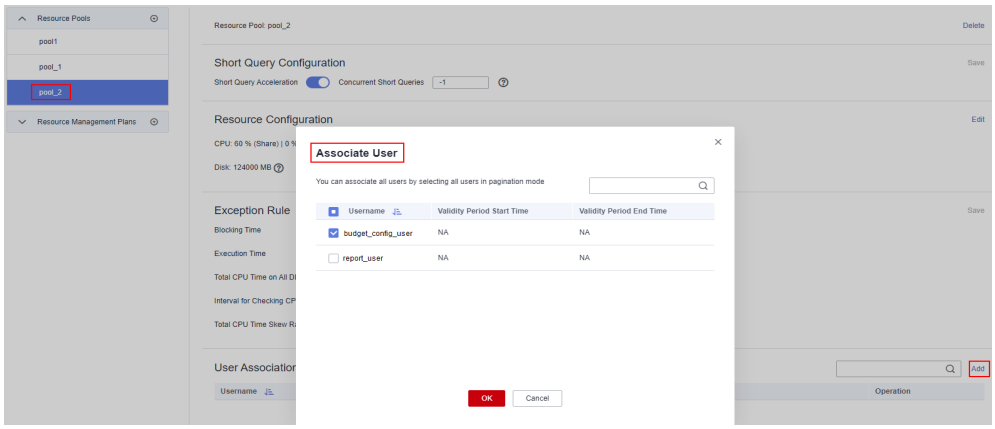
1. Click the created **pool_1**.
2. In the **Exception Rule** area, set **Blocking Time** to 1200s and **Execution Time** to 1800s.
3. Click **Save**.
4. Repeat the preceding steps to configure **pool_2**.



Step 4 Associate users.

1. Click **pool_1** on the left.
2. Click **Add** on the right of **User Association**.
3. Select **report_user** and click **OK**.
4. Repeat the preceding steps to add **budget_config_user** to **pool_2**.





----End

Step 4: Verifying Exception Rules

Step 1 Log in to the database as user **report_user**.

Step 2 Run the following command to check the resource pool to which the **report_user** user belongs:

```
SELECT usename,respool FROM pg_user WHERE usename = 'report_user';
```



```
gaussdb=> select usename,respool from pg_user where usename = 'report_user';
   usename   | respool
-------------+---------
 report_user | pool_1
(1 row)
```

The query result shows that the resource pool to which the **report_user** user belongs is **pool_1**.

Step 3 Verify the exception rule bound to the resource pool **pool_1**.

```
SELECT respool_name,mem_percent,active_statements,except_rule FROM pg_resource_pool WHERE respool_name='pool_1';
```

```
gaussdb=> select respool_name,mem_percent,active_statements,except_rule from pg_resource_pool where respool_name='pool_1';
 respool_name | mem_percent | active_statements | except_rule
--------------+-------------+-------------------+-------------
 pool_1       |          20 |                20 | rule_1
(1 row)
```

It is confirmed that the exception rule **rule_1** is bound to **pool_1**.

Step 4 View the rule type and threshold of the exception rule for the current user.

```
SELECT * FROM pg_except_rule WHERE name = 'rule_1';
```

```
gaussdb=> select * from pg_except_rule where name = 'rule_1';
  name  |    rule     | value
--------+-------------+-------
 rule_1 | action      | abort
 rule_1 | blocktime   | 1200
 rule_1 | elapsedtime | 1800
(3 rows)
```

The return shows that rule_1 has 1200 seconds of block time and 1800 seconds of running duration.

NOTICE

- **PG_EXCEPT_RULE** records information about exception rules and is supported only in cluster 8.2.0 or later.
- The relationship between parameters in the same exception rule is AND.

Step 5 When the block time of a job exceeds 1200s and the running duration exceeds 1800s, an error message is displayed, indicating that the exception rule is triggered and the job is canceled.

```
gaussdb=> insert into mytable select * from table1;
ERROR: canceling statement due to workload manager exception.
DETAIL: except rule [rule_1] is meet condition: rule [elapsedtime] is over limit, current value is: 1800, rule [blocktime] is over limit, current value is: 1200.
```

If error information similar to "ERROR: canceling statement due to workload manager exception." is displayed during job execution, the job is terminated because it exceeds the threshold of the exception rule. If the rules do not need to be modified, you need to optimize the service statements to reduce the execution time.

----End

8.2 Scaling Options for GaussDB(DWS) with a Coupled Storage-Compute Architecture

Scalability is a critical feature for cloud services. It refers to cloud services' ability to increase or decrease compute and storage resources to meet changing demand, achieving a balance between performance and cost.

Typically, a distributed architecture offers the following types of scalability:

- Scale-out (horizontal scaling)

  With a scale-out, more nodes are added to an existing system to increase storage and compute capacities. For GaussDB(DWS), this means to expand the cluster size. To ensure proper resource utilization, make sure the hardware devices you add use the same specifications as the ones already in the cluster do.

- Scale-in (horizontal scaling)

  Scale-in is the opposite of scale-out. With a scale-in, nodes are removed from an existing system to decrease storage and compute capacities and by doing so, increase resource utilization. GaussDB(DWS) is deployed by security ring, which means GaussDB(DWS) clusters are scaled in or out by security ring as well. We will talk about security rings in more detail in a later section.

- Scale-up (vertical scaling)

  With a scale-up, more CPUs, memory, disks, or NICs are added to existing servers to increase the corresponding capacities. In some cases, lower-capacity hardware is replaced by higher-capacity ones. This is also referred to as hardware upgrade, which may entail an OS upgrade sometimes.

- Scale-down (vertical scaling)

  Scale-down is the opposite of scale-up. With a scale-down, the hardware of an existing system is downgraded to match demand.

GaussDB(DWS) offers the standard data warehouse (DWS 2.0) and stream data warehouse, both of which use a distributed architecture with coupled storage and compute. They support both horizontal and vertical scaling. A cluster resizing option allows customers to perform horizontal and vertical scaling at the same time. The cluster topology can also be adjusted.

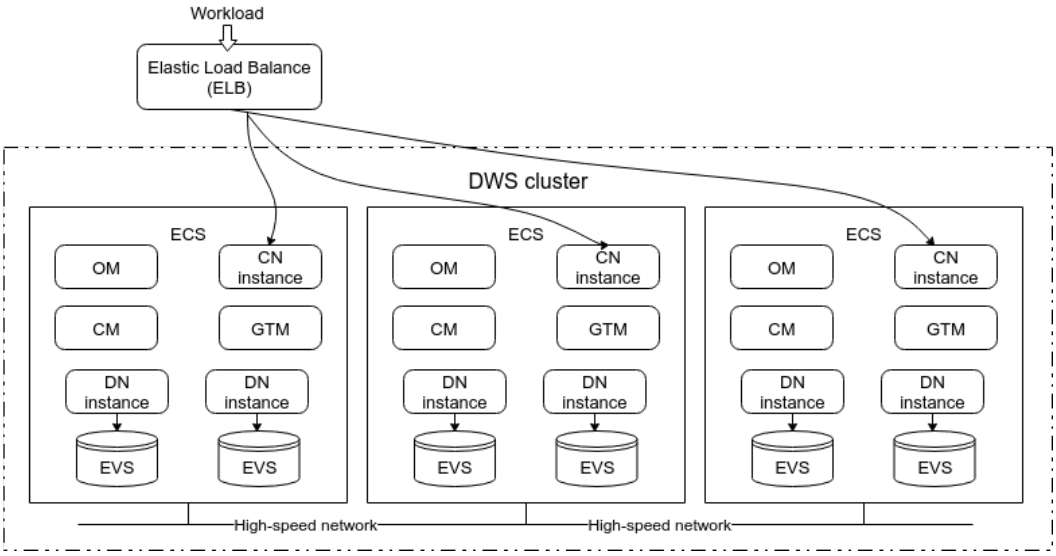
A Closer Look at GaussDB(DWS) Cluster Topology

To fully understand the scalability of GaussDB(DWS), one needs to understand GaussDB(DWS)'s typical cluster topology. The following figure shows a simplified ECS+EVS deployment structure of GaussDB(DWS).

- ECSs provide compute resources, including CPUs and memory. GaussDB(DWS) database instances (such as CNs and DNs) are deployed on ECSs.
- EVS provides storage resources. An EVS disk is attached to each DN.
- All ECSs in a GaussDB(DWS) cluster are within the same VPC to ensure high-speed connections between them.

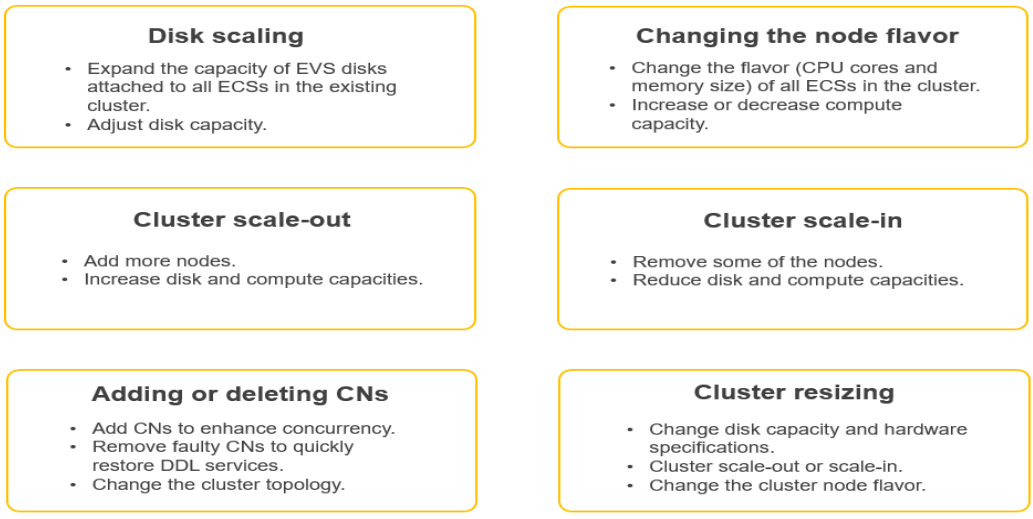
- All the database instances deployed on ECSs form a distributed, massively parallel processing database (MPPDB) cluster to provide data analysis and processing capabilities as a whole.

Figure 8-1 Cluster topology



Once you have had a good look at the typical topology of a GaussDB(DWS) cluster, you can better understand GaussDB(DWS)'s scalability features. At present, GaussDB(DWS) offers the following scaling options: disk scaling, node flavor change, cluster scale-out, cluster scale-in, cluster resizing, and CN addition or deletion, as illustrated by the figure below:

Figure 8-2 GaussDB(DWS) scaling options

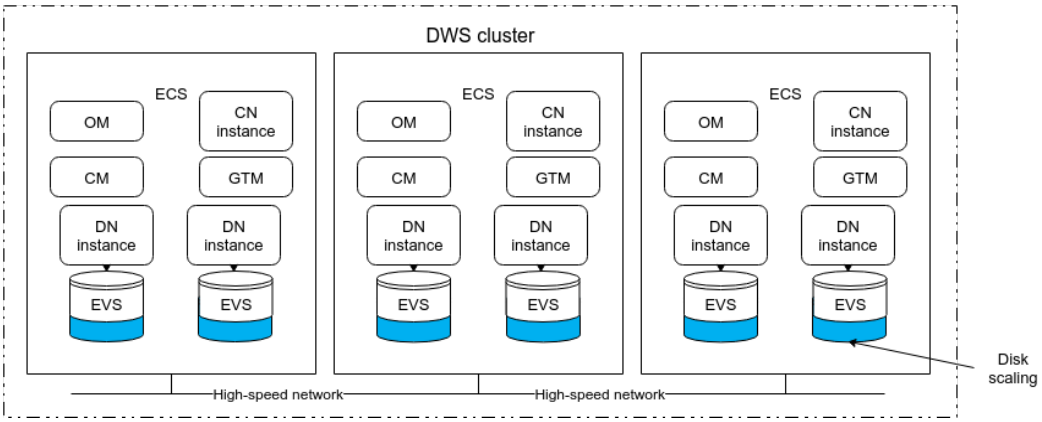


Disk Scaling

- With disk scaling, the size of all EVS disks attached to all ECSs in a cluster is changed. This option can be used to quickly scale disk capacity.

- Disk capacity can only be scaled up, and not down.
- Disk scaling is a lightweight operation that typically can be completed within 5 to 10 minutes. **It does not entail data migration or the restarting of services, so it does not interrupt services**. Nonetheless, you are advised to perform this operation during off-peak hours.
- GaussDB(DWS) standard data warehouses and stream data warehouses support this operation. The cluster version must be 8.1.1.203 or later.
- For details, see **Disk Capacity Expansion of an EVS Cluster**.

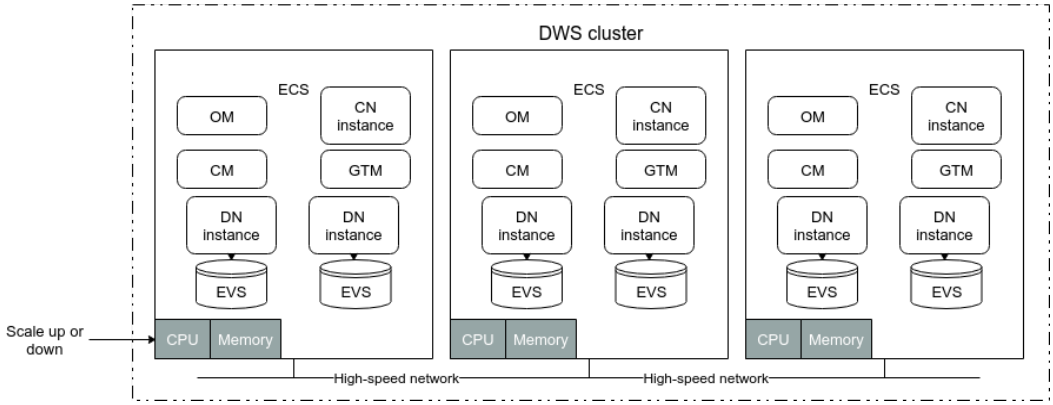
Figure 8-3 Disk scaling



Changing the Node Flavor

- This operation changes the flavor of all ECSs in a cluster. It can be used to quickly change CPU and memory specifications.
- A flavor is a preset resource template of a combination of a specific number of vCPUs and memory. For example, the flavor dwsx.16xlarge includes 64 vCPUs and 512 GB memory.
- Changing the node flavor is a lightweight operation that typically can be completed within 5 to 10 minutes. It does not involve data migration, but **services will need to be restarted once, causing a service interruption in minutes**. You are advised to perform this operation during off-peak hours.
- GaussDB(DWS) standard data warehouses and stream data warehouses support this operation. The cluster version must be 8.1.1.300 or later.
- For details, see **Changing the Node Flavor**.

Figure 8-4 Changing the node flavor

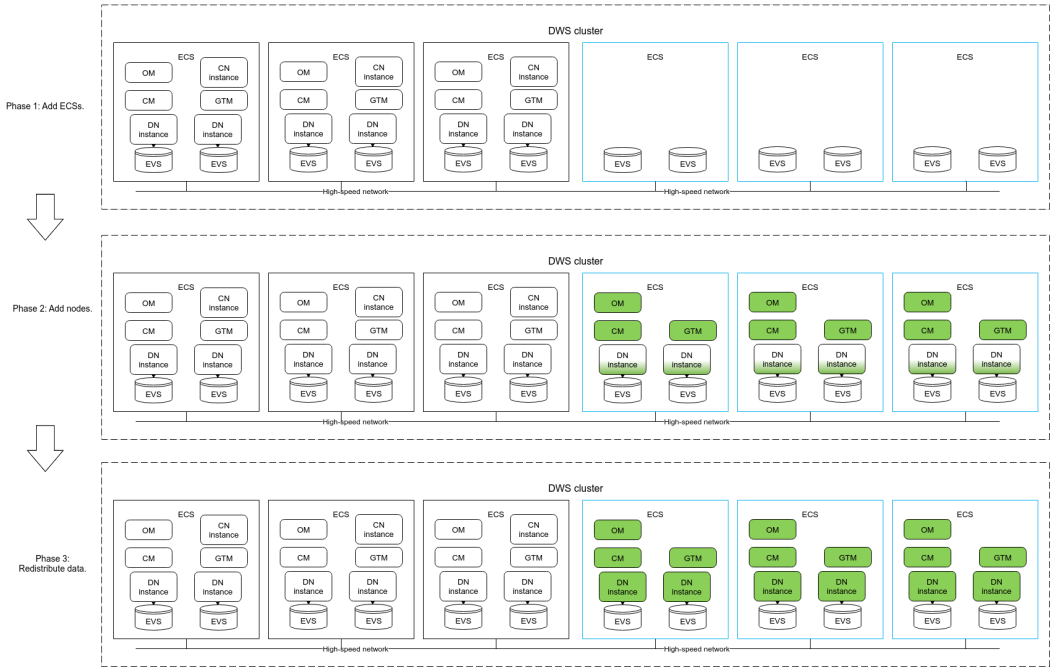


Scaling Out a Cluster

Cluster scale-out is a typical horizontal scaling scenario for MPPDBs, where homogeneous nodes are added to an existing cluster to increase capacity. GaussDB(DWS) 2.0 uses coupled storage and compute, so a cluster scale-out expands both compute and storage capacities.

To balance the load and achieve optimal performance, metadata replication and data redistribution are performed during a cluster scale-out. Therefore, the time needed to complete a cluster scale-out is positively correlated with the number of database objects as well as the data size. To ensure reliability, new nodes are automatically added to security rings. This is why at least three nodes must be added for a scale-out operation.

Figure 8-5 Scaling out a cluster



8.1.1 and later versions support online scale-out. **During an online scale-out, GaussDB(DWS) does not restart and can continue to provide services.** During

data redistribution, you can perform insert, update, and delete operations on tables, but data updates may still be blocked for a short period of time. Redistribution consumes large quantities of CPU and I/O resources, significantly impacting job performance. Therefore, you are advised to perform redistribution when services are stopped or during periods of light load. A phase-by-phase approach is recommended for cluster scale-out: Perform high-concurrency redistribution during periods of light load, and stop redistribution or perform low-concurrency redistribution during periods of heavy load.

Cluster scale-out can be performed phase by phase or in one-click mode.

A phase-by-phase approach separates a scale-out operation into three phases: adding ECSs, adding nodes, and data redistribution. You can schedule the scale-out tasks in a way that can minimize the risk of service interruption.

On the other hand, a one-click scale-out is more convenient to users.

Table 8-1 Comparing two different scale-out approaches

Approach	Characteristics	Impact
Phase-by-phase scale-out	A scale-out operation is divided into three phases: adding ECSs, adding nodes, and data redistribution. You can schedule each phase for the most appropriate times and perform them separately.	The risk of service interruption can be minimized.
One-click scale-out	During a one-click scale-out, adding ECSs, adding nodes, and redistributing data are all performed automatically.	It is more convenient to users.

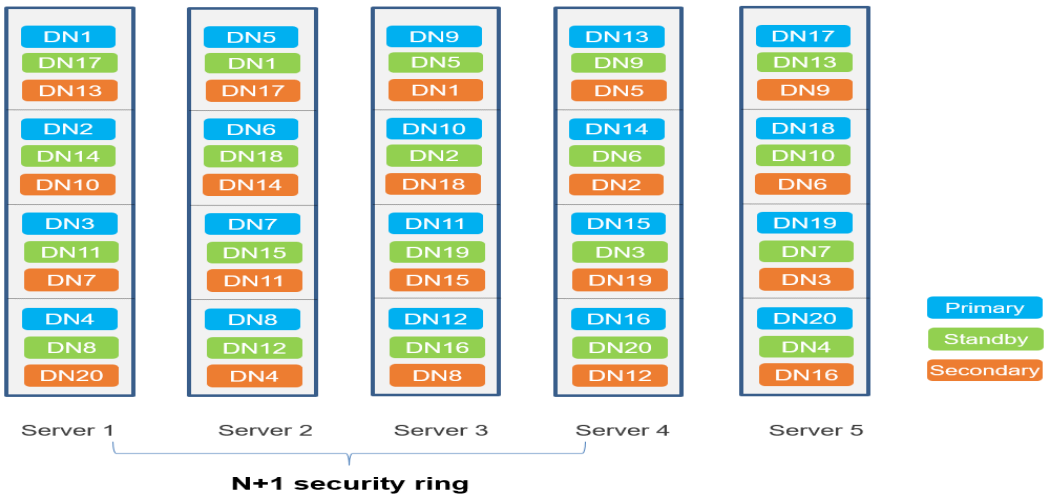
GaussDB(DWS) Cluster Security Ring

A security ring is the minimum set of nodes required for the horizontal deployment of multi-replica DNs. Cluster scale-out and scale-in are both performed by security ring. The main idea behind security rings is fault isolation. Any fault that occurs within a security ring stays within that ring.

GaussDB(DWS) uses a primary-standby-secondary architecture, so the minimum number of nodes in a security ring is **3**. When a fault occurs within a ring, it has no impact on nodes outside that ring. The scope of impact is minimized (3 nodes), and the impact on each node in that faulty ring is 1/(N-1), that is, 1/2. In extreme scenarios, the entire cluster is a security ring. If a fault occurs within this ring, the scope of impact is the largest (the entire cluster), but the impact on each node in the ring is the smallest, that is, 1/(N-1).

A common practice is to form an **N+1 ring**, where each node evenly distributes its N replicas to the remaining N nodes in the same ring. When a fault occurs in the ring, the scope of impact in the entire cluster is N+1 nodes, and the impact on each node in the ring is 1/N.

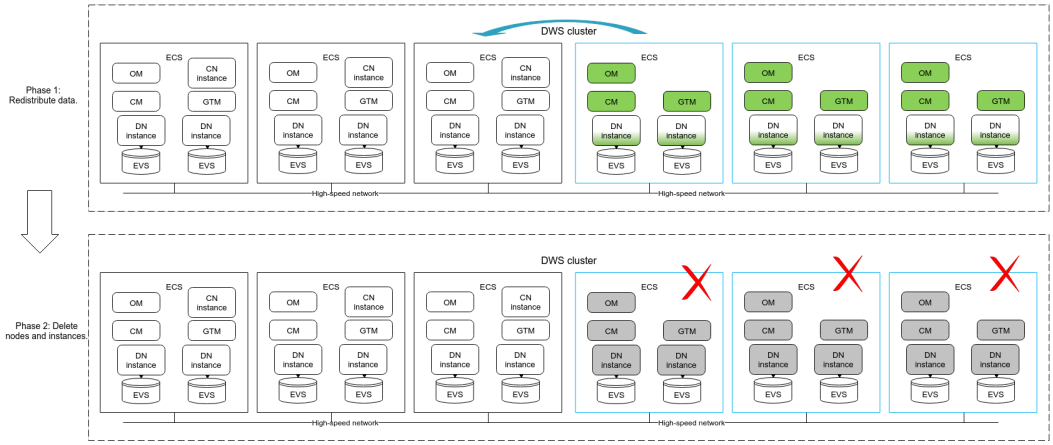
Figure 8-6 Typical N+1 security ring



Scaling In a Cluster

- Cluster scale-in is also a typical horizontal scaling scenario for MPPDBs, where some of the nodes of an existing cluster are removed to reduce capacity. A cluster scale-in reduces both compute and storage capacities.
- Each GaussDB(DWS) cluster physically consists of multiple ECSs. To improve reliability, a set number of ECSs (typically three) form a logical security ring, so each GaussDB(DWS) cluster consists of a number of security rings. **A cluster scale-in is performed by security ring. The security rings at the end of a cluster are first removed.**
- A cluster scale-in involves data migration. Data on the removed nodes needs to be redistributed to the remaining nodes. This means the time needed to complete a cluster scale-in is positively correlated with the number of database objects as well as the data size.
- GaussDB(DWS) standard data warehouses and stream data warehouses support cluster scale-in. 8.1.1.300 and later versions support online scale-in. **During an online scale-in, GaussDB(DWS) does not restart and can continue to provide services**. During data redistribution, you can perform insert, update, and delete operations on tables, but data updates may still be blocked for a short period of time. Redistribution consumes large quantities of CPU and I/O resources, significantly impacting job performance. Therefore, you are advised to perform redistribution when services are stopped or during periods of light load.

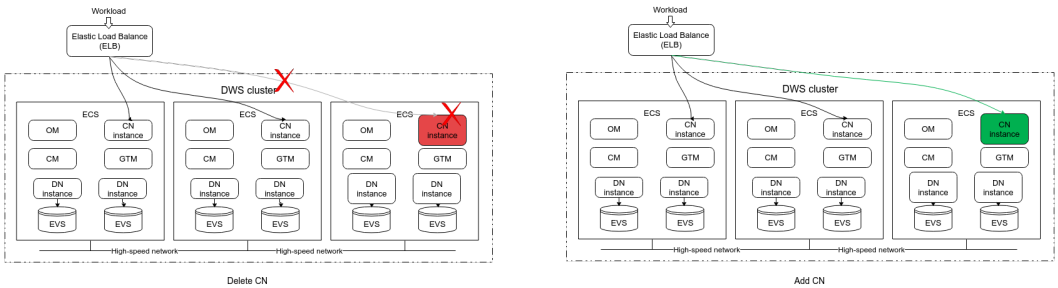
Figure 8-7 Scaling in a cluster



Adding or Deleting CNs

- Adding or deleting coordinator nodes (CNs) is another way of cluster scaling in GaussDB(DWS).
- CNs are an important component of GaussDB(DWS). It provides interfaces to external applications, optimizes global execution plans, distributes execution plans to data nodes (DNs), and summarizes results from each node into a single result set.
- CN capacities determine the entire cluster's concurrency handling capability. By adding more CNs, you increase the cluster's concurrency handling capability.
- CNs use a multi-active architecture. To ensure data consistency, if data on some CNs is damaged, DDL services will be blocked. To quickly restore DDL services, you can remove the faulty CNs.
- In 8.1.1 and later versions, GaussDB(DWS) standard data warehouses and stream data warehouses support this operation.
- When a CN is added, metadata needs to synchronized. The time it takes to add a CN depends on the metadata size. In 8.1.3, CNs can be added and deleted online. **During CN addition, GaussDB(DWS) does not restart and can continue to provide services**. DDL services will be blocked for a short period of time (with no error reported). No other services are affected.

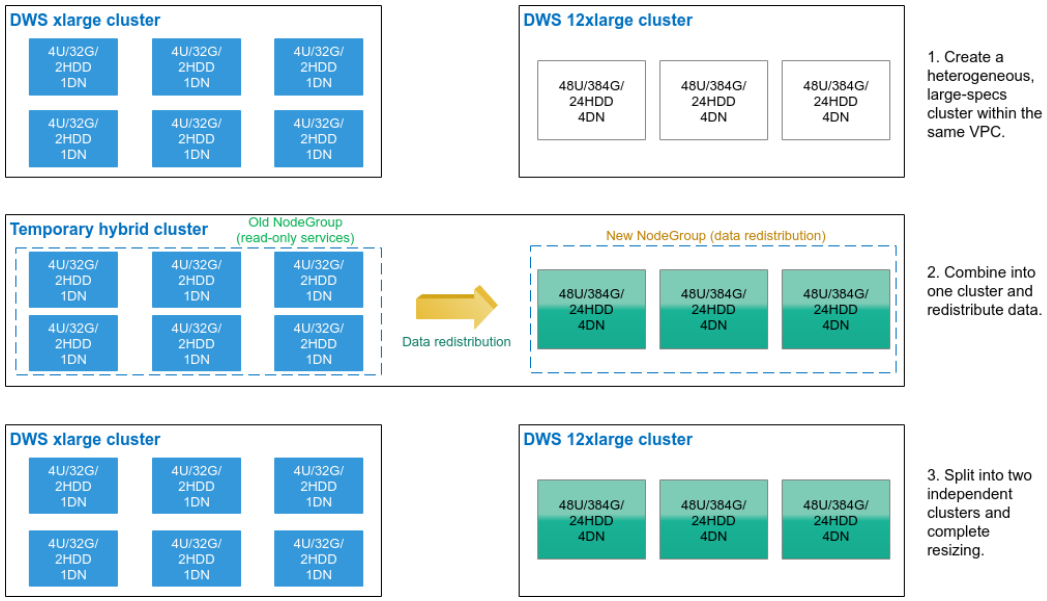
Figure 8-8 Adding or deleting a CN



Resizing a Cluster

- Cluster resizing allows you to perform horizontal and vertical scaling at the same time, including cluster scale-out and scale-in, as well as scale-up and scale-down. The cluster topology can also be adjusted.
- Clustering resizing relies on multiple node groups and data redistribution. During cluster resizing, a new cluster is created based on new resource requirements and cluster planning. Then, data is redistributed between the old and new clusters. Once data migration is complete, services are migrated to the new cluster, and after that, the old cluster is released.
- Cluster resizing involves data migration. Data on the nodes in the old cluster needs to be redistributed to the nodes in the new cluster, with the data still available in the old cluster. The time it takes to resize a cluster is positively correlated with the number of database objects as well as the data size.
- GaussDB(DWS) standard data warehouses support cluster resizing, but agents must be upgraded to 8.2.0.2. **Currently, during cluster resizing, the old cluster can only support read-only services.** Online service capabilities can be expected later.
- For details, see **Changing All Specifications**.

Figure 8-9 Resizing a cluster



Comparing Different Scaling Options

The table below compares different scaling options for GaussDB(DWS).

Table 8-2 Comparing different scaling options for GaussDB(DWS)

Option	Scaled Object	Scope	Impact	Product
Disk scaling	Disk capacity	EVS disks attached to all ECSs in a cluster	Can be completed within 5 to 10 minutes. **There is no need to restart services, so it has no impact on services**. Should be performed during off-peak hours.	Cluster version: 8.1.1.203 or later Product form: standard data warehouse and stream data warehouse
Changing the node flavor	Compute capacity	The flavor (CPU cores and memory size) of all ECSs in a cluster	Can be completed within 5 to 10 minutes. **Services will need to be restarted once, causing a service interruption in minutes**. Should be performed during off-peak hours.	Cluster version: 8.1.1.300 or later Product form: standard data warehouse and stream data warehouse
Cluster scale-out	Disk and compute capacities	Adding homogeneous ECSs in a distributed architecture	Online scale-out supported. **During an online scale-out, GaussDB(DWS) does not restart and can continue to provide services.** The duration is positively correlated with the number of database objects as well as the data size.	Cluster version: all versions. Online scale-out is supported since 8.1.1. Product form: standard data warehouse and stream data warehouse

Option	Scaled Object	Scope	Impact	Product
Cluster scale-in	Disk and compute capacities	Removing some of the ECSs in a distributed architecture	Online scale-in supported. **During an online scale-in, GaussDB(DWS) does not restart and can continue to provide services.** The duration is positively correlated with the number of database objects as well as the data size.	Cluster version: 8.1.1.300 Product form: standard data warehouse and stream data warehouse
Cluster resizing	Disk and compute capacities, and cluster topology	Using a new ECS flavor (new hardware specifications) and new cluster topology to create a new cluster, and redistributing data between the old and new clusters	The duration is positively correlated with the number of database objects as well as the data size. Read-only services can be provided during cluster resizing.	Cluster version: Agent 8.2.0.2 or later Product form: standard data warehouse
Adding or deleting CNs	CN instances	Adding CNs to enhance concurrency, or removing faulty CNs to quickly restore DDL services	Online addition and deletion of CNs is supported in 8.1.3 and later. **During CN addition, GaussDB(DWS) does not restart and can continue to provide services.**	Cluster version: 8.1.1. (Online addition and deletion of CNs is supported in 8.1.3 and later.) Product form: standard data warehouse and stream data warehouse

Application Scenarios for Different Scaling Options

Table 8-3 describes when to use each scaling option.

Table 8-3 Application scenarios for different scaling options for GaussDB(DWS)

Category	Problem to Solve	Recommended Scaling Option	Impact on Services	Estimated Duration
Storage	Insufficient storage space. CPU, memory, and disk I/O capacities are sufficient.	Increase disk capacity.	Online services can be maintained.	No need for data migration. Can be completed within 5 to 10 minutes.
	Excessive storage space, which needs to be reduced to cut costs. CPU, memory, and disk I/O capacities are sufficient.	Create a cluster with smaller disk capacity (but otherwise unchanged), and migrate data to the new cluster by performing a DR switchover.	Data becomes read-only during the DR switchover, which typically takes less than 30 minutes.	The duration is positively correlated with the data size.
Compute	Insufficient CPU or memory capacity	Use a larger ECS flavor.	The cluster needs to restart once.	No need for data migration. Can be completed within 5 to 10 minutes.
	Insufficient disk I/O	Create a cluster with smaller disk capacity (but otherwise unchanged), and migrate data to the new cluster by performing a DR switchover.	Data becomes read-only during the DR switchover, which typically takes less than 30 minutes.	The duration is positively correlated with the data size.
Distributed compute and storage	Insufficient distributed capabilities due to insufficient nodes	Scale out the cluster.	Online services can be maintained (partially impacted).	Data migration is needed. The duration is positively correlated with the sizes of metadata as well as service data.

Category	Problem to Solve	Recommended Scaling Option	Impact on Services	Estimated Duration
	Too many nodes, leading to a high cost	Scale in the cluster.	Online services can be maintained (partially impacted).	Data migration is needed. The duration is positively correlated with the size of service data.
Cluster topology	Change both the cluster topology and node flavor (the number of DNs changes).	Resizes the cluster.	Read-only services	Data migration is needed. The duration is positively correlated with the sizes of metadata as well as service data.
	Change both the cluster topology and node flavor (the number of DNs remains the same).	Perform cluster DR switchover and data migration	Online services can be maintained (partially impacted).	Data migration is needed. The duration is positively correlated with the size of service data.
	Insufficient concurrency support	Add CNs.	Online services can be maintained (partially impacted).	Data migration is needed. The duration is positively correlated with the size of metadata.

