

# Data Warehouse Service

## Quick Start

**Issue** 01  
**Date** 2023-11-14



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2023. 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.

---

# Contents

---

<b>1 Checkpoint Vehicle Analysis.....</b>	<b>1</b>
<b>2 Supply Chain Requirement Analysis of a Company.....</b>	<b>8</b>
<b>3 Operations Status Analysis of a Retail Department Store.....</b>	<b>17</b>
<b>4 Creating a Time Series Table.....</b>	<b>26</b>
<b>5 Best Practices of Hot and Cold Data Management.....</b>	<b>33</b>
<b>6 Best Practices for Automatic Partition Management.....</b>	<b>38</b>
<b>7 Creating a Cluster and Connecting to It.....</b>	<b>45</b>
7.1 Step 1: Starting Preparations.....	45
7.2 Step 2: Creating a Cluster.....	46
7.3 Step 3: Connecting to a Cluster.....	49
7.4 Step 4: Viewing Other Documents and Deleting Resources.....	53
<b>8 Using CDM to Migrate MySQL Data to the GaussDB(DWS) Cluster.....</b>	<b>54</b>
<b>9 Using DLI Flink Jobs to Write Kafka Data to GaussDB(DWS) in Real Time.....</b>	<b>67</b>
<b>10 Basic SQL Operations.....</b>	<b>88</b>
<b>11 Database Quick Start.....</b>	<b>90</b>
11.1 Before You Start.....	90
11.2 Creating and Managing Databases.....	92
11.3 Planning a Storage Model.....	94
11.4 Creating and Managing Tables.....	96
11.4.1 Creating a Table.....	97
11.4.2 Inserting Data to a Table.....	97
11.4.3 Updating Data in a Table.....	101
11.4.4 Viewing Data.....	102
11.4.5 Deleting Data from a Table.....	103
11.5 Loading Sample Data.....	103
11.6 Querying System Catalogs.....	124
11.7 Creating and Managing Schemas.....	127
11.8 Creating and Managing Partitioned Tables.....	129
11.9 Creating and Managing Indexes.....	132

---

11.10 Creating and Managing Views.....	136
11.11 Creating and Managing Sequences.....	137
11.12 Creating and Managing Scheduled Tasks.....	139

# 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 1-1** Regions and OBS bucket names

Region	OBS Bucket
EU-Dublin	dws-demo-eu-west-101

## Making Preparations

- Register a GaussDB(DWS) account and check 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 1-2](#).

**Table 1-2** Basic configurations

Parameter	Configuration
Region	Select <b>CN North-Beijing4</b> or <b>CN-Hong KongEU-Dublin</b> . <b>NOTE</b> <b>EU-Dublin</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 1-3](#).

**Table 1-3** Configuring the network

Parameter	Configuration
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation


Parameter	Configuration
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 1-4](#).

**Table 1-4** Configuring advanced settings

Parameter	Configuration
Cluster Name	dws-demo
Cluster Version	Use the recommended version, for example, 8.1.3.311.
Administrator Account	dbadmin
Administrator Password	-
Confirm Password	-
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**.

Region	Beijing4
Cluster Version	8.1.3.311
Public Network Address	<span style="border: 1px solid red; padding: 2px;">██.249.99.53</span>
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 1-5](#).

**Table 1-5** 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	-
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** Execute the following statement to create the **traffic** database:

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

**Step 4** Create a foreign table, which is used to identify and associate the source data on OBS.

#### NOTICE

- *<obs\_bucket\_name>* indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see [Supported Regions](#). GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- , and replace *<Access\_Key\_Id>* and *<Secret\_Access\_Key>* with the value obtained in [Making Preparations](#).
- If the message "ERROR: schema "xxx" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
create schema tpchobs;
set current_schema = 'tpchobs';
drop FOREIGN table if exists GCJL_OBS;
CREATE FOREIGN TABLE GCJL_OBS
(
    like traffic_data.GCJL
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/traffic-data/gcxx',
    format 'text',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
```

**Step 5** Execute the following statement to import data from the foreign table to the 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. Performing ANALYZE

This statement collects statistics related to ordinary tables in databases. The statistics are saved to the system catalog **PG\_STATISTIC**. When you run the planner, the statistics help you develop an efficient query execution plan.

Execute the following statement to generate the table statistics:

```
Analyze;
```

### 2. Querying the data volume of the data table

Execute the following statement to query the number of loaded data records:

```
set current_schema= traffic_data;
Select count(*) from traffic_data.gcjl;
```

### 3. Accurate vehicle query

Run the following statements to query the driving route of a vehicle by the license plate number and time segment. GaussDB(DWS) responds to the request in seconds.

```
set current_schema= traffic_data;
select hphm, kkbh, gcsj
from traffic_data.gcjl
where hphm = 'YD38641'
and gcsj between '2016-01-06' and '2016-01-07'
order by gcsj desc;
```

### 4. Fuzzy vehicle query

Run the following statements to query the driving route of a vehicle by the license plate number and time segment. GaussDB(DWS) responds to the request in seconds.

```
set current_schema= traffic_data;
select hphm, kkbh, gcsj
from traffic_data.gcjl
where hphm like 'YA23F%'
and kkbh in('508', '1125', '2120')
```

```
and gcsj between '2016-01-01' and '2016-01-07'  
order by hphm,gcsj desc;
```

# 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 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 2-1** Regions and OBS bucket names

Region	OBS Bucket
EU-Dublin	dws-demo-eu-west-101

## Scenario Description

Purpose: 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

- Register a GaussDB(DWS) account and check 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](#).

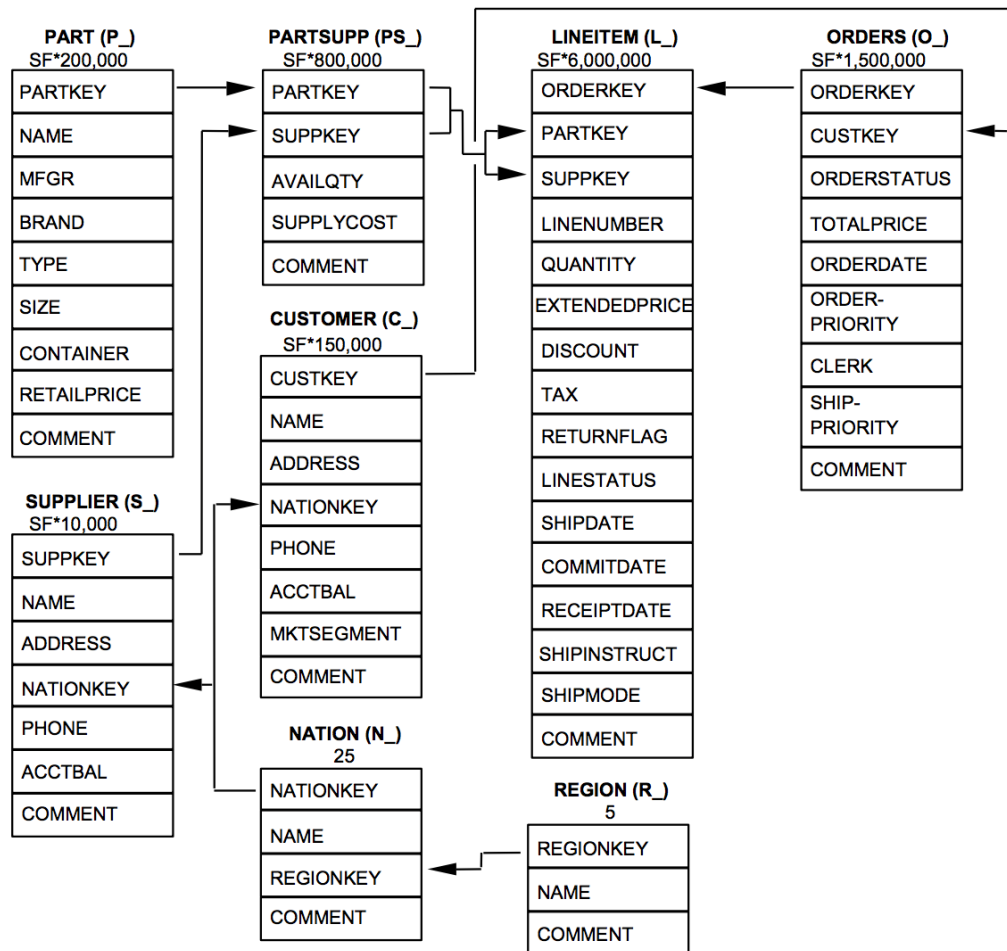
## Step 1: Importing Sample Data

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 2-1](#).

Figure 2-1 TPC-H data tables



Execute the following statements to create tables in the database.

```
CREATE schema tpch;
set current_schema = tpch;

drop table if exists region;
CREATE TABLE REGION
(
    R_REGIONKEY INT NOT NULL ,
    R_NAME CHAR(25) NOT NULL ,
    R_COMMENT VARCHAR(152)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by replication;

drop table if exists nation;
CREATE TABLE NATION
(
    N_NATIONKEY INT NOT NULL,
    N_NAME CHAR(25) NOT NULL,
    N_REGIONKEY INT NOT NULL,
    N_COMMENT VARCHAR(152)
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by replication;

drop table if exists supplier;
CREATE TABLE SUPPLIER
```

```
(
  S_SUPPKEY  BIGINT NOT NULL,
  S_NAME     CHAR(25) NOT NULL,
  S_ADDRESS  VARCHAR(40) NOT NULL,
  S_NATIONKEY INT NOT NULL,
  S_PHONE    CHAR(15) NOT NULL,
  S_ACCTBAL  DECIMAL(15,2) NOT NULL,
  S_COMMENT  VARCHAR(101) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(S_SUPPKEY);

drop table if exists customer;
CREATE TABLE CUSTOMER
(
  C_CUSTKEY  BIGINT NOT NULL,
  C_NAME     VARCHAR(25) NOT NULL,
  C_ADDRESS  VARCHAR(40) NOT NULL,
  C_NATIONKEY INT NOT NULL,
  C_PHONE    CHAR(15) NOT NULL,
  C_ACCTBAL  DECIMAL(15,2) NOT NULL,
  C_MKTSEGMENT CHAR(10) NOT NULL,
  C_COMMENT  VARCHAR(117) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(C_CUSTKEY);

drop table if exists part;
CREATE TABLE PART
(
  P_PARTKEY  BIGINT NOT NULL,
  P_NAME     VARCHAR(55) NOT NULL,
  P_MFGR     CHAR(25) NOT NULL,
  P_BRAND    CHAR(10) NOT NULL,
  P_TYPE     VARCHAR(25) NOT NULL,
  P_SIZE     BIGINT NOT NULL,
  P_CONTAINER CHAR(10) NOT NULL,
  P_RETAILPRICE DECIMAL(15,2) NOT NULL,
  P_COMMENT  VARCHAR(23) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(P_PARTKEY);

drop table if exists partsupp;
CREATE TABLE PARTSUPP
(
  PS_PARTKEY  BIGINT NOT NULL,
  PS_SUPPKEY  BIGINT NOT NULL,
  PS_AVAILQTY BIGINT NOT NULL,
  PS_SUPPLYCOST DECIMAL(15,2) NOT NULL,
  PS_COMMENT  VARCHAR(199) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(PS_PARTKEY);

drop table if exists orders;
CREATE TABLE ORDERS
(
  O_ORDERKEY  BIGINT NOT NULL,
  O_CUSTKEY   BIGINT NOT NULL,
  O_ORDERSTATUS CHAR(1) NOT NULL,
  O_TOTALPRICE DECIMAL(15,2) NOT NULL,
  O_ORDERDATE  DATE NOT NULL,
  O_ORDERPRIORITY CHAR(15) NOT NULL,
  O_CLERK     CHAR(15) NOT NULL,
  O_SHIPPRIORITY BIGINT NOT NULL,
  O_COMMENT   VARCHAR(79) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
```

```
distribute by hash(O_ORDERKEY);

drop table if exists lineitem;
CREATE TABLE LINEITEM
(
  L_ORDERKEY   BIGINT NOT NULL,
  L_PARTKEY    BIGINT NOT NULL,
  L_SUPPKEY    BIGINT NOT NULL,
  L_LINENUMBER BIGINT NOT NULL,
  L_QUANTITY   DECIMAL(15,2) NOT NULL,
  L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
  L_DISCOUNT  DECIMAL(15,2) NOT NULL,
  L_TAX        DECIMAL(15,2) NOT NULL,
  L_RETURNFLAG CHAR(1) NOT NULL,
  L_LINESTATUS CHAR(1) NOT NULL,
  L_SHIPDATE   DATE NOT NULL,
  L_COMMITDATE DATE NOT NULL,
  L_RECEIPTDATE DATE NOT NULL,
  L_SHIPINSTRUCT CHAR(25) NOT NULL,
  L_SHIPMODE    CHAR(10) NOT NULL,
  L_COMMENT    VARCHAR(44) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(L_ORDERKEY);
```

**Step 2** Create a foreign table, which is used to identify and associate the source data on OBS.

#### NOTICE

- *<obs\_bucket\_name>* indicates the OBS bucket name. Only some regions are supported. For details about the supported regions and OBS bucket names, see [Supported Regions](#). GaussDB(DWS) clusters do not support cross-region access to OBS bucket data.
- , and replace *<Access\_Key\_Id>* and *<Secret\_Access\_Key>* with the value obtained in [Making Preparations](#).
- If the message "ERROR: schema "xxx" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
CREATE schema tpchobs;
set current_schema='tpchobs';
drop FOREIGN table if exists region;
CREATE FOREIGN TABLE REGION
(
  like tpch.region
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/tpch/region.tbl',
  format 'text',
  delimiter '|',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on'
);

drop FOREIGN table if exists nation;
CREATE FOREIGN TABLE NATION
(
  like tpch.nation
```



```
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/tpch/nation.tbl',
  format 'text',
  delimiter '|',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on'
);

drop FOREIGN table if exists supplier;
CREATE FOREIGN TABLE SUPPLIER
(
  like tpch.supplier
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/tpch/supplier.tbl',
  format 'text',
  delimiter '|',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on'
);

drop FOREIGN table if exists customer;
CREATE FOREIGN TABLE CUSTOMER
(
  like tpch.customer
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/tpch/customer.tbl',
  format 'text',
  delimiter '|',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on'
);

drop FOREIGN table if exists part;
CREATE FOREIGN TABLE PART
(
  like tpch.part
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/tpch/part.tbl',
  format 'text',
  delimiter '|',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on'
);

drop FOREIGN table if exists partsupp;
CREATE FOREIGN TABLE PARTSUPP
(
  like tpch.partsupp
)
SERVER gsmpp_server
```

```
OPTIONS (  
    encoding 'utf8',  
    location 'obs://<obs_bucket_name>/tpch/partsupp.tbl',  
    format 'text',  
    delimiter '|',  
    access_key '<Access_Key_Id>',  
    secret_access_key '<Secret_Access_Key>',  
    chunksize '64',  
    IGNORE_EXTRA_DATA 'on'  
);  
drop FOREIGN table if exists orders;  
CREATE FOREIGN TABLE ORDERS  
(  
    like tpch.orders  
)  
SERVER gsmpp_server  
OPTIONS (  
    encoding 'utf8',  
    location 'obs://<obs_bucket_name>/tpch/orders.tbl',  
    format 'text',  
    delimiter '|',  
    access_key '<Access_Key_Id>',  
    secret_access_key '<Secret_Access_Key>',  
    chunksize '64',  
    IGNORE_EXTRA_DATA 'on'  
);  
drop FOREIGN table if exists lineitem;  
CREATE FOREIGN TABLE LINEITEM  
(  
    like tpch.lineitem  
)  
SERVER gsmpp_server  
OPTIONS (  
    encoding 'utf8',  
    location 'obs://<obs_bucket_name>/tpch/lineitem.tbl',  
    format 'text',  
    delimiter '|',  
    access_key '<Access_Key_Id>',  
    secret_access_key '<Secret_Access_Key>',  
    chunksize '64',  
    IGNORE_EXTRA_DATA 'on'  
);
```

**Step 3** Copy and execute the following statements to import the foreign table data to the corresponding database table.

Run the **insert** command to import the data in the OBS foreign table to the GaussDB(DWS) database table. The database kernel concurrently imports the OBS data at a high speed to GaussDB(DWS).

```
insert into tpch.lineitem select * from tpchobs.lineitem;  
insert into tpch.part select * from tpchobs.part;  
insert into tpch.partsupp select * from tpchobs.partsupp;  
insert into tpch.customer select * from tpchobs.customer;  
insert into tpch.supplier select * from tpchobs.supplier;  
insert into tpch.nation select * from tpchobs.nation;  
insert into tpch.region select * from tpchobs.region;  
insert into tpch.orders select * from tpchobs.orders;
```

It takes 10 minutes to import data.

----End

## Step 2: Performing Multi-Table Analysis and Theme Analysis

The following uses standard TPC-H query as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG\_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying revenue of a supplier in a region (TPCH-Q5)**

By executing the TPCH-Q5 query statement, you can query the revenue statistics of a spare parts supplier in a region. The revenue is calculated based on **sum(L\_extendedprice \* (1 - L\_discount))**. The statistics can be used to determine whether a local allocation center needs to be established in a given region.

Copy and execute the following TPCH-Q5 statement for query. This statement features multi-table join query with **GROUP BY**, **ORDER BY**, and **AGGREGATE**.

```
set current_schema='tpch';
Select
n_name,
sum(l_extendedprice * (1 - l_discount)) as revenue
from
customer,
orders,
lineitem,
supplier,
nation,
region
where
c_custkey = o_custkey
and l_orderkey = o_orderkey
and l_suppkey = s_suppkey
and c_nationkey = s_nationkey
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'ASIA'
and o_orderdate >= '1994-01-01'::date
and o_orderdate < '1994-01-01'::date + interval '1 year'
group by
n_name
order by
revenue desc;
```

- **Querying relationships between spare parts and suppliers (TPCH-Q16)**

By executing the TPCH-Q16 query statement, you can obtain the number of suppliers that can supply spare parts with the specified contribution conditions. This information can be used to determine whether there are sufficient suppliers when the order quantity is large and the task is urgent.

Copy and execute the following TPCH-Q16 statement for query. The statement features multi-table connection operations with group by, sort by, aggregate, deduplicate, and NOT IN subquery.

```
set current_schema='tpch';
select
p_brand,
p_type,
p_size,
count(distinct ps_suppkey) as supplier_cnt
from
partsupp,
part
where
p_partkey = ps_partkey
and p_brand <> 'Brand#45'
and p_type not like 'MEDIUM POLISHED%'
```

```
and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
and ps_suppkey not in (
  select
    s_suppkey
  from
    supplier
  where
    s_comment like '%Customer%Complaints%'
)
group by
  p_brand,
  p_type,
  p_size
order by
  supplier_cnt desc,
  p_brand,
  p_type,
  p_size
limit 100;
```

- **Querying revenue loss of small orders (TPCH-Q17)**

You can query the average annual revenue loss if there are no small orders. Filter out small orders that are lower than the 20% of the average supply volume, and calculate the total amount of those small orders to figure out the average annual revenue loss.

Copy and execute the following TPCH-Q17 statement for query. The statement features multi-table connection operations with aggregate and aggregate subquery.

```
set current_schema='tpch';
select
  sum(l_extendedprice) / 7.0 as avg_yearly
from
  lineitem,
  part
where
  p_partkey = l_partkey
and p_brand = 'Brand#23'
and p_container = 'MED BOX'
and l_quantity < (
  select 0.2 * avg(l_quantity)
  from lineitem
  where l_partkey = p_partkey
);
```

# 3 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 3-1 Regions and OBS bucket names

Region	OBS Bucket
EU-Dublin	dws-demo-eu-west-101

## Preparations

- You have registered a GaussDB(DWS) account, and the account is not in arrears or frozen.

- You have obtained the AK and SK of the account.
- A cluster has been created and connected using Data Studio. For details, see [Step 1: Creating a Cluster](#) and [Step 2: Using Data Studio to Connect to a Cluster](#).

## Step 1: Importing Sample Data from the Retail Department Store

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the sample data from retail department stores and perform queries.

**Step 1** Execute the following statement to create the **retail** database:

```
create database retail encoding 'utf8' template template0;
```

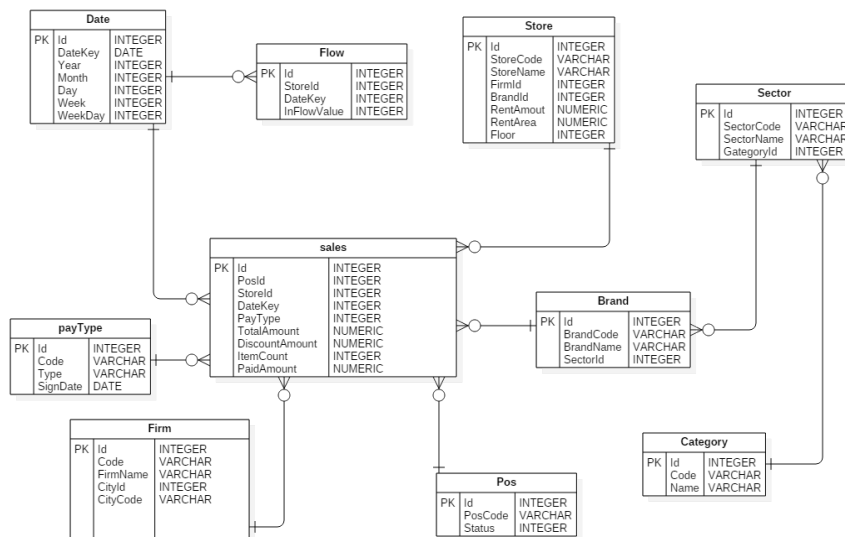
**Step 2** Perform the following steps to switch to the new database:

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

**Step 3** Create a database table.

The sample data consists of 10 database tables whose associations are shown in [Figure 3-1](#).

**Figure 3-1** 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.
- , and replace *<Access\_Key\_Id>* and *<Secret\_Access\_Key>* with the value obtained in [Preparations](#).
- If the message "ERROR: schema "xxx" does not exist Position" is displayed when you create a foreign table, the schema does not exist. Perform the previous step to create a schema.

```
create schema retail_obs_data;
set current_schema='retail_obs_data';
drop FOREIGN table if exists SALES_OBS;
CREATE FOREIGN TABLE SALES_OBS
(
  like retail_data.SALES
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/retail-data/sales',
  format 'csv',
  delimiter ',',
  access_key '<Access_Key_Id>',
```



```
secret_access_key '<Secret_Access_Key>',
chunksize '64',
IGNORE_EXTRA_DATA 'on',
header 'on'
);

drop FOREIGN table if exists FLOW_OBS;
CREATE FOREIGN TABLE FLOW_OBS
(
    like retail_data.flow
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/flow',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists BRAND_OBS;
CREATE FOREIGN TABLE BRAND_OBS
(
    like retail_data.brand
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/brand',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists CATEGORY_OBS;
CREATE FOREIGN TABLE CATEGORY_OBS
(
    like retail_data.category
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/category',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists DATE_OBS;
CREATE FOREIGN TABLE DATE_OBS
(
    like retail_data.date
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
```

```
location 'obs://<obs_bucket_name>/retail-data/date',
format 'csv',
delimiter ',',
access_key '<Access_Key_Id>',
secret_access_key '<Secret_Access_Key>',
chunksize '64',
IGNORE_EXTRA_DATA 'on',
header 'on'
);

drop FOREIGN table if exists FIRM_OBS;
CREATE FOREIGN TABLE FIRM_OBS
(
    like retail_data.firm
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/firm',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists PAYTYPE_OBS;
CREATE FOREIGN TABLE PAYTYPE_OBS
(
    like retail_data.paytype
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/paytype',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists POS_OBS;
CREATE FOREIGN TABLE POS_OBS
(
    like retail_data.pos
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/pos',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

drop FOREIGN table if exists SECTOR_OBS;
CREATE FOREIGN TABLE SECTOR_OBS
(
```

```
    like retail_data.sector
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/retail-data/sector',
  format 'csv',
  delimiter ',',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on',
  header 'on'
);

drop FOREIGN table if exists STORE_OBS;
CREATE FOREIGN TABLE STORE_OBS
(
  like retail_data.store
)
SERVER gsmpp_server
OPTIONS (
  encoding 'utf8',
  location 'obs://<obs_bucket_name>/retail-data/store',
  format 'csv',
  delimiter ',',
  access_key '<Access_Key_Id>',
  secret_access_key '<Secret_Access_Key>',
  chunksize '64',
  IGNORE_EXTRA_DATA 'on',
  header 'on'
);
```

**Step 5** Copy and execute the following statements to import the foreign table data to the cluster:

```
insert into retail_data.store select * from retail_obs_data.STORE_OBS;
insert into retail_data.sector select * from retail_obs_data.SECTOR_OBS;
insert into retail_data.paytype select * from retail_obs_data.PAYTYPE_OBS;
insert into retail_data.firm select * from retail_obs_data.FIRM_OBS;
insert into retail_data.flow select * from retail_obs_data.FLOW_OBS;
insert into retail_data.category select * from retail_obs_data.CATEGORY_OBS;
insert into retail_data.date select * from retail_obs_data.DATE_OBS;
insert into retail_data.pos select * from retail_obs_data.POS_OBS;
insert into retail_data.brand select * from retail_obs_data.BRAND_OBS;
insert into retail_data.sales select * from retail_obs_data.SALES_OBS;
```

It takes some time to import data.

**Step 6** Copy and execute the following statement to create the **v\_sales\_flow\_details** view:

```
set current_schema='retail_data';
CREATE VIEW v_sales_flow_details AS
SELECT
  FIRM.ID FIRMID, FIRM.NAME FIRNAME, FIRM. CITYCODE,
  CATEGORY.ID CATEGORYID, CATEGORY.NAME CATEGORYNAME,
  SECTOR.ID SECTORID, SECTOR.SECTORNAME,
  BRAND.ID BRANDID, BRAND.BRANDNAME,
  STORE.ID STOREID, STORE.STORENAME, STORE.RENTAMOUNT, STORE.RENTAREA,
  DATE.DATEKEY, SALES.TOTALAMOUNT, DISCOUNTAMOUNT, ITEMCOUNT, PAIDAMOUNT, INFLOWVALUE
FROM SALES
INNER JOIN STORE ON SALES.STOREID = STORE.ID
INNER JOIN FIRM ON STORE.FIRMID = FIRM.ID
INNER JOIN BRAND ON STORE.BRANDID = BRAND.ID
INNER JOIN SECTOR ON BRAND.SECTORID = SECTOR.ID
INNER JOIN CATEGORY ON SECTOR.CATEGORYID = CATEGORY.ID
```

```
INNER JOIN DATE ON SALES.DATEKEY = DATE.ID  
INNER JOIN FLOW ON FLOW.DATEKEY = DATE.ID AND FLOW.STOREID = STORE.ID;
```

----End

## Step 2: Performing Operations Status Analysis

The following uses standard query of retail information from department stores as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG\_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying the monthly sales revenue of each store**

Copy and execute the following statements to query the total revenue of each store in a certain month:

```
set current_schema='retail_data';  
SELECT DATE_TRUNC('month',datekey)  
AT TIME ZONE 'UTC' AS __timestamp,  
SUM(paidamount)  
AS sum__paidamount  
FROM v_sales_flow_details  
GROUP BY DATE_TRUNC('month',datekey) AT TIME ZONE 'UTC'  
ORDER BY SUM(paidamount) DESC;
```

- **Querying the sales revenue and price-rent ratio of each store**

Copy and execute the following statement to query the sales revenue and price-rent ratio of each store:

```
set current_schema='retail_data';  
SELECT firname AS firname,  
storename AS storename,  
SUM(paidamount)  
AS sum__paidamount,  
AVG(RENTAMOUNT)/SUM(PAIDAMOUNT)  
AS rentamount_sales_rate  
FROM v_sales_flow_details  
GROUP BY firname, storename  
ORDER BY SUM(paidamount) DESC;
```

- **Analyzing the sales revenue of each city**

Copy and execute the following statement to analyze and query the sales revenue of all provinces:

```
set current_schema='retail_data';  
SELECT citycode AS citycode,  
SUM(paidamount)  
AS sum__paidamount  
FROM v_sales_flow_details  
GROUP BY citycode  
ORDER BY SUM(paidamount) DESC;
```

- **Analyzing and comparing the price-rent ratio and customer flow conversion rate of each store**

```
set current_schema='retail_data';  
SELECT brandname AS brandname,  
firname AS firname,  
SUM(PAIDAMOUNT)/AVG(RENTAREA) AS sales_rentarea_rate,  
SUM(ITEMCOUNT)/SUM(INFLOWVALUE) AS poscount_flow_rate,  
AVG(RENTAMOUNT)/SUM(PAIDAMOUNT) AS rentamount_sales_rate  
FROM v_sales_flow_details
```

```
GROUP BY brandname, firname  
ORDER BY sales_rentarea_rate DESC;
```

- **Analyzing brands in the retail industry**

```
set current_schema='retail_data';  
SELECT categoryname AS categoryname,  
brandname AS brandname,  
SUM(paidamount) AS sum__paidamount  
FROM v_sales_flow_details  
GROUP BY categoryname,  
brandname  
ORDER BY sum__paidamount DESC;
```

- **Querying daily sales information of each brand**

```
set current_schema='retail_data';  
SELECT brandname AS brandname,  
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC' AS __timestamp,  
SUM(paidamount) AS sum__paidamount  
FROM v_sales_flow_details  
WHERE datekey >= '2016-01-01 00:00:00'  
AND datekey <= '2016-01-30 00:00:00'  
GROUP BY brandname,  
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC'  
ORDER BY sum__paidamount ASC  
LIMIT 50000;
```

# 4 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 "Creating an ECS" in the *Huawei Cloud Stack x.x.x User Guide*. When the ECS is created, log in to the ECS. For details, see "Remotely Logging In to a Linux ECS Using a Password (SSH)".

### 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>Europe-Dublin</b> . <b>NOTE</b> <ul style="list-style-type: none"><li>• <b>CN North-Beijing4</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
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.

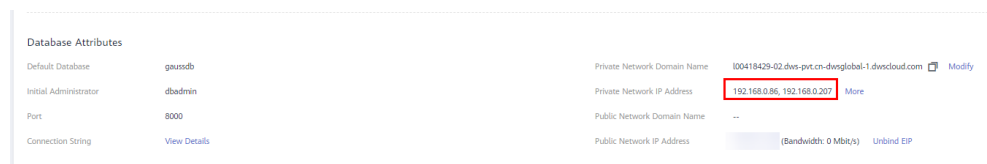


Parameter	Configuration
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24) <b>NOTICE</b> Ensure that the cluster and the ECS are in the same VPC subnet.
Security Group	Automatic creation
EIP	Buy now
Enterprise Project	default
Advanced settings	Default

**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.eu-west-101.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 TSTag,  
  frequency numeric TSField,  
  angle numeric TSField,  
  time timestamptz 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

- When writing table creation statements, you can optimize the sequence of tag columns. More unique columns (more distinct values) are written in the front to improve the performance in time sequence scenarios.
- 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 **ttl** 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.
- **ttl**: 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  $\text{nowtime} - \text{partition boundary} > \text{ttl}$ . 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 boundary value. For example, create the time series table **GENERATOR1** with the default start boundary of partition **P1** as **2022-05-30 16:32:45** and partition **P2** as **2022-05-31 16:56:12**.

```
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 TSTag,  
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"}  
.....
```

# 5 Best Practices of Hot and Cold Data Management

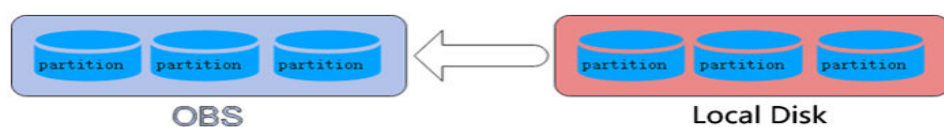
## Scenarios

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.



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. [View data distribution..](#)

## Creating a cluster


- Step 1** Log in to the Huawei Cloud management console.
- Step 2** Choose **Service List > Analytics > Data Warehouse Service**. On the page that is displayed, click **Create Cluster** in the upper right corner.
- Step 3** Configure the parameters according to [Table 5-1](#).

**Table 5-1** Software configuration

Parameter	Configuration
Region	Select EU-Dublin. <b>NOTE</b> EU-Dublin 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

Parameter	Configuration
Administrator Account	dbadmin
Administrator Password	-
Confirm Password	-
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1Mbit/s
Advanced Settings	Default

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

**Step 5** Wait about 6 minutes. After the cluster is created, click  next to the cluster name. On the displayed cluster information page, record the value of **Public Network Address**.

----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.eu-west-101.myhuaweicloud.com/dws/download/dws_client_8.1.x_redhat_x64.zip --no-check-certificate
```

**Step 2** Decompress the client.

```
cd <Path_for_storing_the_client> unzip dws_client_8.1.x_redhat_x64.zip
```

Where,

- *<Path\_for\_storing\_the\_client>*: Replace it with the actual path.
- *dws\_client\_8.1.x\_redhat\_x64.zip*: This is the client tool package name of **RedHat x64**. Replace it with the actual name.

**Step 3** Configure the GaussDB(DWS) client.

```
source gsql_env.sh
```

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

```
All things done.
```

**Step 4** Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

----End

## Creating Hot and Cold Tables

Create column-store cold and hot tables and set the hot data validity period LMT to 100 days.

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:100')  
PARTITION BY RANGE (i)  
(  
PARTITION P1 VALUES LESS THAN(5),  
PARTITION P2 VALUES LESS THAN(10),  
PARTITION P3 VALUES LESS THAN(15),  
PARTITION P8 VALUES LESS THAN(MAXVALUE)  
)  
ENABLE ROW MOVEMENT;
```

## Hot and Cold Data Switchover

Switch cold data to the OBS tablespace.

- Automatic switchover: The scheduler automatically triggers the switchover at 00:00 every day.

The automatic switchover time can be customized. For example, the time can be changed to 06:30 every morning.

```
select * from pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');  
pg_obs_cold_refresh_time  
-----  
SUCCESS  
(1 row)
```

- Manual

Perform the following operations to manually switch a single table:

```
alter table lifecycle_table refresh storage;  
ALTER TABLE
```

Perform the following operations to switch over all cold and hot tables in batches:

```
select pg_catalog.pg_refresh_storage();  
pg_refresh_storage  
-----  
(1,0)  
(1 row)
```

## View data distribution.

View data distribution in hot and cold tables.



View the data distribution in a single table:

```
select * from pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
schemaname | tablename | nodename | hotpartition | coldpartition | switchablepartition | hotdatasize |
| colddatasize | switchabledatasize
-----+-----+-----+-----+-----+-----+-----+-----
public | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8 | | | 96 KB | 0 bytes |
0 bytes
public | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8 | | | 96 KB | 0 bytes |
0 bytes
public | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8 | | | 96 KB | 0 bytes |
0 bytes
(3 rows)
```

View data distribution in all hot and cold tables:

```
select * from pg_catalog.pg_lifecycle_node_data_distribute();
schemaname | tablename | nodename | hotpartition | coldpartition | switchablepartition | hotdatasize |
| colddatasize | switchabledatasize
-----+-----+-----+-----+-----+-----+-----+-----
public | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8 | | | 98304 | 0
| 0
public | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8 | | | 98304 | 0
| 0
public | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8 | | | 98304 | 0
| 0
(3 rows)
```

# 6 Best Practices for Automatic Partition Management

---

## Scenarios

For partition tables whose partition columns are time, the automatic partition management function can be added to automatically create partitions and delete expired partitions, reducing partition table maintenance costs and improving query performance. To facilitate data query and maintenance, the time column is often used as the partition column of a partitioned table that stores time-related data, such as e-commerce order information and real-time IoT data. When the time-related data is imported to a partitioned table, the table should have partitions of the corresponding time ranges. Common partition tables do not automatically create new partitions or delete expired partitions. Therefore, maintenance personnel need to periodically create new partitions and delete expired partitions, leading to increased O&M costs.

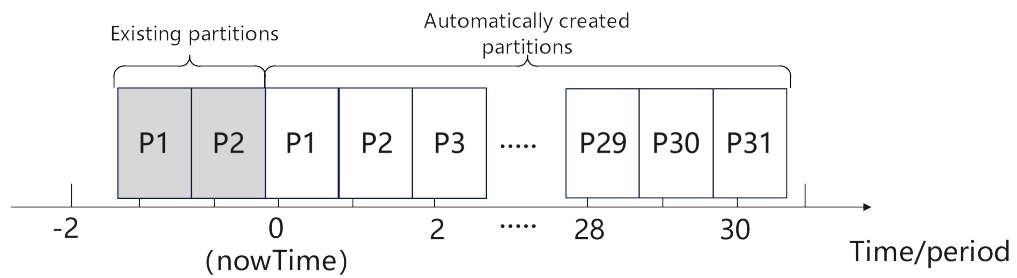
Addressing this, GaussDB(DWS) introduces the automatic partition management feature. You can set the table-level parameters **period** and **ttl** to enable the automatic partition management function, which automatically creates partitions and deletes expired partitions, reducing partition table maintenance costs and improving query performance.

**period**: interval for automatically creating partitions. The default value is 1 day. The value range is 1 hour ~ 100 years.

**ttl**: time for automatically eliminate partitions. The value range is 1 hour ~ 100 years. The partition elimination policy is based on the condition that  $\text{nowtime} - \text{partition boundary} > \text{ttl}$ . Partitions that meet this condition will be eliminated.

- Automatic partition creation

One or more partitions are automatically created at the interval specified by **period** to make the maximum partition boundary time greater than  $\text{nowTime} + 30 \times \text{period}$ . As long as there is an automatically created partition, real-time data will not fail to be imported within the next 30 periods.

**Figure 6-1** 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 **ttl** cannot exist independently. You must set **period** in advance or at the same time, and the value of **ttl** must be greater than or equal to that of **period**.
- During online cluster scale-out, partitions cannot be automatically added. Partitions reserved each time partitions are added will ensure that services are not affected.

## Creating an ECS

For details, see "Creating an ECS" in the *Huawei Cloud Stack x.x.x User Guide*. When the ECS is created, log in to the ECS. For details, see "Remotely Logging In to a Linux ECS Using a Password (SSH)".

**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 6-1](#).

**Table 6-1** Software configuration

Parameter	Configuration
Region	Select EU-Dublin. <b>NOTE</b> EU-Dublin 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	-
Confirm Password	-
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**.

----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.eu-west-101.myhuaweicloud.com/dws/download/dws_client_8.1.x_redhat_x64.zip --no-check-certificate
```

**Step 2** Decompress the client.

```
cd <Path_for_storing_the_client> unzip dws_client_8.1.x_redhat_x64.zip
```

Where,

- *<Path\_for\_storing\_the\_client>*: Replace it with the actual path.
- *dws\_client\_8.1.x\_redhat\_x64.zip*: This is the client tool package name of **RedHat x64**. Replace it with the actual name.

**Step 3** Configure the GaussDB(DWS) client.

```
source gsql_env.sh
```

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

```
All things done.
```

**Step 4** Use the gsql client to connect to a GaussDB(DWS) database (using the password you defined when creating the cluster).

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

```
----End
```

## Automatic partition management

The partition management function is bound to the table-level parameters **period** and **ttl**. Automatic partition creation is enabled with the enabling of **period**, and automatic partition deletion is enabled with the enabling of **ttl**. 30 seconds after **period** or **ttl** is set, the automatic partition creation or deletion works for the first time.

You can enable the partition management function in either of the following ways:

- Specify **period** and **ttl** when creating a table.

This way is applicable when you create a partition management table. There are two syntaxes for creating a partition management table. One specifies partitions, and the other does not.

If partitions are specified when a partition management table is created, the syntax rules are the same as those for creating an ordinary partition table. The only difference is that the syntax specifies the table-level parameters **period** and **ttl**.

The following example shows how to create a partition management table **CPU1** and specify partitions.

```
CREATE TABLE CPU1(
  id integer,
  IP text,
  time timestamp
) with (TTL='7 days',PERIOD='1 day')
partition by range(time)
(
  PARTITION P1 VALUES LESS THAN('2023-02-13 16:32:45'),
  PARTITION P2 VALUES LESS THAN('2023-02-15 16:48:12')
);
```

When creating a partition management table, you can specify only the partition key but not partitions. In this case, two default partitions will be created with **period** as the partition time range. The boundary time of the first default partition is the first hour, day, week, month, or year past the current time. The time unit is selected based on the maximum unit of PERIOD. The boundary time of the second default partition is the boundary time of the first partition plus PERIOD. Assume that the current time is 2023-02-17 16:32:45, and the boundary of the first default partition is described in the following table.

**Table 6-2** Description of the period parameter

period	Maximum PERIOD Unit	Boundary of First Default Partition
1 hour	Hour	2023-02-17 17:00:00
1 day	Day	2023-02-18 00:00:00
1 month	Month	2023-03-01 00:00:00

period	Maximum PERIOD Unit	Boundary of First Default Partition
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 SET** command to set **period** and **tll**.

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 **tll** 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 **tll** to disable the partition management function.

#### NOTE

- The **period** cannot be deleted separately with **TTL**.
- Time series tables do 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);
```



# 7 Creating a Cluster and Connecting to It

---

## 7.1 Step 1: Starting Preparations

This guide is an introductory tutorial that demonstrates how to create a sample GaussDB(DWS) cluster, connect to the cluster database, import the sample data from OBS, and analyze the sample data. You can use this tutorial to evaluate GaussDB(DWS).

Before creating a GaussDB(DWS) cluster, ensure that the following prerequisites are met:

- [Registering a Public Cloud Account](#)
- [Determining the Cluster Ports](#)

### Registering a Public Cloud Account

If you do not have a public cloud account, register one.

1. Open the official public cloud website (<https://www.huaweicloud.com/eu/>) and click **Register** in the upper right corner. The registration page is displayed.
2. Fill in user information as instructed to complete the registration.
3. After the registration is successful, you can be automatically logged in to Huawei Cloud.

### Determining the Cluster Ports

- When creating a GaussDB(DWS) cluster, you need to specify a port for SQL clients or applications to access the cluster.
- If your client is behind a firewall, you need an available port so that you can connect to the cluster and perform query and analysis from the SQL client tool.
- If you do not know an available port, contact the network administrator to specify an open port on your firewall. The ports supported by GaussDB(DWS) range from 8000 to 30000.
- After a cluster is created, its port number cannot be changed. Ensure that the port specified is available.

## 7.2 Step 2: Creating a Cluster

Before using GaussDB(DWS) to analyze data, create a cluster. A cluster consists of multiple nodes in the same subnet. These nodes jointly provide services. This section describes how to create a GaussDB(DWS) cluster.

### Creating a Cluster

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

**Step 2** Choose **Clusters** in the navigation pane on the left.

**Step 3** On the **Clusters** page, click **Create Cluster** in the upper right corner.

**Step 4** Select the region to which the cluster to be created belongs.

- **Region:** Select **EU-Dublin**.
- **AZ:** Retain the default value.

**Step 5** Configure node parameters.

- **Resource:** For example, **Standard**.
- **CPU Architecture:** Select a CPU architecture based on your requirements, for example, **x86**.
- **Node Flavor:** Retain the default value.
- **Nodes:** Retain the default value. At least **3** nodes are required.

Figure 7-1 Configure node parameters.

The screenshot displays the 'Configure node parameters' interface. At the top, there are three tabs: 'Standard' (selected), 'Hybrid', and 'Stream'. Below the tabs, there are several configuration sections:

- Resource:** Includes highlights for Standard, Hybrid, and Stream warehouses.
- Compute Resource:** Set to 'EC2'.
- Storage Type:** Options for 'Cloud SSD' and 'Local SSD'.
- CPU Architecture:** Set to 'x86'.
- Node Flavor Table:**

Flavor Name	vCPUs   Memory	Hot storage
<input checked="" type="radio"/> dwsx2.xlarge.m7	4 vCPUs   32GB	20.00-2,000.00 GB SSD
<input type="radio"/> dwsx2.2xlarge.m7	8 vCPUs   64GB	100.00-4,000.00 GB SSD
<input type="radio"/> dwsx2.4xlarge.m7 (Sold Out)	16 vCPUs   128GB	100.00-8,000.00 GB SSD
<input type="radio"/> dwsx2.8xlarge.m7 (Sold Out)	32 vCPUs   256GB	100.00-16,000.00 GB SSD
<input type="radio"/> dwsx2.16xlarge.m7 (Sold Out)	64 vCPUs   512GB	100.00-32,000.00 GB SSD

Below the table, there are input fields for 'Hot storage' (set to 50 GB per node), 'Cold storage' (with a note about OBS), and 'Nodes' (set to 3). A 'Total' summary at the bottom shows: 'dwsx2.xlarge.m7 | 12 vCPUs | 96 GB Memory | 150.00 GB hot storage'.


**Step 6** Configure cluster parameters.

- **Cluster Name:** Enter **dws-demo**.
- **Cluster Version:** The current cluster version is displayed and cannot be changed.
- **Default Database:** The value is **gaussdb**, which cannot be changed.
- **Administrator Account:** The default value is **dbadmin**. Use the default value. After a cluster is created, the client uses this database administrator account and its password to connect to the cluster's database.
- **Administrator Password:** Enter the password.

- **Confirm Password:** Enter the database administrator password again.
- **Database Port:** Use the default port number. This port is used by the client or application to connect to the cluster's database.

**Figure 7-2** Configuring the cluster

**Step 7** Configure network parameters.

- **VPC:** You can select an existing VPC from the drop-down list. If no VPC has been configured, click **View VPC** to enter the VPC management console to create one, for example, **vpc-dws**. Then, go back to the page for creating a cluster on the GaussDB(DWS) management console, click  next to the **VPC** drop-down list, and select the new VPC.
- **Subnet:** When you create a VPC, a subnet is created by default. You can select the corresponding subnet.
- **Security Group:** Select **Automatic creation**.

The automatically created security group is named **GaussDB(DWS)-<Cluster name>-<GaussDB(DWS) cluster database port>**. The outbound allows all access requests, while the inbound enables only **Database Port** for access requests from clients or applications.

If you select a custom security group, add an inbound rule to it to enable **Database Port** for client hosts to access GaussDB(DWS). [Table 7-1](#) shows an example. For details about how to add an inbound rule, see section [Adding a Security Group Rule](#) in the *Virtual Private Cloud User Guide*.

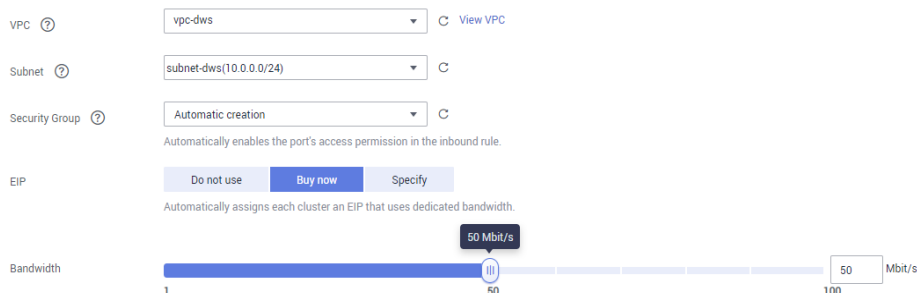
**Table 7-1** Inbound rule example

Parameter	Example Value
Protocol/Application	TCP

Parameter	Example Value
Port	8000 <b>NOTE</b> Enter the value of <b>Database Port</b> set when creating the GaussDB(DWS) cluster. This port is used for receiving client connections to GaussDB(DWS). The default port number is 8000.
Source	Select <b>IP address</b> , and enter the IP address and subnet mask of the client host that accesses GaussDB(DWS), for example, <b>192.168.0.10/16</b> .

- **EIP:** Select **Buy now** to purchase an EIP for the cluster as the cluster's public network IP address, and set its **Bandwidth**.

**Figure 7-3** Configuring the network



**Step 8** Select **Default** for **Advanced Settings** in this example.

- **Default:** indicates that the following advanced settings use the default configurations.
  - **CNs:** Three CNs are deployed by default.
  - **Tag:** By default, no tag is added to the cluster.
  - **Encrypt DataStore:** This parameter is disabled by default, indicating that the database is not encrypted.
- **Custom:** Select this option to configure the following advanced parameters: **CNs, Tag, Encrypt DataStore**

**Step 9** Click **Next**. The **Confirm** page is displayed.

**Step 10** Click **Submit**.

After the submission is successful, the creation starts. Click **Back to Cluster List**. The **Clusters** page is displayed. The initial status of the cluster is **Creating**. Cluster creation takes some time. Wait for a while. Clusters in the **Available** state are ready for use.

----End

## 7.3 Step 3: Connecting to a Cluster

### Scenario


This section describes how to use a database client to connect to the database in a GaussDB(DWS) cluster. In the following example, the Data Studio client tool is used for connection through the public network address. You can also use other SQL clients to connect to the cluster. For more connection methods, see "Connecting to a Cluster" in the *Data Warehouse Service (DWS) Management Guide*.

1. Obtain the name, username, and password of the database to be connected.  
If you use the client to connect to the cluster for the first time, use the administrator username and password set in [Step 2: Creating a Cluster](#) to connect to the default database gaussdb.
2. [Obtaining the Public Network Address of the Cluster](#): Connect to the cluster database using the public network address.
3. [Connecting to the Cluster Database Using Data Studio](#): Download and configure the Data Studio client and connect to the cluster database.

### Obtaining the Public Network Address of the Cluster

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

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

**Step 3** In the cluster list, select a created cluster (for example, **dws-demo**) and click  next to the cluster name to obtain the public network address.

The public network address will be used in [Connecting to the Cluster Database Using Data Studio](#).

----End

### Connecting to the Cluster Database Using Data Studio

**Step 1** GaussDB(DWS) provides a Windows-based Data Studio GUI client. The tool depends on JDK, so you must install Java 1.8.0\_141 or later on the client host.

In the Windows operating system, you can download the required JDK version from the [official website of JDK](#), and install it by following the installation guide.

**Step 2** Log in to the GaussDB(DWS) management console.

**Step 3** Click **Connections**.

**Step 4** On the **Download Client and Driver** page, download **Data Studio GUI Client**.

- Select **Windows x86** or **Windows x64** based on the operating system type and click **Download** to download the Data Studio tool matching the current cluster version.

If clusters of different versions are available, you will download the Data Studio tool matching the earliest cluster version after clicking **Download**. If

there is no cluster, you will download the Data Studio tool of the earliest version after clicking **Download**. GaussDB(DWS) clusters are compatible with earlier versions of Data Studio tools.

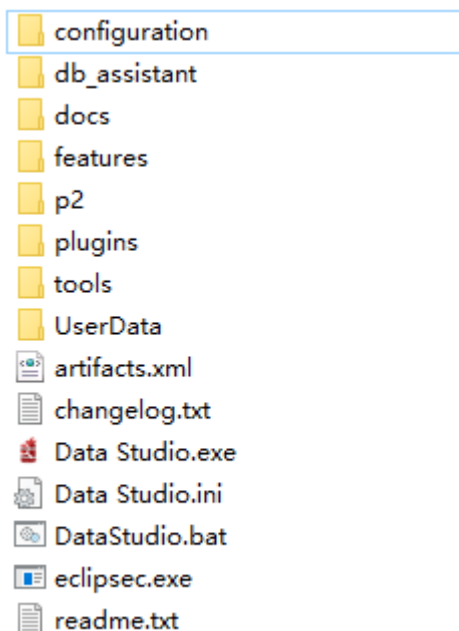
- Click **Historical Version** to download the corresponding Data Studio version. You are advised to download the Data Studio based on the cluster version.

If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the corresponding client. In the cluster list on the **Clusters** page, click the name of the specified cluster to go to the **Cluster Information** page and view the cluster version.

**Step 5** Decompress the downloaded client software package (32-bit or 64-bit) to the installation directory.

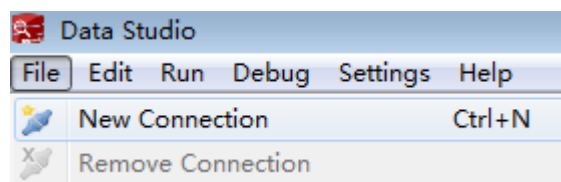
**Step 6** Open the installation directory and double-click **Data Studio.exe** to start the Data Studio client. See [Figure 7-4](#).

**Figure 7-4** Starting the client



**Step 7** Choose **File > New Connection** from the main menu. See [Figure 7-5](#).

**Figure 7-5** Creating a connection



**Step 8** In the displayed **New Database Connection** window, enter the connection parameters.

**Table 7-2** Connection parameters

Parameter	Description	Example
Database Type	Select <b>HUAWEI CLOUD DWS</b> .	HUAWEI CLOUD DWS
Connection Name	Name of a connection	dws-demo
Host	IP address (IPv4) or domain name of the cluster to be connected	-
Host Port	Database port	8000
Database Name	Database name	gaussdb
User Name	Username for connecting to the database	-
Password	Password for logging in to the database to be connected	-
Save Password	Select an option from the drop-down list: <ul style="list-style-type: none"><li>• <b>Current Session Only</b>: The password is saved only in the current session.</li><li>• <b>Do Not Save</b>: The password is not saved.</li></ul>	-
Enable SSL	If <b>Enable SSL</b> is selected, the client can use SSL to encrypt connections. The SSL mode is more secure than common modes, so you are advised to enable SSL connection.	-

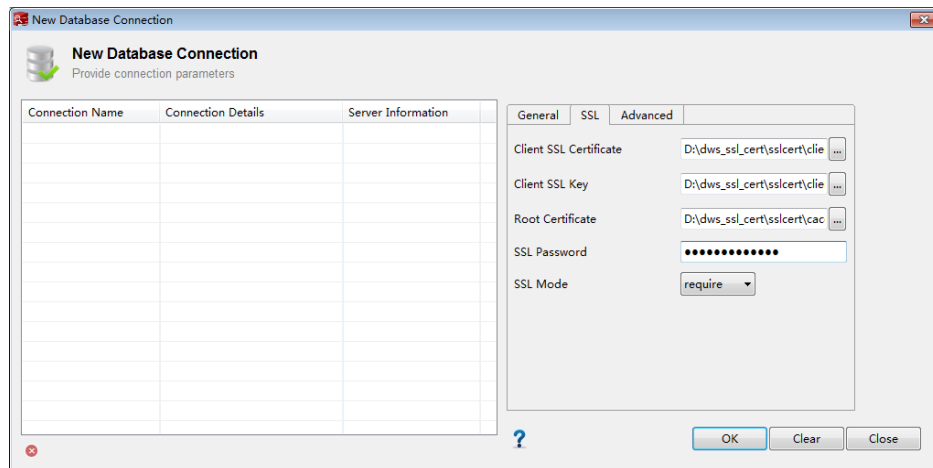
If SSL is enabled, download the SSL certificate by referring to [Downloading an SSL Certificate](#) and decompress the certificate file. Click the **SSL** tab and configure the following parameters:

**Table 7-3** Configuring SSL parameters

Parameter	Description
Client SSL Certificate	Select the <b>sslcert\client.crt</b> file in the decompressed SSL certificate directory.
Client SSL Key	Only the PK8 format is supported. Select the <b>sslcert\client.key.pk8</b> file in the directory where the SSL certificate is decompressed.

Parameter	Description
Root Certificate	When <b>SSL Mode</b> is set to <b>verify-ca</b> , the root certificate must be configured. Select the <b>sslcert\cacert.pem</b> file in the decompressed SSL certificate directory.
SSL Password	Set the password for the client SSL key in PK8 format.
SSL Mode	Supported SSL modes include: <ul style="list-style-type: none"> <li>• require</li> <li>• verify-ca</li> </ul> GaussDB(DWS) does not support the <b>verify-full</b> mode.

Figure 7-6 Configuring SSL parameters

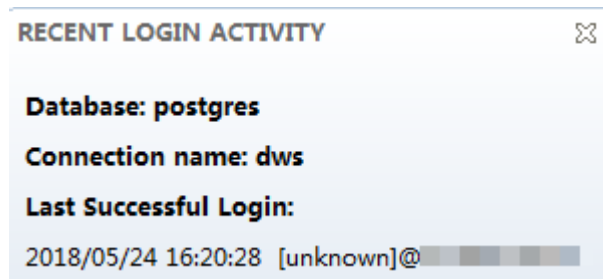


**Step 9** Click **OK** to establish the database connection.

If SSL is enabled, click **Continue** in the displayed **Connection Security Alert** dialog box.

After the login is successful, the **RECENT LOGIN ACTIVITY** dialog box is displayed, indicating that Data Studio is connected to the database. You can run the SQL statement in the **SQL Terminal** window on the Data Studio page.

Figure 7-7 Successful login





For details about how to use other functions of Data Studio, press **F1** to view the Data Studio user manual.

----End

## 7.4 Step 4: Viewing Other Documents and Deleting Resources

### Viewing Other Relevant Documents

After performing the steps in preceding sections, you can refer to the documentation listed as follows for more information about GaussDB(DWS):

- **Data Warehouse Service (DWS) Management Guide:** This guide provides detailed information about the concepts and operations related to creating, managing, monitoring, and connecting clusters.
- **Data Warehouse Service (DWS) Developer Guide:** This guide provides comprehensive and detailed information on how to build, manage, and query GaussDB(DWS) databases, covering SQL syntax, user management, and data import and export.

### Deleting Resources

After performing the steps in "Getting Started", if you do not need to use the sample data, clusters, ECSs, or VPCs, delete the resources so that your service quotas will not be wasted or occupied.

**Step 1** Delete a GaussDB(DWS) cluster.

On the GaussDB(DWS) management console, click **Clusters**, locate the row that contains **dws-demo** in the cluster list, and choose **More > Delete**. In the dialog box that is displayed, select **Release the EIP bound with the cluster** and click **OK**.

If the cluster to be deleted uses an automatically created security group that is not used by other clusters, the security group is automatically deleted when the cluster is deleted.

**Step 2** Delete a subnet. Before deleting the subnet, ensure that it is not bound to other resources.

Log in to the VPC management console. In the navigation tree on the left, click **Virtual Private Cloud**. In the VPC list, click **vpc-dws**. In the subnet list, locate the row that contains **subnet-dws** and click **Delete**.

**Step 3** Delete a VPC. Before deleting the VPC, ensure that it is not bound to other resources.

Log in to the VPC management console, locate the row that contains **vpc-dws** in the VPC list, and click **Delete**.

For details, see "VPC and Subnet > Deleting a VPC" in the *Virtual Private Cloud User Guide*.

----End

# 8 Using CDM to Migrate MySQL Data to the GaussDB(DWS) Cluster

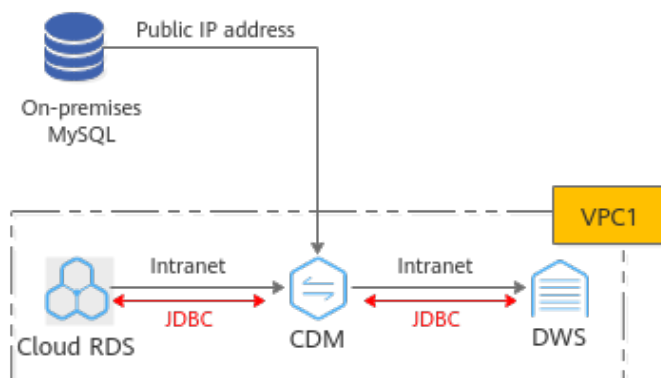
This section describes how to use Cloud Data Migration (CDM) to migrate MySQL data to GaussDB(DWS) clusters in batches.

This section contains the following parts:

1. [Checking Data Before Migration](#)
2. [Creating a GaussDB\(DWS\) Cluster](#)
3. [Creating a CDM Cluster](#)
4. [Creating a Link](#)
5. [Creating and Migrating a Job](#)
6. [Verifying Data Consistency After Migration](#)

## Scenario Description

Figure 8-1 Migration scenario



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

- On-premises MySQL data migration:

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

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

## Checking Data Before Migration

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

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

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

Enter the password of the database account as prompted:

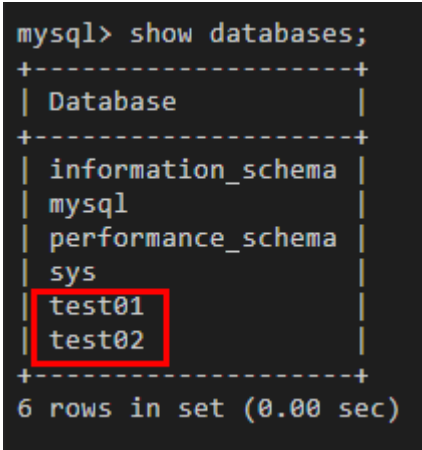
Enter password:

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

For example, the destination MySQL databases to be migrated are **test01**, **test02**, and the encoding format. The test01 library contains the **orders**, **persons**, and **persons\_b** tables and the **persons\_beijing** view. The **test02** library contains the **persons\_c** table.

1. Query the database name.

```
show databases;
```



```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| test01 |
| test02 |
+-----+
6 rows in set (0.00 sec)
```

## 2. Query the database code.

```
use <dbname>;  
status;
```

```
mysql> status;  
-----  
./mysql Ver 14.14 Distrib 5.7.32, for e17 (x86_64) using EditLine wrapper  
  
Connection id:          53  
Current database:      test01  
Current user:          root@localhost  
SSL:                   Not in use  
Current pager:         stdout  
Using outfile:         ''  
Using delimiter:      ;  
Server version:        5.7.32 MySQL Community Server (GPL)  
Protocol version:      10  
Connection:            Localhost via UNIX socket  
Server character set:  utf8mb4  
Db character set:      utf8  
Client character set:  utf8  
Conn. character set:   utf8  
UNIX socket:           /tmp/mysql.sock  
Uptime:                2 hours 44 min 49 sec  
  
Threads: 6 Questions: 658 Slow queries: 0 Opens: 139 Flush tables: 1 Open tables: 114 Queries per second avg: 0.066  
-----
```

```
mysql> status;  
-----  
mysql Ver 14.14 Distrib 5.7.32, for Linux (x86_64) using EditLine wrapper  
  
Connection id:          8  
Current database:      test02  
Current user:          root@127.0.0.1  
SSL:                   Cipher in use is ECDHE-RSA-AES128-GCM-SHA256  
Current pager:         stdout  
Using outfile:         ''  
Using delimiter:      ;  
Server version:        5.7.32 MySQL Community Server (GPL)  
Protocol version:      10  
Connection:            127.0.0.1 via TCP/IP  
Server character set:  utf8  
Db character set:      utf8  
Client character set:  utf8  
Conn. character set:   utf8  
TCP port:              3306  
Uptime:                3 hours 36 min 35 sec  
  
Threads: 1 Questions: 91 Slow queries: 0 Opens: 111 Flush tables: 1 Open tables: 103 Queries per second avg: 0.007  
-----
```

## 3. Query the database table.

```
use <dbname>;  
show full tables;
```

**NOTICE**

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

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

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

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

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

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

----End

## Creating a GaussDB(DWS) Cluster

- Step 1** For details, see [Creating a Cluster](#). You can select **EU-Dublin** for **Region**.

 **NOTE**

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

- Step 2** For details, see [Using the gsql Client to Connect to a Cluster](#).

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

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

----End

## Creating a CDM Cluster

**Step 1** Log in to the HUAWEI CLOUD management console.

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

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

**Table 8-1** Parameter description

Parameter	Value
Region	<b>EU-Dublin</b> . Choose the region of your GaussDB(DWS).
AZ	AZ1 (If the desired resources are sold out in the current AZ, change the AZ and try again.)
Name	CDM-demo
Instance Type	cdm.large (Select other flavors if the flavor is sold out.)
VPC	demo-vpc ( <b>in the same VPC as GaussDB(DWS)</b> )
Subnet	subnet-f377(10.1.0.0/24) (example)
Security Group	-
Enterprise Project	Default

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

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

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

----End

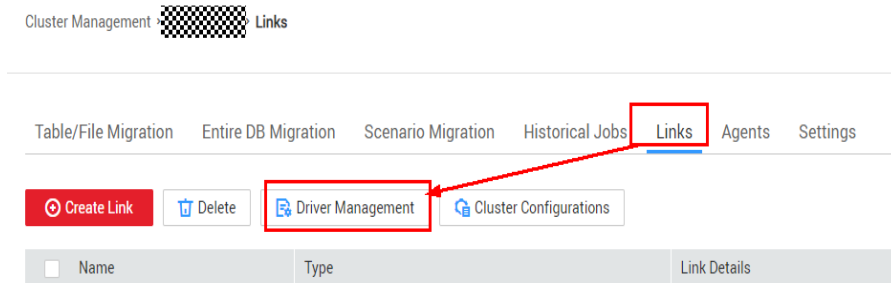
## Creating a Link

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

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

The screenshot shows the 'MySQL Product Archives' page for 'MySQL Connector/J (Archived Versions)'. A yellow warning banner states: 'Please note that these are old versions. New releases will have recent bug fixes and features! To download the latest release of MySQL Connector/J, please visit MySQL Downloads.' Below this, there are dropdown menus for 'Product Version' (set to 5.1.48) and 'Operating System' (set to Platform Independent). A table lists two download options for July 11, 2019: 'Platform Independent (Architecture Independent), Compressed TAR Archive' (4.2M) and 'Platform Independent (Architecture Independent), ZIP Archive' (4.6M). The ZIP Archive option is highlighted with a red box. A 'Download' button is visible next to the ZIP Archive option. At the bottom, a note suggests using MD5 checksums and GnuPG signatures to verify package integrity.

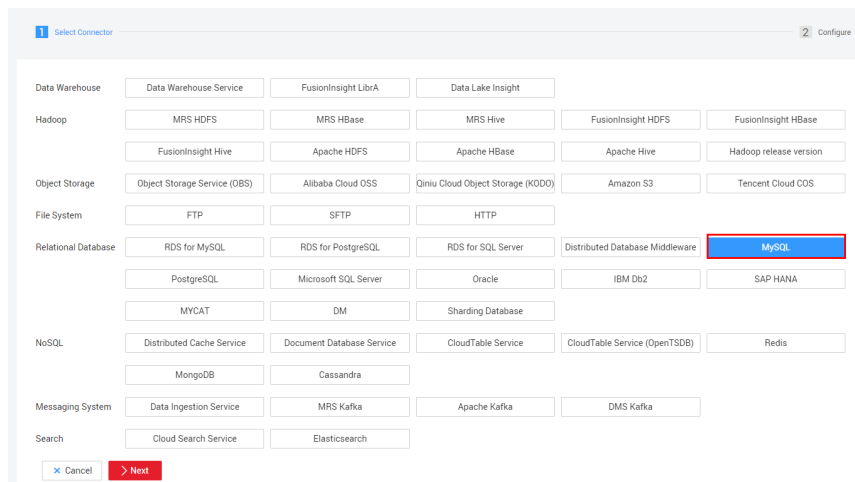
- Download the package to the local host and decompress it to obtain **mysql-connector-java-xxx.jar**.
- On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links > Driver Management**.



- Click **Upload** on the right of MySQL, select mysql-connector-java-xxx.jar, and click **Upload**.

**Step 2** Create a MySQL connection.

- On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links > Create Link**.
- Select **MySQL** and click **Next**. (If the RDS is deployed on the cloud, select RDS for MySQL.)



- Enter the connection information according to **Table 8-2**, and click **Test**. If the test is successful, click **Save**.

**NOTE**

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

**Table 8-2** MySQL link information

Parameter	Value
Name	MySQL

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

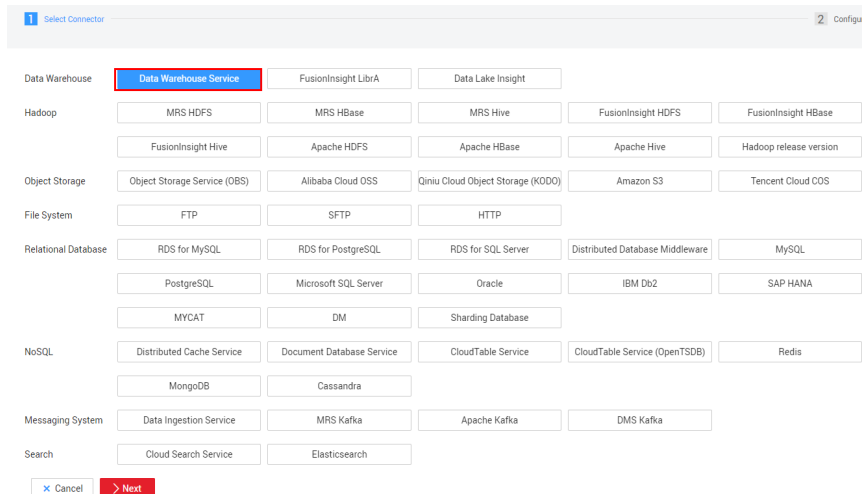


* Name	<input type="text" value="MySQL"/>
* Connector	<input type="text" value="Relational Database"/>
Database Type	<input type="text" value="MySQL"/>
* Database Server	<input type="text" value="██████████"/>
* Port	<input type="text" value="3306"/>
* Database Name	<input type="text" value="test01"/>
* Username	<input type="text" value="root"/>
* Password	<input type="password"/>
Use Local API	<input type="radio"/> Yes <input checked="" type="radio"/> No
Use Agent	<input type="radio"/> Yes <input checked="" type="radio"/> No

[Show Advanced Attributes](#)

**Step 3** Create a GaussDB(DWS) link.

1. On the **Cluster Management** page, click **Job Management** in the **Operation** column of the cluster and choose **Links > Create Link**.
2. Select **Data Warehouse Service** and click **Next**.



3. Enter the connection information according to [Table 8-3](#), and click **Test**. If the test is successful, click **Save**.

**Table 8-3** GaussDB(DWS) connection information

Parameter	Value
Column	DWS-test01
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.
Port	8000
Database Name	test01 (Ensure that the corresponding database has been manually created on GaussDB(DWS) by referring to <a href="#">Step 3.</a> )
User	dbadmin
Password	Password of user <b>dbadmin</b>
Use Agent	No

The screenshot shows a configuration form for a Data Warehouse Service (DWS) cluster. The fields are as follows:

- Name:** DWS-test01
- Connector:** Relational Database
- Database Type:** Data warehouse
- Database Server:** A checkered box with a "Select" link to its right.
- Port:** 8000
- Database Name:** test01
- Username:** dbadmin
- Password:** Masked with dots.
- Use Agent:** Radio buttons for "Yes" and "No", with "No" selected.

Below the form is a link "Show Advanced Attributes". At the bottom are four buttons: "Cancel", "Previous", "Test", and "Save".

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

----End

## Creating and Migrating a Job

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

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

- Job Name: MySQL-DWS-test01
- Source Job Configuration:
  - Source Link Name: MySQL
- Destination Job Configuration:
  - Destination Link Name: DWS-test01
  - Automatic Table Creation: The table is created when it does not exist.

- isCompress: Yes
- Orientation: COLUMN
- Retain the default value for other settings.

**Job Configuration**

\* Job Name: MySQL-DWS-test01

---

**Source Job Configuration**

\* Source Link Name: MySQL

\* Schema/Table Space: test01

Hide Advanced Attributes

Where Clause:

Partition column nullable: Yes No

**Destination Job Configuration**

\* Destination Link Name: DWS-test01

\* Schema/Table Space: public

Auto Table Creation: Auto Creation

isCompress: Yes No

Orientation: COLUMN


Clear Data Before Import: Do not clear

Import Mode: COPY

Hide Advanced Attributes

Extend char length: Yes No

Use non-null constraints: Yes No

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

Configure Basic Information | **2 Map Field** | Configure Task

Tables in test01

Enter search content. Select All Deselect Batch mapping

Table Name

No data available.

Tables to be migrated to public

Enter search content. Select All Deselect

Table Name

- orders
- persons
- persons\_b
- persons\_beijing

>> > < <<

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

Concurrent Extractors tables ?

Concurrent Extractors ?

Write Dirty Data ?

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

Name	Link Details	Created By	Last Execution Time	Duration	Pending	Running	Successful	Failed	Status	Operation
MySQL-DWS-test01	MySQL-DWS-test01		Oct 27, 2021 11:53:28 GMT+08:00	32s	0	0	4	0	Succeeded	Run Historical Record Edit More

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

### NOTICE

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

----End

## Verifying Data Consistency After Migration

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

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

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

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

```
test01-> select * from pg_tables where schemaname='public';
```

schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	hastriggers	tablecreator	created	last_ddl_time
public	persons	dbadmin		f	f	f	dbadmin	2021-10-27 03:43:25.306998+00	2021-10-27 03:43:25.30699
public	persons_beijing	dbadmin		f	f	f	dbadmin	2021-10-27 03:43:25.298073+00	2021-10-27 03:43:25.29807
public	orders	dbadmin		f	f	f	dbadmin	2021-10-27 03:43:25.228591+00	2021-10-27 03:43:25.22859
public	persons_b	dbadmin		f	f	f	dbadmin	2021-10-27 03:43:25.295822+00	2021-10-27 03:43:25.29582

(4 rows)

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

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

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

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

**Step 4** Perform sampling check to verify table data.

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

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

**Step 5** Repeat [Step 2](#) to [Step 4](#) to check whether the data in other databases and tables is correct.

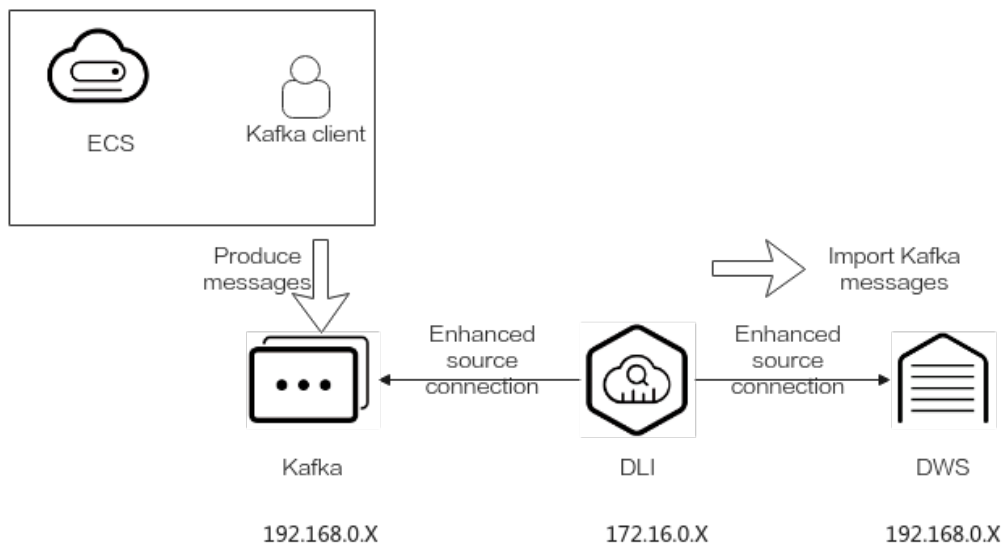
----End

# 9 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 9-1** 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 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 Data Warehouse 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 9-1](#), 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 9-1** 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, Europe-Dublin.
- 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.

## 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 9-2** Kafka instance parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	Europe-Dublin
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 9-2** 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 9-3** 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**.

The **topic-demo** is successfully created in the topic list.

**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 Data Warehouse Cluster and Target Table

**Step 1** **Creating a Cluster**. To ensure network connectivity, the region and VPC of the data warehouse cluster must be the same as those of the Kafka instance. In this practice, the region and VPC are Europe-Dublin. The VPC must be the same as that created for Kafka.

**Step 2** On the **Clusters** page of the GaussDB(DWS) console, click **Login** on the right of the cluster.


 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** 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 9-4** 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 4** Click the database name **gaussdb** and click **SQL Window** in the upper right corner to access the SQL editor.

- Step 5** Copy the following SQL statement. In the SQL window, click Execute SQL to create the target table **user\_dws**.

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

----End

### Step 3: Creating a DLI Queue

- Step 1** Log in to the Huawei Cloud management console and choose **Analytics > Data Lake Insight** from the service list. The DLI management console is displayed.
- Step 2** Choose **Resources > Queue Management** on the left.
- 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 9-3** DLI queue parameters

Parameter	Value
Billing Mode	Pay-per-use
Region	Europe-Dublin
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 9-5** 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\_dws

\* 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 [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**.

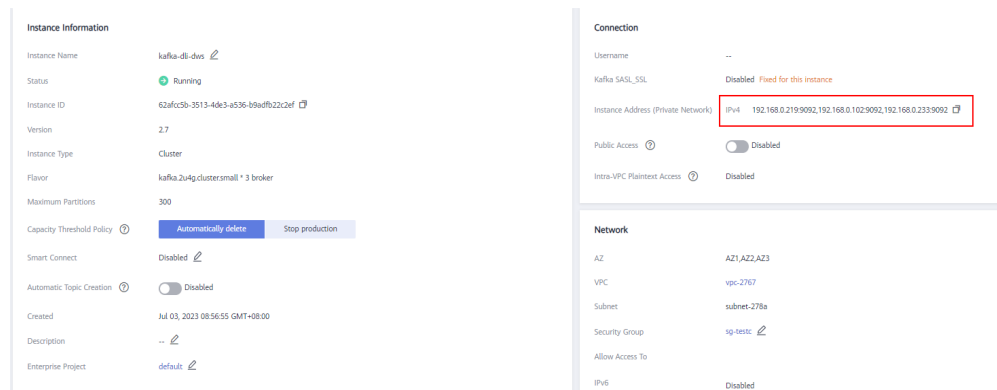
----End

## 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.

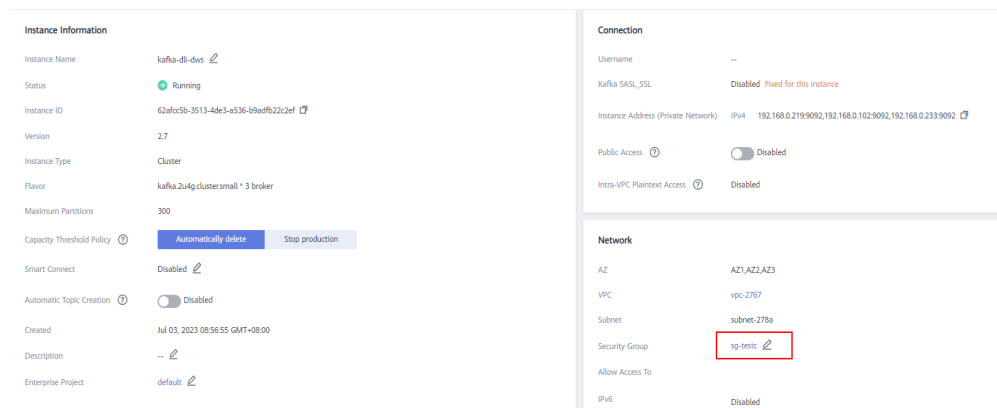
1. Return to the Kafka console and click the Kafka instance name to go to the **Basic Information** page. View the value of **Instance Address (Private Network)** in connection information and record the address for future use.

**Figure 9-6** Kafka private network address



2. Click the security group name.

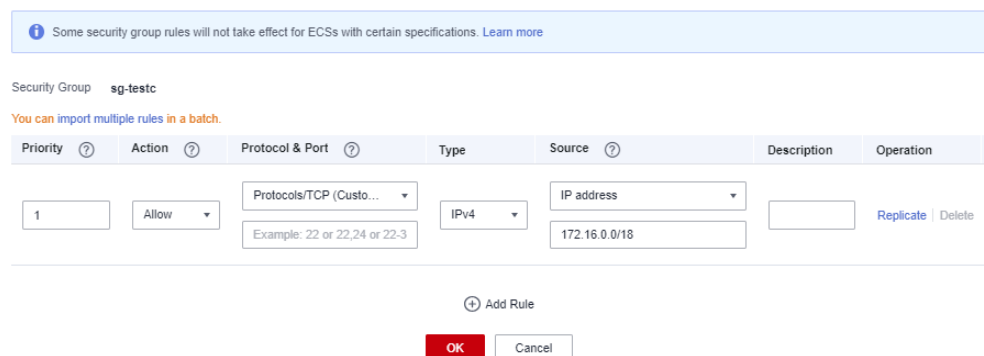
**Figure 9-7** Kafka security group



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 9-8** 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 9-4** 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 9-9** 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**.

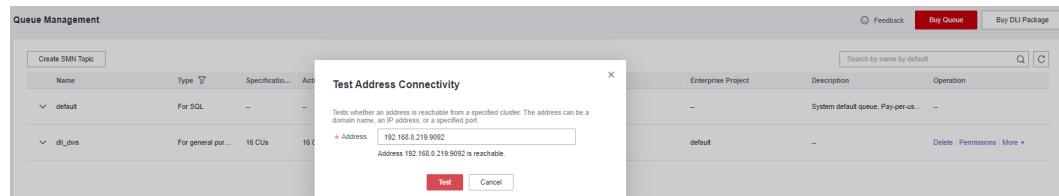
The Kafka connection is successfully created.

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



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

**Figure 9-10** Testing Kafka connectivity

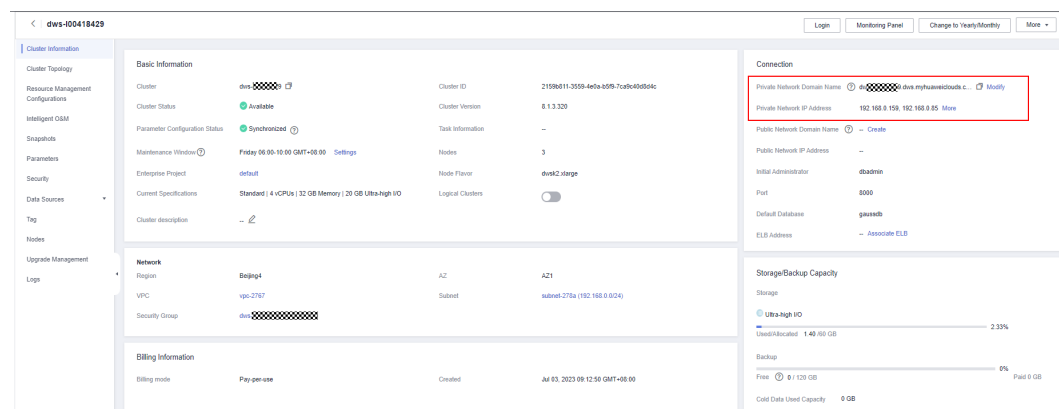


**Step 7** Click **Test** to verify that DLI is successfully connected to Kafka.

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

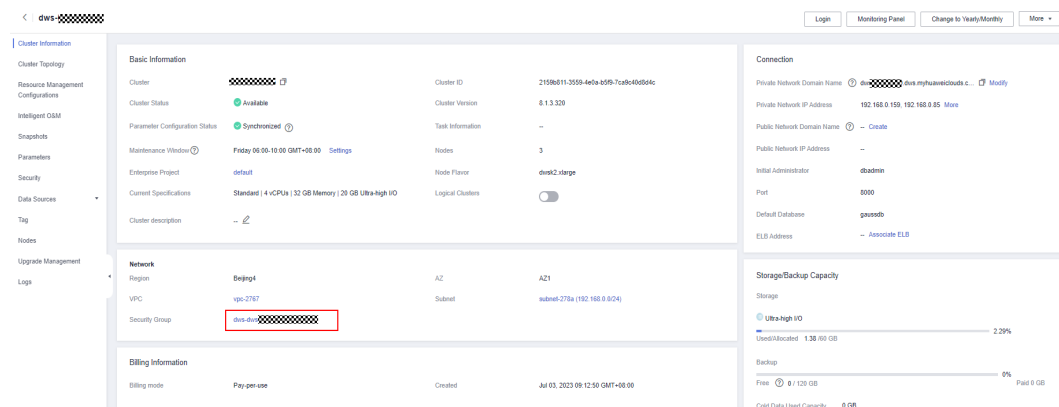
**Step 9** As shown in the following figure, record the private domain name and private IP address of the data warehouse cluster for future use.

**Figure 9-11** Private domain name and IP address



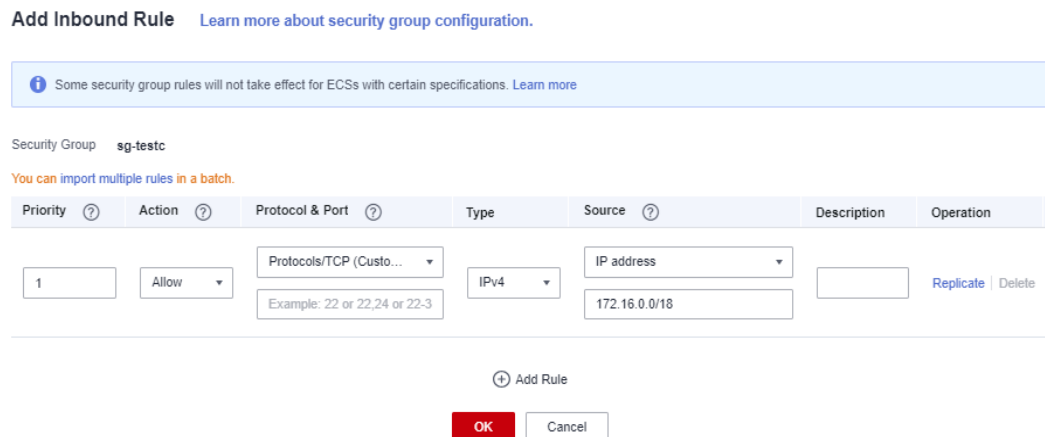
**Step 10** Click the security group name.

**Figure 9-12** GaussDB(DWS) security group



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

**Figure 9-13** Adding a rule to the GaussDB(DWS) security group

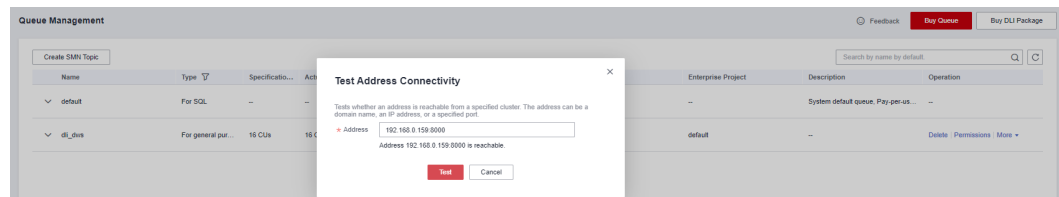


**Step 12** Click **OK**.

**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 private IP address and port number of the Kafka instance obtained in [Step 9](#).

**Figure 9-14** 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**.


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://mvnrepository.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

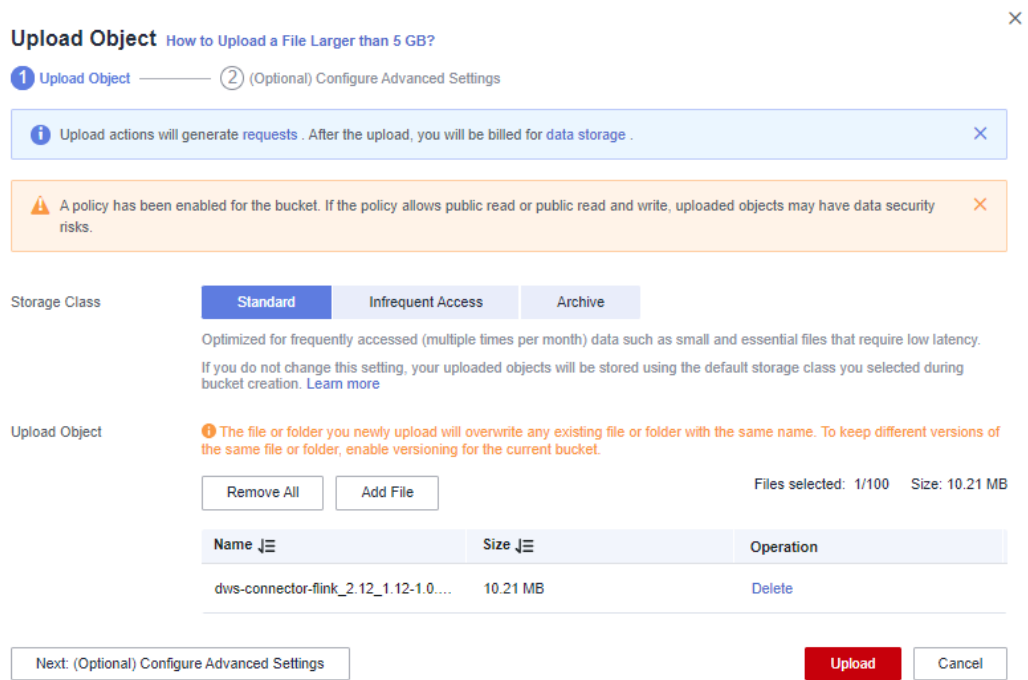
**Step 5** Click [dws-connector-flink\\_2.12\\_1.12-1.0.4-jar-with-dependencies.jar](#) to download it to the local host.

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

Artifact	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 Europe-Dublin.

**Figure 9-15** 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 9-16** 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.

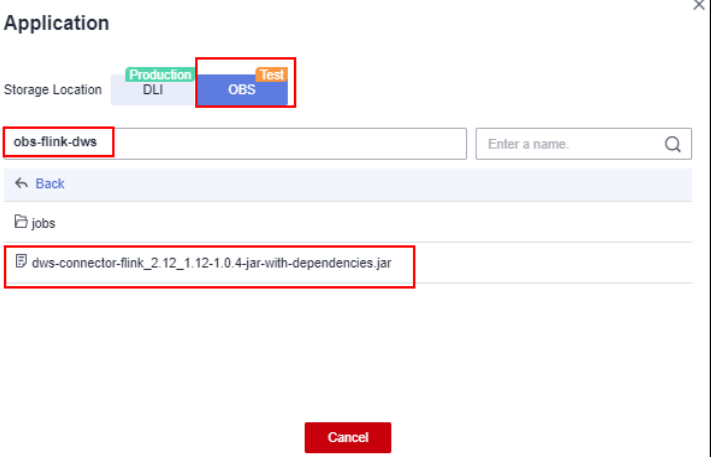
**Step 3** Click **OK**.

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 9-5** 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', there are tabs for 'Production DLI' and 'Test OBS'. A search box contains 'obs-flink-dws'. Below, a file list shows 'dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar' selected. A 'Cancel' button is at the bottom.</p>
OBS Bucket	Select the bucket created in <a href="#">Step 5: Preparing the dws-connector-flink Tool for Interconnecting GaussDB(DWS) with Flink.</a>
Enable Checkpointing	Check the box.
Other parameters	Retain the default value.

**Figure 9-17** 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"
);

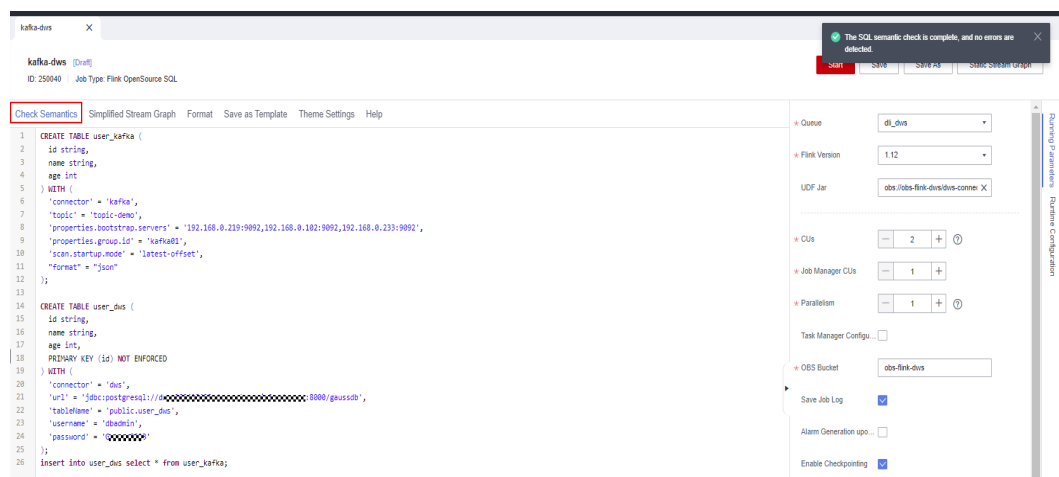
CREATE TABLE user_dws (
  id string,
  name string,
  age int,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'dws',
  'url' = 'jdbc:postgresql://GaussDB(DWS) private network domain name:8000/gaussdb',
  'tableName' = 'public.user_dws',
  'username' = 'dbadmin',
  'password' = 'Password of 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 9-18** 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 9-19** Job execution status

ID	Job Name	Queue	Type	Status	Description	Username	Created	Started	Duration	Operation
250040	kafka-dws	dl_dws	Flink OpenSource SQL	Running	--	XXXXXXXXXX	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安装目录
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:
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-21-9-1001 ~]# ./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-21-9-1001 ~]# ./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. You can find that the data is successfully saved 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

# 10 Basic SQL Operations

This section describes some basic SQL operations of the GaussDB(DWS) database.

## Creating, Viewing, and Deleting a Database

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

```
CREATE DATABASE test_db ENCODING 'UTF8' template = template0;
```
- Query the database list using the **\l** meta-command.  

```
\l
```
- Querying the database list using the **PG\_DATABASE** system catalog  

```
SELECT datname FROM pg_database;
```
- Run the **DROP DATABASE** statement to delete a database.  

```
DROP DATABASE test_db;
```

## Creating, Viewing, Modifying, and Deleting a Table

- Run the **CREATE TABLE** statement to create a table.  

```
CREATE TABLE customer_t1(id INT, name CHAR(40),age TINYINT);
```
- Use the **PG\_GET\_TABLEDEF()** function to view the table creation statement.  

```
SELECT * FROM PG_GET_TABLEDEF('customer_t1');
```
- Run the **ALTER TABLE** statement to modify a table.  
Add a column:  

```
ALTER TABLE customer_t1 ADD (address VARCHAR(100));
```

  
Delete a column:  

```
ALTER TABLE customer_t1 DROP COLUMN address;
```

  
Modify a column type:  

```
ALTER TABLE customer_t1 MODIFY age INTEGER NOT NULL;
```
- Run the **DROP TABLE** statement to delete a table.  

```
DROP TABLE customer_t1;
```

## Creating, Viewing, and Deleting Indexes

- Run the **CREATE INDEX** or **ALTER TABLE** statement to create a common index.  

```
CREATE INDEX c_id_index on customer_t1(id);  
ALTER TABLE customer_t1 ADD INDEX c_id_index (id);
```
- Use the **PG\_INDEXES** system catalog to view all indexes in a table.  

```
SELECT * FROM pg_indexes WHERE tablename = 'customer_t1';
```

- Run the **ALTER TABLE** or **DROP INDEX** statement to delete an index.  
DROP INDEX c\_id\_index;  
ALTER TABLE customer\_t1 DROP INDEX c\_id\_index;

## Adding, Deleting, and Modifying Table Data

- Run the **INSERT INTO** statement to insert table data.  
INSERT INTO customer\_t1 VALUES(1001,'user1',22);
- Run the **SELECT** statement to query table data.  
SELECT \* FROM customer\_t1;
- Run the **UPDATE** statement to update table data.  
UPDATE customer\_t1 SET id = 1009 WHERE id = '1001';
- Use the **DELETE** statement to delete table data.  
DELETE FROM customer\_t1 WHERE id = '1009';

# 11 Database Quick Start

---

## 11.1 Before You Start

This section describes how to quickly create databases and tables, insert data to tables, and query data in tables. Later sections in this chapter will elaborate on common operations.

### Basic Database Operations

#### Step 1 Create a database user.

By default, only administrators that are generated during cluster creation can access the initial database. They need to create user accounts and grant permissions to let other users access the database.

```
CREATE USER joe WITH PASSWORD 'password';
```

If the following information is displayed, the resource pool is created.

```
CREATE USER
```

A user named **joe** is created. You can define its password.

By default, a new user account has the permissions to log in to all databases, create tables, views, and indexes, and perform operations on these objects. For details, see [Users](#).

#### Step 2 Create a database.

```
CREATE DATABASE mytpcds;
```

For details about database management, see [Creating and Managing Databases](#).

#### Step 3 (Optional) Create a schema.

Schemas allow multiple users to use the same database without interfering with each other.

Run the following command to create a schema:

```
CREATE SCHEMA myschema;
```

If the following information is displayed, the schema named **myschema** has been created:

```
CREATE SCHEMA
```

After a schema is created, you can create its objects. When creating an object, specify the required schema using either of the following methods:

Set **search\_path** of the database to the schema.

```
SET SEARCH_PATH TO myschema;  
CREATE TABLE mytable (firstcol int);
```

Specify an object name that contains the schema name. Separate multiple object names by periods (.). The following shows an example.

```
CREATE TABLE myschema.mytable (firstcol int);
```

If no schema is specified during object creation, the object will be created in the current schema. Run the following statement to query the current schema:

```
show search_path;  
search_path  
-----  
"$user",public  
(1 row)
```

After the **mytpcds** database is created, you can run the following command to quit the **gaussdb** database:

```
\q
```

For details about schemas, see [Creating and Managing Schemas](#).

#### Step 4 Create a table.

- Create a table named **mytable** that has only one column. The column name is **firstcol** and the column type is **integer**.

```
CREATE TABLE mytable (firstcol int);
```

If the **DISTRIBUTE BY** statement is not used to specify the distribution column, the system automatically specifies the first column that meets the criteria as a distribution column. If **CREATE TABLE** is displayed at the end of the returned information, the table has been created.

NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'firstcol' as the distribution column by default.

HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.

The system catalog **PG\_TABLES** contains information about all tables in a cluster. You can run the **SELECT** statement to view the attributes of a table in the system catalog.

```
SELECT * FROM PG_TABLES WHERE TABLENAME = 'mytable';
```

- Insert data to the table.

```
INSERT INTO mytable values (100);
```

The **INSERT** statement inserts rows to a database table. For details about batch loading, see [About Parallel Data Import from OBS](#).

- View data in the table.

```
SELECT * from mytable;  
firstcol  
-----  
100  
(1 row)
```

 NOTE

- By default, new database objects, such as the **mytable** table, are created in the **public** schema. For details about schemas, see [Creating and Managing Schemas](#).
- For details about how to create a table, see [Creating a Table](#).
- In addition to the created tables, a database contains many system catalogs. These system catalogs contain cluster installation information and information about various queries and processes of GaussDB(DWS). You can collect information about a database by querying system catalogs. For details, see [Querying System Catalogs](#).
- GaussDB(DWS) supports hybrid row and column storage, which provides high query performance for interaction analysis in complex scenarios. For details about how to select a storage model, see [Planning a Storage Model](#).

----End

## Loading Sample Data

Most examples in this document use the TPC-DS sample table created in the **gaussdb** database. Before you use your SQL query tool to perform operations in the examples, create the TPC-DS sample table and load data to it.

An OBS bucket provides sample data and is accessible to all authenticated cloud users.

For the steps to create a table and load sample data, see [Loading Sample Data](#).

## Releasing Resources

If a cluster is deployed for the practice, delete the cluster after the practice is complete.

For details about how to delete clusters, see [Deleting Clusters](#).

To keep the cluster but delete the **db\_tpcds** database, run the following statement:

```
DROP DATABASE mytpcds;
```

To keep the cluster and the database, run the following statement to delete the tables in the database:

```
DROP TABLE mytable;
```

# 11.2 Creating and Managing Databases

## Prerequisites

To create a database, you must be a database system administrator or have the permission for creating databases. For details, see [Users](#).

## Background

- GaussDB(DWS) has two initial template databases **template0** and **template1** and a default user database **gaussdb**.



- **CREATE DATABASE** creates a database by copying a template database (**template1** by default). Do not use a client or any other tools to connect to or to perform operations on the template databases.
- A maximum of 128 databases can be created in GaussDB(DWS).
- A database system consists of multiple databases. A client can connect to only one database at a time. You are not allowed to query data across databases. If a database cluster contains multiple databases, set the **-d** parameter to specify the database to connect to.

## Procedure

### Step 1 Create database **db\_tpcds**.

```
CREATE DATABASE db_tpcds;
```

If the following information is displayed, the database is created.

```
CREATE DATABASE
```

As stated in [Background](#), the template database **template1** is copied by default to create a database. Its encoding format is SQL\_ASCII. If the name of an object created in this database contains multiple-byte characters (such as Chinese characters) and exceeds the name length limit (63 bytes), the system truncates the name from the last byte instead of the last character. As a result, characters may be incomplete.

To solve the problem, the data object name should not exceed the maximum length or contain multi-byte characters.

If an object whose name is truncated mistakenly cannot be deleted, delete the object using the name before the truncation, or manually delete it from the corresponding system catalog on each node.

You can also use **template0** to create a database by using **CREATE DATABASE** and specify new encoding and locale, for example, use UTF-8 as the default database encoding (**server\_encoding**). For details, see the syntax of **CREATE DATABASE**.

You can run the **show server\_encoding** command to view the current database encoding.

#### NOTE

- Database names must comply with the general naming convention of SQL identifiers. The current user automatically becomes the owner of this new database.
- If a database system supports independent users and projects, you are advised to store them in different databases.
- If the projects or users are associated with each other and share resources, store them in different schemas in the same database. A schema is only a logical structure. For details about user permissions for schemas, see table 1 in [Separation of Permissions](#).

### Step 2 View databases.

- Query the database list using the **\l** meta-command.  

```
\l
```
- Query the database list in the system catalog **pg\_database**:  

```
SELECT datname FROM pg_database;
```

**Step 3** Modify a database.

You can run the **ALTER DATABASE** statement to modify database attributes, such as the owner, name, and default configuration attributes.

- Run the following statement to specify the default schema search path:  
`ALTER DATABASE db_tpcds SET search_path TO pa_catalog,public;`
- Run the following statement to rename the database:  
`ALTER DATABASE db_tpcds RENAME TO human_tpcds;`

**Step 4** Delete a database.

You can run the **DROP DATABASE** statement to delete a database. This command deletes the system directory in the database, as well as the database directory on the disk that stores data. Only the database owner or the system administrator can delete a database. A database accessed by users cannot be deleted. You need to connect to another database before deleting this database.

Run the following statement to delete the database:  
`DROP DATABASE human_tpcds;`

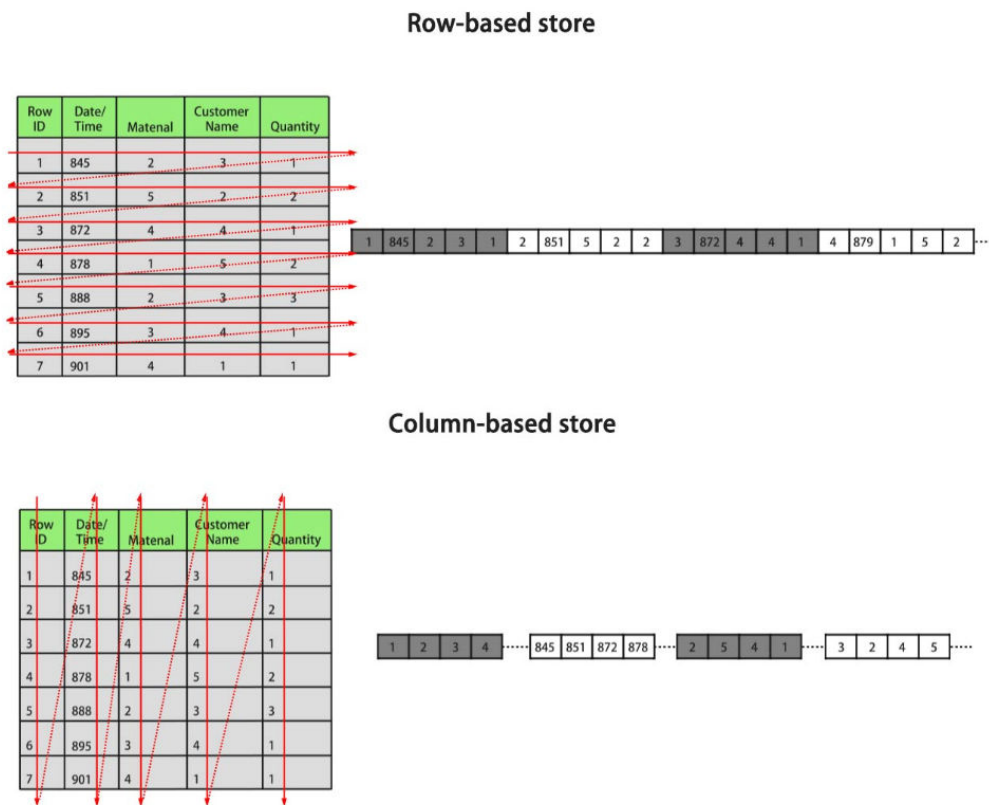
----End

## 11.3 Planning a Storage Model

GaussDB(DWS) supports hybrid row and column storage. Each storage mode applies to specific scenarios. Select an appropriate mode when creating a table.

Row storage stores tables to disk partitions by row, and column storage stores tables to disk partitions by column. By default, a table is created in row storage mode. For details about differences between row storage and column storage, see [Figure 11-1](#).

**Figure 11-1** Differences between row storage and column storage



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

Both storage modes have benefits and drawbacks.

Storage Mode	Benefit	Drawback
Row storage	All the columns of a record are stored in the same partition. Data can be easily inserted and updated.	All the columns of a record are read after the <b>SELECT</b> statement is executed even if only certain columns are required.
Column storage	<ul style="list-style-type: none"> <li>Only necessary columns in a query are read.</li> <li>Projections are efficient.</li> <li>Any column can serve as an index.</li> </ul>	<ul style="list-style-type: none"> <li>The selected columns need to be reconstructed after the <b>SELECT</b> statement is executed.</li> <li>Data cannot be easily inserted or updated.</li> </ul>

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 includes most of the fields, row storage is recommended.

Storage Mode	Application Scenario
Row storage	<ul style="list-style-type: none"><li>• Point queries (simple index-based queries that only return a few records).</li><li>• Scenarios requiring frequent addition, deletion, and modification.</li></ul>
Column storage	<ul style="list-style-type: none"><li>• Statistical analysis queries (requiring a large number of association and grouping operations)</li><li>• Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)</li></ul>

## Row-Store Table

Row-store tables are created by default. In a row-store table, data is stored by row, that is, data in each row is stored continuously. Therefore, this storage model applies to scenarios where data needs to be updated frequently.

```
CREATE TABLE customer_t1
(
  state_ID CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID NUMBER
);
--Delete the table.
DROP TABLE customer_t1;
```

## Column-Store Table

In a column-store table, data is stored by column, that is, data in each column is stored continuously. The I/O of data query in a single column is small, and column-store tables occupy less storage space than row-store tables. This storage model applies to scenarios where data is inserted in batches, less updated, and queried for analysis. A column-store table cannot be used for point queries.

```
CREATE TABLE customer_t2
(
  state_ID CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID NUMBER
)
WITH (ORIENTATION = COLUMN);
--Delete the table.
DROP TABLE customer_t2;
```

# 11.4 Creating and Managing Tables

## 11.4.1 Creating a Table

### Context

A table is created in a database and can be saved in different databases. Tables under different schemas in a database can have the same name. Before creating a table, perform the operations in [Planning a Storage Model](#).

### Creating a Table

Run the following statement to create a 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);
```

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

```
CREATE TABLE
```

**c\_customer\_sk**, **c\_customer\_id**, **c\_first\_name** and **c\_last\_name** are the column names in the table. *integer*, *char(5)*, *char(6)*, and *char(8)* are column name types.

## 11.4.2 Inserting Data to a Table

A new table contains no data. You need to insert data to the table before using it. This section describes how to insert a row or multiple rows of data from a specified table using [INSERT](#). If a large amount of data needs to be imported to a table in batches, see [Import Methods](#).

### Background

The length of a character on the server and client may vary by the used character sets. A string entered on the client will be processed based on the server's character set, so the output may differ from the input.

**Table 11-1** Comparison of character set output between the client and server

Procedure	Same Character Sets	Different Character Sets
No operations are performed on the string while it is saved and read.	Your expected result is returned.	If the character sets for input and output on the client are the same, your expected result is returned.

Procedure	Same Character Sets	Different Character Sets
Operations (such as executing string functions) are performed to the string while it is saved and read.	Your expected result is returned.	The result may differ from expected, depending on the operations performed on the string.
A long string is truncated while it is saved.	Your expected result is returned.	If the character sets used on the client and server are different in character length, an unexpected result may occur.

More than one of the preceding operations can be performed on a string. For example, if the character sets of the client and server are different, a string may be processed and then truncated. In this case, the result will also be unexpected. For details, see [Table 11-2](#).

 **NOTE**

Long strings are truncated only if **DBCMPATIBILITY** is set to **TD** (compatible with Teradata) and **td\_compatible\_truncation** is set to **on**.

Run the following statements to create tables **table1** and **table2** to be used in the examples:

```
CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));  
CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

**Table 11-2** Examples

ID	Server Character Set	Client Character Set	Automatic Truncation Enabled	Examples	Result	Description
1	SQL_ASCII	UTF8	Yes	<code>INSERT INTO table1 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78'));</code>	id  a b c -----+----- +-----+----- 1   87  87  87	A string is reversed on the server and then truncated. Because character sets used by the server and client are different, character A is displayed in multiple bytes on the server and the result is incorrect.
2	SQL_ASCII	UTF8	Yes	<code>INSERT INTO table1 VALUES(2,reverse('123A78'),reverse('123A78'),reverse('123A78'));</code>	id  a b c -----+----- +-----+----- 2   873  873  873	A string is reversed and then automatically truncated. Therefore, the result is unexpected.
3	SQL_ASCII	UTF8	Yes	<code>INSERT INTO table1 VALUES(3,'87A123','87A123','87A123');</code>	id   a   b   c -----+----- +-----+----- 3   87A1   87A1   87A1	The column length in the string type is an integer multiple of the length in client character encoding. Therefore, the result is correct after truncation.

ID	Server Character Set	Client Character Set	Automatic Truncation Enabled	Examples	Result	Description
4	SQL_ASCII	UTF8	No	<pre>INSERT INTO table2 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')); INSERT INTO table2 VALUES(2,reverse('123A78'),reverse('123A78'),reverse('123A78'));</pre>	<pre>id  a b c ---- +-----+ --+----- +----- 1   87 321  87 321   87 321 2   87321  87321  87321</pre>	Similar to the first example, multi-byte characters no longer indicate the original characters after being reversed.

## Common Operations

You need to create a table before inserting data to it. For details about how to create a table, see [Creating and Managing Tables](#).

- Insert a row to table **customer\_t1**:

Data values are arranged in the same order as the columns in the table and are separated by commas (.). Generally, they are text values (constants). Scalar expressions are also allowed.

```
INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
```

If you know the sequence of the columns in the table, you can obtain the same result without listing these columns. For example, the following statement generates the same result as the preceding statement:

```
INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
```

If you do not know some of the values, you can omit them. If no value is specified for a column, the column is set to the default value. The following shows an example.

```
INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');
```

```
INSERT INTO customer_t1 VALUES (3769, 'hello');
```

You can also specify the default value of a column or row:

```
INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);
```

```
INSERT INTO customer_t1 DEFAULT VALUES;
```

- To insert multiple rows, run the following statement:

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

You can also insert multiple rows by running the statement for inserting one row for multiple times. However, you are advised to run this command to improve efficiency.



- Insert data to a table from a specified table. For example, run the following statement to insert the data of table *customer\_t1* to the backup table *customer\_t2*.

```
CREATE TABLE customer_t2
(
  c_customer_sk      integer,
  c_customer_id      char(5),
  c_first_name       char(6),
  c_last_name        char(8)
);

INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

#### NOTE

If there is no implicit conversion between the data types of the specified table and those of the current table, the two tables must have the same data types when data is inserted from the specified table to the current table.

- Delete a backup table.  
**DROP TABLE customer\_t2 CASCADE;**

#### NOTE

If the table to be deleted is associated with other tables, delete the associated tables first.

## 11.4.3 Updating Data in a Table

Data updating is performed to modify data in a database. You can update one row, all rows, or specified rows of data, or update data in a single column without affecting the data in the other columns.

The following types of information are required when the **UPDATE** statement is used to update a row:

- Table name and column names of the data to be updated
- New column data
- Rows of the data to be updated

#### NOTE

- You can use a schema as a modifier of the table name. If no such modifier is specified, the table is located based on the default schema.
- In the statement, **SET** is followed by the target column and the new value of the column. The new value can be a constant or an expression.
- The table can contain the **WHERE** clause to filter the data that is equal to the specified condition.
  - If the statement does not include the **WHERE** clause, all rows are updated.
  - If the statement includes the **WHERE** clause, only the rows matching the clause condition are updated.

In the **SET** clause, the equal sign (=) indicates value setting. In the **WHERE** clause, the equal sign indicates comparison. The **WHERE** clause can specify a condition using the equal and other operators.

Generally, the SQL language does not provide a unique ID for a row of data. Therefore, it is impossible to directly specify the rows of the data to be updated. However, you can specify the rows by setting a condition that only the rows meet. If a table contains primary keys, you can specify a row by primary key.

For details about how to create a table and insert data to it, see [Creating a Table](#) and [Inserting Data to a Table](#).

The following is an example of updating data in the table:

- **c\_customer\_sk** in table **customer\_t1** must be changed from **9527** to **9876**:  
`UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;`
- **c\_customer\_sk** in table **customer\_t1** must be changed from **9527** to **c\_customer\_sk + 100**:  
`UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100 WHERE c_customer_sk = 9527;`
- **c\_customer\_sk** in table **customer\_t1** under the public mode must be changed from **9527** to **9876**:  
`UPDATE public.customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;`
- If the **WHERE** clause is not included, increase all the **c\_customer\_sk** values by **100**.  
`UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;`
- **c\_customer\_sk** values greater than **9527** in table **customer\_t1** must be changed to **9876**:  
`UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk > 9527;`
- You can run an **UPDATE** statement to update multiple columns by specifying multiple values in the **SET** clause. For example:  
`UPDATE customer_t1 SET c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk = 4421;`

After data has been updated or deleted in batches, a large number of deletion markers are generated in the data file. During query, data that is marked out by these deletion markers needs to be scanned as well. In this case, the query performance deteriorates after batch updates or deletions. If data needs to be updated or deleted in batches frequently, you are advised to periodically do **VACUUM FULL** to maintain the query performance.

## 11.4.4 Viewing Data

- Query information about all tables in a database through the system catalog **pg\_tables**:  
`SELECT * FROM pg_tables;`
- Run the **\d+** command of the **gsql** tool to query table attributes:  
`\d+ customer_t1;`
- Query the data volume of the table **customer\_t1**:  
`SELECT count(*) FROM customer_t1;`
- Query all data in table **customer\_t1**:  
`SELECT * FROM customer_t1;`
- Query data in column **c\_customer\_sk**:  
`SELECT c_customer_sk FROM customer_t1;`
- Filter repeated data in column **c\_customer\_sk**:  
`SELECT DISTINCT( c_customer_sk ) FROM customer_t1;`
- Query all data whose column **c\_customer\_sk** is **3869**:  
`SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;`
- Sort data based on column **c\_customer\_sk**.  
`SELECT * FROM customer_t1 ORDER BY c_customer_sk;`

To cancel a query that has been running for a long time, see [Viewing and Stopping the Running Query Statements](#) in [Querying System Catalogs](#).

## 11.4.5 Deleting Data from a Table

You can delete outdated data from a table by row.

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

For example, to delete all the rows whose **c\_customer\_sk** column is **3869** from table **customer\_t1**, run the following statement:

```
DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```

Delete the records whose **c\_customer\_sk** is **6885** and **4321** from the **customer\_t1** table.

```
DELETE FROM customer_t1 WHERE c_customer_sk in (6885, 4321);
```

Delete the records whose **c\_customer\_sk** is greater than 4000 and less than 5000 from the **customer\_t1** table.

```
DELETE FROM customer_t1 WHERE c_customer_sk > 4000 and c_customer_sk < 5000;
```

To delete all rows from the table, run either of the following statements:

```
DELETE FROM customer_t1;  
TRUNCATE TABLE customer_t1;
```

### NOTE

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

To delete a table, execute the following statement:

```
DROP TABLE customer_t1;
```

## 11.5 Loading Sample Data

This section describes how to load data to the default database **gaussdb**. You can obtain sample data from OBS.

### NOTE

- Before performing the following operations, ensure that your SQL client has been connected to the cluster.

1. Create a table.

Copy and run the following statements to create a table in the **gaussdb** database.

```
DROP SCHEMA if exists tpcds cascade;  
CREATE SCHEMA tpcds;  
SET current_schema TO tpcds;  
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);

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);

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);

CREATE TABLE warehouse
(
  w_warehouse_sk    integer          not null,
  w_warehouse_id    char(16)         not null,
  w_warehouse_name  varchar(20)
  w_warehouse_sq_ft integer
  w_street_number   char(10)
  w_street_name     varchar(60)
  w_street_type     char(15)
  w_suite_number    char(10)
  w_city            varchar(60)
  w_county          varchar(30)
  w_state           char(2)
  w_zip             char(10)
```

```
w_country          varchar(20)          ,
w_gmt_offset       decimal(5,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE ship_mode
(
  sm_ship_mode_sk   integer          not null,
  sm_ship_mode_id   char(16)         not null,
  sm_type           char(30)         ,
  sm_code           char(10)         ,
  sm_carrier        char(20)         ,
  sm_contract       char(20)
)WITH (orientation = column, COMPRESSION = MIDDLE);

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);

CREATE TABLE reason
(
  r_reason_sk       integer          not null,
  r_reason_id       char(16)         not null,
  r_reason_desc     char(100)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE income_band
(
  ib_income_band_sk integer          not null,
  ib_lower_bound    integer          ,
  ib_upper_bound    integer
)WITH (orientation = column, COMPRESSION = MIDDLE);

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);

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_percentage  decimal(5,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE call_center
(
cc_call_center_sk   integer          not null,
cc_call_center_id  char(16)         not null,
cc_rec_start_date  date              ,
cc_rec_end_date    date              ,
cc_closed_date_sk  integer          ,
cc_open_date_sk    integer          ,
cc_name           varchar(50)       ,
cc_class         varchar(50)       ,
cc_employees      integer          ,
cc_sq_ft         integer          ,
cc_hours         char(20)          ,
cc_manager       varchar(40)       ,
cc_mkt_id        integer          ,
cc_mkt_class     char(50)          ,
cc_mkt_desc      varchar(100)      ,
cc_market_manager varchar(40)      ,
cc_division      integer          ,
cc_division_name  varchar(50)      ,
cc_company       integer          ,
cc_company_name  char(50)          ,
cc_street_number  char(10)         ,
cc_street_name   varchar(60)       ,
cc_street_type   char(15)         ,
cc_suite_number  char(10)         ,
cc_city         varchar(60)        ,
cc_county       varchar(30)        ,
cc_state        char(2)           ,
cc_zip         char(10)           ,
cc_country     varchar(20)         ,
cc_gmt_offset  decimal(5,2)       ,
cc_tax_percentage decimal(5,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

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);

CREATE TABLE web_site
(
  web_site_sk      integer      not null,
  web_site_id     char(16)    not null,
  web_rec_start_date date      ,
  web_rec_end_date date      ,
  web_name        varchar(50)  ,
  web_open_date_sk integer      ,
  web_close_date_sk integer      ,
  web_class       varchar(50)  ,
  web_manager     varchar(40)  ,
  web_mkt_id      integer      ,
  web_mkt_class   varchar(50)  ,
  web_mkt_desc    varchar(100)  ,
  web_market_manager varchar(40) ,
  web_company_id  integer      ,
  web_company_name char(50)    ,
  web_street_number char(10)   ,
  web_street_name  varchar(60)  ,
  web_street_type  char(15)    ,
  web_suite_number char(10)    ,
  web_city        varchar(60)  ,
  web_county      varchar(30)  ,
  web_state       char(2)     ,
  web_zip        char(10)     ,
  web_country     varchar(20)  ,
  web_gmt_offset  decimal(5,2)  ,
  web_tax_percentage decimal(5,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE store_returns
(
  sr_returned_date_sk integer      ,
  sr_return_time_sk   integer      ,
  sr_item_sk         integer      not null,
  sr_customer_sk     integer      ,
  sr_cdemo_sk        integer      ,
  sr_hdemo_sk        integer      ,
  sr_addr_sk         integer      ,
  sr_store_sk        integer      ,
  sr_reason_sk       integer      ,
  sr_ticket_number   integer      not null,
  sr_return_quantity integer      ,
  sr_return_amt      decimal(7,2)  ,
  sr_return_tax      decimal(7,2)  ,
  sr_return_amt_inc_tax decimal(7,2) ,
  sr_fee            decimal(7,2)  ,
  sr_return_ship_cost decimal(7,2)  ,
  sr_refunded_cash  decimal(7,2)  ,
```

```
sr_reversed_charge    decimal(7,2)    ,
sr_store_credit       decimal(7,2)    ,
sr_net_loss           decimal(7,2)    ,
)WITH (orientation = column, COMPRESSION = MIDDLE);

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);

CREATE TABLE web_page
(
  wp_web_page_sk      integer          not null,
  wp_web_page_id      char(16)         not null,
  wp_rec_start_date   date             ,
  wp_rec_end_date     date             ,
  wp_creation_date_sk integer          ,
  wp_access_date_sk   integer          ,
  wp_autogen_flag     char(1)         ,
  wp_customer_sk      integer          ,
  wp_url              varchar(100)     ,
  wp_type             char(50)         ,
  wp_char_count       integer          ,
  wp_link_count       integer          ,
  wp_image_count      integer          ,
  wp_max_ad_count     integer          ,
)WITH (orientation = column, COMPRESSION = MIDDLE);

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);

CREATE TABLE catalog_page
(
  cp_catalog_page_sk  integer          not null,
  cp_catalog_page_id  char(16)         not null,
  cp_start_date_sk    integer          ,
  cp_end_date_sk      integer          ,
  cp_department       varchar(50)      ,
  cp_catalog_number   integer          ,
  cp_catalog_page_number integer        ,
  cp_description      varchar(100)     ,
  cp_type             varchar(100)     ,
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE inventory
```



```
(
  inv_date_sk      integer      not null,
  inv_item_sk     integer      not null,
  inv_warehouse_sk integer      not null,
  inv_quantity_on_hand integer
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE catalog_returns
(
  cr_returned_date_sk integer      ,
  cr_returned_time_sk integer      ,
  cr_item_sk          integer      not null,
  cr_refunded_customer_sk integer    ,
  cr_refunded_cdemo_sk integer      ,
  cr_refunded_hdemo_sk integer      ,
  cr_refunded_addr_sk integer      ,
  cr_returning_customer_sk integer   ,
  cr_returning_cdemo_sk integer      ,
  cr_returning_hdemo_sk integer      ,
  cr_returning_addr_sk integer      ,
  cr_call_center_sk  integer      ,
  cr_catalog_page_sk integer      ,
  cr_ship_mode_sk    integer      ,
  cr_warehouse_sk    integer      ,
  cr_reason_sk       integer      ,
  cr_order_number    integer      not null,
  cr_return_quantity integer      ,
  cr_return_amount   decimal(7,2) ,
  cr_return_tax      decimal(7,2) ,
  cr_return_amt_inc_tax decimal(7,2) ,
  cr_fee             decimal(7,2) ,
  cr_return_ship_cost decimal(7,2) ,
  cr_refunded_cash   decimal(7,2) ,
  cr_reversed_charge decimal(7,2) ,
  cr_store_credit    decimal(7,2) ,
  cr_net_loss        decimal(7,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE web_returns
(
  wr_returned_date_sk integer      ,
  wr_returned_time_sk integer      ,
  wr_item_sk          integer      not null,
  wr_refunded_customer_sk integer    ,
  wr_refunded_cdemo_sk integer      ,
  wr_refunded_hdemo_sk integer      ,
  wr_refunded_addr_sk integer      ,
  wr_returning_customer_sk integer   ,
  wr_returning_cdemo_sk integer      ,
  wr_returning_hdemo_sk integer      ,
  wr_returning_addr_sk integer      ,
  wr_web_page_sk     integer      ,
  wr_reason_sk       integer      ,
  wr_order_number    integer      not null,
  wr_return_quantity integer      ,
  wr_return_amt      decimal(7,2) ,
  wr_return_tax      decimal(7,2) ,
  wr_return_amt_inc_tax decimal(7,2) ,
  wr_fee             decimal(7,2) ,
  wr_return_ship_cost decimal(7,2) ,
  wr_refunded_cash   decimal(7,2) ,
  wr_reversed_charge decimal(7,2) ,
  wr_account_credit  decimal(7,2) ,
  wr_net_loss        decimal(7,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE web_sales
(
  ws_sold_date_sk integer      ,
```

```

ws_sold_time_sk      integer      ,
ws_ship_date_sk     integer      ,
ws_item_sk          integer      not null,
ws_bill_customer_sk integer      ,
ws_bill_cdemo_sk    integer      ,
ws_bill_hdemo_sk    integer      ,
ws_bill_addr_sk     integer      ,
ws_ship_customer_sk integer      ,
ws_ship_cdemo_sk    integer      ,
ws_ship_hdemo_sk    integer      ,
ws_ship_addr_sk     integer      ,
ws_web_page_sk      integer      ,
ws_web_site_sk      integer      ,
ws_ship_mode_sk     integer      ,
ws_warehouse_sk     integer      ,
ws_promo_sk         integer      ,
ws_order_number     integer      not null,
ws_quantity         integer      ,
ws_wholesale_cost   decimal(7,2) ,
ws_list_price       decimal(7,2) ,
ws_sales_price      decimal(7,2) ,
ws_ext_discount_amt decimal(7,2) ,
ws_ext_sales_price  decimal(7,2) ,
ws_ext_wholesale_cost decimal(7,2) ,
ws_ext_list_price   decimal(7,2) ,
ws_ext_tax          decimal(7,2) ,
ws_coupon_amt       decimal(7,2) ,
ws_ext_ship_cost    decimal(7,2) ,
ws_net_paid         decimal(7,2) ,
ws_net_paid_inc_tax decimal(7,2) ,
ws_net_paid_inc_ship decimal(7,2) ,
ws_net_paid_inc_ship_tax decimal(7,2) ,
ws_net_profit       decimal(7,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE catalog_sales
(
  cs_sold_date_sk      integer      ,
  cs_sold_time_sk     integer      ,
  cs_ship_date_sk     integer      ,
  cs_bill_customer_sk integer      ,
  cs_bill_cdemo_sk    integer      ,
  cs_bill_hdemo_sk    integer      ,
  cs_bill_addr_sk     integer      ,
  cs_ship_customer_sk integer      ,
  cs_ship_cdemo_sk    integer      ,
  cs_ship_hdemo_sk    integer      ,
  cs_ship_addr_sk     integer      ,
  cs_call_center_sk   integer      ,
  cs_catalog_page_sk  integer      ,
  cs_ship_mode_sk     integer      ,
  cs_warehouse_sk     integer      ,
  cs_item_sk          integer      not null,
  cs_promo_sk         integer      ,
  cs_order_number     integer      not null,
  cs_quantity         integer      ,
  cs_wholesale_cost   decimal(7,2) ,
  cs_list_price       decimal(7,2) ,
  cs_sales_price      decimal(7,2) ,
  cs_ext_discount_amt decimal(7,2) ,
  cs_ext_sales_price  decimal(7,2) ,
  cs_ext_wholesale_cost decimal(7,2) ,
  cs_ext_list_price   decimal(7,2) ,
  cs_ext_tax          decimal(7,2) ,
  cs_coupon_amt       decimal(7,2) ,
  cs_ext_ship_cost    decimal(7,2) ,
  cs_net_paid         decimal(7,2) ,
  cs_net_paid_inc_tax decimal(7,2) ,
  cs_net_paid_inc_ship decimal(7,2)
)

```

```
cs_net_paid_inc_ship_tax decimal(7,2) ,
cs_net_profit decimal(7,2)
)WITH (orientation = column, COMPRESSION = MIDDLE);

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 integer 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);
```

## 2. Create an OBS foreign table.

Copy and run the following statements to create an OBS foreign table in the **gaussdb** database, identifying data format and setting error tolerance.

### NOTE

Configure the **ACCESS\_KEY** and **SECRET\_ACCESS\_KEY** parameters as required, and run the statements to create a foreign table using the client tool.

For details about how to obtain the values of **ACCESS\_KEY** and **SECRET\_ACCESS\_KEY**, see [Creating Access Keys \(AK and SK\)](#).

```
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://dws/download/dws_sample_database_data_files/customer_address/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_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://dws/download/dws_sample_database_data_files/customer_demographics/
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_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://dws/download/dws_sample_database_data_files/ date_dim/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_warehouse_001
(
  w_warehouse_sk      integer      not null,
  w_warehouse_id     char(16)     not null,
  w_warehouse_name   varchar(20)  ,
  w_warehouse_sq_ft  integer      ,
  w_street_number    char(10)     ,
  w_street_name      varchar(60)  ,
  w_street_type      char(15)     ,
  w_suite_number     char(10)     ,
  w_city             varchar(60)  ,
  w_county           varchar(30)  ,
  w_state            char(2)       ,
  w_zip              char(10)     ,
  w_country          varchar(20)  ,
  w_gmt_offset       decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/ warehouse/warehouse',
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_warehouse_001;

CREATE FOREIGN TABLE obs_from_ship_mode_001
(
  sm_ship_mode_sk    integer      not null,
  sm_ship_mode_id    char(16)     not null,
  sm_type            char(30)     ,
  sm_code            char(10)     ,
  sm_carrier         char(20)     ,
  sm_contract        char(20)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/ ship_mode/ship_mode' ,
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_ship_mode_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://dws/download/dws_sample_database_data_files/time_dim/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_reason_001
(
r_reason_sk      integer          not null,
r_reason_id      char(16)         not null,
r_reason_desc    char(100)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/reason/reason' ,
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_reason_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://dws/download/dws_sample_database_data_files/income_band/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;

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://dws/download/dws_sample_database_data_files/item/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_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_percentage decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/store/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_call_center_001
(
cc_call_center_sk      integer      not null,
cc_call_center_id     char(16)     not null,
cc_rec_start_date     date          ,
cc_rec_end_date       date          ,
cc_closed_date_sk     integer      ,
cc_open_date_sk       integer      ,
cc_name               varchar(50)  ,
cc_class              varchar(50)  ,
cc_employees          integer      ,
cc_sq_ft              integer      ,
cc_hours              char(20)     ,
cc_manager            varchar(40)  ,
cc_mkt_id             integer      ,
cc_mkt_class          char(50)     ,
cc_mkt_desc           varchar(100) ,
cc_market_manager    varchar(40)  ,
cc_division           integer      ,
cc_division_name     varchar(50)  ,
cc_company            integer      ,
cc_company_name       char(50)     ,
cc_street_number     char(10)     ,
cc_street_name       varchar(60)  ,
cc_street_type       char(15)     ,
cc_suite_number      char(10)     ,
cc_city              varchar(60)  ,
cc_county            varchar(30)  ,
cc_state              char(2)      ,
cc_zip               char(10)     ,
cc_country            varchar(20)  ,
cc_gmt_offset         decimal(5,2) ,
cc_tax_percentage     decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/call_center/call_center',
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_call_center_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_hdemo_sk   integer      ,
c_current_addr_sk    integer      ,
c_first_shipto_date_sk integer      ,
c_first_sales_date_sk integer      ,
c_salutation         char(10)     ,
c_first_name         char(20)     ,
c_last_name          char(30)     ,
c_preferred_cust_flag char(1)     ,
c_birth_day          integer      ,
c_birth_month        integer      ,
c_birth_year         integer      ,

```



```
c_birth_country    varchar(20)      ,
c_login           char(13)        ,
c_email_address   char(50)       ,
c_last_review_date char(10)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/customer/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_web_site_001
(
web_site_sk        integer        not null,
web_site_id       char(16)        not null,
web_rec_start_date date           ,
web_rec_end_date  date           ,
web_name          varchar(50)     ,
web_open_date_sk  integer        ,
web_close_date_sk integer        ,
web_class         varchar(50)     ,
web_manager       varchar(40)     ,
web_mkt_id        integer        ,
web_mkt_class     varchar(50)     ,
web_mkt_desc      varchar(100)    ,
web_market_manager varchar(40)    ,
web_company_id    integer        ,
web_company_name  char(50)        ,
web_street_number char(10)        ,
web_street_name   varchar(60)     ,
web_street_type   char(15)        ,
web_suite_number  char(10)        ,
web_city          varchar(60)     ,
web_county        varchar(30)     ,
web_state         char(2)         ,
web_zip           char(10)        ,
web_country       varchar(20)     ,
web_gmt_offset    decimal(5,2)    ,
web_tax_percentage decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/web_site/web_site' ,
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_web_site_001;

CREATE FOREIGN TABLE obs_from_store_returns_001
(
sr_returned_date_sk integer        ,
sr_return_time_sk   integer        ,
sr_item_sk          integer        not null,
sr_customer_sk      integer        ,
sr_demo_sk          integer        ,

```

```
sr_hdemo_sk      integer      ,
sr_addr_sk       integer      ,
sr_store_sk      integer      ,
sr_reason_sk     integer      ,
sr_ticket_number bigint      not null,
sr_return_quantity integer     ,
sr_return_amt    decimal(7,2)  ,
sr_return_tax    decimal(7,2)  ,
sr_return_amt_inc_tax decimal(7,2) ,
sr_fee          decimal(7,2)    ,
sr_return_ship_cost decimal(7,2) ,
sr_refunded_cash decimal(7,2)  ,
sr_reversed_charge decimal(7,2) ,
sr_store_credit  decimal(7,2)  ,
sr_net_loss     decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/store_returns/store_returns' ,
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_returns_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://dws/download/dws_sample_database_data_files/household_demographics/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_web_page_001
(
wp_web_page_sk      integer      not null,
wp_web_page_id      char(16)     not null,
wp_rec_start_date   date         ,
wp_rec_end_date     date         ,
wp_creation_date_sk integer      ,
wp_access_date_sk   integer      ,
wp_autogen_flag     char(1)      ,
wp_customer_sk      integer      ,
wp_url              varchar(100) ,
wp_type             char(50)     ,
wp_char_count       integer      ,
wp_link_count       integer      ,
wp_image_count      integer      ,
wp_max_ad_count     integer
)
```

```
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/web_page/web_page' ,
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_web_page_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://dws/download/dws_sample_database_data_files/promotion/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_catalog_page_001
(
cp_catalog_page_sk integer          not null,
cp_catalog_page_id char(16)         not null,
cp_start_date_sk   integer           ,
cp_end_date_sk     integer           ,
cp_department      varchar(50)       ,
cp_catalog_number  integer           ,
cp_catalog_page_number integer       ,
cp_description     varchar(100)      ,
cp_type            varchar(100)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/catalog_page/catalog_page' ,
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_catalog_page_001;

CREATE FOREIGN TABLE obs_from_inventory_001
(
    inv_date_sk          integer          not null,
    inv_item_sk          integer          not null,
    inv_warehouse_sk     integer          not null,
    inv_quantity_on_hand integer
)
SERVER gsmpp_server
OPTIONS (
    location 'obs://dws/download/dws_sample_database_data_files/inventory/inventory',
    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_inventory_001;

CREATE FOREIGN TABLE obs_from_catalog_returns_001
(
    cr_returned_date_sk  integer          ,
    cr_returned_time_sk  integer          ,
    cr_item_sk           integer          not null,
    cr_refunded_customer_sk integer        ,
    cr_refunded_cdemo_sk integer          ,
    cr_refunded_hdemo_sk integer          ,
    cr_refunded_addr_sk  integer          ,
    cr_returning_customer_sk integer        ,
    cr_returning_cdemo_sk integer          ,
    cr_returning_hdemo_sk integer          ,
    cr_returning_addr_sk integer          ,
    cr_call_center_sk    integer          ,
    cr_catalog_page_sk   integer          ,
    cr_ship_mode_sk      integer          ,
    cr_warehouse_sk      integer          ,
    cr_reason_sk         integer          ,
    cr_order_number      bigint           not null,
    cr_return_quantity   integer          ,
    cr_return_amount     decimal(7,2)     ,
    cr_return_tax        decimal(7,2)     ,
    cr_return_amt_inc_tax decimal(7,2)     ,
    cr_fee               decimal(7,2)     ,
    cr_return_ship_cost  decimal(7,2)     ,
    cr_refunded_cash     decimal(7,2)     ,
    cr_reversed_charge   decimal(7,2)     ,
    cr_store_credit      decimal(7,2)     ,
    cr_net_loss          decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
    location 'obs://dws/download/dws_sample_database_data_files/catalog_returns/catalog_returns',
    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',
```

```
chunksizes '64'
)
with err_obs_from_catalog_returns_001;

CREATE FOREIGN TABLE obs_from_web_returns_001
(
  wr_returned_date_sk integer ,
  wr_returned_time_sk integer ,
  wr_item_sk integer not null,
  wr_refunded_customer_sk integer ,
  wr_refunded_cdemo_sk integer ,
  wr_refunded_hdemo_sk integer ,
  wr_refunded_addr_sk integer ,
  wr_returning_customer_sk integer ,
  wr_returning_cdemo_sk integer ,
  wr_returning_hdemo_sk integer ,
  wr_returning_addr_sk integer ,
  wr_web_page_sk integer ,
  wr_reason_sk integer ,
  wr_order_number bigint not null,
  wr_return_quantity integer ,
  wr_return_amt decimal(7,2) ,
  wr_return_tax decimal(7,2) ,
  wr_return_amt_inc_tax decimal(7,2) ,
  wr_fee decimal(7,2) ,
  wr_return_ship_cost decimal(7,2) ,
  wr_refunded_cash decimal(7,2) ,
  wr_reversed_charge decimal(7,2) ,
  wr_account_credit decimal(7,2) ,
  wr_net_loss decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
  location 'obs://dws/download/dws_sample_database_data_files/web_returns/web_returns',
  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',
  chunksizes '64'
)
with err_obs_from_web_returns_001;

CREATE FOREIGN TABLE obs_from_web_sales_001
(
  ws_sold_date_sk integer ,
  ws_sold_time_sk integer ,
  ws_ship_date_sk integer ,
  ws_item_sk integer not null,
  ws_bill_customer_sk integer ,
  ws_bill_cdemo_sk integer ,
  ws_bill_hdemo_sk integer ,
  ws_bill_addr_sk integer ,
  ws_ship_customer_sk integer ,
  ws_ship_cdemo_sk integer ,
  ws_ship_hdemo_sk integer ,
  ws_ship_addr_sk integer ,
  ws_web_page_sk integer ,
  ws_web_site_sk integer ,
  ws_ship_mode_sk integer ,
  ws_warehouse_sk integer ,
  ws_promo_sk integer ,
  ws_order_number bigint not null,
  ws_quantity integer ,
  ws_wholesale_cost decimal(7,2) ,
  ws_list_price decimal(7,2) ,
  ws_sales_price decimal(7,2) ,

```

```
ws_ext_discount_amt    decimal(7,2)    ,
ws_ext_sales_price     decimal(7,2)    ,
ws_ext_wholesale_cost  decimal(7,2)    ,
ws_ext_list_price      decimal(7,2)    ,
ws_ext_tax             decimal(7,2)    ,
ws_coupon_amt         decimal(7,2)    ,
ws_ext_ship_cost      decimal(7,2)    ,
ws_net_paid           decimal(7,2)    ,
ws_net_paid_inc_tax   decimal(7,2)    ,
ws_net_paid_inc_ship  decimal(7,2)    ,
ws_net_paid_inc_ship_tax decimal(7,2)    ,
ws_net_profit         decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/web_sales/web_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_web_sales_001;

CREATE FOREIGN TABLE obs_from_catalog_sales_001
(
cs_sold_date_sk        integer          ,
cs_sold_time_sk        integer          ,
cs_ship_date_sk        integer          ,
cs_bill_customer_sk    integer          ,
cs_bill_cdemo_sk       integer          ,
cs_bill_hdemo_sk       integer          ,
cs_bill_addr_sk        integer          ,
cs_ship_customer_sk    integer          ,
cs_ship_cdemo_sk       integer          ,
cs_ship_hdemo_sk       integer          ,
cs_ship_addr_sk        integer          ,
cs_call_center_sk      integer          ,
cs_catalog_page_sk     integer          ,
cs_ship_mode_sk        integer          ,
cs_warehouse_sk       integer          ,
cs_item_sk             integer          not null,
cs_promo_sk            integer          ,
cs_order_number        bigint          not null,
cs_quantity            integer          ,
cs_wholesale_cost      decimal(7,2)    ,
cs_list_price          decimal(7,2)    ,
cs_sales_price         decimal(7,2)    ,
cs_ext_discount_amt    decimal(7,2)    ,
cs_ext_sales_price     decimal(7,2)    ,
cs_ext_wholesale_cost  decimal(7,2)    ,
cs_ext_list_price      decimal(7,2)    ,
cs_ext_tax             decimal(7,2)    ,
cs_coupon_amt         decimal(7,2)    ,
cs_ext_ship_cost      decimal(7,2)    ,
cs_net_paid           decimal(7,2)    ,
cs_net_paid_inc_tax   decimal(7,2)    ,
cs_net_paid_inc_ship  decimal(7,2)    ,
cs_net_paid_inc_ship_tax decimal(7,2)    ,
cs_net_profit         decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://dws/download/dws_sample_database_data_files/catalog_sales/catalog_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_catalog_sales_001;

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://dws/download/dws_sample_database_data_files/store_sales/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;
```

### 3. Run **INSERT** to insert data.

```
INSERT INTO customer_address SELECT * FROM obs_from_customer_address_001 ;
INSERT INTO customer_demographics SELECT * FROM obs_from_customer_demographics_001 ;
INSERT INTO date_dim SELECT * FROM obs_from_date_dim_001;
INSERT INTO warehouse SELECT * FROM obs_from_warehouse_001;
INSERT INTO ship_mode SELECT * FROM obs_from_ship_mode_001;
INSERT INTO time_dim SELECT * FROM obs_from_time_dim_001;
INSERT INTO reason SELECT * FROM obs_from_reason_001;
INSERT INTO income_band SELECT * FROM obs_from_income_band_001;
INSERT INTO item SELECT * FROM obs_from_item_001;
INSERT INTO store SELECT * FROM obs_from_store_001;
INSERT INTO call_center SELECT * FROM obs_from_call_center_001;
INSERT INTO customer SELECT * FROM obs_from_customer_001;
INSERT INTO web_site SELECT * FROM obs_from_web_site_001;
INSERT INTO store_returns SELECT * FROM obs_from_store_returns_001;
INSERT INTO household_demographics SELECT * FROM obs_from_household_demographics_001;
INSERT INTO web_page SELECT * FROM obs_from_web_page_001;
INSERT INTO promotion SELECT * FROM obs_from_promotion_001;
INSERT INTO catalog_page SELECT * FROM obs_from_catalog_page_001;
INSERT INTO inventory SELECT * FROM obs_from_inventory_001;
INSERT INTO catalog_returns SELECT * FROM obs_from_catalog_returns_001;
```

```
INSERT INTO web_returns SELECT * FROM obs_from_web_returns_001;
INSERT INTO web_sales SELECT * FROM obs_from_web_sales_001;
INSERT INTO catalog_sales SELECT * FROM obs_from_catalog_sales_001;
INSERT INTO store_sales SELECT * FROM obs_from_store_sales_001;
```

#### 4. Optimize performance.

```
ANALYZE customer_address;
ANALYZE customer_demographics;
ANALYZE date_dim;
ANALYZE warehouse;
ANALYZE ship_mode;
ANALYZE time_dim;
ANALYZE reason;
ANALYZE income_band;
ANALYZE item;
ANALYZE store;
ANALYZE call_center;
ANALYZE customer;
ANALYZE web_site;
ANALYZE store_returns;
ANALYZE household_demographics;
ANALYZE web_page;
ANALYZE promotion;
ANALYZE catalog_page;
ANALYZE inventory;
ANALYZE catalog_returns;
ANALYZE web_returns;
ANALYZE web_sales;
ANALYZE catalog_sales;
ANALYZE store_sales;
```

#### 5. Run **SELECT** to query the database. For details, see "SELECT."

```
-- Query all records and sort them in alphabetic order.
SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

-- Filter based on query conditions, and group the results:
SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;
```

## 11.6 Querying System Catalogs

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

The description of each table in [System Catalogs and System Views](#) specifies whether the table is visible to all users or only to the initial user. To query tables that are visible only to the initial user, log in as the initial user.

### Querying Database Tables

For example, you can run the following statement to query the **PG\_TABLES** system catalog for all tables in the **public** schema:

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

Information similar to the following is displayed:

```
tablename
-----
err_hr_staffs
test
err_hr_staffs_ft3
web_returns_p1
```



```
mig_seq_table
films4
(6 rows)
```

## Querying Database Users

You can run **PG\_USER** to query all users in the database. User IDs (**USESYSID**) and permissions can also be queried.

```
SELECT * FROM pg_user;
       username          | usesysid | usecreatedb | usesuper | usecatupd | userepl | passwd | valbegin |
valuntil | respool
       | parent | spacelimit | useconfig
-----+-----+-----+-----+-----+-----+-----+-----+
dfc22b86afbd9a745668c3ecd0f15ec18 | 17107 | f          | f        | f         | f        |  |  |
default_p
ool | 0 |  |
guest | 17103 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
Ruby | 10 | t          | t        | t         | t        |  |  | default_p
ool | 0 |  |
dbadmin | 16404 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
lily | 16482 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
jack | 16478 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
(6 rows)
```

GaussDB(DWS) uses **Ruby** to perform routine management and O&M. You can add **WHERE usesysid > 10** to the **SELECT** statement so that only specified usernames are displayed.

```
SELECT * FROM pg_user WHERE usesysid > 10;
       username          | usesysid | usecreatedb | usesuper | usecatupd | userepl | passwd | valbegin |
valuntil | respool
       | parent | spacelimit | useconfig
-----+-----+-----+-----+-----+-----+-----+
dfc22b86afbd9a745668c3ecd0f15ec18 | 17107 | f          | f        | f         | f        |  |  |
default_p
ool | 0 |  |
guest | 17103 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
dbadmin | 16404 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
lily | 16482 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
jack | 16478 | f          | f        | f         | f        |  |  | default_p
ool | 0 |  |
(5 rows)
```

## Querying User Attributes

**PG\_AUTHID** can be used to view the attribute list of all users in the database.

```
SELECT * FROM pg_authid;
rolname | rolsuper | rolinherit | rolcreatorole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit | rolpassword | rolvalidbegin | rolvaliduntil | rolrespool |
roluseft | rolparentid | roltabspace | rolkind | rolnodegroup | roltempSPACE | rolspillspace | rolexcpdata |
```

```

rolmonitoradmin | rooperatoradmin | rolpolicyadmin
-----+-----+-----+-----+-----+-----+-----+-----
|          |          |          |          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |          |          |          |
-----+-----+-----+-----+-----+-----+-----+-----
|          |          |          |          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |          |          |          |
-----+-----+-----+-----+-----+-----+-----+-----
dbadmin | f | t | f | f | f | f | f | f | t | 
-1 | sha256ce0ea617e7b7c0f2d38b00a12261b5e98ce18c218
89a30ebf410631c52f882d376141f31b47b67b0ceec9d10c931358140d276009bd8d19ac6a5647558c8b70d009
986e774c2ba9563e42f4331629379d40720e4a3e0997c2b592833db778908md5eb0c1ffc5c76ef6272debb03f58
5b0b9ecdfecefade |
|          | default_pool | f |          | 0 |          | n |          |          |          |          |
f |          | f |          | f |          |          |          |          |          |          |
Ruby | t | t | t | t | t | t | t | t | t | t | 
-1 | sha256d2058470eb2cead16d1d85a2b69207bc33e020bd4
530b67102c9c237dd2cb5d1ebae9d98c88ebf8f51950a333a4bb436488df40e645eb3d346af6c401c8fe5f83b208
7349dccc38fd1eb8ec828b27f28af4e5066549ba0bb6d249f82c664151md5417898ceaa6a205cb03d1b8df8fb9
2f7ecdfecefade |
|          | default_pool | t |          | 0 |          | n |          | 0 |          |          |          |
t |          | t |          | t |          |          |          |          |          |          |
(2 rows)

```

Query the permissions of user **joe**.

```
SELECT * FROM pg_authid where rolname = 'joe';
```

## Querying and Stopping the Running Query Statements

You can view the running query statements in the [PG\\_STAT\\_ACTIVITY](#) view.

### Step 1 Set `track_activities` to **on**.

```
SET track_activities = on;
```

The database collects the running information about active queries only when this parameter is set to **on**.

### Step 2 Query the information about running query statements, such as the user who runs the statements and the connected database, query status, and PID of the statements.

```

SELECT datname, username, state,pid FROM pg_stat_activity;
datname | username | state | pid
-----+-----+-----+-----
postgres | Ruby | active | 140298793514752
postgres | Ruby | active | 140298718004992
postgres | Ruby | idle | 140298650908416
postgres | Ruby | idle | 140298625742592
postgres | dbadmin | active | 140298575406848
(5 rows)

```

If **state** is **idle**, the connection is idle and requires a user to enter a command.

To identify queries that are not idle, run the following command:

```
SELECT datname, username, state FROM pg_stat_activity WHERE state != 'idle';
```

### Step 3 To cancel queries that have been running for a long time, use the **PG\_TERMINATE\_BACKEND** function to end sessions based on the thread ID.

```
SELECT PG_TERMINATE_BACKEND(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, a user terminates the current session.

```
FATAL: terminating connection due to administrator command  
FATAL: terminating connection due to administrator command
```

#### NOTE

If the **PG\_TERMINATE\_BACKEND** function is used to terminate the backend threads of the current session, the `gsql` client will be reconnected automatically rather than be logged out. Information **The connection to the server was lost. Attempting reset: Succeeded.** is returned.

```
FATAL: terminating connection due to administrator command  
FATAL: terminating connection due to administrator command  
The connection to the server was lost. Attempting reset: Succeeded.
```

----End

## 11.7 Creating and Managing Schemas

### Background

Based on schema management, multiple users can use the same database without conflicts. Database objects can be organized as manageable logical groups. In addition, third-party applications can be added to the same schema without causing conflicts. Schema management involves creating a schema, using a schema, deleting a schema, setting a search path for a schema, and setting schema permissions.

### Important Notes

- The database cluster has one or more named databases. Users and user groups are shared within a cluster, but their data is exclusive. Any user who has connected to a server can only access the database that is specified in the connection request.
- A database can have one or more schemas, and a schema can contain tables and other data objects, such as data types, functions, and operators. One object name can be used in different schemas. For example, both `schema1` and `schema2` can have a table named `mytable`.
- Different from databases, schemas are not isolated. You can access the objects in a schema of the connected database based on your schema permissions. To manage schema permissions, you need to have a good understanding of the database permissions.
- A schema named with the **PG\_** prefix cannot be created because this type of schema is reserved for the database system.
- If a user is created, a schema named after the user will also be created in the current database.
- To reference a table that is not modified with a schema name, the system uses **search\_path** to find the schema that the table belongs to. **pg\_temp** and **pg\_catalog** are always the first two schemas to be searched no matter whether or how they are specified in **search\_path**. **search\_path** is a schema name list, and the first table detected in it is the target table. If no target

table is found, an error will be reported. (If a table exists but the schema it belongs to is not listed in **search\_path**, the search fails as well.) The first schema in **search\_path** is called **current schema**. This schema is the first one to be searched. If no schema name is declared, newly created database objects are saved in this schema by default.

- Each database has a **pg\_catalog** schema, which contains system catalogs and all built-in data types, functions, and operators. **pg\_catalog** is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search\_path**. This search order ensures that database built-in objects can be found. To use a custom object that has the same name as a built-in object, you can specify the schema of the custom object.

## Procedure

- Create a schema.
  - Run the following command to create a schema:  

```
CREATE SCHEMA myschema;
```

If the following information is displayed, the schema named **myschema** has been successfully created:

```
CREATE SCHEMA
```

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

```
CREATE SCHEMA myschema AUTHORIZATION dbadmin;
```

If the following information is displayed, the **myschema** schema that belongs to **dbadmin** has been created successfully:

```
CREATE SCHEMA
```
- Use a schema.

If you want to create or access an object in a specified schema, the object name must contain the schema name. To be specific, the name consists of a schema name and an object name, which are separated by a dot (.).

  - Run the following command to create table **mytable** in **myschema**:  

```
CREATE TABLE myschema.mytable(id int, name varchar(20));
```

To specify the location of an object, the object name must contain the schema name.
  - Run the following command to query all data of table **mytable** in **myschema**:  

```
SELECT * FROM myschema.mytable;
```

```
id | name  
----+-----  
(0 rows)
```
- View **search\_path** of a schema.

You can set **search\_path** to specify the sequence of schemas in which objects are searched. The first schema listed in **search\_path** will become the default schema. If no schema is specified during object creation, the object will be created in the default schema.

  - Run the following command to view **search\_path**:  

```
SHOW SEARCH_PATH;
```

```
search_path
```

```
-----  
"$user",public  
(1 row)
```

- Run the following command to set **search\_path** to **myschema** and **public** (**myschema** is searched first):

```
SET SEARCH_PATH TO myschema, public;  
SET
```

- Set permissions for a schema.

By default, a user can only access database objects in its own schema. Only after a user is granted with the usage permission on a schema by the schema owner, the user can access the objects in the schema.

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

- Run the following command to view the current schema:

```
SELECT current_schema();  
current_schema  
-----  
myschema  
(1 row)
```

- Run the following commands to create user **jack** and grant the usage permission on **myschema** to the user:

```
CREATE USER jack IDENTIFIED BY 'password';  
GRANT USAGE ON schema myschema TO jack;
```

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

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

- Delete a schema.

- If a schema is empty, that is, it contains no database object, you can execute the **DROP SCHEMA** statement to delete it. For example, run the following command to delete an empty schema named **nullschema**:

```
DROP SCHEMA IF EXISTS nullschema;
```

- To delete a schema that is not null, use the keyword **CASCADE** to delete it and all its objects. For example, run the following command to delete **myschema** and all objects in it:

```
DROP SCHEMA myschema CASCADE;
```

- Delete user **jack**.

```
DROP USER jack;
```

## 11.8 Creating and Managing Partitioned Tables

### Background

GaussDB(DWS) supports range partitioned tables and list partitioned tables.

Range partitioned table: Data within a specific range is mapped onto each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning mode is most commonly used. The partition key is usually a date. For example, sales data is partitioned by month.

List partitioned table: Data is mapped to partitions based on partition keys. These keys do not overlap in different partitions. Create a partition for each group of key values to store corresponding data. List partitioning is supported only by clusters of 8.1.3 and later versions.

A partitioned table has the following advantages over an ordinary table:

- High query performance: The system queries only the concerned partitions rather than the whole table, improving the query efficiency.
- High availability: If a partition is faulty, data in the other partitions is still available.
- Easy maintenance: You only need to fix the faulty partition.
- Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

To convert an ordinary table to a partitioned table, you need to create a partitioned table and import data to it from the ordinary table. When you design tables, plan whether to use partitioned tables based on service requirements.

## Procedure

- Perform the following operations on a range partitioned table.

- Create a range partitioned table:

```
CREATE TABLE tpcds.customer_address
(
  ca_address_sk integer NOT NULL ,
  ca_address_id character(16) NOT NULL ,
  ca_street_number character(10) ,
  ca_street_name character varying(60) ,
  ca_street_type character(15) ,
  ca_suite_number character(10) ,
  ca_city character varying(60) ,
  ca_county character varying(30) ,
  ca_state character(2) ,
  ca_zip character(10) ,
  ca_country character varying(20) ,
  ca_gmt_offset numeric(5,2) ,
  ca_location_type character(20)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
  PARTITION P1 VALUES LESS THAN(5000),
  PARTITION P2 VALUES LESS THAN(10000),
  PARTITION P3 VALUES LESS THAN(15000),
  PARTITION P4 VALUES LESS THAN(20000),
  PARTITION P5 VALUES LESS THAN(25000),
  PARTITION P6 VALUES LESS THAN(30000),
  PARTITION P7 VALUES LESS THAN(40000),
  PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

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

```
CREATE TABLE
```

### NOTE

Create a maximum of 1000 column-store partitioned tables.

- Insert data.

Insert data from the **tpcds.customer\_address** table to the **tpcds.web\_returns\_p2** table.

For example, you can run the following command to insert the data of the **tpcds.customer\_address** table into its backup table **tpcds.web\_returns\_p2**:

```
CREATE TABLE tpcds.web_returns_p2
(
  ca_address_sk integer NOT NULL ,
  ca_address_id character(16) NOT NULL ,
  ca_street_number character(10) ,
  ca_street_name character varying(60) ,
  ca_street_type character(15) ,
  ca_suite_number character(10) ,
  ca_city character varying(60) ,
  ca_county character varying(30) ,
  ca_state character(2) ,
  ca_zip character(10) ,
  ca_country character varying(20) ,
  ca_gmt_offset numeric(5,2) ,
  ca_location_type character(20)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
  PARTITION P1 VALUES LESS THAN(5000),
  PARTITION P2 VALUES LESS THAN(10000),
  PARTITION P3 VALUES LESS THAN(15000),
  PARTITION P4 VALUES LESS THAN(20000),
  PARTITION P5 VALUES LESS THAN(25000),
  PARTITION P6 VALUES LESS THAN(30000),
  PARTITION P7 VALUES LESS THAN(40000),
  PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
CREATE TABLE
INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

#### NOTE

**ROW MOVEMENT** is disabled by default. In this case, cross-partition update is not allowed. To enable cross-partition update, specify **ENABLE ROW MOVEMENT**. However, if **SELECT FOR UPDATE** is executed concurrently to query the partitioned table, the query results may be inconsistent. Therefore, exercise caution when performing this operation.

- Modify the row movement attributes of a partitioned table.

```
ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
```

- Delete a partition.

Run the following command to delete partition **P8**:

```
ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
```

- Add a partition.

Run the following command to add partition **P8** and set its range to [40000, MAXVALUE]:

```
ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN (MAXVALUE);
```

- Rename a partition.

- Run the following command to rename partition **P8** to **P\_9**:

```
ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
```

- Run the following command to rename partition **P\_9** to **P8**:

```
ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
```

- Query a partition.

Run the following command to query partition **P7**:

```
SELECT * FROM tpcds.web_returns_p2 PARTITION (P7);
```

```
SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
```

- View partitioned tables using the system catalog **dba\_tab\_partitions**.

- ```
SELECT * FROM dba_tab_partitions WHERE table_name='tpcds.customer_address';
```
- Delete a partitioned table.  

```
DROP TABLE tpcds.web_returns_p2;
```
  - Perform the following operations on a list partitioned table:
    - Create a list partitioned table.  

```
CREATE TABLE data_list  
(  
  id int,  
  time int,  
  sarlay decimal(12,2)  
)  
PARTITION BY LIST (time)  
(  
  PARTITION P1 VALUES (202209),  
  PARTITION P2 VALUES (202210,202208),  
  PARTITION P3 VALUES (202211),  
  PARTITION P4 VALUES (202212),  
  PARTITION P5 VALUES (202301)  
);
```
    - If the following information is displayed, the tablespaces are created.  

```
CREATE TABLE
```
    - Insert data.  

```
INSERT INTO data_list VALUES (1,202209,10000),(2,202210,20000),(3,202211,30000),  
(4,202212,40000),(5,202301,50000),(6,202301,60000);
```
    - Add a partition.  

```
ALTER TABLE data_list ADD PARTITION P6 VALUES (202302,202303);
```
    - Split partitions.  

```
ALTER TABLE data_list SPLIT PARTITION P2 VALUES(202210) INTO (PARTITION p2a,PARTITION  
p2b);
```
    - Merge partitions.  

```
ALTER TABLE data_list MERGE PARTITIONS p2a,p2b INTO PARTITION P2;
```
    - Rename a partition.
      - Rename partition **P4** to **P\_5**.  

```
ALTER TABLE data_list RENAME PARTITION P4 TO P_5;
```
      - Rename partition **P\_5** to **P4**.  

```
ALTER TABLE data_list RENAME PARTITION FOR (202212) TO P4;
```
    - Delete a partition.  
**Delete partition P1.**  

```
ALTER TABLE data_list DROP PARTITION P1;
```
    - Query a partition.  
**Query partition P5.**  

```
SELECT * FROM data_list PARTITION (P5);  
SELECT * FROM data_list PARTITION FOR (202301);
```
    - View partitioned tables using the system catalog **dba\_tab\_partitions**.  

```
SELECT * FROM dba_tab_partitions where table_name='data_list';
```
    - Delete a partitioned table.  

```
DROP TABLE data_list;
```

## 11.9 Creating and Managing Indexes

### Background

Indexes accelerate the data access speed but also add the processing time of the insert, update, and delete operations. Therefore, before creating an index, consider



whether it is necessary and determine the columns where indexes will be created. You can determine whether to add an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be sorted.

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

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

#### NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequenced scanning, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located. Therefore, data operations increase. Therefore, delete unnecessary indexes periodically.
- After an index is created, it takes effect on the existing data in the table.

## Procedure

For details about the procedure for creating a partitioned table, see [Creating and Managing Partitioned Tables](#).

- Creating an Index
  - Create the partitioned table index **tpcds\_web\_returns\_p2\_index1** without specifying the partition name.  

```
CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2 (ca_address_id) LOCAL;
```

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

```
CREATE INDEX
```
  - Create the partitioned table index **tpcds\_web\_returns\_p2\_index2** and specify index names for all partitions. Currently, specifying index names for partial partitions is not allowed.  

```
CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2 (ca_address_sk) LOCAL  
(  
PARTITION web_returns_p2_P1_index,  
PARTITION web_returns_p2_P2_index TABLESPACE example3,  
PARTITION web_returns_p2_P3_index TABLESPACE example4,  
PARTITION web_returns_p2_P4_index,  
PARTITION web_returns_p2_P5_index,  
PARTITION web_returns_p2_P6_index,  
PARTITION web_returns_p2_P7_index,
```

```
PARTITION web_returns_p2_P8_index
) TABLESPACE example2;
```

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

```
CREATE INDEX
```

- Renaming an index partition

Rename the name of index partition **web\_returns\_p2\_P8\_index** to **web\_returns\_p2\_P8\_index\_new**.

```
ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION web_returns_p2_P8_index TO
web_returns_p2_P8_index_new;
```

If the following information is displayed, the index has been renamed.

```
ALTER INDEX
```

- Querying indexes

- Run the following command to query all indexes defined by the system and users:

```
SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```

- Run the following command to query information about a specified index:

```
\di+ tpcds.tpcds_web_returns_p2_index2
```

- Deleting an index

```
DROP INDEX tpcds.tpcds_web_returns_p2_index1;
DROP INDEX tpcds.tpcds_web_returns_p2_index2;
```

If the following output is displayed, the index has been deleted.

```
DROP INDEX
```

GaussDB(DWS) supports four methods for creating indexes. For details, see [Table 11-3](#).

#### NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequenced scanning, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located. Therefore, data operations increase. Therefore, delete unnecessary indexes periodically.

**Table 11-3 Indexing Method**

| Indexing Method | Description                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unique index    | Refers to an index that constrains the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, GaussDB(DWS) automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, only B-tree can create a unique index in GaussDB(DWS). |
| Composite index | Refers to an index that can be defined for multiple attributes of a table. Currently, composite indexes can be created only for B-tree in GaussDB(DWS) and a maximum of 32 columns can share a composite index.                                                                                                                                                 |

| Indexing Method  | Description                                                                                                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Partial index    | Refers to an index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.                                                                                |
| Expression index | Refers to an index that is built on a function or an expression calculated based on one or more attributes of a table. An expression index works only when the queried expression is the same as the created expression. |

- Run the following command to create an ordinary table:  

```
CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address,  
INSERT 0 0
```
- Create a common index.  
You need to query the following information in the **tpcds.customer\_address\_bak** table:  

```
SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888;
```

Generally, the database system needs to scan the **tpcds.customer\_address\_bak** table row by row to find all matched tuples. If the size of the **tpcds.customer\_address\_bak** table is large but only a few (possibly zero or one) of the WHERE conditions are met, the performance of this sequential scan is low. If the database system uses an index to maintain the **ca\_address\_sk** attribute, the database system only needs to search a few tree layers for the matched tuples. This greatly improves data query performance. Furthermore, indexes can improve the update and delete operation performance in the database.

Run the following command to create an index:

```
CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak (ca_address_sk);
```
- Create a multi-column index.  
Assume you need to frequently query records with **ca\_address\_sk** being **5050** and **ca\_street\_number** smaller than **1000** in the **tpcds.customer\_address\_bak** table. Run the following command:  

```
SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE ca_address_sk =  
5050 AND ca_street_number < 1000;
```

Run the following command to define a multiple-column index on **ca\_address\_sk** and **ca\_street\_number** columns:

```
CREATE INDEX more_column_index ON  
tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
```
- Create a partition index.  
If you only want to find records whose **ca\_address\_sk** is **5050**, you can create a partial index to facilitate your query.  

```
CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE ca_address_sk =  
5050;
```
- Create an expression index.  
Assume you need to frequently query records with **ca\_street\_number** smaller than **1000**, run the following command:  

```
SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
```

The following expression index can be created for this query task:

```
CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
```

- Delete the **tpcds.customer\_address\_bak** table.  

```
DROP TABLE tpcds.customer_address_bak;
```

## 11.10 Creating and Managing Views

### Background

If some columns in one or more tables in a database are frequently searched for, an administrator can define a view for these columns, and then users can directly access these columns in the view without entering search criteria.

A view is different from a basic table. It is only a virtual object rather than a physical one. A database only stores the definition of a view and does not store its data. The data is still stored in the original base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

### Managing a View

- Creating a view

Run the following command to create MyView:

```
CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns WHERE  
trunc(wr_refunded_cash) > 10000;
```

#### NOTE

**OR REPLACE** in **CREATE VIEW** is optional. The parameter **OR REPLACE** is specified to redefine an existing view.

- Query a view.

Query the *MyView* view. Real-time data will be returned.

```
SELECT * FROM MyView;
```

- Run the following command to query the views in the current user:

```
SELECT * FROM user_views;
```

- Run the following command to query all views:

```
SELECT * FROM dba_views;
```

- View details about a specified view.

Run the following command to view details about the *dba\_users* view:

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

- Rebuild a view.

Run the following command to rebuild a view without entering a query statement:

```
ALTER VIEW MyView REBUILD;
```

- Delete a view

Run the following command to delete MyView:

```
DROP VIEW MyView;
```

## 11.11 Creating and Managing Sequences

### Context

A sequence is a database object that generates unique integers. The values of a sequence are integers that automatically increase according to a certain rule. Sequences generate unique values because they increase automatically. This is why sequence numbers are often used as the primary keys.

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

- Set the data type of a column to sequence integer. A sequence will be automatically created by the database for this column.
- Run the CREATE SEQUENCE statement to create a sequence. Set the initial value of the **nextval('sequence\_name')** function to the default value of a column.

### Procedure

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

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

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

```
CREATE TABLE
```

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

1. Create a sequence.  

```
CREATE SEQUENCE seq1 cache 100;
```

If the following information is displayed, the sequence has been created:

```
CREATE SEQUENCE
```

2. Set the initial value of the **nextval('sequence\_name')** function to the default value of a column.

```
CREATE TABLE T2  
(  
  id int not null default nextval('seq1'),  
  name text  
);
```

If the following information is displayed, the initial value of the function has been set:

```
CREATE TABLE
```

3. Associate the sequence with a column.

Associate the sequence with a specified column in a table. The sequence will be deleted when you delete its associated field or the table where the field belongs.

```
ALTER SEQUENCE seq1 OWNED BY T2.id;
```

If the following information is displayed, the owner has been set:

```
ALTER SEQUENCE
```

#### NOTE

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

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

## Precautions

Sequence values are generated by the GTM. By default, each request for a sequence value is sent to the GTM. The GTM calculates the result of the current value plus the step and then returns the result. The GTM is the only node that can generate sequence values and probably becomes the performance bottleneck. Therefore, you are not advised to use sequences when sequence values need to be generated frequently (for example, using BulkLoad to import data). For example, the **INSERT FROM SELECT** statement has poor performance in the following scenario:

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

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

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

#### NOTE

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

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

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

## 11.12 Creating and Managing Scheduled Tasks

### Context

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

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

#### NOTE

The hybrid data warehouse (standalone) does not support scheduled tasks.

### Periodic Task Management

#### Step 1 Creates a test table.

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

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

```
CREATE TABLE
```

#### Step 2 Create the customized storage procedure.

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

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

```
CREATE PROCEDURE
```

#### Step 3 Create a task.

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

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

```
1
(1 row)
```

- Create task with specified **job\_id**.

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

#### Step 4 View the created task information about the current user.

```
select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what from
user_jobs;
job | dbname | start_date | last_date | this_date | next_date | broken |
status | interval | failures | what
-----+-----+-----+-----+-----+-----+-----+-----
1 | | 2017-07-18 11:38:03 | 2017-07-18 13:53:03.607838 | 2017-07-18 13:53:03.607838 | 2017-07-18
13:54:03 | n | s | interval '1 minute' | 0 | call public.prc_job_1();
(1 row)
```

#### Step 5 Stop a task.

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

#### Step 6 Start a task.

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

#### Step 7 Modify attributes of a task.

- Modify the **Next\_date** parameter information about a task.

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

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

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

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

```
-- Change What to the SQL statement insert into public.test values(333,
sysdate+5); for Job1.
call dbms_job.what(1,'insert into public.test values(333, sysdate+5);');
what
-----
(1 row)
```

- Modify **Next\_date**, **Interval**, and **What** parameter information of **JOB**.

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



```
(1 row)
```

**Step 8** Delete a **JOB**.

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

```
-----  
(1 row)
```

**Step 9** Set JOB permissions.

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

----End