Data Warehouse Service

Quick Start

Issue 10

Date 2020-12-15

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

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

Contents

1 Creating a Cluster and Connecting to It

1.1 Step 1: Starting Preparations

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

Before creating a data warehouse cluster, ensure that the following prerequisites are met:

- **Registering a Public Cloud Account and Authenticating It Using Your Real Name**
- **Determining the Cluster Ports**

Registering a Public Cloud Account and Authenticating It Using Your Real Name

If you do not have a public cloud account, register one. If you have an account that has passed real-name authentication, skip this step and use your existing account.

1. Open the official public cloud website (**https://www.huaweicloud.com/en-us/**) 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. Click the username in the upper right corner to enter the **Account Info** page. Then click **Real-Name Authentication** in the left navigation pane.
4. Perform real-name authentication as prompted.

   NOTE

   You must perform real-name authentication before enabling cloud services.

Determining the Cluster Ports

- When creating a data warehouse 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 the cluster is created, its port number cannot be changed. Ensure that the port specified during cluster creation is available.**

1.2 Step 2: Creating a Cluster

Before using GaussDB(DWS) to execute data analysis tasks, you need to create a data warehouse cluster. A cluster consists of multiple nodes in the same subnet. These nodes together provide services. To create a cluster, perform the following steps:

Creating a Cluster

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

Step 2 In the navigation pane on the left, choose **Cluster Management**. On the displayed page, click **Buy DWS Cluster** in the upper right corner.

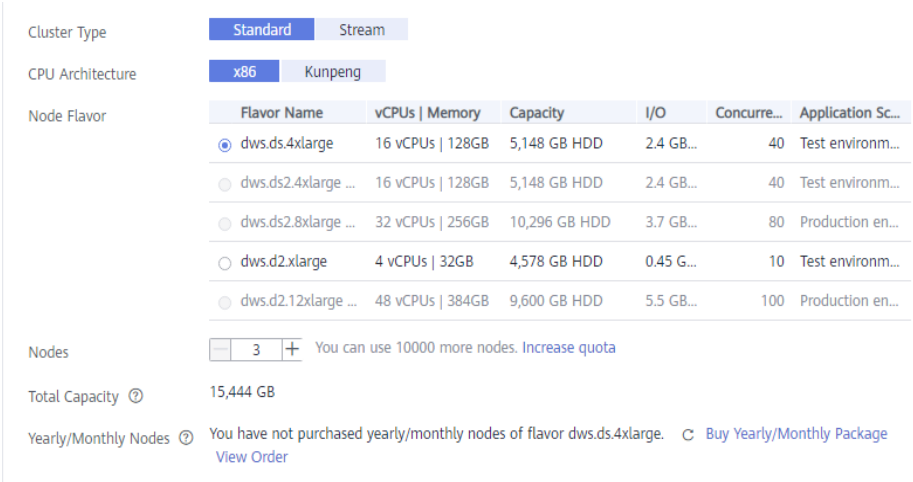
Step 3 Select the region to which the cluster to be created belongs.

- **Region**: Select **CN North-Beijing4**.
- **AZ**: Use the default value.

Step 4 Configure node-related parameters.

- **Cluster Type**: Select **Standard**.
- **CPU Architecture**: Select **x86**.
- **Node Flavor**: Retain the default value.
- **Nodes**: Retain the default value. At least **3** nodes are required.

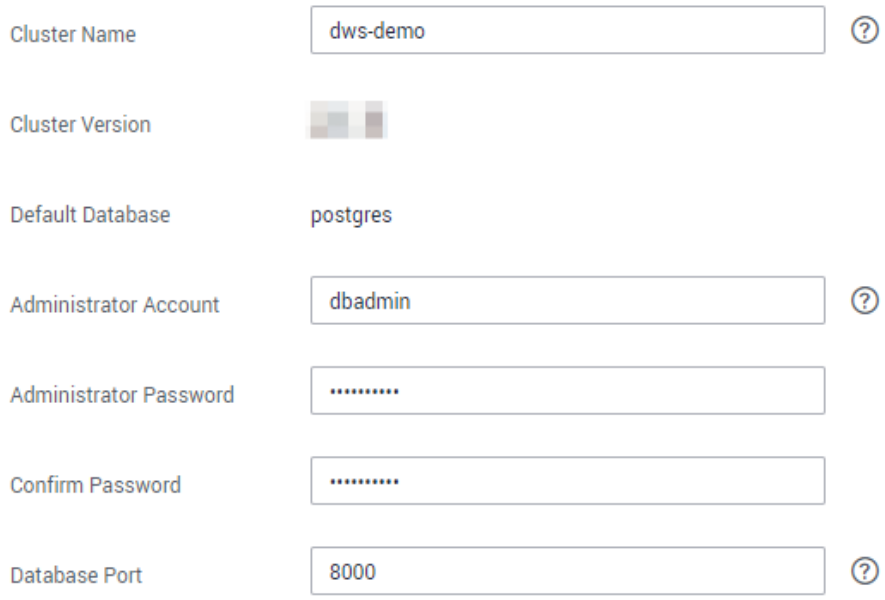
Figure 1-1 Configuring node-related parameters



Step 5 Configure cluster-related parameters.

- **Cluster Name**: Enter **dws-demo**.
- **Administrator Account**: The default value is **dbadmin**. Use the default value. After a cluster is created, the client uses this administrator account and its password to connect to the cluster's database.
- **Administrator Password**: Enter the password.
- **Confirm Password**: Enter the 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 1-2 Configuring the cluster



Step 6 Configure network parameters.

- **VPC**: You can select an existing VPC from the drop-down list. If no VPC has been configured, click **Create 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.

  For details, see the *Virtual Private Cloud User Guide*.
- **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 opens only the **Database Port** to allow access requests from clients or applications.

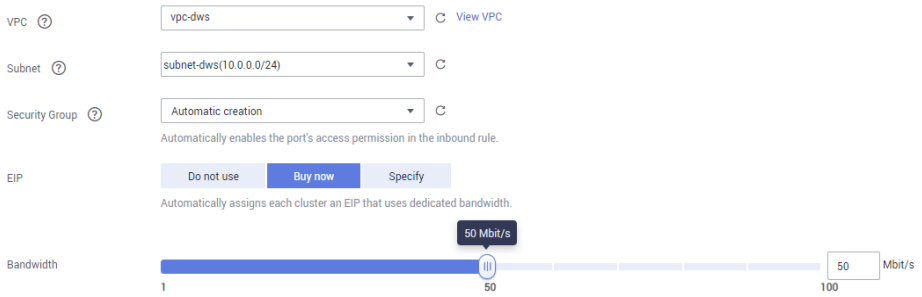
  If you select a customized security group, add an inbound rule to it to open the **Database Port** of the GaussDB(DWS) cluster for client hosts that access GaussDB(DWS). **Table 1-1** shows an example of the inbound rule. For details about how to add the inbound rule, see **Adding a Security Group Rule** in the *Virtual Private Cloud User Guide*.

Table 1-1 Inbound rule example

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

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

Figure 1-3 Network parameters



Step 7 Configure the **Enterprise Project** to which the cluster belongs. You can configure this parameter only when the Enterprise Project Management service is enabled. The default value is **default**.

An enterprise project facilitates project-level management and grouping of cloud resources and users.

You can select the default enterprise project (**default**) or other existing enterprise projects. To create an enterprise project, log in to the Enterprise Management console. For details, see the *Enterprise Management User Guide*.

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

- **Default**: Indicates that the following advanced settings use the default configurations.
  - **Automated Snapshot**: By default, the policy for automatically creating cluster snapshots is enabled.
  - **CNs**: Two CNs are deployed by default.
  - **Parameter Template**: The default database parameter template is associated with the cluster.
  - **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**: If you select this option, parameters **Automated Snapshot**, **CNs**, **Tag**, **Encrypt DataStore**, and **Parameter Template** are displayed on the page. You can customize these settings as required.

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

Step 10 Click **Submit**.

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

----End

1.3 Step 3: Connecting to a Cluster

Scenario

This section describes how to use a database client to connect to a database in a data warehouse cluster. In the following example, the Data Studio client tool is used to connect to the database in the cluster through the public network address. You can also use other SQL clients to connect to the cluster. For more connection methods, see **Connecting to Clusters** in the *Data Warehouse Service 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 section **Step 2: Creating a Cluster** to connect to the default database **postgres**.
2. **Obtaining the Public Network Address of the Cluster**: Connect to the cluster's database using the public network address.
3. **Using Data Studio to Connect to the Cluster's Database**: Download and configure the Data Studio client and connect to the cluster's database.

Obtaining the Public Network Address of the Cluster

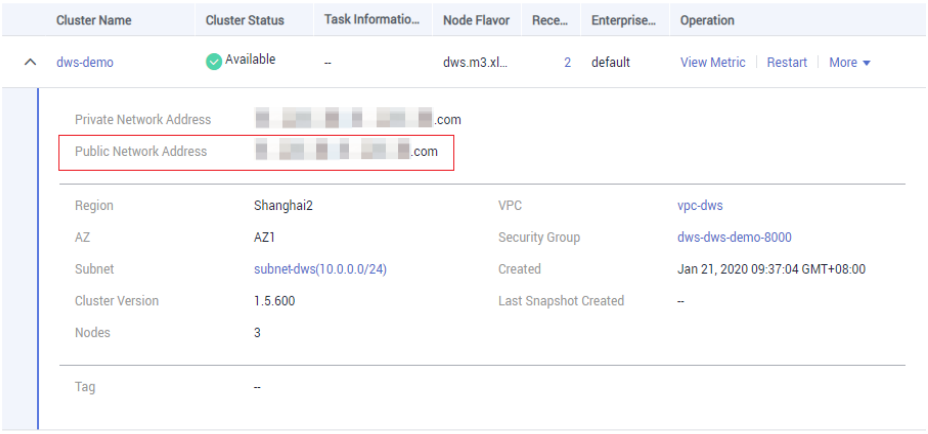
Step 1 Log in to the GaussDB(DWS) management console.

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

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. Save this information.

The public network address will be used in **Using Data Studio to Connect to the Cluster's Database**.

Figure 1-4 Cluster management



----End

Using Data Studio to Connect to the Cluster's Database

Step 1 GaussDB(DWS) provides the Data Studio GUI client running the Windows system. The tool depends on the JDK, so install Java 1.8.0_141 or a later version of JDK on the client host, but only the Java 8 version is supported.

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.

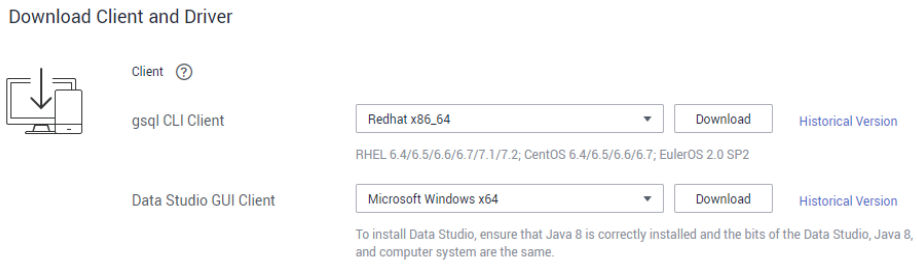
Step 3 Click **Connection Management**.

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.

Figure 1-5 Downloading clients

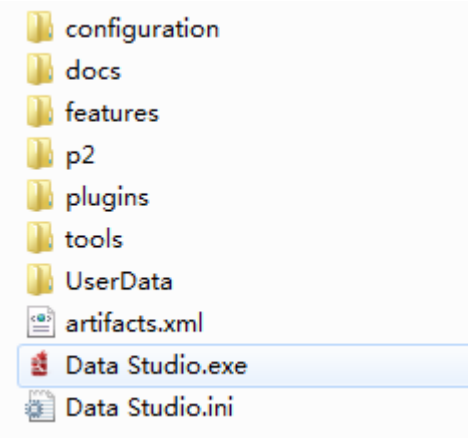


If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the corresponding client. In the cluster list on the **Cluster Management** page, click the name of the specified cluster and click the **Basic Information** tab to view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

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

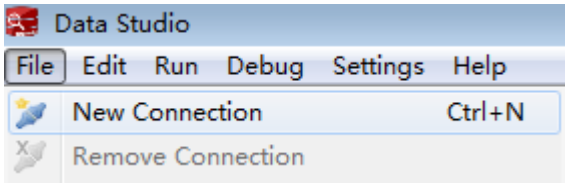
Step 6 Open the installation directory and double-click **Data Studio.exe** to start the Data Studio client. See **Figure 1-6**.

Figure 1-6 Starting the client



Step 7 Choose **File** > **New Connection** from the main menu. See **Figure 1-7**.

Figure 1-7 Creating a connection



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

Figure 1-8 Configuring connection parameters

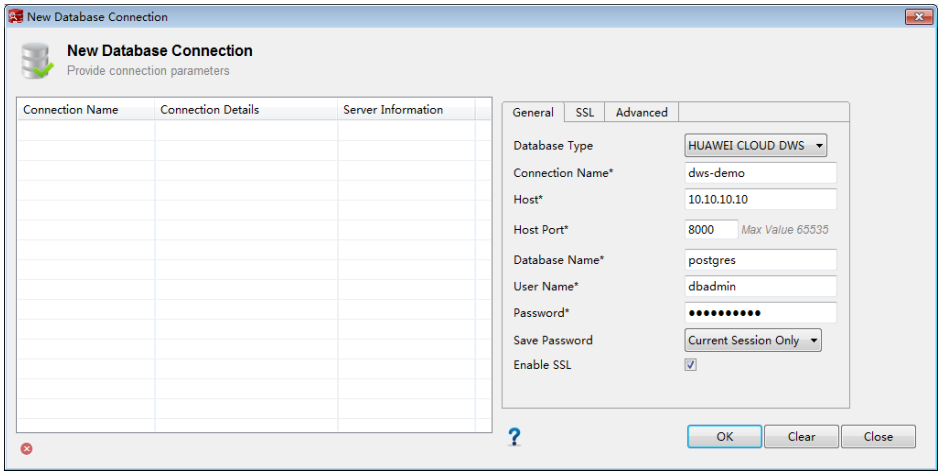


Table 1-2 Connection parameters

Parameter	Description	Example
Database Type	Select **GaussDB(DWS)**.	GaussDB(DWS)
Connection Name	Enter the connection name.	dws-demo
Host	Enter the IP address (IPv4) or domain name of the cluster to be connected.	-
Host Port	Enter the port address.	8000
Database Name	Enter the database name.	postgres
User Name	Enter the name of the user who wants to connect to the database.	-
Password	Enter the password for logging in to the database to be connected.	-
Save Password	Select an option from the drop-down list: • **Current Session Only**: The password is saved only in the current session. • **Do Not Save**: The password is not saved.	-
Enable SSL	If **Enable SSL** 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.	-

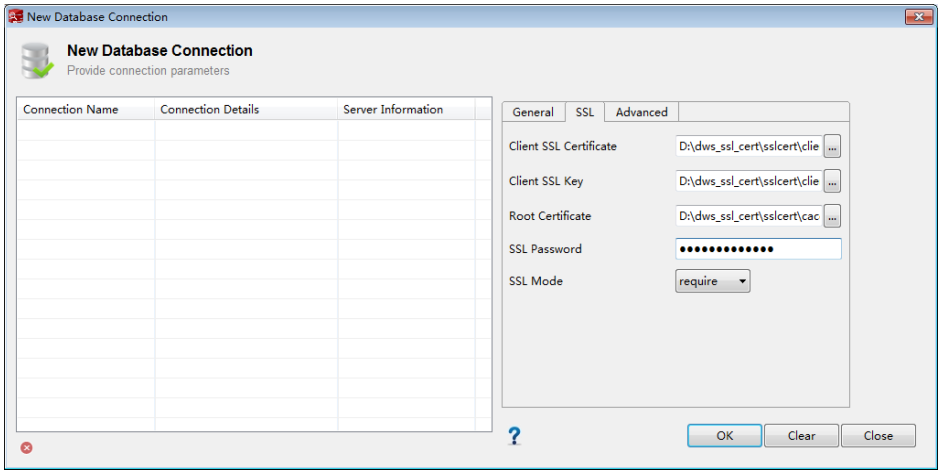
When **Enable SSL** is selected, download the SSL certificate and decompress it by referring to **Downloading the SSL Certificate**. Then, click the **SSL** tab in the window shown in **Figure 1-8** and set the following parameters:

Table 1-3 Configuring SSL parameters

Parameter	Description
Client SSL Certificate	Select the **sslcert\client.crt** file in the decompressed SSL certificate directory.

Parameter	Description
Client SSL Key	For Data Studio 18.2.0 SPC1 and later versions, the client SSL key can only be in the PK8 format. Select the **sslcert\client.key.pk8** file in the decompressed SSL certificate directory. For versions earlier than Data Studio 18.2.0 SPC1, select the **sslcert\client.key** file in the decompressed SSL certificate directory.
Root Certificate	When **SSL Mode** is set to **verify-ca**, the root certificate must be configured. Select the **sslcert\cacert.pem** file in the decompressed SSL certificate directory.
SSL Password	Set the password for the client SSL key in PK8 format. The default password is **Gauss@MppDB**.
SSL Mode	Supported SSL modes include: • require • verify-ca GaussDB(DWS) does not support the **verify-full** mode.

Figure 1-9 Configuring SSL parameters

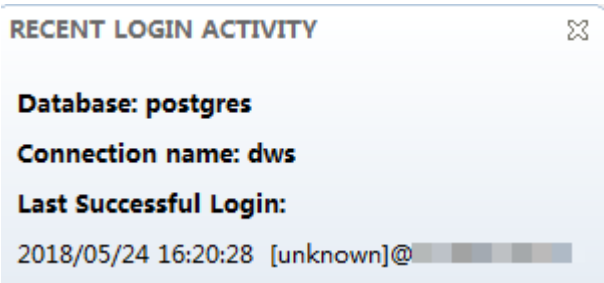


Step 9 Click **OK** to establish the database connection.

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 the Data Studio is connected to the database. You can run the SQL statement in the **SQL Terminal** window on the Data Studio page.

Figure 1-10 Successful login



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

----End

1.4 Step 4: Importing Sample Data and Performing Queries

This document provides you with three independent samples. You can select one or more to experience GaussDB(DWS) based on your requirements.

1.4.1 Analysis of Passed Vehicles at Traffic Checkpoints

This practice demonstrates the analysis of passed vehicles at traffic checkpoints. In this practice, 890 million pieces of data from traffic checkpoints are loaded to a single database table on GaussDB(DWS) for performing accurate query and fuzzy query. It is an example of high-performance query of historical data on GaussDB(DWS).

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 to access the OBS bucket.

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**

Making Preparations

- Register a HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.

Step 1: Creating a Cluster

Step 1 Log in to the HUAWEI CLOUD management console.

Step 2 Choose **Service List** > **EI Enterprise Intelligence** > **Data Warehouse Service**.

Step 3 In the navigation pane on the left, choose **Cluster Management**. On the displayed page, click **Buy DWS Cluster** in the upper right corner.

Step 4 Configure the parameters according to **Table 1-4**.

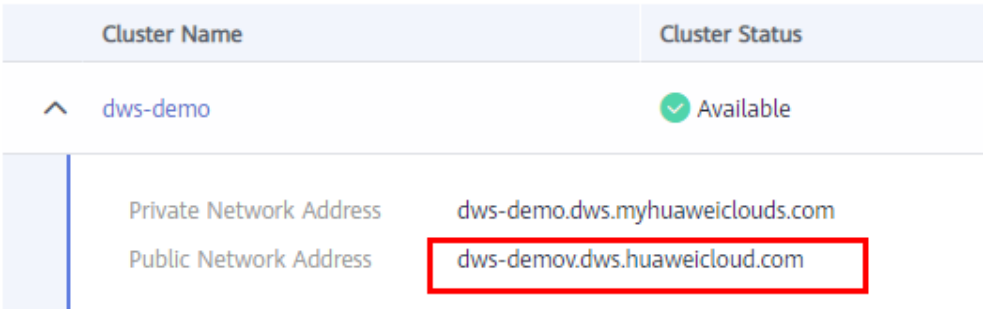
Table 1-4 Software configuration

Parameter	Configuration Method
Region	Select the **CN North-Beijing4**. **NOTE** This section uses **CN North-Beijing4** as an example. If you want to perform operations in other regions, ensure that all operations are performed in the same region.
AZ	AZ2
Cluster Type	Standard
CPU Architecture	x86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPU | 128 GB | 2000 GB SSD)
Nodes	3
Cluster Name	dws-demo
Administrator Account	dbadmin
Administrator Password	Demo@123456
Confirm Password	Demo@123456
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation

Parameter	Configuration Method
EIP	Buy now
Bandwidth	1 Mbit/s
Advanced Settings	Default

Step 5 Confirm the information, click **Buy Now**, and then click **Submit**.

Step 6 Wait about 6 minutes. After the cluster is created, click next to the cluster name. On the displayed cluster information page, record the **Public Network Address**, for example, **dws-demov.dws.huaweicloud.com**.



----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 **Connection Management** 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 Method
Database Type	GaussDB(DWS)
Column	dws-demo

Parameter	Configuration Method
Host	dws-demov.dws.huaweicloud.com The value of this parameter must be the same as the value of **Public Network Address** queried in **Step 1: Creating a Cluster**.
Host Port	8000
Database Name	postgres
User Name	dbadmin
Password	Demo@123456
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 in 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),
    cllx   VARCHAR(8),
    csys   VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by hash(hphm);
```


Step 4 Create a foreign table, which is used to identify and associate the source data on OBS. Replace *<Access_Key_Id>* and *<Secret_Access_Key>* with the actual values obtained in **Making Preparations**.

```
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://dws-demo-cn-north-4/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. **Executing Analyze**

   This statement collects statistics related to common table content in databases. The statistics are saved under the system directory PG_STATISTIC. The statistics are useful when you run the planner, which provides you with an efficient query execution plan.

   Execute the following statement to generate the statistics related to the tables:

   ```
   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**

   Execute the following statement to query driving route by 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 = 'YueD38641'
   and gcsj between '2016-01-06' and '2016-01-07'
   order by gcsj desc;
   ```

4. **Fuzzy vehicle query**

Execute the following statement to query driving route by 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 '%A23F%'
and kkbh in('508', '1125', '2120')
and gcsj between '2016-01-01' and '2016-01-07'
order by hphm,gcsj desc;
```

1.4.2 Supply Chain Requirement Analysis of a Company

This practice describes how to load the sample data set from OBS to a data warehouse cluster and perform data queries. This example comprises multi-table analysis and theme analysis in the data analysis scenario.

NOTE

In this example, a standard TPC-H-1x data set of 1 GB size has been generated on GaussDB(DWS), and has been uploaded to the **tpch** folder of an OBS bucket. All HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket. Users can easily import the data set using their accounts.

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

1. **Making Preparations**
2. **Step 1: Importing Sample Data**
3. **Step 2: Performing Multi-Table Analysis and Theme Analysis**

Scenario Description

Purpose: Understand the basic functions and data import of GaussDB (DWS) and analyze the order data of a company and its suppliers. The analysis dimensions are 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. By analyzing the relationship between parts and suppliers, you can obtain the number of suppliers that can supply parts based on 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.
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 the 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 HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.

- A cluster has been created and connected using Data Studio. For details, see **Analysis of Passed Vehicles at Traffic Checkpoints**.

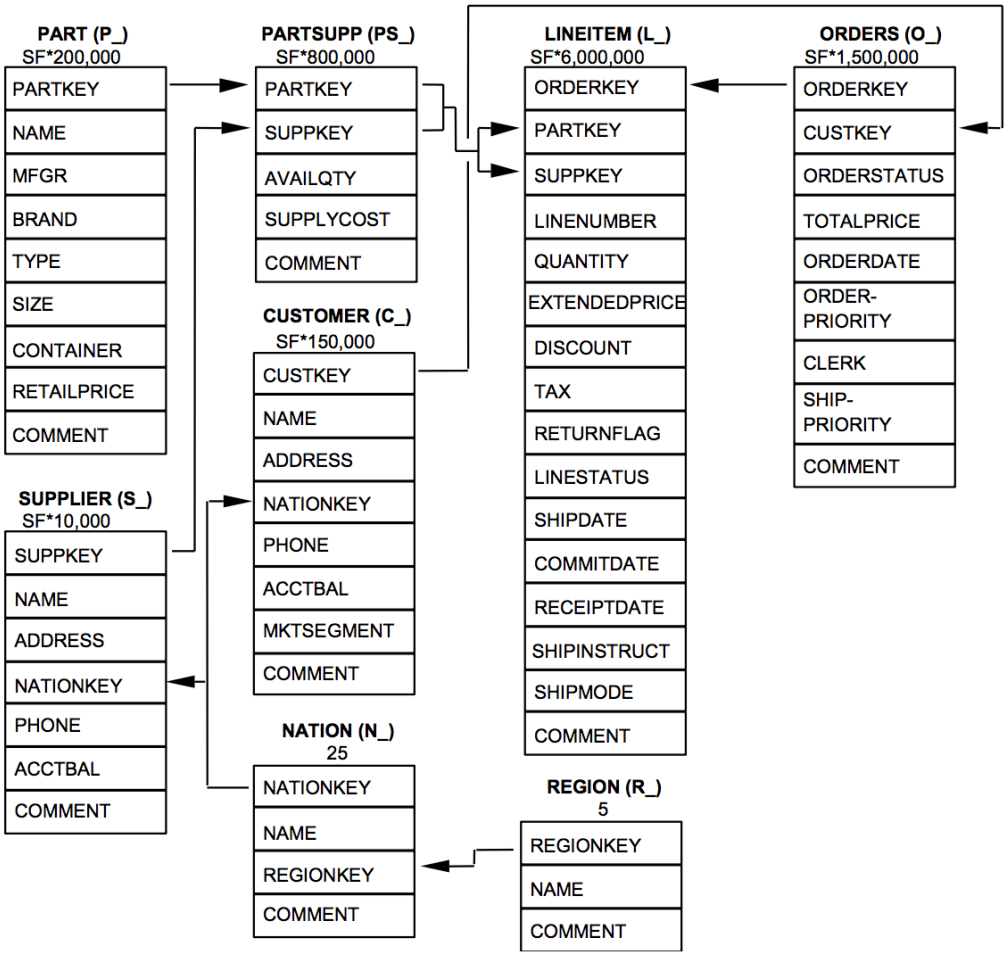
Step 1: Importing Sample Data

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 1-11**.

Figure 1-11 TPC-H data tables



Copy and execute the following table creation statements to create corresponding data tables in the **postgres** 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.

<obs_bucket_name > indicates the OBS bucket name. In this practice, **CN North-Beijing4** is used as an example. You can enter **dws-demo-cn-north-4**. Replace *<Access_Key_Id >* and *<Secret_Access_Key >* with the actual values obtained in **Making Preparations**.

```
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 connection query operations with group by, sort by, aggregate, and subquery.

  ```
  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
);
```

1.4.3 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.

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

1. **Making Preparations**
2. **Step 1: Importing Sample Data from the Retail Department Store**
3. **Step 2: Performing Operations Status Analysis**

Making Preparations

- Register a HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.
- A cluster has been created and connected using Data Studio. For details, see **Analysis of Passed Vehicles at Traffic Checkpoints**.

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;
```

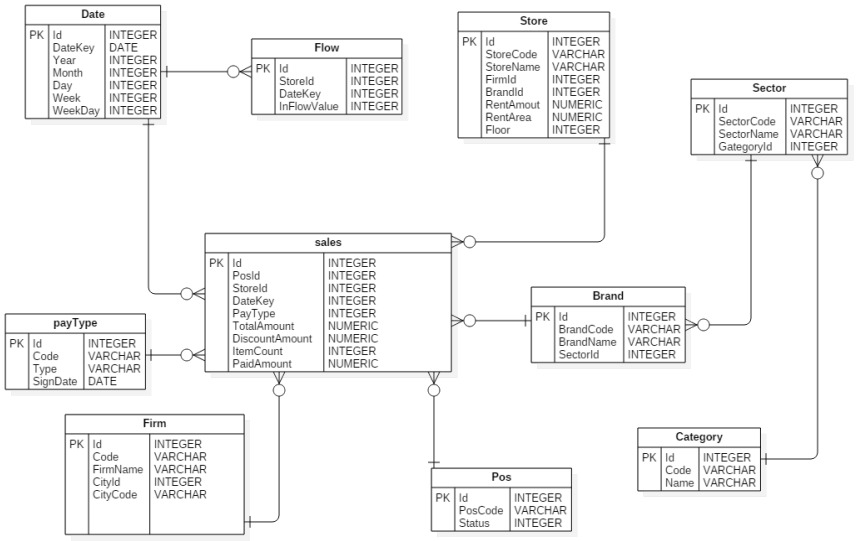
Step 2 Perform the following steps to switch to the new database:

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 1-12**.

Figure 1-12 Sample data tables of retail department stores



Copy and execute the following statements to switch to create a database table of retail department store information.

```
create schema retail_data;
set current_schema='retail_data';

DROP TABLE IF EXISTS STORE;
CREATE TABLE STORE (
	ID INT,
	STORECODE VARCHAR(10),
	STORENAME VARCHAR(100),
	FIRMID INT,
	FLOOR INT,
	BRANDID INT,
	RENTAMOUNT NUMERIC(18,2),
	RENTAREA NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS POS;
CREATE TABLE POS(
	ID INT,
	POSCODE VARCHAR(20),
	STATUS INT,
	MODIFICATIONDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS BRAND;
CREATE TABLE BRAND (
	ID INT,
	BRANDCODE VARCHAR(10),
	BRANDNAME VARCHAR(100),
	SECTORID INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SECTOR;
CREATE TABLE SECTOR(
	ID INT,
	SECTORCODE VARCHAR(10),
	SECTORNAME VARCHAR(20),
	CATEGORYID INT
)
```



```
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS CATEGORY;
CREATE TABLE CATEGORY(
	ID INT,
	CODE VARCHAR(10),
	NAME VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS FIRM;
CREATE TABLE FIRM(
	ID INT,
	CODE VARCHAR(4),
	NAME VARCHAR(40),
	CITYID INT,
	CITYNAME VARCHAR(10),
	CITYCODE VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS DATE;
CREATE TABLE DATE(
	ID INT,
	DATEKEY DATE,
	YEAR INT,
	MONTH INT,
	DAY INT,
	WEEK INT,
	WEEKDAY INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS PAYTYPE;
CREATE TABLE PAYTYPE(
	ID INT,
	CODE VARCHAR(10),
	TYPE VARCHAR(10),
	SIGNDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SALES;
CREATE TABLE SALES(
	 ID INT,
	 POSID INT,
	 STOREID INT,
	 DATEKEY INT,
	 PAYTYPE INT,
	 TOTALAMOUNT NUMERIC(18,2),
	 DISCOUNTAMOUNT NUMERIC(18,2),
	 ITEMCOUNT INT,
	 PAIDAMOUNT NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);

DROP TABLE IF EXISTS FLOW;
CREATE TABLE FLOW (
	 ID INT,
	 STOREID INT,
	 DATEKEY INT,
	 INFLOWVALUE INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);
```

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

<obs_bucket_name > indicates the OBS bucket name.In this practice, **CN North-Beijing4** is used as an example. You can enter **dws-demo-cn-north-4**. Replace *<Access_Key_Id >* and *<Secret_Access_Key >* with the actual values obtained in **Making Preparations**.

```
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 total sales revenue of each store**

  Copy and execute the following statements to query the total turnover of each store:

  ```
  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 province**

  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;
```

1.5 Step 5: Viewing Other Documents and Clearing Resources

Viewing Other Relevant Documents

After performing the preceding steps, you can refer to the following resources to learn more about DWS:

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

Clearing Resources

After performing the preceding steps, if you do not need to use the sample data, clusters, ECSs, and VPCs in the created examples, delete these resources to prevent them from wasting and occupying quotas.

Step 1 Delete a data warehouse cluster.

On the GaussDB(DWS) management console, click **Cluster Management**, 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** 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**.

----End

2 Database Quick Start

2.1 Before You Start

This chapter describes common operations in a database.

Before starting this chapter, creat a GaussDB(DWS) cluster and establish the connection between the SQL query tool and the cluster described in **Prerequisites**.

This chapter 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. If other users want to access the database, run the following statement to create a new user account with permission granted:

```
CREATE USER joe WITH PASSWORD "Bigdata@123";
```

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

```
CREATE USER
```

In this case, you have created a user named **joe** and its password is **Bigdata@123**.

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 more information, see **Users**.

Step 2 Create a database.

```
CREATE DATABASE mytpcds;
CREATE DATABASE
```

For details, 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 objects under it. When creating an object, specify the required schema using either of the following methods:

Set **search_path** of the database to the specified schema.

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

Specify a complete object name consisting of the schema and object names separated by periods (.). 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 command to query the current schema:

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

After the **mytpcds** database is created, run the following command to quit the **postgres** database:

```
\q
```

For more details about schemas, see **Creating and Managing Schemas**.

Step 4 Create a table.

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

  ```
  mytpcds=>  CREATE TABLE mytable (firstcol int);
  ```

  If the **DISTRIBUTE BY** statement is not used to specify distribution columns, 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.
  CREATE TABLE
  ```

  PG_TABLES contains information about all tables in a cluster. You can run the **SELECT** statement to view the attributes of a table in it.

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

- Run the following command to insert data to the table:

  ```
  mytpcds=> INSERT INTO mytable values (100);
  INSERT 0 1
  ```

  The **INSERT** statement is used to insert rows to the database table. For details about standard batch loading, see **About Parallel Data Import from OBS**.

- Run the following command to view data in the table:

  ```
  mytpcds=> 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 more details about schemas, see **Creating and Managing Schemas**.
- For details about table creation, see **Creating a Table**.
- In addition to the created tables, a database contains many system catalogs which contain cluster installation information and information about various queries and processes in GaussDB(DWS). You can collect information about the database by querying the system catalog. For details, see **Querying System Catalogs**.
- GaussDB(DWS) supports hybrid row-column storage, providing high query performance for interactive analysis in complex scenarios. For details about storage model selection, see **Planning a Storage Model**.

----End

Loading Sample Data

Most examples in this document are based on the TPC-DS sample table created in the **postgres** database. Before you use your SQL query tool to perform operations, create the TPC-DS sample table first 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.

To delete a cluster, follow the steps in section **Deleting a Cluster** in *Data Warehouse Service Management Guide*.

To retain the cluster but clear the **db_tpcds** database, run the following command:

```
DROP DATABASE mytpcds;
```

To retain the cluster and the database, run the following command to clear only the tables in the database:

```
DROP TABLE mytable;
```

2.2 Creating and Managing Databases

Prerequisites

Only database system administrators or users granted with database creation permissions can create a database. For details about how to grant database creation permissions to a user, see **Users**.

Background

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

- Create a database **db_tpcds**:
  ```
  CREATE DATABASE db_tpcds;
  ```
  If the following information is displayed, the database has been 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 resolve this problem, you need to:

  – Ensure that the name of the data object does not exceed the maximum length.

  – Exclude multi-byte characters from object names.

  If you fail to delete an object by specifying its name after truncation, specify its original name to delete it, or manually delete it from the system catalogs 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 a pure logical structure. For permission of a schema, see **Table 1 Default user permissions** in **Separation of Duty**.

- View a database.

  – Run the **\l** meta-command to view the database list of the database system.


```
\l
```

  - Run the following command to query the database list in the system catalog **pg_database**:

```
SELECT datname FROM pg_database;
```

- 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 command to set the default search path for the database:

```
ALTER DATABASE db_tpcds SET search_path TO pa_catalog,public;
ALTER DATABASE
```

  - Run the following command to modify the database tablespaces:

```
ALTER DATABASE db_tpcds SET TABLESPACE tpcds;
ALTER DATABASE
```

  - Run the following command to rename the database:

```
ALTER DATABASE db_tpcds RENAME TO human_tpcds;
ALTER DATABASE
```

- 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 command to delete a database:

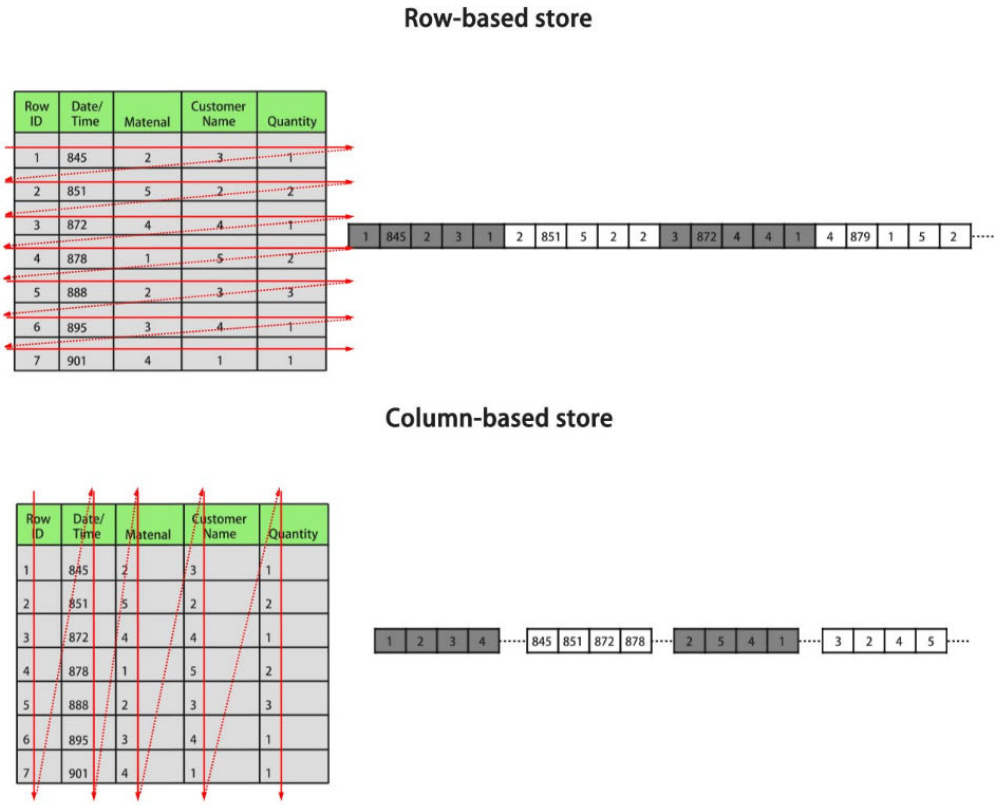
```
DROP DATABASE human_tpcds;
DROP DATABASE
```

2.3 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 2-1**.

Figure 2-1 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 **SELECT** statement is executed even if only certain columns are required.
Column storage	• Only necessary columns in a query are read. • Projections are efficient. • Any column can serve as an index.	• The selected columns need to be reconstructed after the **SELECT** statement is executed. • Data cannot be easily inserted or updated.

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	• Point queries (simple index-based queries that only return a few records). • Scenarios requiring frequent addition, deletion, and modification.
Column storage	• Statistical analysis queries (requiring a large number of association and grouping operations) • Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

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;
```

2.4 Creating and Managing Tables

2.4.1 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.

2.4.2 Inserting Data to a Table

You need to insert data to the table before using it. This section describes how to insert a row or multiple rows of data using the **INSERT** statement and to insert data from a specified table. If a large amount of data needs to be imported to the table in batches, see **Import Modes**.

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 2-1 Comparison between the outputs of character sets configured for the client and server

Procedure	Same Encoding Between Server and Client	Different Encoding Between Server and Client
No operation is performed on the character string during storage and extraction.	Output the expected result.	Output the expected result (the entered client encoding format must be the same as the displayed one).

Procedure	Same Encoding Between Server and Client	Different Encoding Between Server and Client
Some operations are performed on the character string during storage and extraction (such as operations performed on the character string function).	Output the expected result.	Unexpected results may be generated based on specific operations on character strings.
Ultra-long character strings are truncated during the storage process.	Output the expected result.	If the character encoding length in the character set is inconsistent, unexpected results may be generated.

The effect of the preceding character string function operations and automatic truncation can be overlapped. For example, if the character set of the client is different from that of the server, the character string operation is performed, and the character string is truncated, the character string is truncated after the character string is processed. As a result, unexpected results are generated. For details, see **Table 2-2**.

NOTE

Long strings are truncated only if **DBCOMPATIBILITY** is set to **TD** (compatible with Teradata) and **td_compatible_truncation** is set to **on**.

Run the following commands to create **table1** and **table2** to be used in the example:

```
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 2-2 Example

ID	Character Set of the Server	Character Set of the Client	Enable Automatic Truncation	Example	Result	Description
1	SQL_ASCII	UTF8	Yes	**INSERT INTO** *table1* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*);	id |a|b|c ----+------+------+------ 1 | 87| 87| 87	Character strings are truncated after being reversed on the server. Because the character sets of the server and client are different, character A is represented by multiple bytes on the client. As a result, the result is abnormal.
2	SQL_ASCII	UTF8	Yes	**INSERT INTO** *table1* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	Character strings are automatically truncated after being reversed. Therefore, an unexpected result is generated.

ID	Character Set of the Server	Character Set of the Client	Enable Automatic Truncation	Example	Result	Description
3	SQL_ASCII	UTF8	Yes	**INSERT INTO** *table1* **VALUES**(*3,'87A123','87A123','87A123'*	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	The field length of the character string type is an integer multiple of the character encoding length of the client. Therefore, the result is normal after the truncation.
4	SQL_ASCII	UTF8	No	**INSERT INTO** *table2* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*); **INSERT INTO** *table2* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*); 	id |a|b|c ---- +---------- --+-------- +-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	Similar to example 1, the original character is no longer represented after the multi-byte character inversion.

Procedure

Insert data to an existing table. For details about how to create a table, see **Creating and Managing Tables**.

- Insert a row to table **customer_t1**:

  Field values are arranged in the same order as they are arranged 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 fields in the table, you can obtain the same result without listing them. 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 field, the field is set to the default value. 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 for a field 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 single statement to improve efficiency.

- Assume that you have created a backup table **customer_t2** for table **customer_t1**. To insert data from **customer_t1** to **customer_t2**, run the following statements:

```
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 file.

```
DROP TABLE customer_t2 CASCADE;
```

  NOTE

  If the table to be deleted is dependent on other tables, you need to delete its dependent tables first.

2.4.3 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.

2.4.4 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**.

2.4.5 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;
```

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;
```

2.5 Loading Sample Data

This section describes how to load data to the default database **postgres** from OBS.

NOTE

Connect your SQL client to the cluster before performing operations in this section.

1. Create a table.

   Copy and run the following statements to create a table in the **postgres** 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_precentage         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 **postgres** database. The table is used for identifying data format and setting error tolerance.

```
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_precentage          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_cdemo_sk        integer                  ,
    c_current_hdemo_sk        integer                  ,
    c_current_addr_sk         integer                  ,
    c_first_shipto_date_sk    integer                  ,
    c_first_sales_date_sk     integer                  ,
    c_salutation              char(10)                 ,
    c_first_name              char(20)                 ,
    c_last_name               char(30)                 ,
    c_preferred_cust_flag     char(1)                  ,
    c_birth_day               integer                  ,
    c_birth_month             integer                  ,
    c_birth_year              integer                  ,
    c_birth_country           varchar(20)              ,
    c_login                   char(13)                 ,
    c_email_address           char(50)                 ,
    c_last_review_date        char(10)
)
SERVER gsmpp_server
OPTIONS (
location 'obs://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_cdemo_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',
chunksize '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',
chunksize '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. Set **ACCESS_KEY** and **SECRET_ACCESS_KEY** parameters as needed in the foreign table creation statement, and run this statement in a client tool to create a foreign table.

   For the values of **ACCESS_KEY** and **SECRET_ACCESS_KEY**, see **Creating Access Keys (AK and SK)** of this document.

4. Run the **INSERT** command 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;
```

5. 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;
```

6. Run the **SELECT** command 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 results based on query conditions, and group them.
SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;
```

2.6 Querying System Catalogs

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

You can view whether the table is visible to all users or only the initial user in **Overview of System Catalogs and System Views**. Log in as the initial user to query tables that are visible only to the initial user.

Querying Database Tables

For example, query the **PG_TABLES** system catalog for all tables in the **public** schema.

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

Information similar to the following is displayed:

```
    tablename
------------------
 err_hr_staffs
 test
 err_hr_staffs_ft3
 web_returns_p1
 mig_seq_table
 films4
(6 rows)
```

Viewing Database Users

You can run the **PG_USER** command to view the list of all users in the database, and view the user ID (**USESYSID**) and permissions.

```
SELECT * FROM pg_user;
             usename             | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin |
valuntil |   respool
   | parent | spacelimit | useconfig
-----------------------------------+----------+-------------+----------+-----------+---------+----------+----------
+----------+----------
----+---------+------------+-----------
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 maintenance. You can add **WHERE usesysid > 10** to the **SELECT** statement to filter queries so that only specified user names are displayed.

```
SELECT * FROM pg_user WHERE usesysid > 10;
             usename             | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin |
valuntil |   respool
   | parent | spacelimit | useconfig
-----------------------------------+----------+-------------+----------+-----------+---------+----------+----------
+----------+----------
----+---------+------------+-----------
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)
```

Viewing 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 View the running query statements. Run the following command to view the database names, user who performs queries, query status, and the corresponding PID which are connected to the running query statements:

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
 datname  | usename | 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, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

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

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

If information similar to the following is displayed, the session is successfully terminated:

```
PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```

If information similar to the following is displayed, a user 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. The message "The connection to the server was lost." is returned. Attempting reset: Succeeded."

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

----End

2.7 Creating and Managing Schemas

Context

Based on schema management, multiple users can use the same database without mutual impacts. 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.

Precautions

- 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.
- When a user is created in the initial postgres database, the system automatically creates a schema with the same name as the new user. In other databases, such a schema needs to be manually created.
- 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** is 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** is 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));
    CREATE TABLE
    ```
    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 the 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. By default, all roles have the **CREATE** and **USAGE** permissions for the **public** schema. In this way, users can connect to a specified database and create objects in the **public** schema of the database. For security purposes, you are advised to run the following statement to revoke the **CREATE** permission.

  - Run the following command to revoke **PUBLIC**'s permission to create objects in the **public** schema. **public** indicates the schema and **PUBLIC** indicates all roles.

    ```
    REVOKE CREATE ON SCHEMA public FROM PUBLIC;
    REVOKE
    ```

  - 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 'Bigdata@123';
    CREATE USER
    GRANT USAGE ON schema myschema TO jack;
    GRANT
    ```

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

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

- 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;
    DROP SCHEMA
    ```

  - 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
    DROP SCHEMA
    ```

  - Run the following command to delete user **jack**:

    ```
    DROP USER jack;
    DROP USER
    ```

2.8 Creating and Managing Partitioned Tables

Background

GaussDB(DWS) supports range 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.

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 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)
)
TABLESPACE example1
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) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

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

```
CREATE TABLE
```

  NOTE

  You are advised to create a maximum of 1000 column-based 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)
)
TABLESPACE example1
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) TABLESPACE example2
)
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 if it is not specified in the partitioned table. In this case, cross-partition update is not allowed. If **ENABLE ROW MOVEMENT** is specified, cross-partition update is allowed. However, if **SELECT FOR UPDATE** is executed concurrently to query the partitioned table, the query results may be instantaneously 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;
ALTER TABLE
```

- Delete a partition.

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

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

- 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);
ALTER TABLE
```

- 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;
ALTER TABLE
```


  - Run the following command to rename partition **P_9** to **P8**:
    ```
    ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
    ALTER TABLE
    ```

- Query a partition.

  Run the following command to query partition **P6**:
  ```
  SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
  SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
  ```

- Delete a partitioned table.
  ```
  DROP TABLE tpcds.web_returns_p2;
  DROP TABLE
  ```

2.9 Creating and Managing Indexes

Context

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.

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** with the partition name specified.
    ```
    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 2-3**.

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 2-3 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.
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 INDEX
  ```
- 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 INDEX
```

- 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 INDEX
  ```

- 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));
  CREATE INDEX
  ```

- Delete the **tpcds.customer_address_bak** table.

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

2.10 Creating and Managing Views

Context

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;
  CREATE VIEW
  ```

  NOTE

  The **OR REPLACE** parameter in this command is optional. It indicates that if the view exists, the new view will replace the existing view.

- Query a view.

  Run the following command to query MyView:

  ```
  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;
  ```
- Deleting a view

  Run the following command to delete MyView:
  ```
  DROP VIEW MyView;
  DROP VIEW
  ```

2.11 Creating and Managing Sequences

Context

A sequence is a database object that generates unique integers according to a certain rule and is usually used to generate primary key values.

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

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

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.

   In this way, the sequence will be deleted when you delete the column or the table where the column resides.

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

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

   ```
   ALTER SEQUENCE
   ```

NOTE

After the cache is specified, the sequence may have gaps (for example, the sequence numbers are 1, 4, and 5) and cannot be saved. 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 effects on nextval in the current session in any cases and take effects in other sessions only when no cache is specified for them. If cache is specified for a session, it takes effect only after all the cached values have been used. To avoid duplicate values, use setval only when necessary. Do not set it to an existing sequence value or a cached sequence value.

If BulkLoad is used, set sufficient cache for *newSeq1* and do not set **Maxvalue** or **Minvalue**. To improve the performance, database may push down the invocation

of **nextval**('*sequence_name*') to DNs. Currently, the concurrent connection requests that can be processed by the GTM are limited. If there are too many DNs, a large number of concurrent connection requests will be sent to the GTM. In this case, you need to limit the concurrent connection of BulkLoad to save the GTM connection resources. If the table is in REPLICATION mode, the invocation cannot be pushed down and the database may break down. 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**.

2.12 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.JOBS 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).

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 | postgres   | 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 catalog, 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

