

**Data Warehouse Service**

# **Best Practices**

**Issue** 35  
**Date** 2024-06-07



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI 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

---

<b>1 Import and Export.....</b>	<b>1</b>
1.1 Best Practices for Data Import.....	1
1.2 GDS Practice Guide.....	3
1.3 Tutorial: Importing Data from OBS to a Cluster.....	5
1.4 Tutorial: Using GDS to Import Data from a Remote Server.....	9
1.5 Tutorial: Viewing or Importing Data from MRS Hive.....	14
1.6 Tutorial: Importing Remote GaussDB(DWS) Data Sources.....	23
1.7 Tutorial: Exporting ORC Data to MRS.....	31
<b>2 Data Migration.....</b>	<b>38</b>
2.1 Migrating Data From Oracle to GaussDB(DWS).....	38
2.1.1 Migration Process.....	38
2.1.2 Required Tools.....	39
2.1.3 Migrating Table Definitions.....	40
2.1.3.1 Installing the PL/SQL Developer on the Local Host.....	40
2.1.3.2 Migrating Table Definitions and Syntax.....	41
2.1.4 Migrating Full Table Data.....	45
2.1.4.1 Configuring a GaussDB(DWS) Data Source Connection.....	45
2.1.4.2 Configuring an Oracle Data Source Connection.....	46
2.1.4.3 Migrating Tables.....	47
2.1.4.4 Verification.....	49
2.1.5 Migrating SQL Statements.....	49
2.1.5.1 Migrating Syntax.....	49
2.1.5.2 Verification.....	51
2.2 Synchronizing MySQL Table Data to GaussDB(DWS) in Real Time.....	51
2.3 Using DLI Flink Jobs to Write Kafka Data to GaussDB(DWS) in Real Time.....	60
2.4 Practice of Data Interconnection Between Two GaussDB(DWS) Clusters Based on GDS.....	80
<b>3 Table Optimization Practices.....</b>	<b>88</b>
3.1 Table Structure Design.....	88
3.2 Table Optimization Overview.....	94
3.3 Selecting a Table Model.....	94
3.4 Step 1: Creating an Initial Table and Loading Sample Data.....	95
3.5 Step 2: Testing System Performance of the Initial Table and Establishing a Baseline.....	100

3.6 Step 3: Optimizing a Table.....	104
3.7 Step 4: Creating Another Table and Loading Data.....	106
3.8 Step 5: Testing System Performance in the New Table.....	108
3.9 Step 6: Evaluating the Performance of the Optimized Table.....	110
3.10 Appendix: Table Creation Syntax.....	112
3.10.1 Usage.....	112
3.10.2 Creating an Initial Table.....	113
3.10.3 Creating a Another Table After Design Optimization.....	116
3.10.4 Creating a Foreign Table.....	120
<b>4 Advanced Features.....</b>	<b>126</b>
4.1 Creating a Time Series Table.....	126
4.2 Best Practices of Hot and Cold Data Management.....	132
4.3 Best Practices for Automatic Partition Management.....	137
4.4 GaussDB(DWS) View Decoupling and Automatic Rebuilding.....	143
4.5 Best Practices of Column-Store Delta Tables.....	146
4.6 Best Practices of GIN Index.....	149
<b>5 Database Management.....</b>	<b>152</b>
5.1 Best Practices of Resource Management.....	152
5.2 Excellent Practices for SQL Queries.....	157
5.3 Analyzing SQL Statements that Are Being Executed.....	158
5.4 Excellent Practices for Data Skew Queries.....	162
5.4.1 Real-Time Detection of Storage Skew During Data Import.....	162
5.4.2 Quickly Locating the Tables That Cause Data Skew.....	163
5.5 Best Practices for User Management.....	165
5.6 Viewing Table and Database Information.....	169
5.7 Best Practices of Database SEQUENCE.....	175
<b>6 Sample Data Analysis.....</b>	<b>181</b>
6.1 Checkpoint Vehicle Analysis.....	181
6.2 Supply Chain Requirement Analysis of a Company.....	187
6.3 Operations Status Analysis of a Retail Department Store.....	196
<b>7 Security Management.....</b>	<b>206</b>
7.1 Role-based Access Control (RBAC).....	206
7.2 Encrypting and Decrypting Data Columns.....	209
7.3 Managing Data Permissions Through Views.....	212
7.4 Configuring Read-Only Permissions.....	214

# 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 [Suggestions for SMP Parameter Settings](#).
- 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. For details about how to create a table, see [Creating a Table](#).

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 details about how to use the COPY statement, see [Running the COPY FROM STDIN Statement to Import Data](#).

## Using a `gsql` Meta-Command to Import Data

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

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 details about how to use the `\copy` command, see [Using the `\copy` 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 GDS Practice Guide

- Before installing GDS, ensure that the system parameters of the server where GDS is deployed are consistent with those of the database cluster.
- Ensure the physical network works properly for communication between GDS and GaussDB(DWS). A 10GE network is recommended. The 1GE network cannot guarantee smooth communication between GDS and GaussDB(DWS), because it cannot bear the high-speed data transmission pressure and is prone to disconnection. 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).
- Plan service deployment in advance. It is recommended that one or two GDSs be deployed on a RAID of a data server. It is recommended that the ratio of GDS quantity to DN quantity be in the range of 1:3 to 1:6. Do not deploy too many GDS processes on a loader. Deploy only one GDS process if a 1GE NIC is used, and no more than four GDS processes if a 10GE NIC is used.
- Hierarchically divide the data directories for data imported and exported by GDS in advance. Do not put too many files under a data directory, and delete expired files in a timely manner.
- Properly plan the character set of the target database. You are advised to use UTF8 instead of the SQL\_ASCII characters which can easily incur mixed encoding. When exporting data using GDS, ensure that the character set of the foreign table is the same as that of the client. When importing data, ensure that the client and data file content use the same encoding method.
- If the character set of the database, client, or foreign table cannot be changed, run the `iconv` command to manually change the character set.  
#Note: `-f` indicates the character set of the source file, and `-t` indicates the target character set.  
`iconv -f utf8 -t gbk utf8.txt -o gbk.txt`
- For details about GDS import practices, see [Using GDS to Import Data](#).
- GDS supports CSV, TEXT, and FIXED formats. The default format is TEXT. The binary format is not supported. However, the encode/decode function can be used to process data of the binary type. Example:  
Export a binary table.

```
-- Create a table.
CREATE TABLE blob_type_t1
(
  BT_COL BYTEA
) DISTRIBUTE BY REPLICATION;
-- Create a foreign table.
CREATE FOREIGN TABLE f_blob_type_t1( BT_COL text ) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://127.0.0.1:7789/', FORMAT 'text', DELIMITER E'\x08', NULL '', EOL '0x0a' ) WRITE ONLY;
INSERT INTO blob_type_t1 VALUES(E'\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\xDEADBEEF');
INSERT INTO f_blob_type_t1 select encode(BT_COL,'base64') from blob_type_t1;
```

### Import a binary table.

```
-- Create a table.
CREATE TABLE blob_type_t2
(
  BT_COL BYTEA
) DISTRIBUTE BY REPLICATION;
-- Create a foreign table.
CREATE FOREIGN TABLE f_blob_type_t2( BT_COL text ) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://127.0.0.1:7789/f_blob_type_t1.dat.0', FORMAT 'text', DELIMITER E'\x08', NULL '', EOL '0x0a' );
insert into blob_type_t2 select decode(BT_COL,'base64') from f_blob_type_t2;
SELECT * FROM blob_type_t2;
  bt_col
-----
\xdeadbeef
\xdeadbeef
\xdeadbeef
\xdeadbeef
(4 rows)
```

- Do not repeatedly export data from the same foreign table. Otherwise, the previously exported file will be overwritten.
- If you are not sure whether the file is in the standard CSV format, you are advised to set **quote** parameter to invisible characters such as **0x07**, **0x08**, or **0x1b** to import and export data using GDS. This prevents task failures caused by incorrect file format.

```
CREATE FOREIGN TABLE foreign_HR_staffs_ft1
(
  MANAGER_ID NUMBER(6),
  section_ID NUMBER(4)
) SERVER gsmpp_server OPTIONS (location 'file:///input_data/*', format 'csv', mode 'private', quote
'0x07', delimiter ',') WITH err_HR_staffs_ft1;
```

- GDS supports concurrent import and export. The **gds -t** parameter is used to set the size of the thread pool and control the maximum number of concurrent working threads. But it does not accelerate a single SQL task. The default value of **gds -t** is **8**, and the upper limit is **200**. When using the pipe function to import and export data, ensure that the value of **-t** is greater than or equal to the number of concurrent services.
- When specifying a multi-character delimiter for a GDS foreign table, confirm that each character is distinct in TEXT format. For example, delimiter '---' is not recommended.
- GDS imports a single file through multiple tables in parallel to improve data import performance. (Only CSV and TXT files can be imported.)

```
-- Create a target table.
CREATE TABLE pipegds_widetb_1 (city integer, tel_num varchar(16), card_code varchar(15),
phone_code vcreate table pipegds_widetb_3 (city integer, tel_num varchar(16), card_code varchar(15),
phone_code varchar(16), region_code varchar(6), station_id varchar(10), tmsi varchar(20), rec_date
integer(6), rec_time integer(6), rec_type numeric(2), switch_id varchar(15), attach_city varchar(6),
opc varchar(20), dpc varchar(20));
```

```
-- Create a foreign table that contains the file_sequence column.
CREATE FOREIGN TABLE gds_pip_csv_r_1( like pipegds_widetb_1) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+', NULL '', file_sequence
'5-1');

CREATE FOREIGN TABLE gds_pip_csv_r_2( like pipegds_widetb_1) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+', NULL '', file_sequence
'5-2');

CREATE FOREIGN TABLE gds_pip_csv_r_3( like pipegds_widetb_1) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+', NULL '', file_sequence
'5-3');

CREATE FOREIGN TABLE gds_pip_csv_r_4( like pipegds_widetb_1) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+', NULL '', file_sequence
'5-4');

CREATE FOREIGN TABLE gds_pip_csv_r_5( like pipegds_widetb_1) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+', NULL '', file_sequence
'5-5');

-- Import the wide_tb.txt file to the pipegds_widetb_1 table in parallel.
\parallel on
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_1;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_2;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_3;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_4;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_5;
\parallel off
```

For details about **file\_sequence**, see [CREATE FOREIGN TABLE \(for GDS Import and Export\)](#).

## 1.3 Tutorial: Importing Data from OBS to a 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 are in the same region as the GaussDB(DWS) cluster. 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 Console Operation 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 [Uploading an Object](#) in the *Object Storage Service Console Operation Guide*.

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

# 1.4 Tutorial: Using GDS to Import Data from a Remote Server

## 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 DWS are in the same region, VPC, and subnet.
- The ECS security group rule must allow access to the 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, click **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"
80,"GKLW-L",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."
30,"HWECL",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;
```

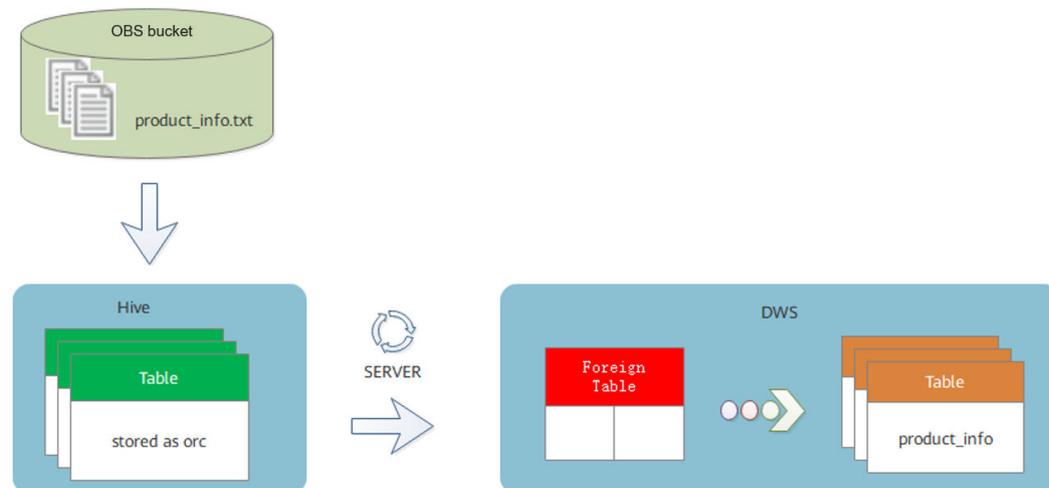
If the following information is displayed, the table has been deleted:

```
DROP FOREIGN TABLE
```

----End

## 1.5 Tutorial: Viewing or Importing Data from MRS Hive

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) <b>NOTE</b> <ul style="list-style-type: none"><li>• 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).</li><li>• 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).</li></ul>
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

Parameter	Value
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 <b>admin</b> 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 <b>root</b> again.
Secure Communications	Select <b>Enable</b> .

----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

Parameter	Value
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.
- 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.
  - 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.
  - Choose **Cluster > Name of the desired cluster > Dashboard > More > Download Client**. The **Download Cluster Client** dialog box is displayed.

### Download Cluster Client

Download the  client. The cluster client provides all services.

Select Client Type:  **Complete Client**  Configuration Files Only

Select Platform Type:  x86\_64  aarch64

Save to Path:  

OK

Cancel

### 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.
- Use SSH to log in to the preceding node as user **root**. Run the following command to switch to user **omm**:  
**su - omm**
  - 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 Data Source 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**.

**Create MRS Cluster Connection** ×

\* MRS Data Source  ? ↻ [View MRS Cluster](#)  
Kerberos Authentication: Disabled

\* MRS Account  ?

\* Password  ?

Description  ?  
0/256

----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	srvfwdw	srvtype	srversion	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)

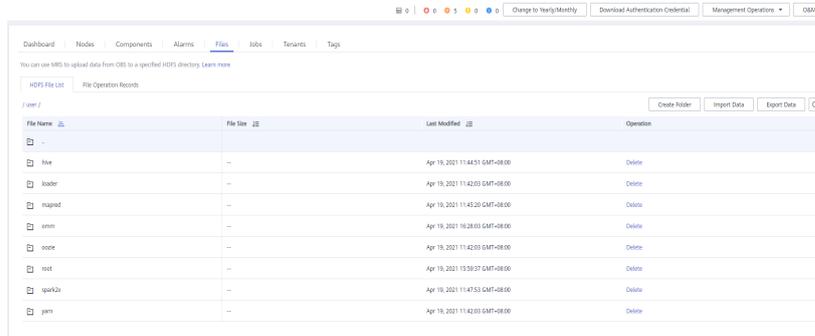
----End

## 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

## 1.6 Tutorial: Importing Remote GaussDB(DWS) Data Sources

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 1-5](#).

**Table 1-5** Software configuration

Parameter	Configuration
Region	Select CN-Hong Kong . <b>NOTE</b> <ul style="list-style-type: none"><li>• <b>CN-Hong Kong</b> is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region.</li><li>• Ensure that GaussDB(DWS) and the ECS are in the same region, AZ, and VPC subnet.</li></ul>
AZ	AZ2
Resource	Standard data warehouse
Compute Resource	ECS

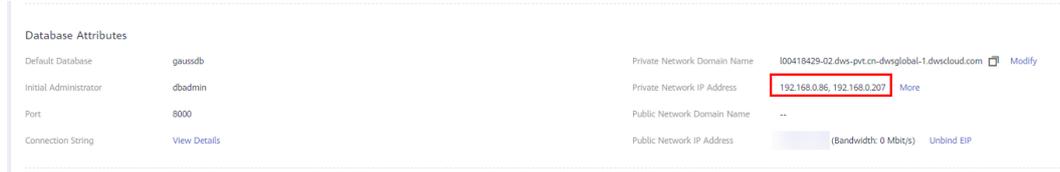
Parameter	Configuration
Storage Type	Cloud SSD
CPU Architecture	x86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPUs   128 GB   2000 GB SSD) <b>NOTE</b> If this flavor is sold out, select other AZs or flavors.
Hot Storage	100 GB/node
Nodes	3
Cluster Name	dws-demo01
Administrator Account	dbadmin
Administrator Password	User-defined password
Confirm Password	password
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24) <b>NOTICE</b> 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**.

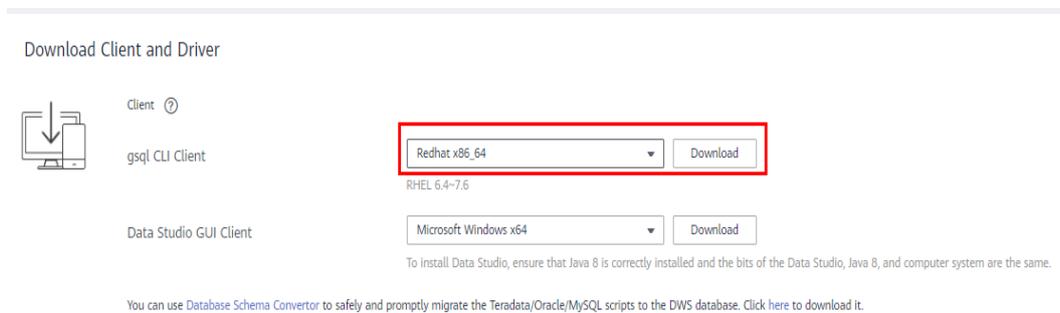
**Step 5** Wait for about 10 minutes. After the cluster is created, click the cluster name to go to the **Basic Information** page. Choose **Network**, click a security group name, and verify that a security group rule has been added. In this example, the client IP address is 192.168.0.x (the private 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 homepage of the GaussDB(DWS) console. Choose **Connections** in the navigation pane on the left, select the ECS OS (for example, select **Redhat x86\_64** for CentOS 7.6), and click **Download** to save the tool package to the local host. 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 Tutorial: Exporting ORC Data to MRS

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 1-6** Software configuration

Parameter	Example Value
Region	CN-Hong Kong
Cluster Name	mrs_01

Parameter	Example Value
Cluster Version	MRS 1.9.2 (recommended) <b>NOTE</b> <ul style="list-style-type: none"> <li>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).</li> <li>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).</li> </ul>
Cluster Type	Analysis Cluster

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

**Table 1-7** 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 1-8** 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.

Parameter	Example Value
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.
User Name	admin
Password	This password is used to log in to the cluster management page.
Confirm Password	Enter the password of user <b>admin</b> 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 <b>root</b> again.
Secure Communications	Select <b>Enable</b> .

----End

## Creating an MRS Data Source 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**.

## Create MRS Cluster Connection

\* MRS Data Source  ? ↻ [View MRS Cluster](#)

Kerberos Authentication: Enabled

\* MRS Account

\* Password

Description  ?

0/256

----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 | srfdw | srvtype | srvversion | srvacl
-----+-----+-----+-----+-----+-----
          |          |      |        |           |         |
+-----+-----+-----+-----+-----+-----
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 | srfdw | srvtype | srvversion | srvacl
-----+-----+-----+-----+-----+-----
          |          |      |        |           |         |
+-----+-----+-----+-----+-----+-----
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.

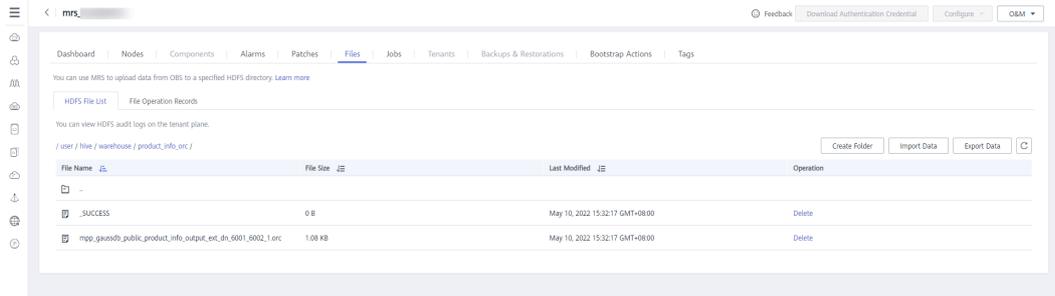
```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

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

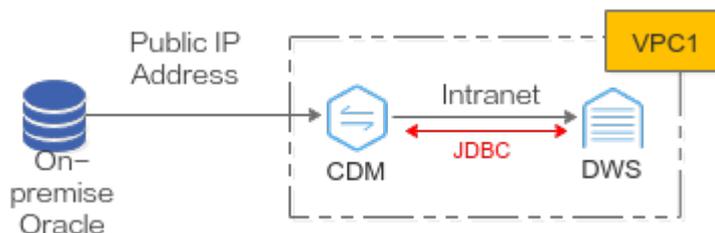
# 2 Data Migration

## 2.1 Migrating Data From Oracle to GaussDB(DWS)

### 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.

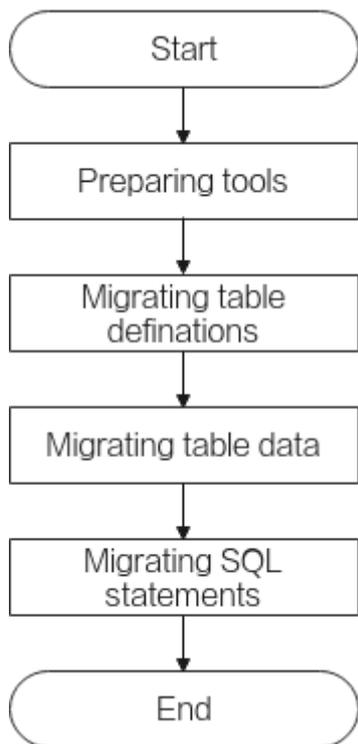
Figure 2-1 Migration scenario



#### 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
<b>Required Tools</b>	Software tools to be prepared before the migration.
<b>Migrating Table Definitions</b>	Use the PL/SQL Developer to migrate table definitions.
<b>Migrating Full Table Data</b>	Use Huawei Cloud Data Migration Service (CDM) to migrate data.
<b>Migrating SQL Statements</b>	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	<a href="#">PL/SQL Developer download address</a>

Tool	Description	Download Address
Oracle Instant Client	Oracle client	<a href="#">Instant Client download address</a>
DSC	Syntax migration tool for GaussDB(DWS)	<a href="#">DSC Download Address</a>

## 2.1.3 Migrating Table Definitions

### 2.1.3.1 Installing the PL/SQL Developer on the Local Host

#### Procedure

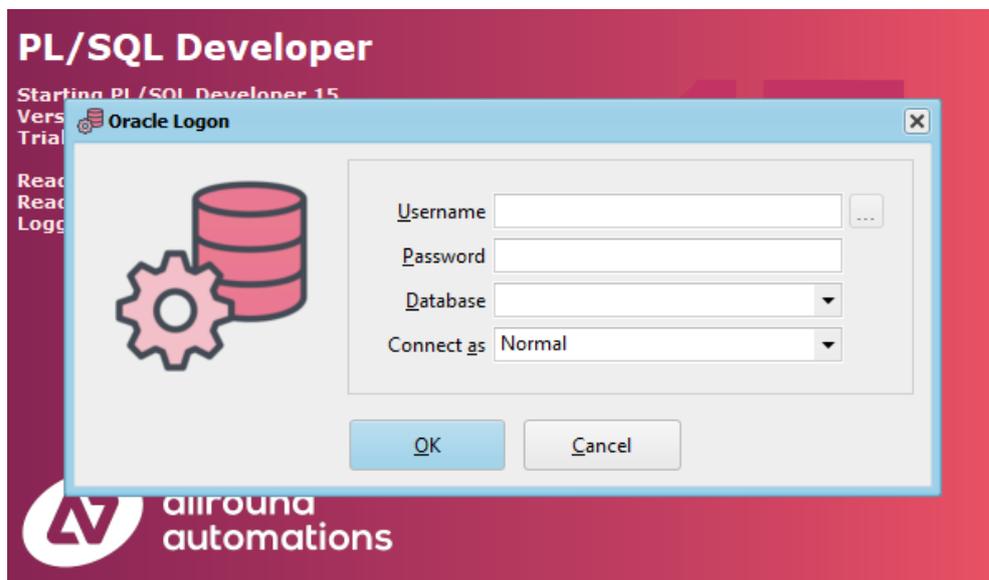
**Step 1** Decompress the PL/SQL Developer, Instant Client, and DSC packages.

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

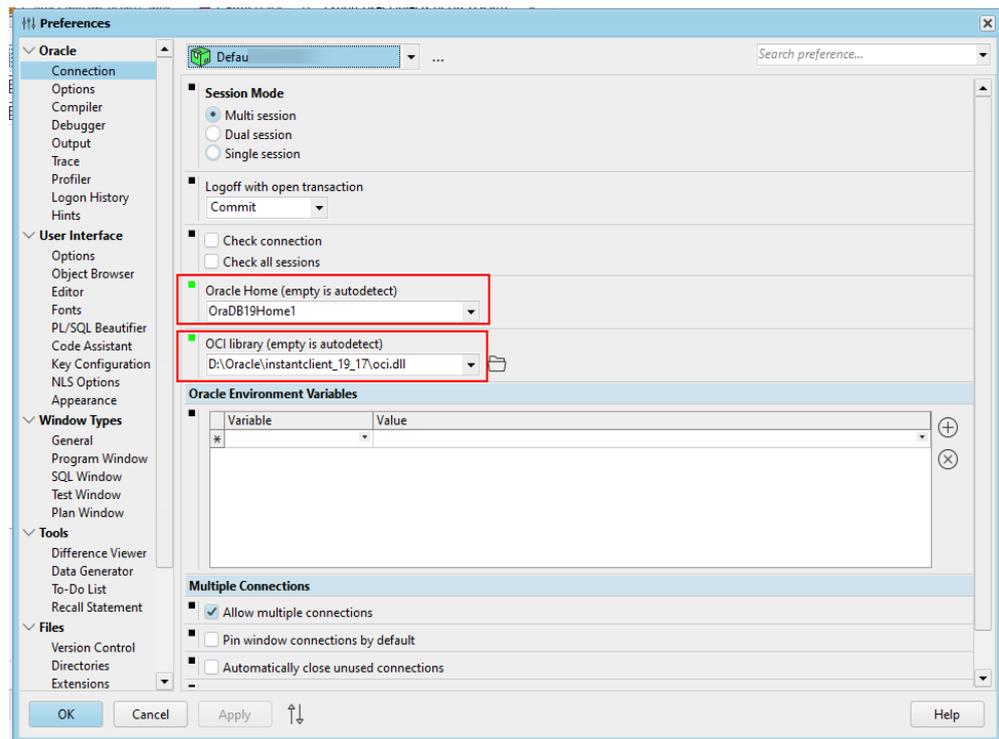
 **NOTE**

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

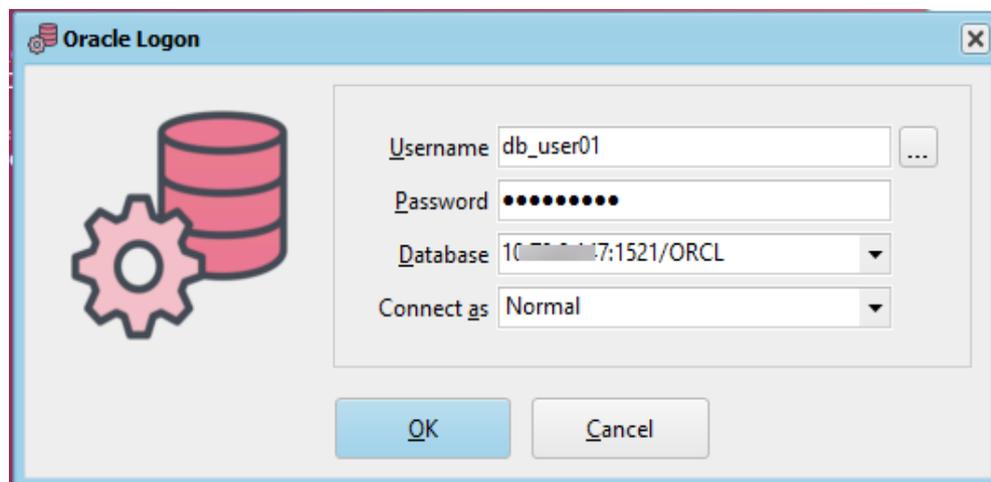
1. On the login page, click Cancel.



2. Choose **Configure > Preferences > Connection**, and add the Oracle Home and OCI 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, for example, xx.xx.xx.xx:1521/ORCL.



**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

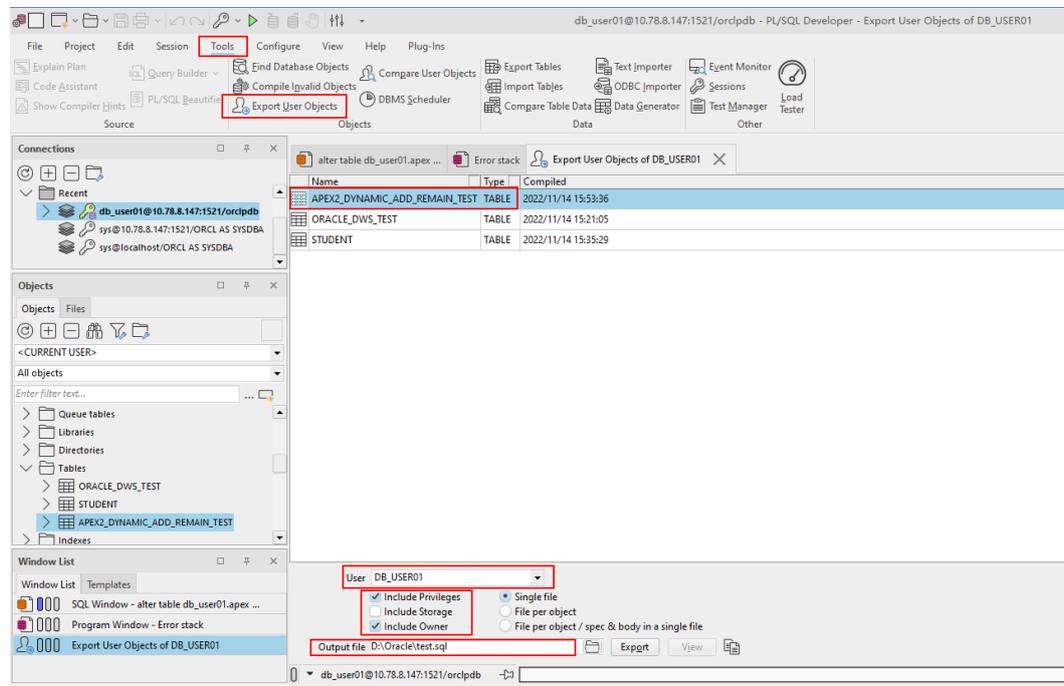
**Step 1** Log in to the PL/SQL Developer use an account with the **sysdba** permission. In this example, the account **db\_user01** is used.

#### NOTE

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

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

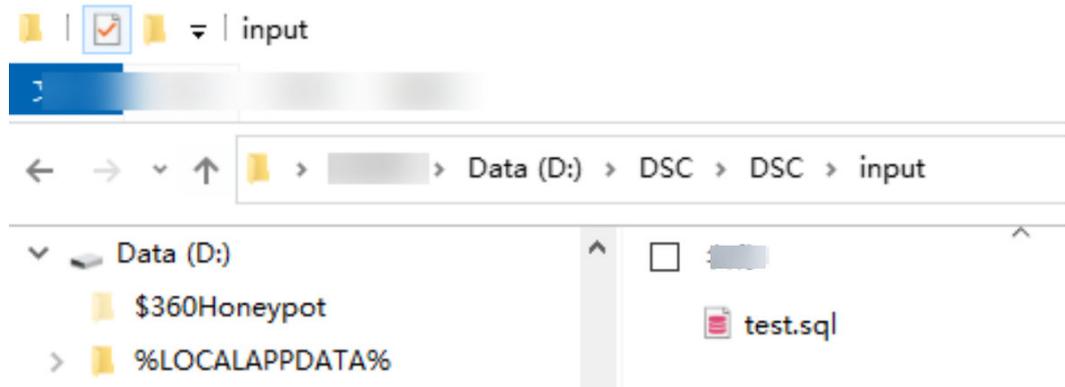
**Step 3** Select the logged-in user **db\_user01**, select the table object **APEX2\_DYNAMIC\_ADD\_REMAIN\_TEST** of the user, select the path to the output file (name the output SQL file as **test**), and click **Export**.



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.

```
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 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
1 prompt PL
2 /
3 SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by 1c 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 ;
```

- After the change:

```
1 prompt PL
2 /
3 SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by 1c 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 ) DISTRIBUTE BY HASH (ID);
19 \echo Done
20 /*spool off*/
21 SET define
22 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 `dwcj` 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(*) FROM 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 cluster and bind an EIP to the cluster. For details, see section [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 <b>Select</b> and select the GaussDB(DWS) cluster to be connected from the cluster list. <b>NOTE</b> 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 <b>dbadmin</b>

Parameter	Value
Use Agent	No

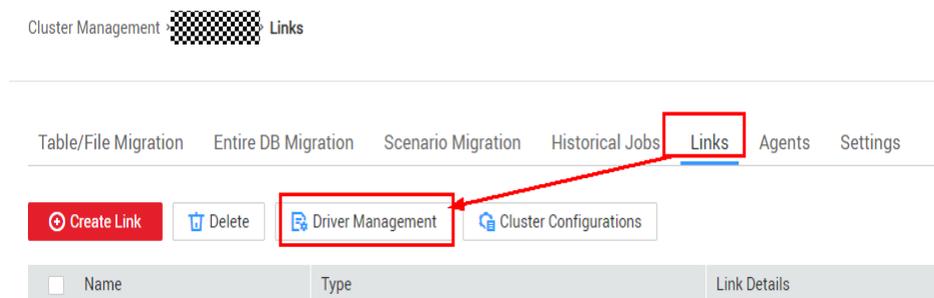
----End

### 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**.

Download Driver

Updated drivers take effect after the CDM cluster is restarted.

Driver Name	Driver Package Name	Driver Type	Description	Operation
MYSQL	None	Preset		Upload Copy from SFTP
ORACLE_6	None	Preset	oracle + 12.1	Upload Copy from SFTP
ORACLE_7	None	Preset	oracle + 12.1	Upload Copy from SFTP
ORACLE_8	None	Preset	oracle + 12.1	Upload Copy from SFTP
POSTGRESQL	None	Preset		Upload Copy from SFTP
DB2	None	Preset		Upload Copy from SFTP
SQLSERVER	None	Preset		Upload Copy from SFTP
DDM	None	Preset		Upload Copy from SFTP
MYCAT	None	Preset		Upload Copy from SFTP
DM	None	Preset		Upload Copy from SFTP

**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.

Job Configuration

\* Job Name

**Source Job Configuration**

\* Source Link Name  [Configuration Guide](#)

Use SQL Statement  Yes  No

\* Schema/Table Space

\* Table Name

[Show Advanced Attributes](#)

**Destination Job Configuration**

\* Destination Link Name  [Configuration Guide](#)

\* Schema/Table Space

Auto Table Creation

\* Table Name

Clear Data Before Import

Import Mode

[Show Advanced Attributes](#)

**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.

## Configure Task

Retry if failed ?

Group ?  + Add ✎ Edit 🗑 Delete

Schedule Execution

**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 SQL Statements

### 2.1.5.1 Migrating Syntax

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

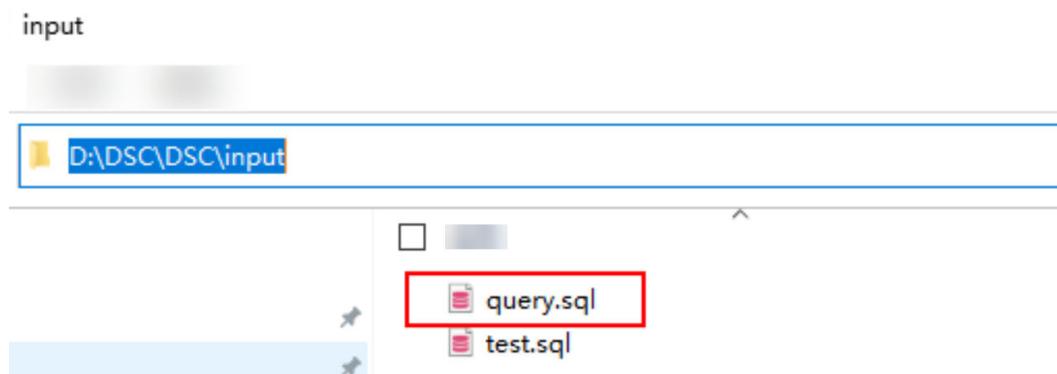
```
-- Generally, the HAVING clause must appear after the GROUP BY clause, but Oracle allows HAVING to appear before or after the GROUP BY clause. Therefore, you need to move the HAVING clause after the GROUP BY clause in the target database.
SELECT
id,
count(*),
sum(remain_users)
FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST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

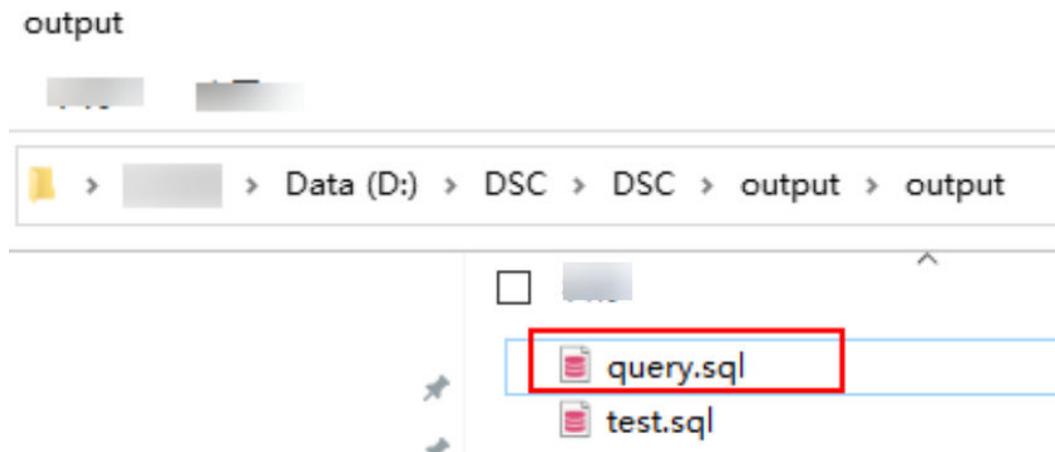
***** Schema Conversion Started *****

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 Synchronizing MySQL Table Data to GaussDB(DWS) 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?](#)

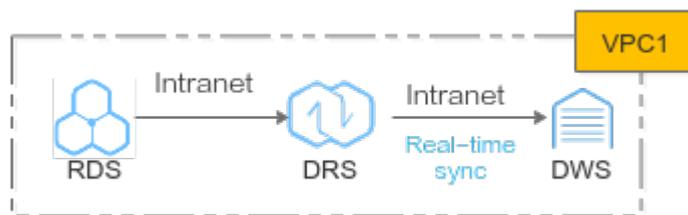
This practice takes about 60 minutes. The process is as follows:

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 2-3 DRS real-time synchronization



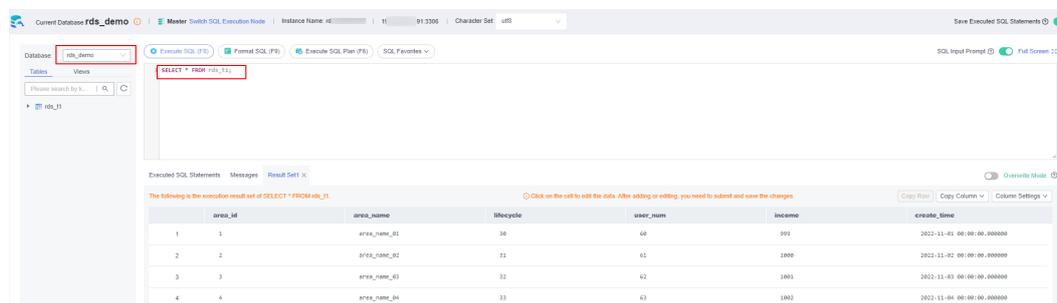
## Preparations

- 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** On the **Clusters** page of the GaussDB(DWS) console, 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**.

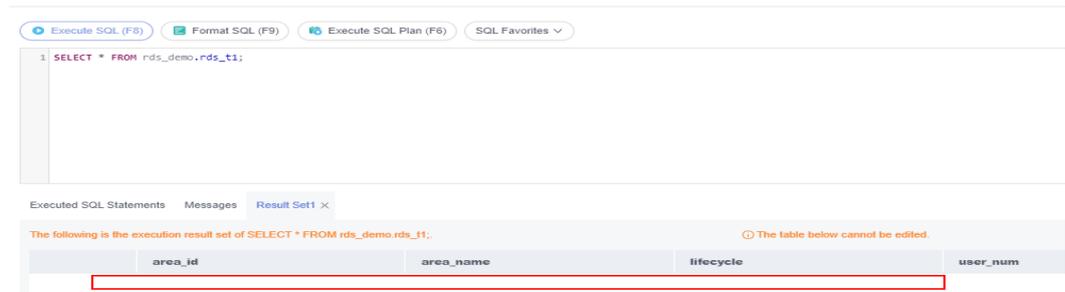
```
CREATE SCHEMA rds_demo;
```

**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.

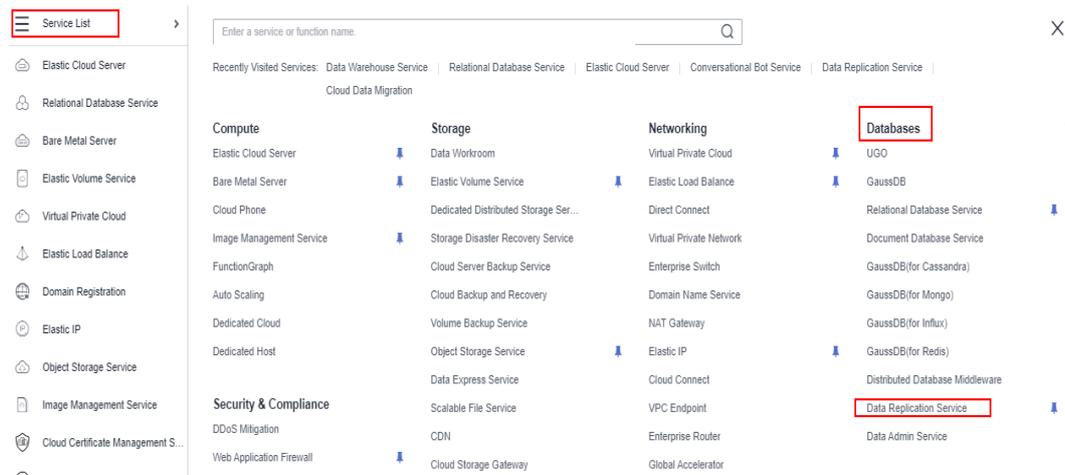
```
SELECT * FROM rds_demo.rds_t1;
```



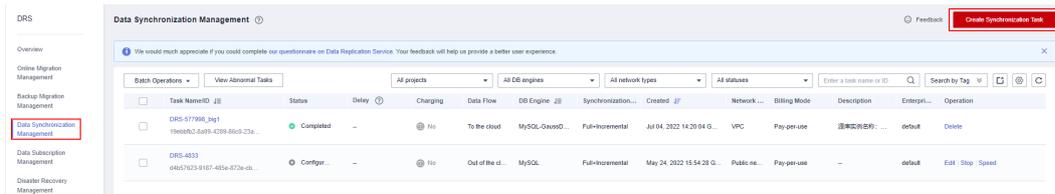
----End

### 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 2-7](#).

**Table 2-7** 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 2-8](#).

**Table 2-8** 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 <b>VPC</b> . If the MySQL database is offline, select <b>Public Network</b> .
Instance Type	Single
Destination DB Instance	Select the cluster created in <a href="#">Step 2: Create a GaussDB(DWS) Cluster</a> .
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 <b>Micro</b> . This option is selected based on the data volume and synchronization rate.

**Synchronization Instance Details** ⓘ

The following information cannot be modified after you go to the next page.

- Data Flow**:  To the cloud  Out of the cloud  Self-built to self-built  
The destination database must be a database in the current cloud. If you want to synchronize data between databases, select To the cloud.
- Source DB Engine**:  MySQL  Oracle  DB2  DDM  PostgreSQL
- Destination DB Engine**:  MySQL  GaussDB(DWS)  GaussDB(for openGauss) Distributed Edition  PostgreSQL  GaussDB(for MySQL) Primary/Standby Ed...  
The syntax of databases such as MySQL and Oracle is different from that of GaussDB(DWS). DDL statements may fail to be synchronized. Contact DWS experts to evaluate DDL support before the synchronization.
- Network Type**:  ⓘ
- Instance Type**:  Single  Primary/Standby
- Destination DB Instance**:
- Synchronization Instance Subnet**:  ⓘ
- Synchronization Mode**:  Full+Incremental  Full  Incremental  
This synchronization type synchronizes data in real time. After a full synchronization initializes the destination database, an incremental synchronization parses logs to ensure data consistency between the source and destination databases.

---

- Specifications**:  Micro  Small  Medium  Large  
Micro: up to 300 statements per second Small: up to 3,000 statements per second Medium: up to 7,500 statements per second Large: unlimited

---

- Enterprise Project**:   ⓘ

**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.

 **Notice** ✕

For details about GaussDB(DWS) synchronization solution selection, see [Solution Selection Recommendations](#).  
For details about GaussDB(DWS) synchronization suggestions, see [Failure Details and Support Evaluation](#).

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

**Table 2-9** Source database information

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

✔ Synchronization instance created (IP address [redacted] 17). Add the IP addresses to the whitelists of the source and destination d

### Source Database

System databases, users, parameters, and jobs will not be migrated. You need to manually import users and jobs to the destination database and configure paramete

Database Type: Self-built on ECS **RDS DB instance**

DB Instance Name:  [View DB Instance](#) [View Unselectable DB Instance](#)

Database Username:

Database Password:

✔ Test successful

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

**Table 2-10** Destination database information

Parameter	Value
Database Username	dbadmin
Database Password	*****

### Destination Database

DB Instance Name: DWS-... (192.1...:000)

Database Username:

Database Password:

✔ Test successful

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

**Step 9** Set the synchronization policy. For details, see [Table 2-11](#).

**Table 2-11** 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	<p>Tables</p> <p>Select the table to be synchronized from the source database. In this practice, select <b>rds_t1</b> under <b>rds_demo</b>.</p> <p>Enter the name of the GaussDB(DWS) database that data is synchronized to: <b>rds_demo</b></p>

Flow Control  Yes  No ?

---

Synchronization Object Type  Table structure  Data  Constraint  
When you manually create a table structure in the destination database, for details about the data type, see [Mapping Data Types](#).

Incremental Conflict Policy  Ignore  Report error  Overwrite ?  
If synchronized data conflicts with existing data (such as the source and destination databases containing the same primary or unique keys) in the destination database, the existing data will be overwritten.

Data Synchronization Topology  One-to-one  One-to-many  Many-to-one ?

Synchronize DDLs  Default  Custom ?  
During database-level synchronization, all DDL operations in the binlog related to database objects, except DDL related those to permissions, are synchronized to the destination. Common DDL statements are CREATE\_TABLE and RENAME\_TABLE. During table-level synchronization, only DDL operations in the binlog related to the selected tables are synchronized. Common DDL statements are ADD\_COLUMN, MODIFY\_COLUMN, and ALTER\_COLUMN.

Synchronization Object  Tables  Databases  Import object file  
Only some DDL statements can be synchronized. For details, see precautions of the current scenario in [Real-Time Synchronization > Before You Start](#).  
 If any data in the source database changes, click the refresh button below.  
 Move objects to be migrated from list of unselected objects on left side to the list of selected objects on right side.

Select All

For tables, only expanded databases are searched.

>

<

Select All

For tables, only expanded databases are searched.

rds\_demo X database

rds\_demo Edit

rds\_t1 Edit

>

<

**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**.

Check Again

Check success rate  100% All checks must pass before you can continue. If any check requires confirmation, check and confirm the results before proceeding to the next step.

Check Item	Check Result
<b>Database parameters</b>	
Whether a table without a primary key is selected for a synchronization object for initial object selection	✔ Passed
Whether source database tables contain unique keys	✔ Passed
Whether the source database contains tables with the same name	✔ Passed
Whether the selected source tables contain additional columns	✔ Passed
Whether the source database contains unsupported table field types	✔ Passed
Whether the compatible database type meets the requirements	✔ Passed
Whether the character set type is supported	✔ Passed
Whether the SSL connection is correctly configured	✔ Passed
Whether the source database binlog is row-based	✔ Passed
Whether the binlog_row_image value of the source database is FULL	✔ Passed
Whether the source database binlog is enabled	✔ Passed
Whether the source database name is valid	✔ Passed
Whether the source database server_id meets the incremental migration requirements	✔ Passed
Whether there are tables containing fields of the longtext or longblob type in the synchronization object	✔ Passed
Whether a table without a primary key is selected for a newly-added synchronization object when the task is edited again	✔ Passed

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

Start Time  Start upon task creation  Start at a specified time

Send Notifications

Stop Abnormal Tasks After  Abnormal tasks run longer than the period you set (unit: day) will automatically stop.

Details

Product Name	Configuration
<b>Task Information</b>	
Name	DRS-1668
Description	Source Database Instance Name: rds-00418429 Destination DB Instance Name: DWS-00419429
Synchronization Mode	Full-Incremental synchronization
Data Flow	To the cloud
Enterprise Project	default
<b>Synchronization Instance Details</b>	
Specifications	Micro
Source DB Engine	MySQL
Target DB Engine	GaussDB(DWS)
Network Type	VPC

Price: ¥0.80/hour

Previous

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

### Notice

**!** During the synchronization, do not perform any operations on the destination DB instance through the management console. To ensure migration success, we strongly recommend that you read the [migration precautions](#) carefully before starting migration tasks and follow the instructions to ensure migration stability.

**!** Any task that is active will be billed, even if its status becomes abnormal. If a task is no longer needed, stop the task to avoid unnecessary fees.

If the task status is abnormal for more than 14 days, the task automatically stops. Pay attention to the alarms you received and handle the task in time to resume the download and avoid task retry failure.

I have read the precautions.

**Submit**

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

Task Name/ID	Status	Delay	Charging	Data Flow	DB Engine	Synchronization...	Created	Network ...	Billing Mode	Description	Enterpri...	Operation
DR- d0009564-f793-442b-aa5-461...	Starting	--	No	To the cloud	MySQL-GaussD...	Full-Incremental	Nov 15, 2022 10:00:11 G...	VPC	Pay-per-Use Created on Nov ...	Source Database...	default	Stop

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

Task Name/ID	Status	Delay	Charging	Data Flow	DB Engine	Synchronization...	Created	Network ...	Billing Mode	Description	Enterpri...	Operation
d0009564-f793-442b-aa5-461...	Income...	0s	Yes	To the cloud	MySQL-GaussD...	Full-Incremental	Nov 15, 2022 10:00:11 G...	VPC	Pay-per-Use Created on Nov ...	Source Database...	default	Edit Stop More +

----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;
```

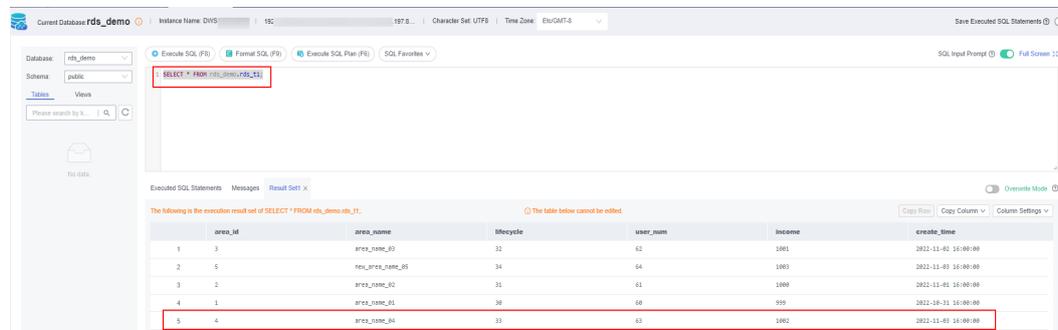
area_id	area_name	lifecycle	user_num	income	create_time
1	area_name_01	32	62	1001	2022-11-02 16:00:00
2	new_area_name_05	34	64	1003	2022-11-03 16:00:00
3	area_name_02	31	61	1000	2022-11-03 16:00:00
4	area_name_01	30	60	999	2022-10-31 16:00:00

**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;`



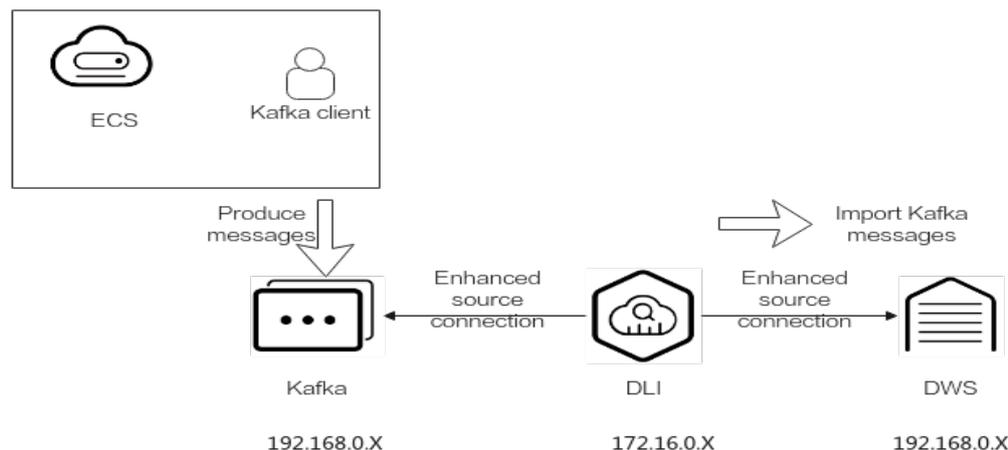
----End

## 2.3 Using DLI Flink Jobs to Write Kafka Data to GaussDB(DWS) 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 2-4** 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 2-12](#), 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 2-12** 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 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 2-13** 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 2-5 Creating a Kafka instance

Billing Mode: Yearly/Monthly, **Pay-per-use**

Region: [Region Selection]

Project: [Project Selection]

AZ: **AZ1**, **AZ2**, **AZ3**, AZ7

Select one AZ or at least three AZs. Do not select two AZs. [Learn more](#)  
The more AZs selected, the better the reliability and SLA coverage.  
AZs that support IPv6: AZ7

Instance Name: kafka-dli-dws

Enterprise Project: default [View Enterprise Project](#)

Specifications: **Default**, Custom

Version: **2.7**, 1.1.0

CPU Architecture: **x86**

Broker Flavor:

Flavor Name
<input checked="" type="radio"/> kafka.2u4g.cluster.small
<input type="radio"/> kafka.2u4g.cluster
<input type="radio"/> kafka.4u8g.cluster
<input type="radio"/> kafka.8u16g.cluster
<input type="radio"/> kafka.12u24g.cluster
<input type="radio"/> kafka.16u32g.cluster

To ensure stable services, choose a bandwidth 30% higher than what is required under normal conditions.

Currently Selected: kafka.2u4g.cluster.small | TPS Limit per Broker 20,000 | Maximum Partitions per Broker

Brokers: [ - ] 3 [ + ]

**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 2-6 Creating a topic

### Create Topic

Topic Name	<input type="text" value="topic-demo"/>
Partitions <span>?</span>	<input type="text" value="3"/> Value range: 1 to 100
Replicas	<input type="text" value="3"/> Value range: 1 to 3 Number of message copies.
Aging Time (h)	<input type="text" value="72"/> Value range: 1 to 720 Time after which data in the topic expires.
Synchronous Replication <span>?</span>	<input type="checkbox"/>
Synchronous Flushing <span>?</span>	<input type="checkbox"/>
message.timestamp.type <span>?</span>	<input type="text" value="LogAppendTime"/>
max.message.bytes <span>?</span>	<input type="text" value="10,485,760"/>

**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** On the **Clusters** page of the GaussDB(DWS) console, 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** The login username is **dbadmin**, the database name is **gaussdb**, and the password is the password of user **dbadmin** set during data warehouse cluster creation. Select **Remember Password**, enable **Collect Metadata Periodically** and **Show Executed SQL Statements**, and click **Log In**.

**Figure 2-7** Logging In to GaussDB(DWS)

**Instance Login Information**

DB Instance Name	dws- 	DB Engine Version	GaussDB(DWS) 8.1.3.320
------------------	--	-------------------	------------------------

\* Login Username:

\* Database Name:

\* Password:   ✔ Connection is successful.

Remember Password Your password will be encrypted and stored securely.

Collect Metadata Periodically  If not enabled, DAS can query the real-time structure information only from databases, which may affect the real-time performance of databases.

Show Executed SQL Statements  If not enabled, the executed SQL statements cannot be viewed, and you need to input each SQL statement manually.

- Step 5** Click the database name **gaussdb** and click **SQL Window** in the upper right corner to access the SQL editor.

- Step 6** 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 > Queue Management**.
- Step 3** Click **Buy Queue** 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 2-14** DLI queue parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	CN-Hong Kong
Project	Default
Name	dli_dws
Type	For a general queue, select <b>Dedicated Resource Mode</b> .
AZ Mode	Single-AZ deployment
Specifications	16 CUs
Enterprise Project	default
Advanced Settings	Custom
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.

Figure 2-8 Creating a DLI queue

Billing Mode: Yearly/Monthly | **Pay-per-use**

Billing for CUH used = Number of CUs x Usage duration x Unit price. You are billed for used CUs on an hourly basis (rounded up to the nearest hour). You can also buy a DLI package, which is more cost-effective.

Region: [Region Selection]

Project: [Project Selection]

---

\* Name: dli\_dvs

\* Type: For SQL | **For general purpose**

**Dedicated Resource Mode** ? If this is disabled, enhanced datasource connections cannot be created.

AZ Mode ?): Single AZ | Dual-AZ

Dual-AZ improves data availability by creating a duplicate queue in the second AZ, but at an increased cost (twice as much as that of single AZ mode).

\* Specifications ?): 16 CUs **Test** | 64 CUs **Production** | 256 CUs **Production** | 512 CUs **Production**

Each CU includes one core and 4 GB memory

\* Enterprise Project: default C ? Create Enterprise Project

Description: [Text Area] 0/128

---

Advanced Settings: Default | **Custom**

CIDR Block: **172.16.0.0** / 18 ?

If you need to use DLI enhanced datasource connections, the CIDR block entered here cannot be the same as that of the data source.  
Recommended CIDR blocks: 10.0.0.0-10.255.0.0/8-22, 172.16.0.0-172.31.0.0/12-22, 192.168.0.0-192.168.0.0/16-22

**Step 4** Click **Buy**.

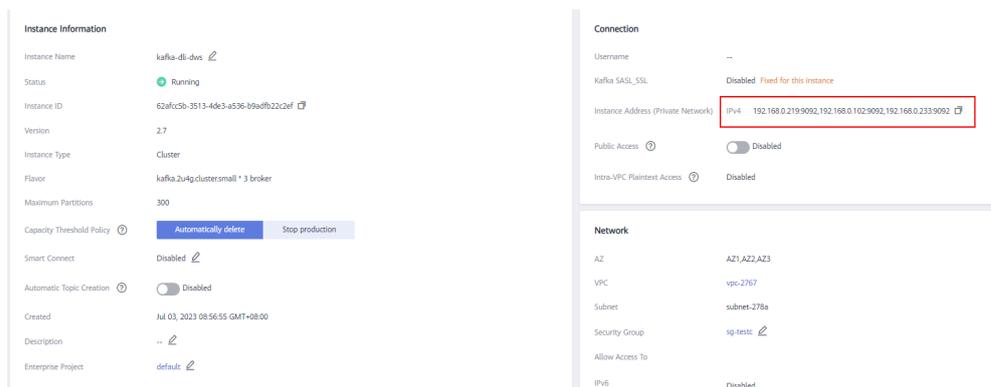
----End

## 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.

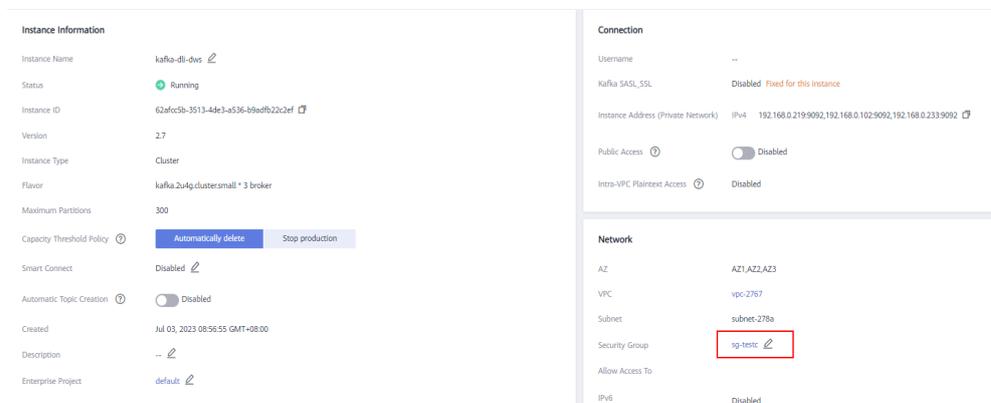
1. Return to the Kafka console and click the Kafka instance name to go to the **Basic Information** page. View the value of **Instance Address (Private Network)** in connection information and record the address for future use.

Figure 2-9 Kafka private network address



2. Click the security group name.

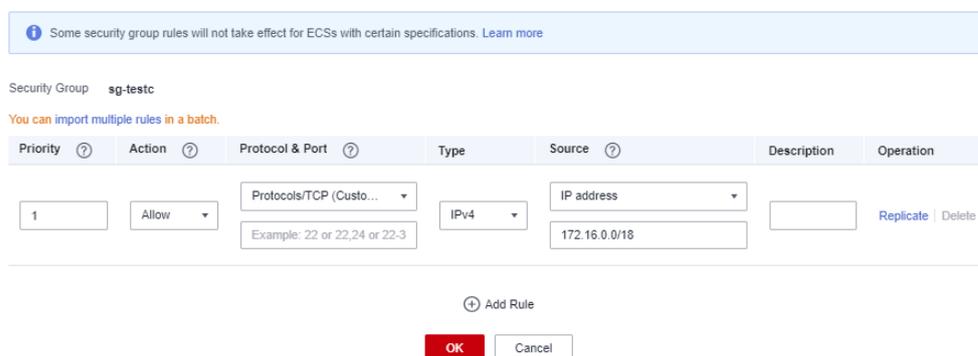
Figure 2-10 Kafka security group



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 2-11 Adding rules to the Kafka security group

Add Inbound Rule [Learn more about security group configuration.](#)



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 2-15** Connection from DLI to Kafka

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

**Figure 2-12** Creating an enhanced connection

### Create Enhanced Connection

After you create the enhanced datasource connection, the system will automatically create a VPC peering connection and required routes. [Learn more about how to connect DLI queues.](#)

★ Connection Name

Resource Pool

★ VPC

★ Subnet

Route Table

Host Information

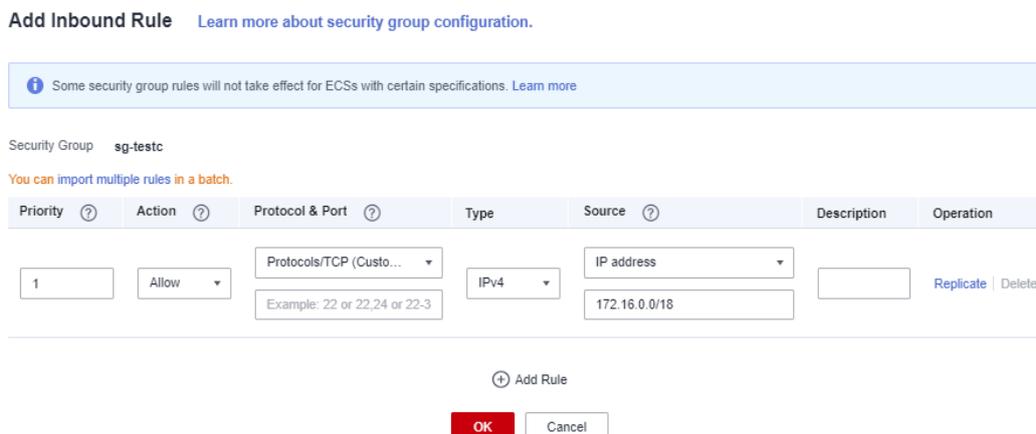
Tags

**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**.



**Figure 2-16** Adding a rule to the GaussDB(DWS) security group

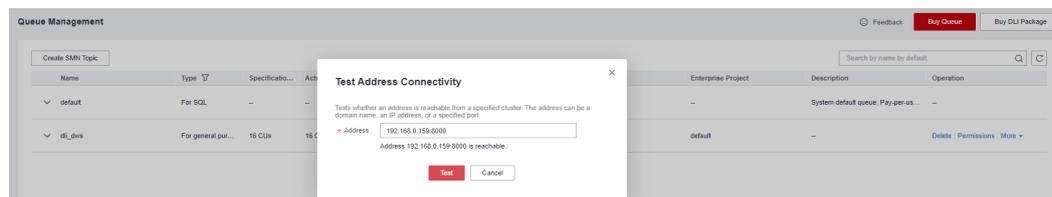


**Step 12** Click **OK**.

**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 2-17** 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 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**.

home » com.huaweicloud » dws

### Group: HuaweiCloud DWS

Sort: **popular** | newest

-  **1. DWS Client**  
com.huaweicloud.dws » [dws-client](#)  
DWS Client  
Last Release on Jun 13, 2023
-  **2. HuaweiCloud DWS JDBC**  
com.huaweicloud.dws » [huaweicloud-dws-jdbc](#)  
Data Warehouse Service JDBC driver  
Last Release on May 19, 2023
-  **3. DWS Connectors**  
com.huaweicloud.dws » [huaweicloud-dws-connectors-parent](#)  
connectors for dws  
Last Release on Jun 13, 2023
-  **4. DWS Connector Flink 2 12 1 12**  
com.huaweicloud.dws » [dws-connector-flink\\_2.12\\_1.12](#)  
DWS Connector Flink 2 12 1 12  
Last Release on Jun 13, 2023

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

 **DWS Connector Flink 2 12 1 12**  
DWS Connector Flink 2 12 1 12

Tags: [flink](#) [cloud](#) [connector](#)

Ranking: #649163 in MvnRepository (See Top Artifacts)

Central (3)

Version	Vulnerabilities	Repository	Usages	Date
<a href="#">1.0.4</a>		<a href="#">Central</a>	0	Jun 13, 2023
<a href="#">1.0.3</a>		<a href="#">Central</a>	0	Mar 30, 2023
<a href="#">1.0.2</a>		<a href="#">Central</a>	0	Mar 13, 2023

**Step 4** Click **View ALL**.

**DWS Connector Flink 2.12.1.12 » 1.0.4**  
DWS Connector Flink 2.12.1.12

Tags: [flink](#) [cloud](#) [connector](#)

Date: Jun 13, 2023

Files: [pom \(6 KB\)](#) [jar \(44 KB\)](#) [View All](#)

Repositories: [Central](#)

Ranking: #649163 in MvnRepository (See Top Artifacts)

Vulnerabilities: **Vulnerabilities from dependencies:**  
[CVE-2022-4065](#)

Maven [Gradle](#) [Gradle \(Short\)](#) [Gradle \(Kotlin\)](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mavenrepository.com/artifact/com.huaweicloud.dws/dws-connector-flink_2.12_1.12 -->
<dependency>
  <groupId>com.huaweicloud.dws</groupId>
  <artifactId>dws-connector-flink_2.12_1.12</artifactId>
  <version>1.0.4</version>
</dependency>
```

Include comment with link to declaration

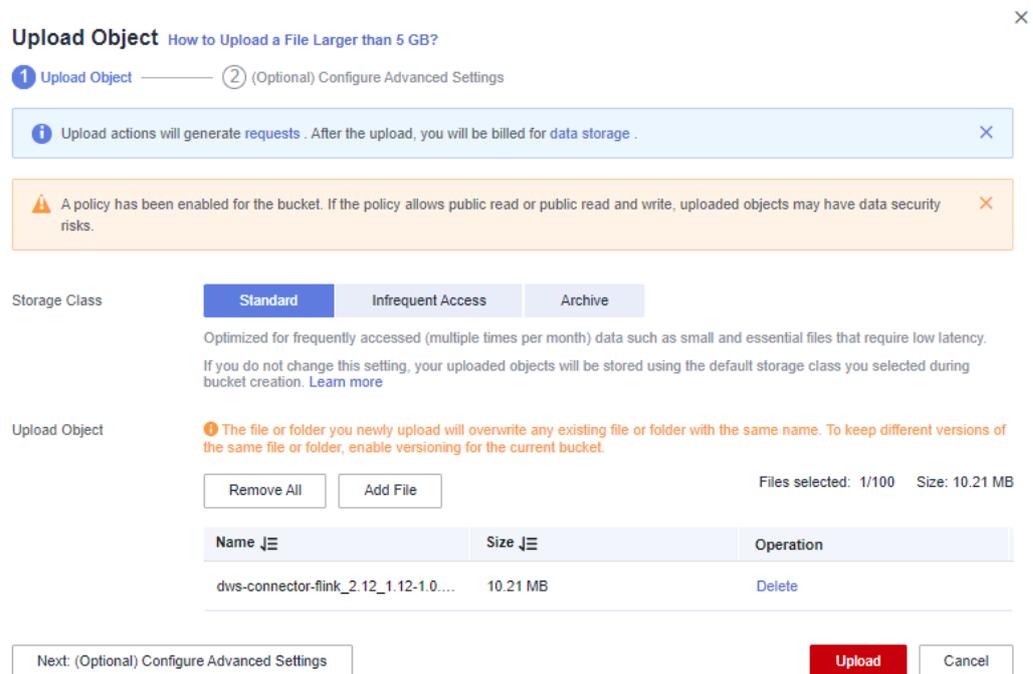
**Step 5** Click `dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar` to download it to the local host.

[com/huaweicloud/dws/dws-connector-flink\\_2.12\\_1.12/1.0.4](https://mavenrepository.com/huaweicloud/dws/dws-connector-flink_2.12_1.12/1.0.4)

File Name	Date	Size
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar</a>	2023-06-13 06:46	10703994
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar.asc</a>	2023-06-13 06:46	235
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar.md5</a>	2023-06-13 06:46	32
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar.sha1</a>	2023-06-13 06:46	40
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-javadoc.jar</a>	2023-06-13 06:46	187712
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-javadoc.jar.asc</a>	2023-06-13 06:46	235
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-javadoc.jar.md5</a>	2023-06-13 06:46	32
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-javadoc.jar.sha1</a>	2023-06-13 06:46	40
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-sources.jar</a>	2023-06-13 06:46	24883
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-sources.jar.asc</a>	2023-06-13 06:46	235
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-sources.jar.md5</a>	2023-06-13 06:46	32
<a href="#">dws-connector-flink_2.12_1.12-1.0.4-sources.jar.sha1</a>	2023-06-13 06:46	40
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.jar</a>	2023-06-13 06:46	45271
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.jar.asc</a>	2023-06-13 06:46	235
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.jar.md5</a>	2023-06-13 06:46	32
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.jar.sha1</a>	2023-06-13 06:46	40
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.pom</a>	2023-06-13 06:46	6544
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.pom.asc</a>	2023-06-13 06:46	235
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.pom.md5</a>	2023-06-13 06:46	32
<a href="#">dws-connector-flink_2.12_1.12-1.0.4.pom.sha1</a>	2023-06-13 06:46	40

**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 2-18** 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 2-19** Creating a job

x

### Create Job

Type:

\* Name:

Description:

Template Name:

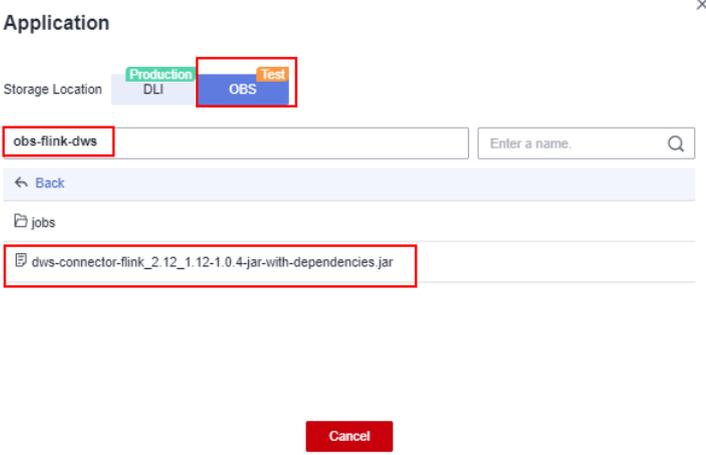
Tags:   
 It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. [View predefined tags](#)   
 To add a tag, enter a tag key and a tag value below.   
   
     
 20 tags available for addition.

**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 2-16** Flink job parameters

Parameter	Value
Queue	dli_dws
Flink Version	1.12

Parameter	Value
UDF Jar	<p>Select the JAR file in the OBS bucket created in <a href="#">Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink.</a></p>  <p>The screenshot shows a configuration window titled 'Application'. Under 'Storage Location', 'OBS' is selected. The bucket name 'obs-flink-dws' is entered in the search field. A file named 'dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar' is selected in the file list. A 'Cancel' button is visible at the bottom.</p>
OBS Bucket	<p>Select the bucket created in <a href="#">Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink.</a></p>
Enable Checkpointing	<p>Check the box.</p>
Other parameters	<p>Retain the default value.</p>

**Figure 2-20** Editing a job

\* Queue

\* Flink Version

UDF Jar

---

\* CUs

\* Job Manager CUs

\* Parallelism

Task Manager Configu...

\* OBS Bucket

Save Job Log

Alarm Generation upo...

Enable Checkpointing

Checkpoint Interval  s

Checkpoint Mode

Auto Restart upon Exc...

Idle State Retention Time  h

Dirty Data Policy

**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"
);

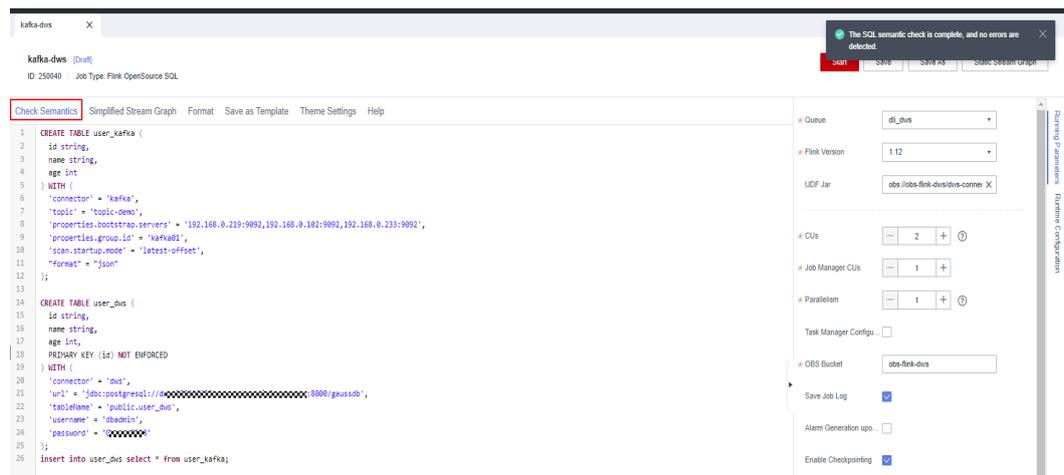
CREATE TABLE user_dws (
  id string,
  name string,
  age int,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'dws',
  'url' = 'jdbc:postgresql://GaussDB(DWS) private network domain name:8000/gaussdb',
  'tableName' = 'public.user_dws',
  'username' = 'dbadmin',
  'password' = 'Password of 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 2-21** 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 2-22** Job execution status

ID	JE	Name	Queues	Type	Status	Description	Username	Created	Started	Duration	Operation
250040		kafka-dws	dli_dws	Flink OpenSource SQL	Running			Jul 03, 2023 09:44:16 GM...	Jul 03, 2023 09:50:17 GM...	4min 41.10s	Edit Start More

----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:${JAVA_HOME}/lib/
gsjdb4.jar:${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}
```

```
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:${JAVA_HOME}/lib/
gsjdb4.jar:${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
```

```
[root@ecs-100418420 jdk-17.0.7]# source /etc/profile
[root@ecs-100418420 jdk-17.0.7]# java -version
java version "17.0.7" 2023-04-18 LTS
Java(TM) SE Runtime Environment (build 17.0.7+8-LTS-224)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.7+8-LTS-224, mixed mode, sharing)
[root@ecs-100418420 jdk-17.0.7]#
```

**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
```

```
[root@ecs ~]# ./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"}
```

```
[root@ecs ~]# ./kafka-console-producer.sh --broker-list 192.168.0.136:9092,192.168.0.214:9092,192.168.0.217:9092 --topic topic-demo
> {"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 **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

## 2.4 Practice of Data Interconnection Between Two GaussDB(DWS) Clusters Based on 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 2-17](#) describes the regions where OBS data has been uploaded.

**Table 2-17** 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

Region	OBS Bucket
RU-Moscow2	dws-demo-ru-northwest-2
LA-Sao Paulo1	dws-demo-sa-brazil-1

## Constraints

In this practice, two sets of 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 in the China-Hong Kong region. For details, see [Creating a Cluster](#). 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** Enter the login username **dbadmin**, the database name **gaussdb**, and the password of user **dbadmin** set during GaussDB(DWS) cluster creation. Select **Remember Password**, enable **Collect Metadata Periodically** and **Show Executed SQL Statements**, and click **Log In**.

**Figure 2-23** Logging in to GaussDB(DWS)

### Instance Login Information

DB Instance Name	dws-██████████	DB Engine Version	GaussDB(DWS) 8.1.3.320
------------------	----------------	-------------------	------------------------

\* Login Username:

\* Database Name:

\* Password:   ✔ Connection is successful.

Remember Password Your password will be encrypted and stored securely.

Collect Metadata  Periodically ?  
 If not enabled, DAS can query the real-time structure information only from databases, which may affect the real-time performance of databases.

Show Executed SQL Statements  ?  
 If not enabled, the executed SQL statements cannot be viewed, and you need to input each SQL statement manually.

**Step 3** Click the database name **gaussdb** and click **SQL Window** in the upper right corner to access the SQL editor.

**Step 4** 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 5** 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 6** 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 7** 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 pane on the left, click **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**.

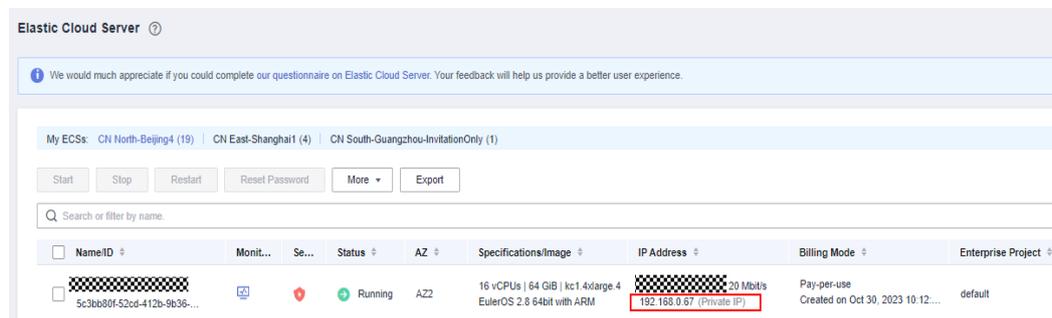
```
su - gds_user
```

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

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

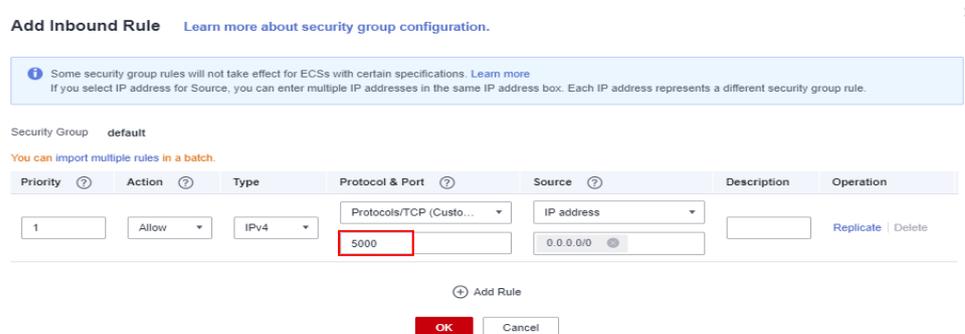
**Step 9** Run the following command to start GDS. You can view the private IP address of the ECS on the ECS console.

```
/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 DWS.

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



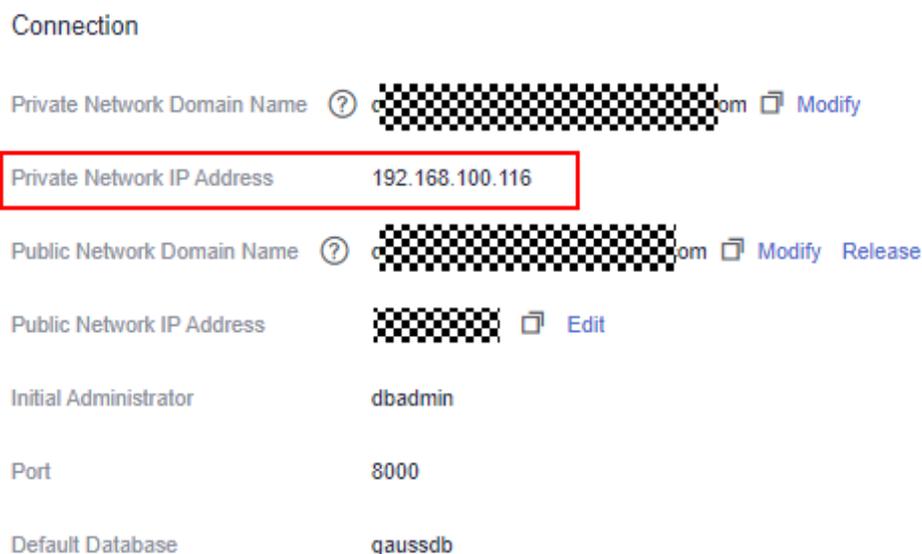
4. Repeat the preceding steps to add an inbound rule of 8000.

----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. To do so, go to the DWS console, switch to the cluster management page, 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 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 Table Optimization Practices

## 3.1 Table Structure Design

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 configuring the table storage mode, compression level, distribution mode, distribution column, partitioned tables, and local clustering).

### 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 <b>INSERT</b> , <b>UPDATE</b> , and <b>DELETE</b> 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	<ul style="list-style-type: none"> <li>• 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).</li> <li>• 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.</li> </ul>
Hash	Table data is distributed on all DNs in the cluster.	Fact tables containing a large amount of data	<ul style="list-style-type: none"> <li>• The I/O resources of each node can be used during data read/write, greatly improving the read/write speed of a table.</li> <li>• Generally, a large table (containing over 1 million records) is defined as a hash table.</li> </ul>

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	<ul style="list-style-type: none"> <li>Round-robin can avoid data skew, improving the space utilization of the cluster.</li> <li>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.</li> <li>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.</li> </ul>

## 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:

- 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.
- 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 `dwcj` table, you are not advised to use `zqdh` as the distribution key.
- 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 the common query is `SELECT * FROM t where c1 = 10 and c2 = 100 and c3 > 10`, you can create the combination index `Index cidx (c1, c2, c3)`. In this way, you can use the query conditions to construct an index prefix for 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.
- The query condition is used. `=`. When `NOT IN` is used, indexes cannot be used.
- 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.

## 3.2 Table Optimization Overview

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.

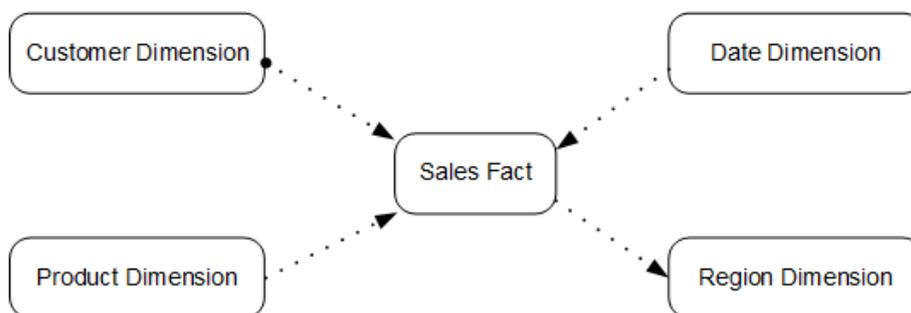
Estimated time: 60 minutes

## 3.3 Selecting a Table Model

The most common types of data warehouse schemas are star and snowflake schemas. Consider service and performance requirements when you choose a schema for your tables.

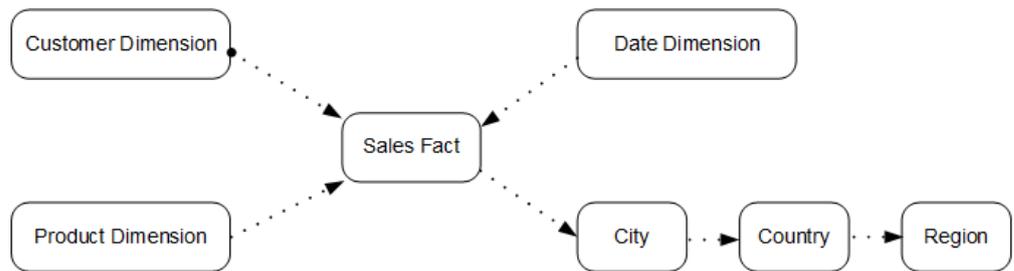
- In the star schema, a central fact table contains the core data for the database and several dimension tables provide descriptive attribute information for the fact table. The primary key of a dimension table associates a foreign key in a fact table, as shown in [Figure 3-1](#).
  - All facts must have the same granularity.
  - Different dimensions are not associated.

**Figure 3-1** Star schema



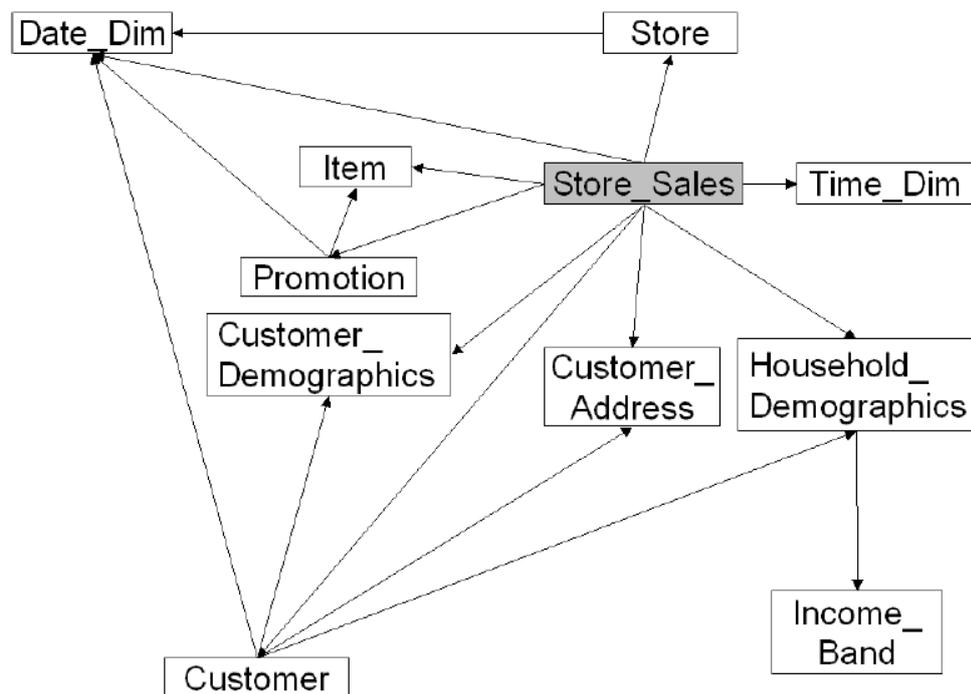
- The snowflake schema is developed based on the star schema. In this schema, each dimension can be associated with multiple dimensions and split into tables of different granularities based on the dimension level, as shown in [Figure 3-2](#).
  - Dimension tables can be associated as needed, and the data stored in them is reduced.
  - This schema has more dimension tables to maintain than the star schema does.

Figure 3-2 Snowflake schema



This practice verifies performance using the Store Sales (SS) model of TPC-DS. The model uses the snowflake schema. Figure 3-3 illustrates its structure.

Figure 3-3 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](http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp).

## 3.4 Step 1: Creating an Initial Table and Loading Sample Data

### 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
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 details about how to create a cluster, see [Creating a GaussDB\(DWS\) 2.0 Cluster](#).

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 3-3](#). This section only provides the syntax for creating the **store\_sales** table. To create all tables, copy the syntax in [Creating an Initial 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)
);
```

**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 [Creating a Foreign Table](#).

 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 3-1](#). 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

## 3.5 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 3-2** Cluster specifications

<b>Model</b>	dws.d2.xlarge VM
<b>CPU</b>	4*CPU E5-2680 v2 @ 2.80GHZ
<b>Memory</b>	32 GB
<b>Network</b>	1 GB
<b>Disk</b>	1.63 TB
<b>Number of Nodes</b>	8

Record the results using the following benchmark table.

**Table 3-3** Recording results

<b>Benchmark</b>	<b>Before</b>	<b>After</b>
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	-

## 3.6 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	-

## 3.7 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 [Creating a Another Table After Design Optimization](#).

```
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_demo_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	-

Benchmark	Before	After
Query 3	17721.15 ms	-
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

## 3.8 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		
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

## 3.9 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_De mographics	171 MB	11 MB	-160 MB	-93.6
Customer_Add ress	170 MB	27 MB	-143 MB	-84.1%
Household_De mographics	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 [Table Optimization Practices](#) to further improve the distribution of tables and the performance of data loading, storage, and query.

## Deleting Resources

After the exercise is completed, delete the cluster by referring to [Deleting a 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;
```

## 3.10 Appendix: Table Creation Syntax

### 3.10.1 Usage

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.

## 3.10.2 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_demo_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
);
```

### 3.10.3 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_demo_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;
```

### 3.10.4 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_demo_sk  integer
  ,
  c_current_demo_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;
```

# 4 Advanced Features

---

## 4.1 Creating a Time Series Table

### Scenarios

Time series tables inherit the syntax of common column-store and row-store tables, making it easier to understand and use.

Time series tables can be managed through out data life cycle. Data increases explosively every day with a lot of dimensions. New partitions need to be added to the table periodically to store new data. Data generated a long time ago usually is of low value and is not frequently accessed. Therefore, it can be periodically deleted. Therefore, time series tables must have the capabilities of periodically adding and deleting partitions.

This practice demonstrates how to quickly create your time series tables and manage them by partitions. Specifying a proper type for a column helps improve the performance of operations such as import and query, making your service more efficient. The following figure uses genset data sampling as an example.

Figure 4-1 Genset data sample

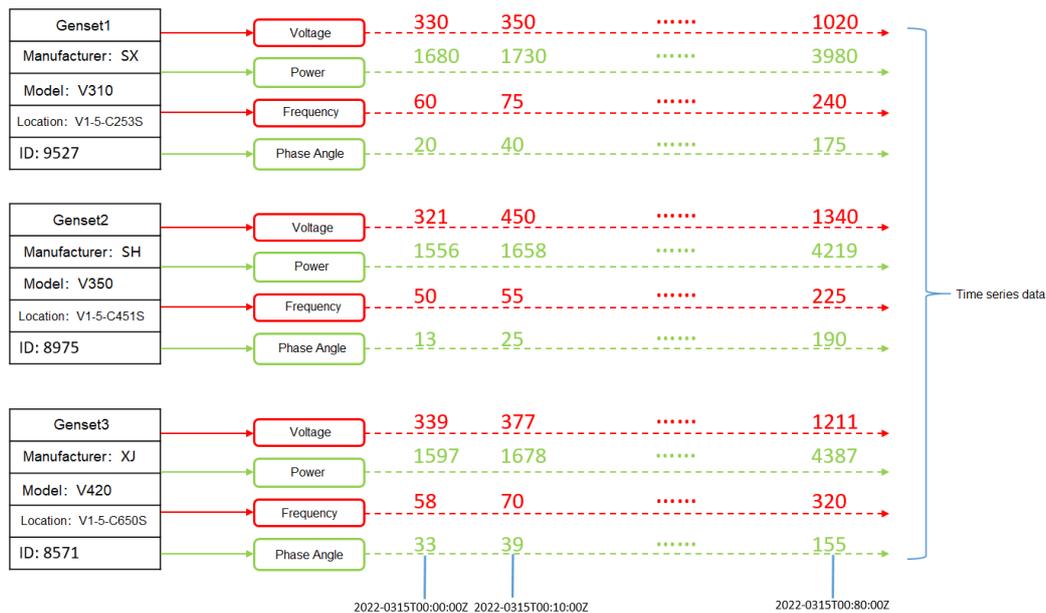


Figure 4-2 Genset data table

tag					field				time
Genset	Manufacturer	Model	Location	ID	Voltage	Power	Frequency	Phase Angle	Timestamp
Genset1	SX	V310	V1-5-C253S	9527	330	1680	60	20	2022-0315T00:00:00Z
Genset2	SH	V350	V1-5-C451S	8975	321	1556	50	13	2022-0315T00:00:00Z
Genset3	XJ	V420	V1-5-C650S	8571	339	1597	58	33	2022-0315T00:00:00Z
Genset1	SX	V310	V1-5-C253S	9527	350	1730	75	40	2022-0315T00:10:00Z
Genset2	SH	V350	V1-5-C451S	8975	450	1658	55	25	2022-0315T00:10:00Z
Genset3	XJ	V420	V1-5-C650S	8571	337	1678	70	39	2022-0315T00:10:00Z
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
Genset1	SX	V310	V1-5-C253S	9527	1020	3980	240	175	2022-0315T00:80:00Z
Genset2	SH	V350	V1-5-C451S	8975	1340	4219	225	190	2022-0315T00:80:00Z
Genset3	XJ	V420	V1-5-C650S	8571	1211	4387	320	155	2022-0315T00:80:00Z

- The columns that describe generator attributes (generator information, manufacturer, model, location, and ID) are set as tag columns. During table creation, they are specified as **TSTag**
- The values of the sampling data metrics (voltage, power, frequency, and current phase angle) vary with time. During table creation, they are specified as **TSField**.
- The last column is specified as the time column, which stores the time information corresponding to the data in the field columns. During table creation, it is specified as **TSTime**.

## Procedure

This practice takes about 30 minutes. The basic process is as follows:

1. [Creating an ECS.](#)
2. [Creating a Stream Data Warehouse.](#)
3. [Using the gsql CLI Client to Connect to a Cluster.](#)
4. [Creating a time series table.](#)

## 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 Stream Data Warehouse

- 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 4-1](#).

**Table 4-1** Software configuration

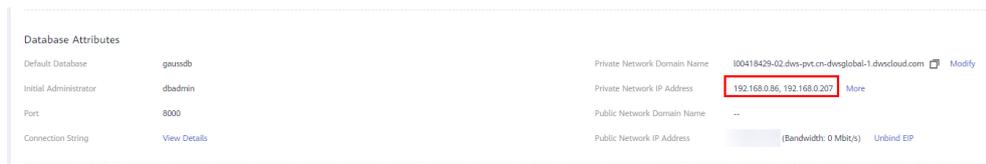
Parameter	Configuration
Region	Select <b>CN-Hong Kong</b> . <b>NOTE</b> <ul style="list-style-type: none"><li>• <b>CN-Hong Kong</b> is used as an example. You can select other regions as required. Ensure that all operations are performed in the same region.</li><li>• Ensure that GaussDB(DWS) and the ECS are in the same region, AZ, and VPC subnet.</li></ul>
AZ	AZ2
Product	Stream data warehouse
Compute Resource	ECS
Storage Type	Cloud SSD
CPU Architecture	X86

Parameter	Configuration
Node Flavor	dwsx2.rt.2xlarge.m6 (8 vCPU   64GB   100-4,000 GB SSD) <b>NOTE</b> If this flavor is sold out, select other AZs or flavors.
Hot Storage	200 GB/node
Nodes	3
Cluster Name	dws-demo01
Administrator Account	dbadmin
Administrator Password	<i>User-defined</i>
Confirm Password	Enter the user-defined administrator password again.
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24) <b>NOTICE</b> Ensure that the cluster and the ECS are in the same VPC subnet.
Security Group	Automatic creation
EIP	Buy now
Enterprise Project	Default
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**.



----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 a Time Series Table

1. The following describes how to create a time series table **GENERATOR** for storing the sample data of gensets.

```
CREATE TABLE IF NOT EXISTS GENERATOR(  
genset text TSTag,  
manufacturer text TSTag,  
model text TSTag,  
location text TSTag,  
ID bigint TSTag,  
voltage numeric TSField,  
power bigint TSField,  
frequency numeric TSField,  
angle numeric TSField,  
time timestampz TSTime) with (orientation=TIMESERIES, period='1 hour', ttl='1 month') distribute by  
hash(model);
```

2. Query the current time.

```
select now();  
now
```

```
-----  
2022-05-25 15:28:38.520757+08  
(1 row)
```

### 3. Query the default partition and partition boundary.

```
SELECT relname, boundaries FROM pg_partition where parentid=(SELECT oid FROM pg_class where  
relname='generator') order by boundaries ;
```

```
  relname  | boundaries  
-----+-----  
default_part_1 | {"2022-05-25 16:00:00+08"}  
default_part_2 | {"2022-05-25 17:00:00+08"}  
p1653505200 | {"2022-05-25 18:00:00+08"}  
p1653541200 | {"2022-05-25 19:00:00+08"}  
p1653577200 | {"2022-05-25 20:00:00+08"}  
.....
```

The **TSTAG** columns support the text, char, bool, int, and big int types.

The **TSTime** column supports the timestamp with time zone and timestamp without time zone types. It also supports the date type in databases compatible with the Oracle syntax. If time zone-related operations are involved, select a time type with time zone.

The data types supported by **TSField** columns are the same as those supported by column-store tables.

#### NOTE

- To optimize the performance in time sequence scenarios, it is recommended to optimize the sequence of tag columns. To do so, prioritize writing more unique columns with distinct values at the beginning of table creation statements.
- When creating a time series table, set the table-level parameter **orientation** to **timeseries**.
- You do not need to manually specify **DISTRIBUTE BY** and **PARTITION BY** for a time series table. By default, data is distributed based on all tag columns, and the partition key is the TSTime column.
- In the **create table like** syntax, the column names and the **kv\_type** types are automatically inherited from the source table. If the source table is a non-time series table and the new table is a time series table, the **kv\_type** type of the corresponding column cannot be determined. As a result, the creation fails.
- One and only one **TSTIME** attribute must be specified. Columns of the TSTIME type cannot be deleted. There must be at least one **TSTag** and **TSField** columns. Otherwise, an error will be reported during table creation.

Time series tables use the TSTIME column as the partition key and have the function of automatic partition management. Partition tables with the automatic partition management function help users greatly reduce O&M time. In the preceding table creation statement, you can see in the table-level parameters that two parameters **period** and **ttd** are specified for the time series table.

- **period**: interval for automatically creating partitions. The default value is 1 day. The value range is 1 hour ~ 100 years. By default, an auto-increment partition task is created for the time series table. The auto-increment partition task dynamically creates partitions to ensure that sufficient partitions are available for importing data.
- **ttd**: time for automatically eliminate partitions. The value range is 1 hour ~ 100 years. By default, no partition elimination task is created. You need to manually specify the partition elimination task when creating a table or use the ALTER TABLE syntax to set the partition elimination task after creating a table. The partition elimination policy is based on the condition that  $\text{nowtime} - \text{partition boundary} > \text{ttd}$ . Partitions that meet this

condition will be eliminated. This feature helps users periodically delete obsolete data.

#### NOTE

For partition boundaries

- If the **period** unit is hour, the start boundary value is the coming hour, and the partition interval is the value of **period**.
- If the **period** unit is day, the start boundary value is 00:00 of the coming day, and the partition interval is the value of **period**.
- If the **period** unit is month, the start boundary value is 00:00 of the coming month, and the partition interval is the value of **period**.
- If the **period** unit is year, the start boundary value is 00:00 of the next year, and the partition interval is the value of **period**.

## Creating a Time Series Table (Manually Setting Partition Boundaries)

1. Manually specify the start value of the partition boundary. For example, to manually set the default partition boundary time **P1** to **2022-05-30 16:32:45** and **P2** to **2022-05-31 16:56:12**, and create a time series table **GENERATOR1**, run the following command:

```
CREATE TABLE IF NOT EXISTS GENERATOR1(  
  genset text TSTag,  
  manufacturer text TSTag,  
  model text TSTag,  
  location text TSTag,  
  ID bigint TSTag,  
  voltage numeric TSField,  
  power bigint TSField,  
  frequency numeric TSField,  
  angle numeric TSField,  
  time timestampz TSTime) with (orientation=TIMESERIES, period='1 day') distribute by hash(model)  
  partition by range(time)  
  (  
    PARTITION P1 VALUES LESS THAN('2022-05-30 16:32:45'),  
    PARTITION P2 VALUES LESS THAN('2022-05-31 16:56:12')  
  );
```

2. Query the current time:

```
select now();  
      now  
-----  
2022-05-31 20:36:09.700096+08(1 row)
```

3. Run the following command to query partitions and partition boundaries:

```
SELECT relname, boundaries FROM pg_partition where parentid=(SELECT oid FROM pg_class where  
relname='generator1') order by boundaries ;  
  relname |      boundaries  
-----+-----  
p1       | {"2022-05-30 16:32:45+08"}  
p2       | {"2022-05-31 16:56:12+08"}  
p1654073772 | {"2022-06-01 16:56:12+08"}  
p1654160172 | {"2022-06-02 16:56:12+08"}  
.....
```

## 4.2 Best Practices of Hot and Cold Data Management

### 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

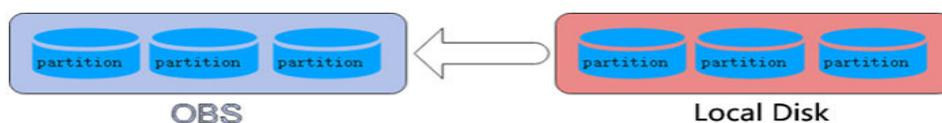
periods, therefore, data is managed in a hierarchical manner, improving data analysis performance and reducing service costs. In some data usage scenarios, data can be classified into hot data and cold data by accessing frequency.

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 4-3** Hot and cold data management



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.
- **HPN:** indicates the number of hot partitions to be reserved. During the cold and hot switchover, data needs to be migrated to OBS. HPN is an integer ranging from 0 to 1600.

## Constraints

- 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 hot data can be switched to cold data. Cold data cannot be switched to hot data.

## 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 4-2](#).

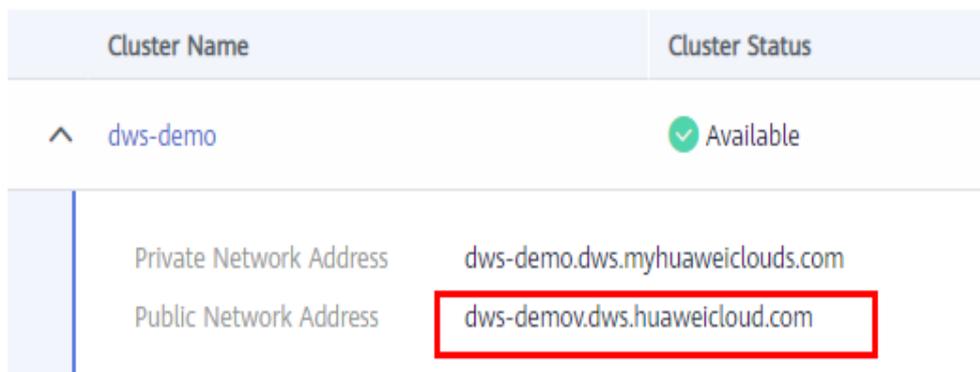
**Table 4-2** Software configuration

Parameter	Configuration
Region	Select CN-Hong Kong. <b>NOTE</b> 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) <b>NOTE</b> 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)

Parameter	Configuration
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-demo.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.

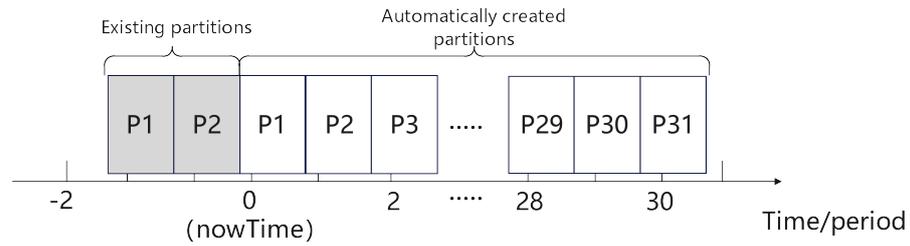
```
source gsql_env.sh
```

If the following information is displayed, the gsql client is successfully configured:





**Figure 4-4** 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  $(\text{nowTime} - \text{boundaryTime})/\text{period}$  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 **ttn** cannot exist independently. You must set **period** in advance or at the same time, and the value of **ttn** 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 4-3](#).

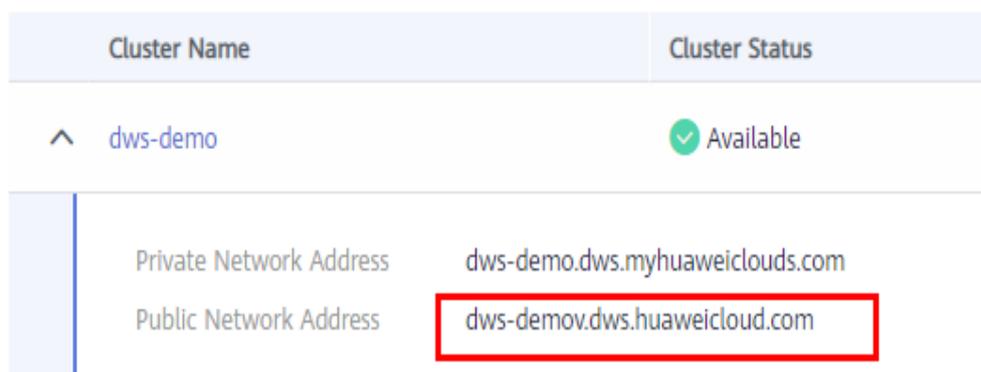
**Table 4-3** Software configuration

Parameter	Configuration
Region	Select CN-Hong Kong. <b>NOTE</b> 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) <b>NOTE</b> 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 **ttd**. Automatic partition creation is enabled with the enabling of **period**, and automatic partition deletion is enabled with the enabling of **ttd**. 30 seconds after **period** or **ttd** 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 **ttd** 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 an ordinary partition table. The only difference is that the syntax specifies the table-level parameters **period** and **ttd**.

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 4-4** 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 **ttd**.

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 **ttd** 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 **ttd** 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);
```

## 4.4 GaussDB(DWS) View Decoupling and Automatic Rebuilding

To solve the problem that base table objects cannot be modified independently due to view and table dependency, GaussDB(DWS) implements view decoupling and rebuilding. This document describes the application scenarios and use methods of 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.

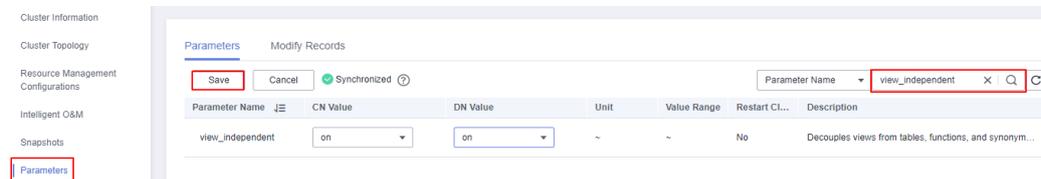
### Usage

**Step 1** Create a cluster on the management console. For details, see section [Creating a 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 DAS to connect to a cluster. Locate the required cluster in the cluster list and click **Log In** in the **Operation** column. On the DAS page that is displayed, enter the username, database name, and password, and test the connection. If the connection is successful, log in to the cluster. For details, see [Using DAS to Connect to a Cluster](#).



**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

## 4.5 Best Practices of Column-Store Delta Tables

### Working Principles

In GaussDB(DWS), data in a column-store table is stored by column. By default, 60,000 rows in each column are stored in a CU. A CU is the minimum unit for storing data in a column-store table. After a CU is generated, data in it is fixed and cannot be modified. No matter whether one or 60,000 data records are inserted into a column-store table, only one CU is generated. When a small amount of data is inserted into a column-store table for multiple times, it cannot be well depressed. As a result, data bloating occurs, which affects the query performance and disk usage.

Data in a CU file cannot be modified and can only be appended. Deleting the CU file data is to mark the old data as invalid in the dictionary. Updating the CU file data is to mark the old data as invalid and write a new record to the new CU. If a column-store table is updated or deleted for multiple times or only a small amount of data is inserted each time, the column-store table space bloats and a large amount of space cannot be effectively used.

Column-store tables are designed to import a large amount of data and store it by column for query. To solve the preceding problems, the delta table is introduced, which is a row-store table attached to a column-store table. After the delta table is enabled, when a single piece of data or a small batch of data is imported, the data is stored in the delta table to avoid small CUs. The addition, deletion, modification, and query of the delta table are the same as those of row-store tables. After the delta table is enabled, the performance of importing column-store tables is greatly improved.

### Use Cases

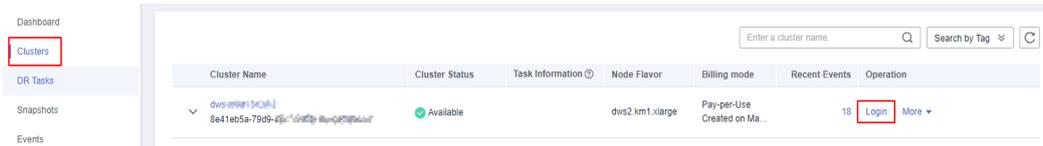
The column-store delta table is used for hybrid row-column storage and is suitable for real-time analysis and statistics. It solves the performance problem caused by importing small batches of data and periodically merges the data to the primary table to ensure the analysis and query performance. You need to determine whether to enable delta tables based on the actual situation. Otherwise, the advantages of GaussDB(DWS) column-store tables cannot be fully utilized, wasting extra space and time.

### 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.
- 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. For details, see [Checkpoint Vehicle Analysis](#).

## Procedure

- Step 1** Use DAS to connect to a cluster. Locate the required cluster in the cluster list and click **Log In** in the **Operation** column. On the DAS page that is displayed, enter the username, database name, and password, and test the connection. If the connection is successful, log in to the cluster. For details, see [Using DAS to Connect to a Cluster](#).



- Step 2** Execute the following statement to create the **traffic** database:

```
CREATE DATABASE traffic encoding 'utf8' template template0;
```

- Step 3** Run the following statements to create database tables **GCJL** and **GCJL2** for storing checkpoint vehicle information: By default, the delta table is not enabled for **GCJL** but is enabled for **GCJL2**.

```
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),
    clx VARCHAR(8),
    csys VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by hash(hphm);

DROP TABLE if exists GCJL2;
CREATE TABLE GCJL2
(
    kkbh VARCHAR(20),
    hphm VARCHAR(20),
    gcsj DATE ,
    cplx VARCHAR(8),
    clx VARCHAR(8),
    csys VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE, ENABLE_DELTA = TRUE)
distribute by hash(hphm);
```

### NOTE

- Delta tables are disabled by default. To enable delta tables, set **enable\_delta** to **true** when creating column-store tables.
- If the delta table is not enabled when the table is created, you can run the **ALTER TABLE** command to enable the delta table after the table is created.  
`ALTER TABLE table_name SET (enable_delta=TRUE);`
- To disable the enabled the delta table, run the following command:  
`ALTER TABLE table_name SET (enable_delta=FALSE);`

- 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.
- In this practice, the **CN-Hong Kong** region is used as an example. Enter **dws-demo-ap-southeast-1**, and replace `<Access_Key_Id>` and `<Secret_Access_Key>` with the actual value.
- 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.

```
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 GCJL_OBS;
INSERT INTO traffic_data.GCJL2 select * from GCJL_OBS;
```

It takes some time to import data.

**Step 6** Run the following statement to check the size of the storage space after the database table is imported:

```
SELECT pg_size_pretty(pg_total_relation_size('traffic_data.GCJL'));
SELECT pg_size_pretty(pg_total_relation_size('traffic_data.GCJL2'));
```

After the delta table is enabled, the storage space usage is reduced from 8953 MB to 6053 MB, greatly improving the import performance.

The screenshot shows a database interface with a tab labeled "Result Set1". Below the tab, there is a warning message: "The following is the execution result set of SELECT pg\_size\_pretty(pg\_total\_relation\_size('traffic\_data.GCJL'));" and "The table below cannot be edited." The result set is displayed in a table with two columns: an index and the storage size in MB.

	pg_size_pretty
1	8953 MB
	pg_size_pretty
1	6053 MB

**Step 7** Query data in the table. It is found that the query speed is improved after the delta table is enabled.

```
SELECT * FROM traffic_data.GCJL where hphm = 'YD38641';  
SELECT * FROM traffic_data.GCJL2 where hphm = 'YD38641';
```

----End

## Impact of Enabling the Delta Table

- Enabling the delta table function for a column-store table helps avoid the creation of small CUs during the import of individual or minimal data entries. This can significantly improve the performance. For instance, importing 100 data records into a cluster with three CNs and six DNAs can cut import time by 25% and reduce storage usage by 97%. It is recommended to enable the delta table function prior to multiple insertions of small data batches. After ensuring that subsequent operations do not involve small data imports, the delta table function can be disabled.
- A delta table, which operates as a row-store table associated to a column-store table, typically remains disabled to preserve the latter's high compression ratio. When enabled, the delta table can significantly increase both time and space requirements for large-scale data import. For instance, in a cluster with three CNs and six DNAs, importing 10,000 records with the delta table enabled can be up to four times slower and occupy over ten times the space compared to when it's disabled. It's recommended to enable or disable the delta table based on specific service needs.

## 4.6 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 section [Creating a Cluster](#).

**Step 2** Use DAS to connect to a cluster. Locate the required cluster in the cluster list and click **Log In** in the **Operation** column. On the DAS page that is displayed, enter the username, database name, and password, and test the connection. If the connection is successful, log in to the cluster. For details, see [Using DAS to Connect to a Cluster](#).



**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 Database Management

---

## 5.1 Best Practices of Resource Management

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** Connect to a cluster by referring to [Using the gsql CLI Client to Connect to a 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');
```

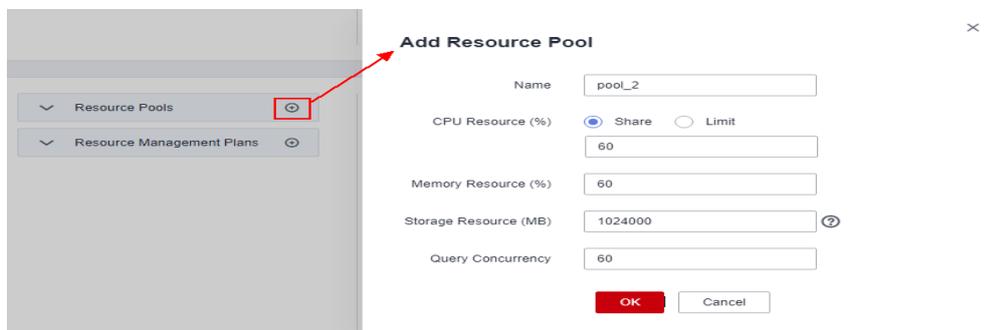
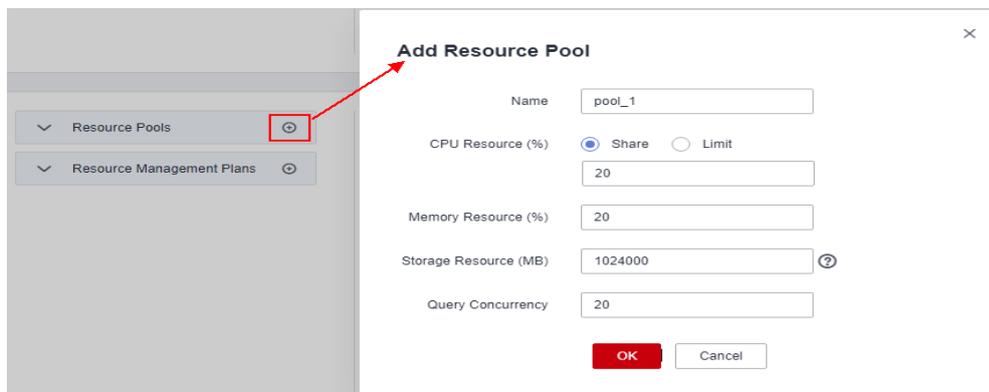
```
tpch-> SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO where username in ('budget_config_user', 'report_user');
s | read_counts | write_counts | read_speed | write_speed
-----|-----|-----|-----|-----
budget_config_user | 0 | 10796 | 0 | 0
report_user | 0 | 10796 | 0 | 0
(2 rows)
```

----End

### Step 3: Creating a Resource Pool

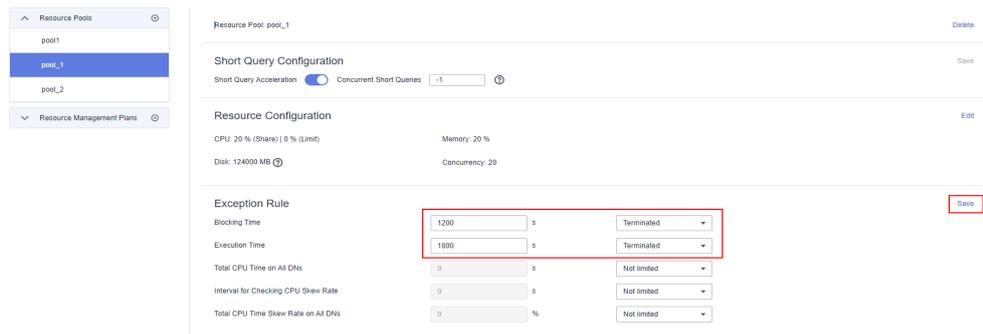
**Step 1** Log in to the GaussDB(DWS) management console, click a cluster name in the cluster list. The **Resource Management Configurations** page is displayed.

**Step 2** Click **Add Workload Queue**. Create the report resource pool **pool\_1** and transaction resource pool **pool\_2** by referring to [Scenarios](#).



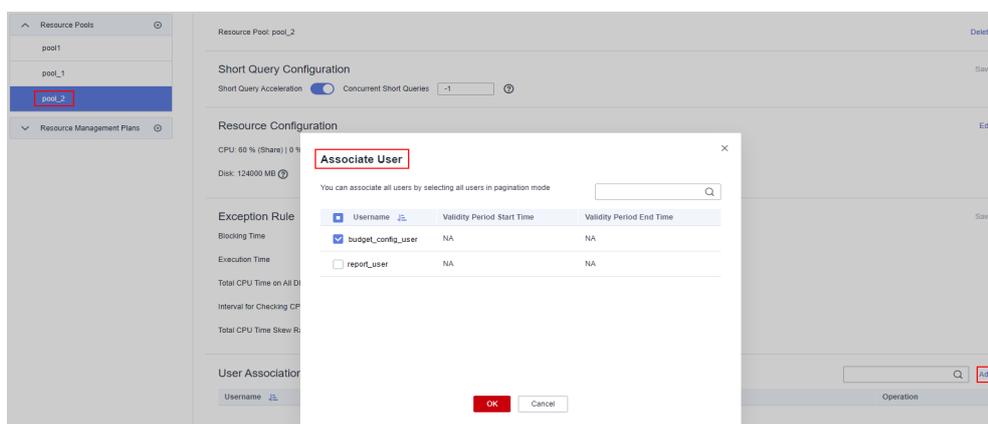
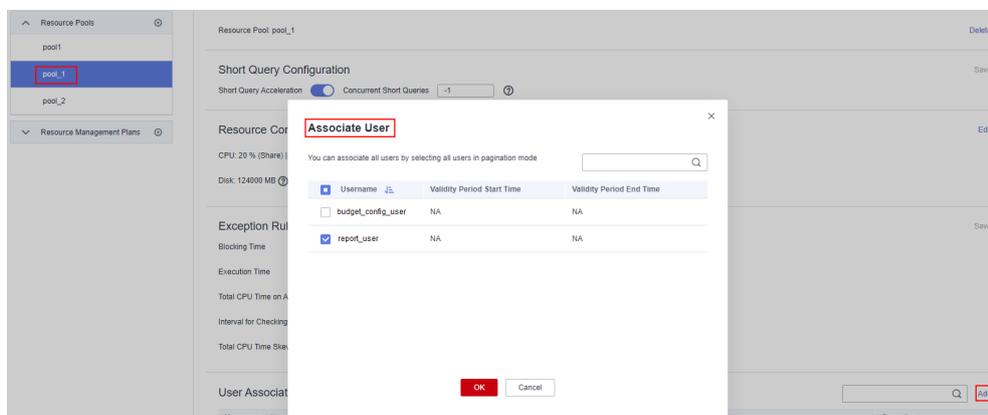
**Step 3** Modify the exception rules.

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 username,respool FROM pg_user WHERE username = 'report_user';
```

```
gaussdb=> select username, respool from pg_user where username = 'report_user';
username | 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 tables;
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

## 5.2 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 5-1 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 about SQL tuning, see [Typical SQL Optimization Methods](#).

## 5.3 Analyzing SQL Statements that Are Being Executed

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 5-1** Some PG\_STAT\_ACTIVITY fields

Name	Type	Description
username	name	Name of the user logging in to the backend
client_addr	inet	IP address of the client connected to the backend <b>null</b> 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

Name	Type	Description
state	text	<p>Overall state of the backend. The value can be:</p> <ul style="list-style-type: none"> <li>● <b>active</b>: The backend is executing queries.</li> <li>● <b>idle</b>: The backend is waiting for new client commands.</li> <li>● <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>● <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>● <b>fastpath function call</b>: The backend is executing a <b>fast-path</b> function.</li> <li>● <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul> <p><b>NOTE</b> Common users can view only the session status of their own accounts. That is, the state information of other accounts is empty.</p>
waiting	boolean	<p>If the back end is currently waiting for a lock, the value is <b>t</b>. Otherwise, the value is <b>f</b>.</p> <ul style="list-style-type: none"> <li>● <b>t</b> stands for true.</li> <li>● <b>f</b> stands for false.</li> </ul>

Name	Type	Description
enqueue	text	<p>Queuing status of a statement. Its value can be:</p> <ul style="list-style-type: none"> <li>• <b>waiting in global queue:</b> The statement is queuing in the global concurrency queue. The number of concurrent statements exceeds the value of <b>max_active_statements</b> configured for a single CN.</li> <li>• <b>waiting in respool queue:</b> 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 <b>max_dop</b> of the fast track.</li> <li>• <b>waiting in ccn queue:</b> The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing. The scenarios are: <ol style="list-style-type: none"> <li>1. The available global memory exceeds the upper limit, the job is queuing in the global memory queue.</li> <li>2. Concurrent requests on the slow lane in the resource pool exceed the upper limit, which is specified by <b>active_statements</b>.</li> <li>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 <b>mem_percent</b>.</li> </ol> </li> <li>• Empty or <b>no waiting queue:</b> The statement is running.</li> </ul>
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 username,client_addr,application_name,state,waiting,enqueue,pid FROM PG_STAT_ACTIVITY WHERE DATNAME='database name';
```

The following command output is displayed:

```
username | 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 username,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:

```
username | state | query
-----+-----+-----
leo | idle | select * from joe.mytable;
dbadmin | active | SELECT username,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,username,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, username, 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 | username | 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,
username, 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, username, 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.username as w_user,
l.query as locking_query,
l.pid as l_pid,
l.username 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.

## 5.4 Excellent Practices for Data Skew Queries

### 5.4.1 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:  $\text{Skew ratio} = (\text{Maximum number of rows imported on a DN} - \text{Minimum number of rows imported on a DN}) / \text{Number of imported rows}$ . 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
```

## 5.4.2 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.
- 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)
```

## 5.5 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;
```







## 5.6 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 /*+ nestloop(a c)*/ c.column_name, c.data_type, c.ordinal_position, pgd.description, pp.partkey,
c.is_nullable, c.column_default, c.character_maximum_length, c.numeric_precision, c.numeric_scale,
c.datetime_precision, c.interval_type, c.udt_name from information_schema.columns as c left join
pg_namespace sp on sp.nspname = c.table_schema left join pg_class cla on cla.relname =
c.table_name and cla.relnamespace = sp.oid left join pg_catalog.pg_partition pp on (pp.parentid =
cla.oid and pp.parttype = 'r') left join pg_catalog.pg_description pgd on (pgd.objoid=cla.oid and
pgd.objsubid = c.ordinal_position)where c.table_name in ('tablename') and c.table_schema = 'public';

```

For example, to quickly query the column information of the **customer\_t1** table, run the following command:

```

SELECT /*+ nestloop(a c)*/ c.column_name, c.data_type, c.ordinal_position, pgd.description, pp.partkey,
c.is_nullable, c.column_default, c.character_maximum_length, c.numeric_precision, c.numeric_scale,
c.datetime_precision, c.interval_type, c.udt_name from information_schema.columns as c left join
pg_namespace sp on sp.nspname = c.table_schema left join pg_class cla on cla.relname =
c.table_name and cla.relnamespace = sp.oid left join pg_catalog.pg_partition pp on (pp.parentid =
cla.oid and pp.parttype = 'r') left join pg_catalog.pg_description pgd on (pgd.objoid=cla.oid and
pgd.objsubid = c.ordinal_position) where c.table_name in ('customer_t1') and c.table_schema = 'public';
column_name | data_type | ordinal_position | description | partkey | is_nullable | column_default |
character_maximum_length | numeric_precision | numeric_scale | datetime_precision | interval_type |
udt_name
-----+-----+-----+-----+-----+-----+-----
c_last_name | character | 4 | | | YES | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
c_first_name | character | 3 | | | YES | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
c_customer_id | character | 2 | | | YES | | | | | |
| | | | | | | | | | | |

```

```

|          |          | bpchar
c_customer_sk | integer |          1 |          | YES |          |
|          32 |          |          |          |
0 |          |          | int4
(4 rows)

```

- 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(timestampz starttime,timestampz 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('2022-08-26
8:00:00','2022-08-26 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 avgsz,
pg_catalog.max(dnsize)
AS maxsize,
pg_catalog.min(dnsize)
AS minsize,
(maxsize
- avgsz) * 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,
avgsz::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
-----+-----+-----+-----+-----+-----
 gausldb | 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
 gausldb
(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.tablename
WHERE t.schemaname='public'
ORDER BY 1,2;
```

## 5.7 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:

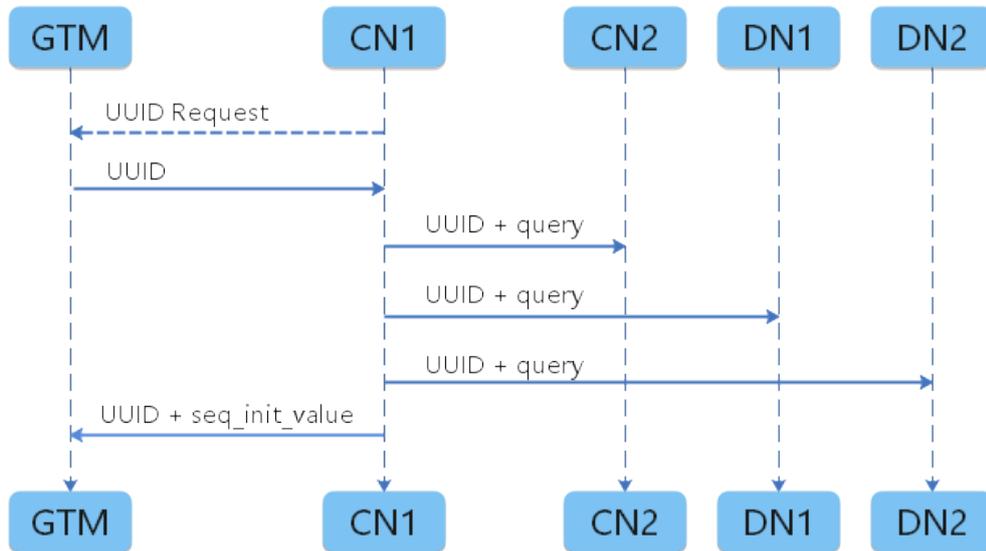
- **sequence\_name** indicates the name of the sequence.
- **last\_value** is meaningless.
- **start\_value** indicates the initial value of the sequence.
- **increment\_by** indicates the step of the sequence.
- **max\_value** indicates the maximum value of the sequence.
- **min\_value** indicates the 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 field 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 5-2** Creating a sequence



The specific process is as follows:

1. The CN that receives the SQL command requests 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

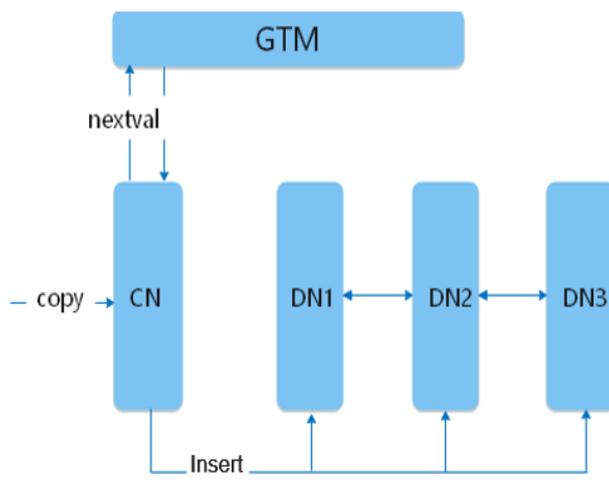
RunTime Analyze Information
-----
"dbadmin.test2" runtime: 9.586ms, sync stats

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)

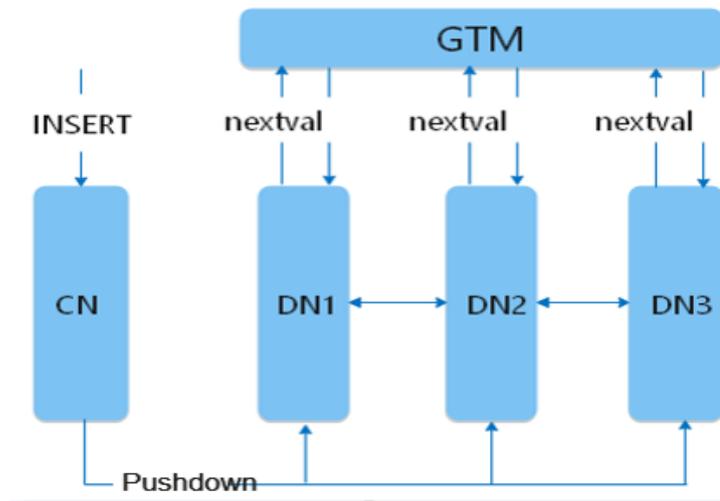
```

During an INSERT operation, nextval is executed on the DN. 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 DN. 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.

```

8888=# select * from pg_stat_wait_status where query_id in (217298687383447130,217298687383475707,217298687383956317,217298687383962387,217298687384026648) order by query_id,
node_name,db_name,thread_name,query_id,tid,lwtid,ptid,tlevel,spid,wait_status,wait_event
node_name | db_name | thread_name | query_id | tid | lwtid | ptid | tlevel | spid | wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
dn_6001_6002 | cn_5003 | kfpst | 217298687383447130 | 281459614074028 | 2490718 | | | 0 | 0 | wait cmd |
dn_6005_6006 | cn_5003 | kfpst | 217298687383447130 | 281459123161136 | 211948 | | | 0 | 0 | wait cmd |
dn_6003_6004 | cn_5003 | kfpst | 217298687383447130 | 281460010731568 | 3913471 | | | 0 | 0 | wait cmd |
dn_6007_6008 | cn_5003 | kfpst | 217298687383447130 | 28145996444976 | 2974033 | | | 0 | 0 | wait cmd |
cn_5003 | cn_5003 | kfpst | 217298687383447130 | 281454976845872 | 211939 | | | 0 | 0 | gtm get sequence val |
cn_5003 | cn_5003 | kfpst | 217298687383475707 | 28145496664560 | 211941 | | | 0 | 0 | acquire lwlock | PgStatObjectLock
dn_6001_6002 | cn_5003 | kfpst | 217298687383475707 | 281459828259356 | 2490701 | | | 0 | 0 | wait cmd |
dn_6005_6006 | cn_5003 | kfpst | 217298687383475707 | 281460079614008 | 211844 | | | 0 | 0 | wait cmd |
dn_6003_6004 | cn_5003 | kfpst | 217298687383475707 | 281458634766384 | 3913447 | | | 0 | 0 | wait cmd |
dn_6007_6008 | cn_5003 | kfpst | 217298687383475707 | 281460012633008 | 2974022 | | | 0 | 0 | wait cmd |
dn_6003_6004 | cn_5003 | kfpst | 217298687383956317 | 281458383046704 | 3913477 | | | 0 | 0 | wait cmd |
cn_5003 | cn_5003 | kfpst | 217298687383956317 | 28145426501936 | 211943 | | | 0 | 0 | acquire lwlock | PgStatObjectLock
dn_6005_6006 | cn_5003 | kfpst | 217298687383956317 | 281459898572592 | 211952 | | | 0 | 0 | wait cmd |
dn_6001_6002 | cn_5003 | kfpst | 217298687383956317 | 281459345590320 | 2490722 | | | 0 | 0 | wait cmd |
dn_6007_6008 | cn_5003 | kfpst | 217298687383956317 | 281458762401840 | 2974037 | | | 0 | 0 | wait cmd |
dn_6005_6006 | cn_5003 | kfpst | 217298687383962387 | 281459139942448 | 211949 | | | 0 | 0 | wait cmd |
cn_5003 | cn_5003 | kfpst | 217298687383962387 | 281454943283248 | 211942 | | | 0 | 0 | acquire lwlock | PgStatObjectLock
dn_6003_6004 | cn_5003 | kfpst | 217298687383962387 | 281458433390640 | 3913473 | | | 0 | 0 | wait cmd |
dn_6001_6002 | cn_5003 | kfpst | 217298687383962387 | 281459412215568 | 2490719 | | | 0 | 0 | wait cmd |
dn_6007_6008 | cn_5003 | kfpst | 217298687383962387 | 28145879564464 | 2974033 | | | 0 | 0 | wait cmd |
dn_6001_6002 | cn_5003 | kfpst | 217298687384026648 | 281459395934256 | 2490721 | | | 0 | 0 | wait cmd |
dn_6005_6006 | cn_5003 | kfpst | 217298687384026648 | 281459898989812 | 211951 | | | 0 | 0 | wait cmd |
dn_6007_6008 | cn_5003 | kfpst | 217298687384026648 | 281458779183152 | 2974036 | | | 0 | 0 | wait cmd |
cn_5003 | cn_5003 | kfpst | 217298687384026648 | 281454993627184 | 211940 | | | 0 | 0 | acquire lwlock | PgStatObjectLock
dn_6003_6004 | cn_5003 | kfpst | 217298687384026648 | 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

# 6 Sample Data Analysis

## 6.1 Checkpoint Vehicle Analysis

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 6-1](#) describes the regions where OBS data has been uploaded.

**Table 6-1** Regions and OBS bucket names

Region	OBS Bucket
CN North-Beijing1	dws-demo-cn-north-1
CN North-Beijing2	dws-demo-cn-north-2

Region	OBS Bucket
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** In the navigation pane on the left, choose **Clusters**. On the displayed page, click **Create Cluster** in the upper right corner.

**Step 4** Configure the parameters according to [Table 6-2](#).

**Table 6-2** Basic configurations

Parameter	Configuration
Region	Select <b>CN North-Beijing4</b> or <b>CN-Hong KongEU-Dublin</b> . <b>NOTE</b> <b>CN-Hong Kong</b> 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) <b>NOTE</b> 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 6-3](#).

**Table 6-3** 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** Verify that the information is correct and click **Next: Configure Advanced Settings**. Configure the network by referring to [Table 6-4](#).

**Table 6-4** 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 configuration, and click **Next**.

**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 6-1** Cluster information

Region	Beijing4
Cluster Version	8.1.3.311
Public Network Address	 .249.99.53
Subnet	subnet-278a (192.168.0.0/24)
Nodes	3
Tag	--

----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 **Connections** page of the GaussDB(DWS) console, download the Data Studio GUI 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 6-5](#).

**Table 6-5** 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 <b>Public Network Address</b> queried in <a href="#">Step 1: Creating a Cluster</a> .
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.
- In this practice, the **CN-Hong Kong** region is used as an example. Enter **dws-demo-ap-southeast-1**, 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/gcxl',
    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.gcj;
```

### 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.gcj  
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.gcj  
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;
```

## 6.2 Supply Chain Requirement Analysis 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 6-6](#) describes the regions where OBS data has been uploaded.

**Table 6-6** 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 [Checkpoint Vehicle Analysis](#).

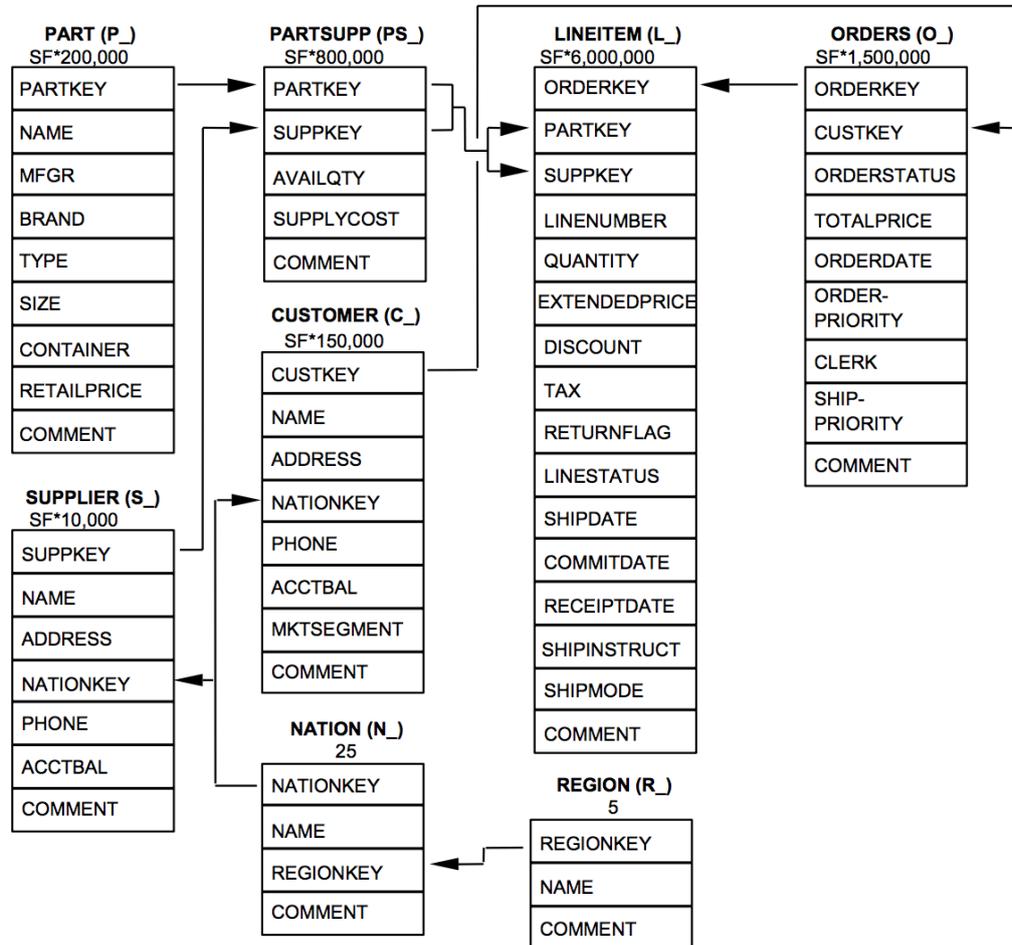
## Step 1: Importing Sample Data

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 6-2](#).

Figure 6-2 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.
- In this practice, the **CN-Hong Kong** region is used as an example. Enter **dws-demo-ap-southeast-1**, 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  
);
```

## 6.3 Operations Status Analysis 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 6-7** describes the regions where OBS data has been uploaded.

**Table 6-7** 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;
```

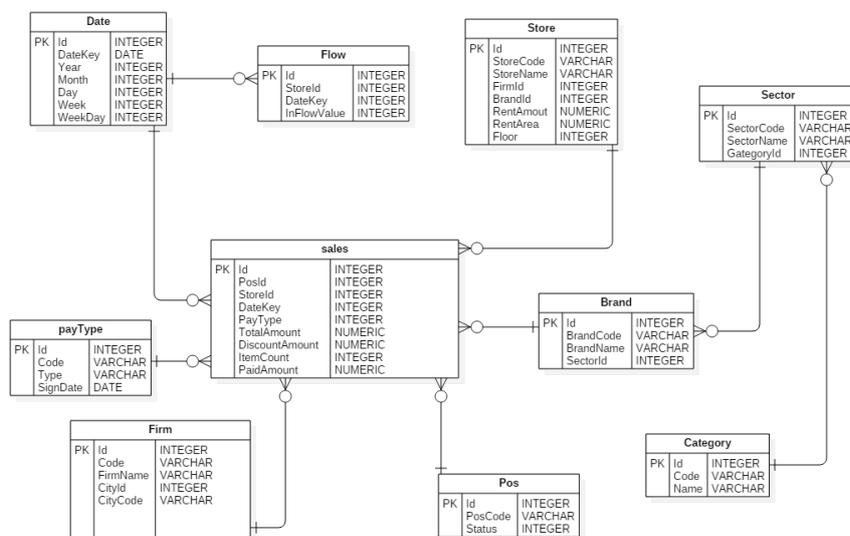
**Step 2** Perform the following steps to switch to the new database:

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 6-3](#).

**Figure 6-3** 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),
  FIRMLID 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.
- In this practice, the **CN-Hong Kong** region is used as an example. Enter **dws-demo-ap-southeast-1**, 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 FIR MID, 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;
```

# 7 Security Management

## 7.1 Role-based Access Control (RBAC)

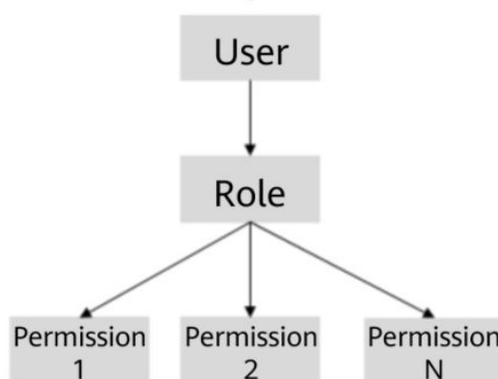
### What is RBAC?

- 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.



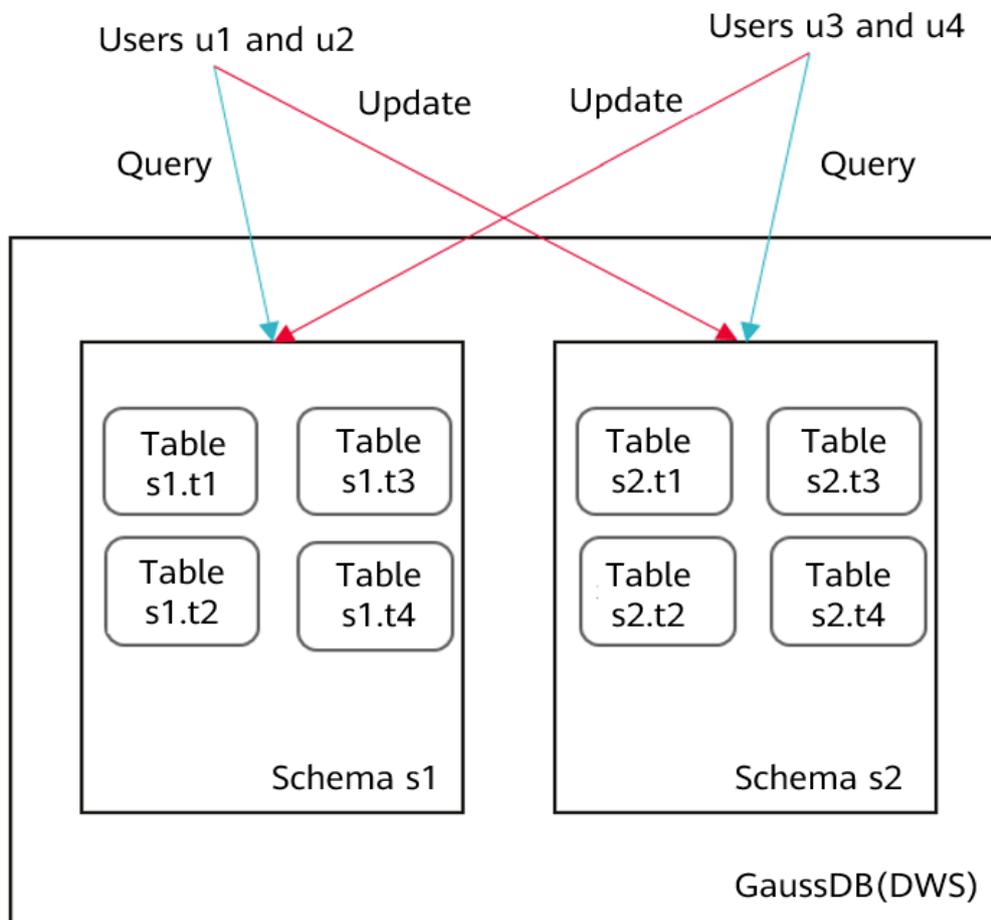
### Scenarios

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;
```

```
test_lhy=> \du u1
                List of roles
Role name | Attributes | Member of
-----+-----+-----
u1        |           | {rs1_select,rs2_update}
```

**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;
```

```
test_lhy=> UPDATE s1.t1 SET c1 = 2 WHERE c2 = 2;
ERROR: Distributed key column can't be updated in current version
test_lhy=> SELECT * FROM s1.t1;
 c1 | c2
----+----
  1 |  2
(1 row)
test_lhy=> UPDATE s1.t1 SET c2 = 3 WHERE c1 = 1;
ERROR: permission denied for relation t1
```

**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

## 7.2 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.

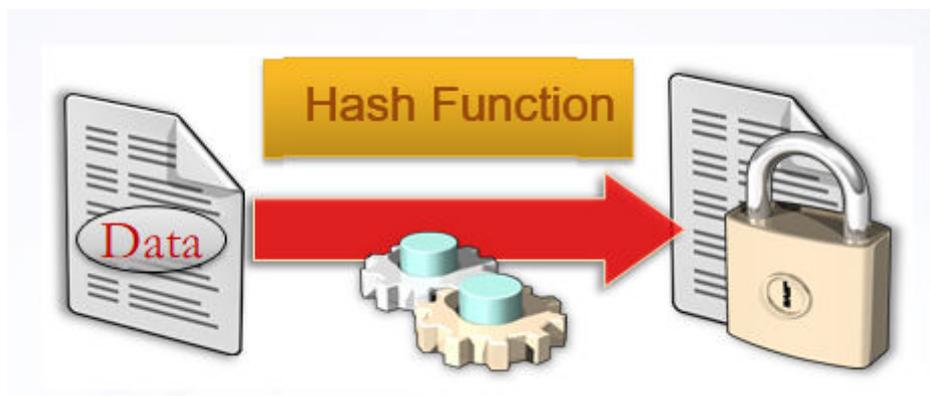
### Background

- 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.

**Figure 7-1** Hash functions



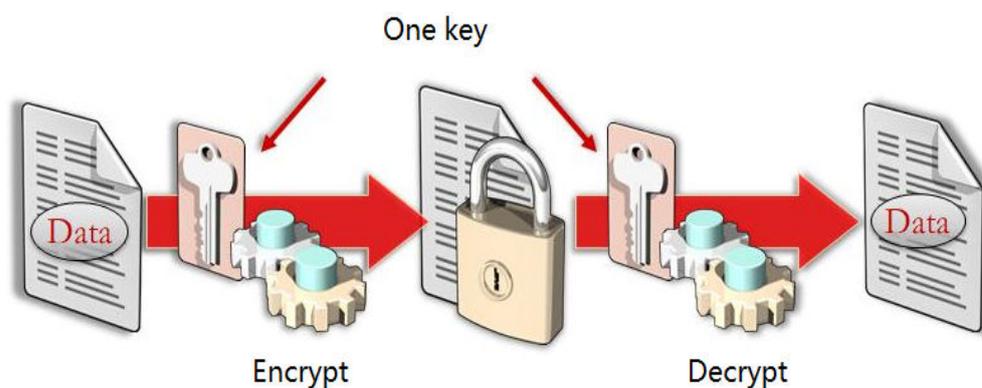
- Symmetric Encryption Algorithms

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 7-2 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 `gsql` CLI Client to Connect to a 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 | AAAAAAAAAAABAUUC3VQ+MvPCDAaTUySl1e2gGLr4/ATdCUjTEvova3cb/Ba3ZKqIn1yNVGEFBvJnTq/3sLF4//Gm8qG7AyfNbbqdW3aYErLVpbE/QWFX9Ilg== | aFEWQR2gkj
iu6sfsAad+dHzfFDHePZ6xd44zyekh+qVFlh9FODZ0DoaFAJXctwUsiqaiiTxW8cSEaNjS/E7Ke1ruY=
2 | 81b637d8fcd2c6da6359e6963113a1170de795e4b725b84d1e0b4cfd9ec58ce9 | AAAAAAAAAAABAUUC3VQ+MvPCDAaTUySl1taXxAoDqE793hgyCjvC0ESdAX5Mtgdq2LX11f5ZxraQ73WIJvtlBX8oe3gTDxoXGIHbHht4kzM4U8dOwr5rjgg== | aFEWQR2gkj
iu6sfsAad+dM8tPTDo/Pds6ZmqdmjGiKxf39+Wzx5NoQ6c8FrzihnRzgc0fycWSu5YGNWOKYWhRsE84Ac=
3 | 026ad9b14a7453b7488daa0c6acbc258b1506f52c441c7c465474c1a564394ff | AAAAAAAAAACnyusORPeApqMUgh56ucQu3uso/Llw5MbPFMkOXuspEzhhnc9vErwOFe6cuGtx8muEyHCX7V5yXs+8FfxNh3n5L3419LDWJLY2O4merHpSg== | 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

## 7.3 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**).

## 7.4 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.

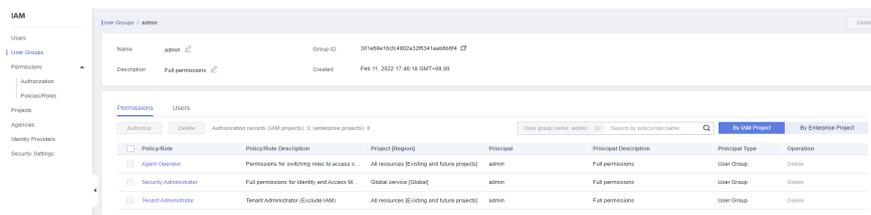
- **Scenario 1:** Allow software developers in your enterprise to use GaussDB(DWS) resources, but do not allow them to delete the resources or perform any high-risk operations. To this end, you can create IAM users for these developers and grant them only the permissions required for using GaussDB(DWS) 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