Data Warehouse Service

Data Migration

Issue	01
Date	2025-09-18

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

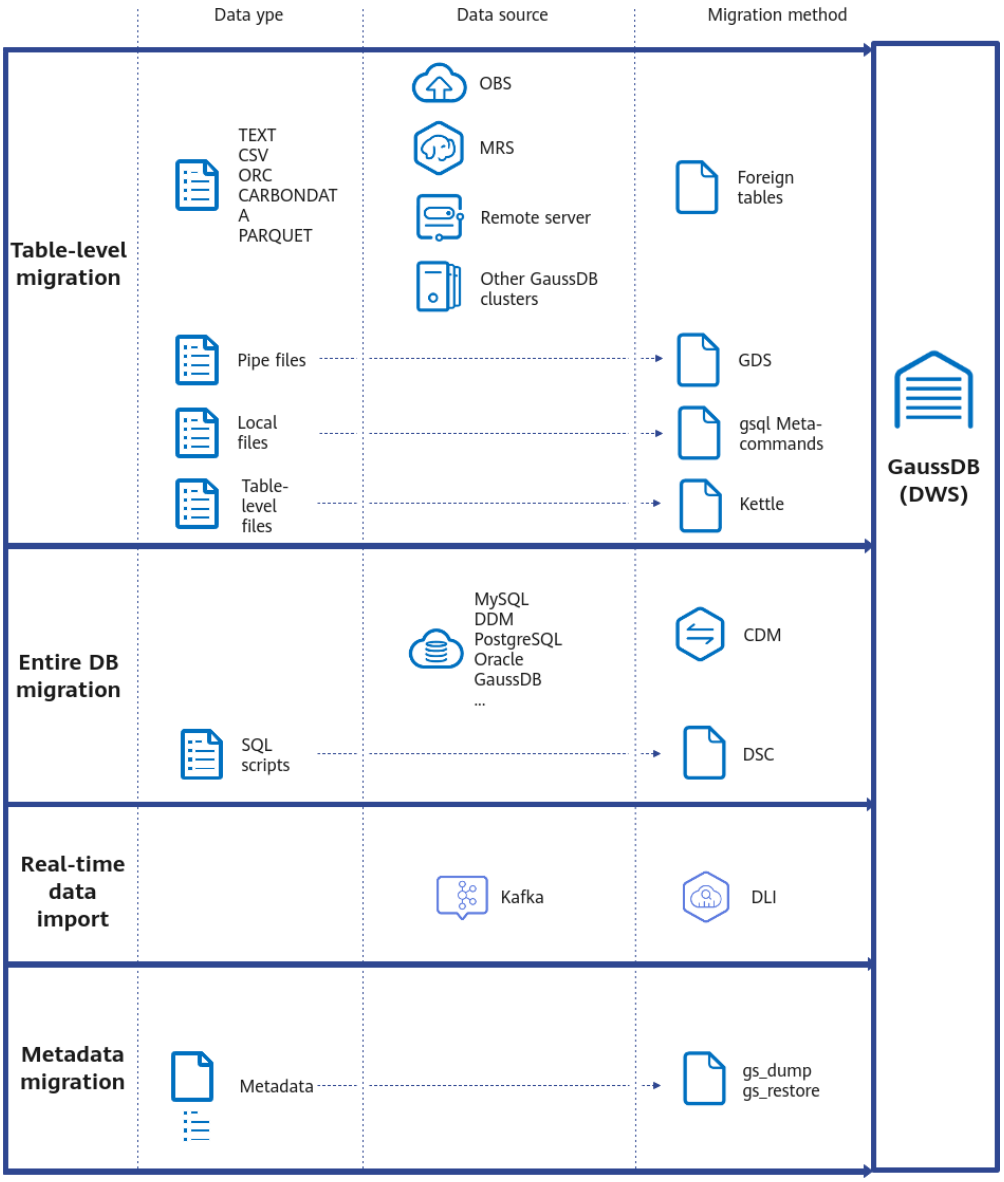
Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Data Migration to DWS

You can import data from different sources to DWS. For details, see **Figure 1-1**. The features of each method are listed in **Table 1-1**. You can select a method as required. To optimize your data migration and ETL process, we suggest using Cloud Data Migration (CDM) and DataArts Studio in combination. CDM specializes in batch data migration, while DataArts Studio offers a visually intuitive development environment and seamlessly manages the entire ETL process.

Figure 1-1 Data migration



NOTE

- CDM, OBS, DLI, and MRS are cloud services.
- GDS, DSC, and **gs_restore**, and **gs_dump** are internal tools.

Table 1-1 Import methods

Import Method	Data Source	Description	Advantage
Importing Data from OBS in Parallel	OBS	You can import data in TXT, CSV, ORC, or CarbonData format from OBS to DWS for query, and can remotely read data from OBS. It is recommended for DWS.	This method features high performance and flexible scale-out.
Using GDS to Import Data from a Remote Server	Servers (remote servers)	You can use the GDS tool provided by DWS to import data from the remote server to DWS in parallel. Multiple DNs are used for the import. This method is efficient and applicable to importing a large amount of data to the database.	
Importing Data from MRS to a Cluster	MRS (HDFS)	You can configure a DWS cluster to connect to an MRS cluster. In DWS, read data from the HDFS of MRS.	This method features high performance and flexible scale-out.
GDS-based Cross-Cluster Interconnection	-	GDS is used for data transit to implement data synchronization between multiple clusters.	This method is available for data synchronization between multiple DWS clusters.
Using a gsql Meta-Command to Import Data	Local files	Unlike the SQL **COPY** statement, the **\copy** command can read data from or write data into only local files on a **gsql** client.	This method is easy to operate and suitable for importing a small amount of data to the database.
Using COPY FROM STDIN to Import Data	Other files or databases	When you use Java to develop applications, the CopyManager interface of the JDBC driver is invoked to write data from files or other databases to DWS.	Data is directly written from other databases to DWS. Service data does not need to be stored in files.

Import Method	Data Source	Description	Advantage
Importing Kafka Data to DWS in Real Time	Kafka	You can use DLI Flink jobs to import Kafka data to DWS in real time.	This method enables Kafka real-time data import.
Using CDM to Migrate Data to DWS	Databases, NoSQL, file systems, and big data platforms	CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources. CDM migrates data to DWS using the copy method or the GDS parallel import method.	This method supports data import from abundant data sources and is easy-to-operate.
Using DSC to Migrate SQL Scripts	Databases, NoSQL, file systems, and big data platforms	For details, see the documents of the third-party ETL tool. DWS provides the DSC tool to migrate Teradata/Oracle scripts to DWS.	Provides abundant data sources and powerful data conversion capabilities through OBS.
Using gs_dump and gs_dumpall to Export Metadata	• Plain text • Custom • Directory • .tar	gs_dump exports a single database or its objects. gs_dumpall exports all databases or global objects in a cluster. To migrate database information, you can use a tool to import the exported metadata to a target database.	This method is applicable to metadata migration.
Using gs_restore to Import Data	SQL, TMP, and TAR file formats	During database migration, you can export files using **gs_dump tool** and import them to DWS by using **gs_restore**. In this way, metadata, such as table definitions and database object definitions, can be imported. The following definitions need to be imported: • All database objects • A single database object • A single schema • A single table	

2 Importing Data

2.1 Importing Data from OBS in Parallel

2.1.1 Parallel Data Import from OBS

The object storage service (OBS) is an object-based cloud storage service, featuring data storage of high security, proven reliability, and cost-effectiveness. OBS provides large storage capacity for you to store files of any type.

DWS uses OBS as a platform for converting cluster data and external data, satisfying the requirements for secure, reliable, and cost-effective storage.

You can import data in TXT, CSV, ORC, PARQUET, CARBONDATA, or JSON format from OBS to DWS for query, and can remotely read data from OBS. You are advised to import frequently accessed hot data to DWS to facilitate queries and store cold data to OBS for remote read to reduce cost.

Currently, data can be imported using either of the following methods:

- Method 1: You do not need to create a server. Use the default server to create a foreign table. Data in TXT or CSV format is supported. For details, see **Importing CSV/TXT Data from OBS**.
- Method 2: You need to create a server and use the server to create a foreign table. Data in ORC, CarbonData, text, CSV, Parquet, or JSON format is supported. For details, see **Importing ORC, CarbonData, Parquet, and JSON Data from OBS**.

NOTICE

- Ensure that no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the DWS cluster are in the same region.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.

  If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

During data migration and Extract-Transform-Load (ETL), a massive volume of data needs to be imported to DWS in parallel. The common import mode is time-consuming. When you import data in parallel using OBS foreign tables, source data files to be imported are identified based on the import URL and data formats specified in the tables. Data is imported in parallel through DNs to DWS, improving the overall import performance.

Advantages:

- The CN only plans and delivers data import tasks, and the DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to import data.
- You can preprocess data before the import.
- Fault tolerance can be configured for data format errors during the data import. You can locate incorrect data based on displayed error information after the data is imported.

Disadvantage:

You need to create OBS foreign tables and store to-be-imported data on OBS.

Application Scenario:

A large volume of local data is imported concurrently on many DNs.

Related Concepts

- **Source data file**: a TEXT, CSV, ORC, CARBONDATA, PARQUET or JSON file that stores data. to be imported in parallel.
- **OBS**: a cloud storage service used to store unstructured data, such as documents, images, and videos. Data is imported in parallel from the OBS server to DWS.
- **Bucket**: a container storing objects on OBS.
  - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
  - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the

ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.

- A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.

- **Object**: a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.

  Generally, objects are managed as files. However, OBS has no file system–related concepts, such as files and folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **Key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.

- **Key**: name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name*+*Object name*.
- **Metadata**: object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
  - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
  - User metadata contains object descriptions specified by users for uploading objects.
- **Data**: object content. OBS does not sense the content and regards it as stateless binary data.
- **Ordinary table**: A database table that stores data imported to data files in parallel. Ordinary tables are classified into row-store tables and column-store tables.
- **Foreign table**: A foreign table is used to identify data in a source data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a source data file.

How Data Is Imported

Figure 2-1 shows how data is imported from OBS. The CN plans and delivers data import tasks. It delivers tasks to each DN by file.

The delivery method is as follows:

In **Figure 2-1**, there are four DNs (DN0 to DN3) and OBS stores six files numbered from t1.data.0 to t1.data.5. The files are delivered as follows:

t1.data.0 -> DN0

t1.data.1 -> DN1

t1.data.2 -> DN2

t1.data.3 -> DN3

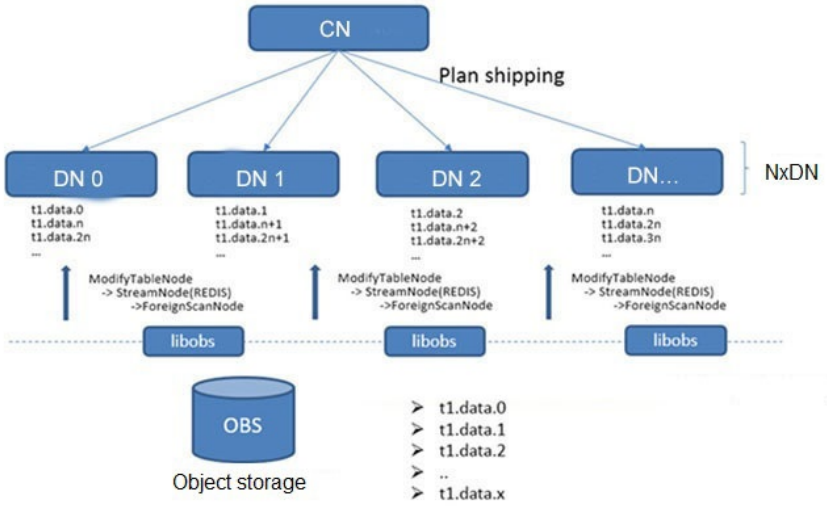
t1.data.4 -> DN0

t1.data.5 -> DN1

Two files are delivered to DN0 and DN1, respectively. One file is delivered to each of the other DNs.

The import performance is the best when one OBS file is delivered to each DN and all the files have the same size. To improve the performance of loading data from OBS, split the data file into multiple files as evenly as possible before storing it to OBS. The recommended number of split files is an integer multiple of the DN quantity.

Figure 2-1 Parallel data import using OBS foreign tables



Import Flowchart

Figure 2-2 Parallel import procedure

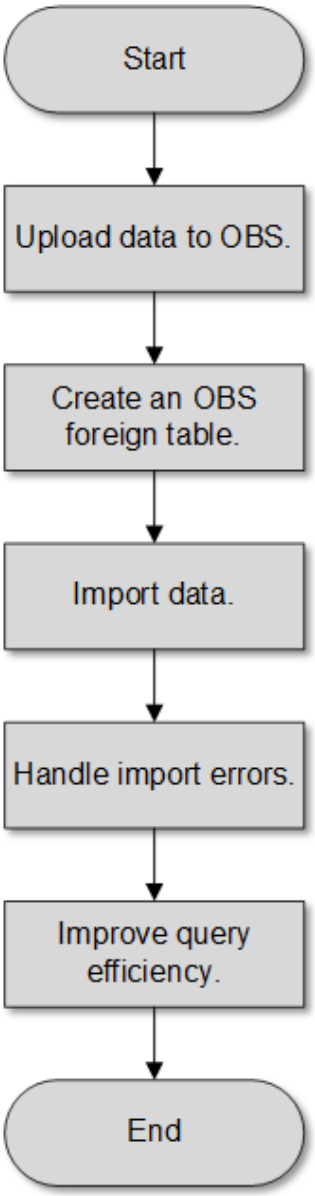


Table 2-1 Procedure description

Procedure	Description	Subtask
Upload data to OBS.	Plan the storage path on the OBS server and upload data files. For details, see **Uploading Data to OBS**.	-

Procedure	Description	Subtask
Create an OBS foreign table.	Create a foreign table to identify source data files on the OBS server. The OBS foreign table stores data source information, such as its bucket name, object name, file format, storage location, encoding format, and delimiter. For details, see **Creating an OBS Foreign Table**.	-
Import data.	After creating the foreign table, run the **INSERT** statement to efficiently import data to the target tables. For details, see **Importing Data**.	-
Handle import errors.	If errors occur during data import, handle them based on the displayed error information described in **Handling Import Errors** to ensure data integrity.	-
Improve query efficiency.	After data is imported, run the **ANALYZE** statement to generate table statistics. The **ANALYZE** statement stores the statistics in the **PG_STATISTIC** system catalog. When you run the plan generator, the statistics help you generate an efficient query execution plan.	-

2.1.2 Importing CSV/TXT Data from OBS

2.1.2.1 Creating Access Keys (AK and SK)

In this example, OBS data is imported to DWS databases. When users who have registered with the cloud platform access OBS using clients, call APIs, or SDKs, access keys (AK/SK) are required for user authentication. If you want to connect to the DWS database through a client or a JDBC/ODBC application to access OBS, obtain the access keys (AK) and secret keys (SK) first.

- Access Key ID (AK): indicates the ID of the access key, which is a unique identifier used in conjunction with a Secret Access Key to sign requests cryptographically.
- Secret Access Key (SK): indicates the key used with its associated AK to cryptographically sign requests and identify request senders to prevent requests from being modified.

Creating Access Keys (AK and SK)

Before creating an AK/SK pair, ensure that your account (used to log in to the management console) has passed real-name authentication.

To create an AK/SK pair on the management console, perform the following steps:

Step 1 Log in to the **DWS console**.

Step 2 Click the username in the upper right corner and choose **My Credentials** from the drop-down list.

Step 3 In the navigation tree on the left, click **Access Keys**.

If an access key already exists in the access key list, you can directly use it. However, you can view only **Access Key ID** in the access key list. You can download the key file containing the AK and SK only when adding an access key. If you do not have the key file, click **Create Access Key** to create one.

NOTE

- Each user can create a maximum of two valid access keys. If there are already two access keys, delete them and create one. To delete an access key, you need to enter the current login password and email address or SMS verification code. Deletion is successful only after the verification is passed.
- To ensure account security, change your access keys periodically and keep them secure.

Step 4 Click **Add Access Key**.

Step 5 In the displayed **Add Access Key** dialog box, enter the password and its verification code and click **OK**.

NOTE

- If you have not bound an email address or a mobile number, enter only the login password.
- If you have bound an email address and a mobile phone number, you can use either of them for verification.

Step 6 In the displayed **Download Access Key** dialog box, click **OK** to save the access keys to your browser's default download path.

NOTE

- Keep the access keys secure to prevent them from being leaked.
- If you click **Cancel** in the dialog box, the access keys will not be downloaded, and you cannot download them later. In this case, re-create access keys.

Step 7 Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

----End

Precautions

If you find that your AK/SK pair is abnormally used (for example, the AK/SK pair is lost or leaked) or will be no longer used, delete your AK/SK pair in the access key list or contact the administrator to reset your AK/SK pair.

When deleting the access keys, you need to enter the login password and either an email or mobile verification code.

NOTE

Deleted AK/SK pairs cannot be restored.

2.1.2.2 Uploading Data to OBS

Scenarios

Before importing data from OBS to a cluster, prepare source data files and upload these files to OBS. If the data files have been stored on OBS, you only need to complete **Step 2** to **Step 3** in **Uploading Data to OBS**.

Preparing Data Files

Prepare source data files to be uploaded to OBS. DWS supports only source data files in CSV, TEXT, ORC, or CarbonData format.

If user data cannot be saved in CSV format, store the data as any text file.

NOTE

According to **Parallel Data Import from OBS**, when the source data file contains a large volume of data, evenly split the file into multiple files before storing it to OBS. The import performance is better when the number of files is an integer multiple of the DN quantity.

Assume that you have stored the following three CSV files in OBS:

- Data file **product_info.0**

  The file contains the following data:

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

- Data file **product_info.1**

  The file contains the following data:

  ```
  200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants
  men,black,L,997,2017-09-10,301,The clothes are of good quality.
  250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress
  women,black,S,841,2017-09-15,299,Follow the store for a long time.
  108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's
  really amazing to buy.
  450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket
  women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.
  260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat
  women,red,L,2004,2017-09-15,826,Very favorite clothes.
  ```


- Data file **product_info.2**

  The file contains the following data:

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

Uploading Data to OBS

Step 1 Upload data to OBS.

Store the source data files to be imported in the OBS bucket in advance.

1. **Log in to the OBS console**.

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

2. Create a bucket.

   For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Buckets > Creating a Bucket" in the *Object Storage Service User Guide*.

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

3. Create a folder.

   For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Creating a Folder" in the *Object Storage Service User Guide*.

   For example:

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

4. Upload the files.

   For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Uploading a File" in the *Object Storage Service User Guide*.

   For example:

   - Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:

     ```
     product_info.0
     product_info.1
     ```

   - Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:

     ```
     product_info.2
     ```

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. The OBS path of the source data files is the value of the **location** parameter used for creating a foreign table.

The OBS path in the **location** parameter is in the format of **obs://***bucket_name*/*file_path*/

For example, the OBS paths are as follows:

```
obs://mybucket/input_data/product_info.0
obs://mybucket/input_data/product_info.1
obs://mybucket02/input_data/product_info.2
```

Step 3 Grant the OBS bucket read permission for the user who will import data.

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

For details, see "Console Operation Guide > Permission Control > Configuring a Bucket ACL" in *Object Storage Service User Guide*.

----End

2.1.2.3 Creating an OBS Foreign Table

Procedure

Step 1 Based on the path planned in **Uploading Data to OBS**, determine the value of the **location** parameter used for creating a foreign table.

Step 2 Obtain the keys (AK and SK) to access OBS.

To obtain the keys, log in to the **DWS console**, click the username in the upper right corner, choose **My Credentials**, and click **Access Keys** in the left navigation tree. On the displayed page, view the existing **Access Key ID** (AK). To obtain both the AK and SK, click **Create Access Key** to create and download an access key.

Step 3 Set data format parameters for the foreign table based on the formats of data to be imported. You need to collect the following source data information:

- **format**: format of the source data file in the foreign table. OBS foreign tables support CSV and TEXT formats. The default value is **TEXT**.
- **header**: Whether the data file contains a table header. Only CSV files can have headers.
- **delimiter**: Delimiter specified to separate data fields in a file. If no delimiter is specified, the default one will be used.
- For more parameters used for foreign tables, see data format parameters.

Step 4 Plan the error tolerance of parallel import to specify how errors are handled during the import.

- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.

NOTE

Valid value: **true**, **on**, **false**, and **off**.

- If this parameter is set to **true** or **on** and the last column of a data row in a source data file is lost, the column will be replaced with **null** and no error message will be generated.
- If this parameter is set to **false** or **off** and the last column of a data row in a source data file is lost, the following error information will be displayed:
  ```
  missing data for column "tt"
  ```

Default value: **false** or **off**

- **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.

  NOTE

  Value range: true/on, false/off.

  - When this parameter is **true** or **on** and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.
  - If the parameter is set to **false** or **off**, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:
    ```
    extra data after last expected column
    ```

  Default value: **false** or **off**

- **per node reject limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import fails and an error message will be reported. This parameter is optional.
- **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

  NOTE

  Valid value: **true**, **on**, **false**, and **off**.

  - When the parameter is **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
  - If the parameter is **false** or **off**, and an error occurs when there are invalid characters, the import will be interrupted.

  Default value: **false** or **off**

  The following describes the rules for converting an invalid character:

  - **\0** is converted to a space.
  - Other invalid characters are converted to question marks (?).
  - If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to adjust parameter settings to avoid import errors.
- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- For details about the parameters, see error tolerance parameters.

Step 5 Create an OBS table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

Create a foreign table in the DWS database. Parameters are described as follows:

- **Data format parameter access keys (AK and SK)**
  - Set **access_key** to the AK you have obtained.
  - Set **secret_access_key** to the SK you have obtained.

  NOTE

  The values of **access_key** and **secret_access_key** are examples only.
- **Set data format parameters.**
  - Set **format** to **CSV**.
  - Set **encoding** to **UTF-8**.
  - Configure **encrypt**. Its default value is **off**.
  - Set **delimiter** to **,**.
  - Retain the default value (double quotation marks) of **quote**.
  - Set **null** (null value in a source data file) to a null string without quotation marks.
  - Set **header** (whether the exported data file contains the header row) to the default value **false**. If the first row of the data file is not a header, retain the default value.

    NOTE

    When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**.
- **Set fault-tolerant parameters for data import.**
  - To allow all data format errors detected during data import, set the **PER NODE REJECT LIMIT** to **'unlimited'**.
  - To record data format errors detected during data import, set **LOG INTO** to **product_info_err**, which will store the errors in the **product_info_err** table.
  - When **fill_missing_fields** is set to **true** and the last column of a data row in a source data file is missing, it will be replaced with **NULL** and no error message will be displayed.
  - When **ignore_extra_data** is set to **true** and the number of columns in the source data file exceeds the number defined for the foreign table, any extra columns at the end of the row will be ignored and no error message will be displayed.

Based on the preceding settings, the foreign table is created using the following statements:

NOTICE

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

```
DROP FOREIGN TABLE product_info_ext;

CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(

LOCATION 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
fill_missing_fields 'true',
ignore_extra_data 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

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

```
CREATE FOREIGN TABLE
```

2.1.2.4 Importing Data

Background

Before importing data, you are advised to optimize your design and deployment based on the following excellent practices, helping maximize system resource utilization and improving data import performance.

- In most cases, OBS data import performance is limited by concurrent network access rate. Therefore, you are advised to deploy multiple buckets on the OBS server to import data in parallel from buckets, better utilizing DN data transfer.
- Similar to the single table import, ensure that the I/O performance is greater than the maximum network throughput in the concurrent import.
- Set the **raise_errors_if_no_files**, **partition_mem_batch**, and **partition_max_cache_size** parameters. Specify whether to distinguish "the number of imported file records is empty" and "the imported file does not exist" when data is imported, and specify the number and size of data buffers when data is imported.

- If a table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Procedure

Step 1 Create a table in the DWS database to store the data imported from the OBS.

The structure of the table must be consistent with that of the fields in the source data file. That is, the number of fields and field types must be the same. In addition, the structure of the target table must be the same as that of the foreign table. The field names can be different.

Step 2 (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Step 3 Import data.

```
INSERT INTO [Target table name] SELECT * FROM [Foreign table name]
```

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see **Handling Import Errors**.

  ```
  INSERT 0 20
  ```

- If data fails to be loaded, rectify the problem by following the instructions provided in **Handling Import Errors** and try again.

----End

Example

For example, create a table named **product_info**.

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

Run the following statement to import data from the *product_info*_**ext** foreign table to the *product_info* table:

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


2.1.2.5 Handling Import Errors

Scenarios

If you encounter an error during data import, use the instructions in this document to resolve it. Note that the error table records only data format errors.

Querying Error Information

Errors that arise during data import are categorized as either data format errors or non-data format errors.

- Data format error

  When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

  ```
  SELECT * FROM error_table_name;
  ```

  Table 2-2 lists the columns of the **error_table_name** table.

  Table 2-2 Columns of the **error_table_name** table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where an error about data format occurs
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

  If a non-data format error occurs during data import, the entire process is halted and the error is not recorded in the error table. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 2-3 Handling data import errors

Error Information	Error Type	Cause	Solution
missing data for column "r_reason_desc"	Format error	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter (|) into the value of the second column, causing loss of the value of the third column. `BE|Belgium\\|1`	1. If an error is reported due to missing columns, perform the following operations: • Add the **r_reason_desc** column to the source data file. • When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it is set to **NULL** and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.

Error Information	Error Type	Cause	Solution
extra data after last expected column	Format error	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.
invalid input syntax for type numeric: "a"	Format error	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	Non-format error	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	Non-format error	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. **INSERT INTO** reasons **SELECT DISTINCT * FROM** foreign_tpcds_reasons;
value too long for type character varying(16)	Format error	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes.

2.1.3 Importing ORC, CarbonData, Parquet, and JSON Data from OBS

2.1.3.1 Preparing Data on OBS

Scenarios

Before you use the SQL on OBS feature to query OBS data:

1. You have stored the ORC data on OBS.

   For example, the ORC table has been created when you use the Hive or Spark component, and the ORC data has been stored on OBS.

   Assume that there are two ORC data files, named **product_info.0** and **product_info.1**, whose original data is stored in the **demo.db/product_info_orc/** directory of the **mybucket** OBS bucket. You can view their original data in **Original Data**.

2. If your data files are already on OBS, perform steps in **Obtaining the OBS Path of Original Data and Setting Read Permission**.

   NOTE

   This section uses the ORC format as an example to describe how to import data. The methods for importing CarbonData, Parquet, and JSON data are similar.

Original Data

Assume that you have stored the two ORC data files on OBS and their original data is as follows:

- Data file **product_info.0**

  The file contains the following data:

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

- Data file **product_info.1**

  The file contains the following data:

  ```
  200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
  250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
  108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
  450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.
  260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes.
  ```


Obtaining the OBS Path of Original Data and Setting Read Permission

Step 1 **Log in to the OBS console**.

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

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. You need to specify the OBS paths of source data files when creating a foreign table.

For details about how to view the OBS path for storing the data source files, see "OBS Console Operation Guide > Managing Objects > Accessing an Object Using Its URL" in the *Object Storage Service User Guide*.

For example, the OBS paths are as follows:

```
https://obs.eu-west-101.myhuaweicloud.eu/mybucket/demo.db/product_info_orc/product_info.0
https://obs.eu-west-101.myhuaweicloud.eu/mybucket/demo.db/product_info_orc/product_info.1
```

Step 3 Grant the OBS bucket read permission for the user.

The user who executes the SQL on OBS function needs to obtain the read permission on the OBS bucket where the source data file is located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see "Console Operation Guide > Permission Control > Configuring a Bucket ACL" in *Object Storage Service User Guide*.

----End

2.1.3.2 Creating a Foreign Server

Create an external server to define the OBS server information for the foreign table. The creation method varies by cluster version.

- For 8.2.0 and later versions, delegate access to OBS bucket data by creating an OBS data source on the OBS console.
- For versions before 8.2.0, you will need to manually create an OBS server. For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Clusters of Version Earlier than 8.2.0: (Optional) Creating a User and a Database and Granting Foreign Table Permissions

Common users do not have permissions to create foreign servers and tables. If you want to use a common user to create foreign servers and tables in a customized database, perform the following steps to create a user and a database, and grant the user foreign table permissions.

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the gsql client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

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

Switch to the created user.

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

Run the following command to create the database demo:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit |
datlastsysoid | datfrozenxid | dattablespace | datcompatibility |                    datacl

------------+--------+----------+------------+----------+---------------+--------------+--------------+---------------
+--------------+--------------+-----------------+-------------------------------------
--------------
 template1  |     10 |        0 | C          | C        | t             | t            |           -1 |        14146 |        1351
|          1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 template0  |     10 |        0 | C          | C        | t             | f            |           -1 |        14146 |        1350
|          1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 gaussdb    |     10 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1352 |
1663 | ORA             | {=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase |  17000 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1351
|          1663 | ORA             |
(4 rows)
```

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Connect to the new database as a database administrator through the database client tool provided by DWS.

You can use the gsql client to run the following command to switch to an administrator user and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password of the system administrator as prompted.

CAUTION

You must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON SCHEMA public TO dbuser;
GRANT ALL ON FOREIGN DATA WRAPPER dfs_fdw TO dbuser;
```

where *fdw_name* can be **hdfs_fdw** or **dfs_fdw**, and **dbuser** is the name of the user who creates SERVER.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the UseFT permission.

```
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvalidbegin |
rolvaliduntil | memberof | rolreplication | rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+--------------+---------------
+---------------+----------+----------------+---------------+----------------+----------
 dbuser   | f      | t        | f           | t        | t        |         -1 |           |           | {}      | f
| f           | f           | t
 lily     | f      | t        | f           | f        | t        |         -1 |           |           | {}      | f          |
f           | f           | f
 Ruby     | t      | t        | t           | t        | t        |         -1 |           |           | {}      | t
| t           | t           | t
```

----End

Clusters of Version Earlier than 8.2.0: Creating an External Server

Step 1 Use the user who is about to create a foreign server to connect to the corresponding database.

In this example, use common user **dbuser** created in **Clusters of Version Earlier than 8.2.0: (Optional) Creating a User and a Database and Granting Foreign Table Permissions** to connect to **mydatabase** created by the user. You need to connect to the database through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  ```
  \c mydatabase dbuser;
  ```
  Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:
  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```
  Enter the password as prompted.

Step 2 Create a foreign server.

Run the commands below to create a foreign server named **obs_server**.

NOTE

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

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
  address 'obs.eu-west-101.myhuaweicloud.eu' ,
  ACCESS_KEY 'access_key_value_to_be_replaced',
  SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
  encrypt 'on',
  type 'obs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*

  You can customize a name.

  In this example, the name is set to **obs_server**.
- **FOREIGN DATA WRAPPER**

  fdw_name can be **hdfs_fdw** or **dfs_fdw**, which already exists in the database.
- **OPTIONS parameters**
  - **address**

    Specifies the endpoint of the OBS service.

    Obtain the address as follows:

    i. Obtain the OBS path by performing **Step 2** in **Preparing Data on OBS**.

    ii. The OBS path on OBS is **obs.example.com**, which is the endpoint of OBS.
  - (Mandatory) **Access keys (AK and SK)**

    DWS needs to use access keys (AK) and secret keys (SK) to access OBS. Therefore, you must obtain the keys first.

    - (Mandatory) **access_key**: specifies users' AK information.
    - (Mandatory) **secret_access_key**: specifies users' SK information.

    For details about how to obtain the access keys, see **Creating Access Keys (AK and SK)**.

– **type**

Its value is **obs**, which indicates that **dfs_fdw** connects to OBS.

Step 3 View the foreign server.

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

The server is successfully created if the returned result is as follows:

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

------------+----------+--------+---------+------------+--------
+------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
-----------
 obs_server |    24661 |  13686 |         |            |        |
{address=xxx.xxx.x.xxx,access_key=xxxxxxxxxxxxxxxxxxxx,type=obs,secret_access_key=xxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxx}
(1 row)
```

----End

2.1.3.3 Creating a Foreign Table

After performing steps in **Creating a Foreign Server**, create an OBS foreign table in the DWS database to access the data stored in OBS. An OBS foreign table is read-only. It can only be queried using **SELECT**. The operations vary according to the cluster version.

- For 8.2.0 and later versions, you can refer to the operations described in **Managing OBS Data Sources**.
- For versions earlier than 8.2.0, refer to the operations outlined in the following section.

CAUTION

If error message "permission denied for foreign server xxx" is displayed during foreign table creation, the current user does not have permissions on the foreign server. To grant the user the permissions on the foreign server, run the following command (replace *obs_server* with the foreign server name and **u1** with the current user name):

```
GRANT usage ON foreign server obs_server TO u1;
```

Creating a Foreign Table

The syntax for creating a foreign table is as follows:

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
    [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
      column_constraint [...]} ] |
      table_constraint [, ...]} [, ...] ] )
    SERVER dfs_server
    OPTIONS ( { option_name ' value ' } [, ...] )
    DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
    [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table **product_info_ext_obs**, configure the parameters in the syntax as follows.

- **table_name**

  Specifies the name of the foreign table to be created.

- **Table column definitions**
  - **column_name**: specifies the name of a column in the foreign table.
  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

  The number of fields and field types in the foreign table must be the same as those in the data stored on OBS.

- **SERVER dfs_server**

  This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign server connects to OBS to read data by setting its foreign server.

  Enter the name of the foreign server created by following steps in **Creating a Foreign Server**.

- **OPTIONS parameters**

  These are parameters associated with the foreign table. The key parameters are as follows:
  - **format**: indicates the file format on OBS. The ORC, CarbonData, and Parquet formats are supported.
  - **foldername**: This parameter is mandatory. It indicates the OBS path of the data source file. You only need to enter */Bucket name/Folder directory level/*.

    You can use **Step 2** in **Preparing Data on OBS** to obtain the complete OBS path of the data source file. The path is the endpoint of OBS.
  - **totalrows**: This parameter is optional. It does not indicate the total rows of the imported data. Because OBS may store many files, it is slow to analyze data. This parameter allows you to set an estimated value so that the optimizer can estimate the table size according to the value. Generally, query efficiency is relatively high when the estimated value is almost the same as the actual value.
  - **encoding**: encoding of data source files in foreign tables. The default value is **utf8**. This parameter is mandatory for OBS foreign tables.

- **DISTRIBUTE BY**:

  This clause is mandatory. The **ROUNDROBIN** and **REPLICATION** distribution modes are supported. By default, the **ROUNDROBIN** distribution mode is used.

  ROUNDROBIN indicates that when a foreign table reads data from the data source, each node in the DWS cluster randomly reads some data and integrates the random data to a complete data set.

  REPLICATION indicates that when a foreign table reads data from the data source, the DWS cluster selects a node to read all data. This is because each data node has complete table data.

- **Other parameters in the syntax**

Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the preceding settings, the command for creating the foreign table is as follows:

Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is **obs_server**, the file format on OBS corresponding to the table is ORC, and the data storage path on OBS is **/mybucket/demo.db/product_info_orc/**.

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
)
DISTRIBUTE BY ROUNDROBIN;
```

Create an OBS foreign table that contains partition columns. The **product_info_ext_obs** foreign table uses the **product_manufacturer** column as the partition key. The following partition directories exist in **obs/mybucket/demo.db/product_info_orc/**:

Partition directory 1: product_manufacturer=10001

Partition directory 2: product_manufacturer=10010

Partition directory 3: product_manufacturer=10086

...

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)   ,
    product_manufacturer     integer
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
```



```
)
DISTRIBUTE BY ROUNDROBIN
PARTITION BY (product_manufacturer) AUTOMAPPED;
```

2.1.3.4 Querying Data on OBS Through Foreign Tables

Viewing Data on OBS by Directly Querying the Foreign Table

If the data amount is small, you can directly run **SELECT** to query the foreign table and view the data on OBS.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM product_info_ext_obs;
```

If the query result is the same as the data in **Original Data**, the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

Step 1 Create a table in DWS to store imported data.

The target table structure must be the same as the structure of the foreign table created in **Creating a Foreign Table**. That is, both tables must have the same number of columns and column types.

For example, create a table named *product_info*. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;

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

Step 2 Run the **INSERT INTO.. SELECT ..** command to import data from the foreign table to the target table.

Example:

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

If information similar to the following is displayed, the data has been imported.


```
INSERT 0 10
```

Step 3 Run the following **SELECT** command to view data imported from OBS to DWS:

```
SELECT * FROM product_info;
```

If the query result is the same as the data in **Original Data**, the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

----End

2.1.3.5 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation. The procedure is as follows:

1. **Deleting the Foreign Table and Target Table**
2. **Deleting the Created Foreign Server**
3. **Deleting the Database and the User to Which the Database Belongs**

   If you have performed steps in **Clusters of Version Earlier than 8.2.0: (Optional) Creating a User and a Database and Granting Foreign Table Permissions**, delete the database and the user to which the database belongs.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If operations in **Querying Data After Importing It** have been performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

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

```
DROP TABLE
```

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

```
DROP FOREIGN TABLE product_info_ext_obs;
```

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

```
DROP FOREIGN TABLE
```

----End

Deleting the Created Foreign Server

Step 1 Use the user who created the foreign server to connect to the database where the foreign server is located.

In this example, common user **dbuser** is used to create the foreign server in **mydatabase**. You need to connect to the database through the database client tool provided by DWS. You can use the gsql client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

  ```
  \c mydatabase dbuser;
  ```

  Enter the password as prompted.

- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:
  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```
  Enter the password as prompted.

Step 2 Delete the created foreign server.

Run the following command to delete it:

```
DROP SERVER obs_server;
```

The database is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

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

The server is successfully deleted if the returned result is as follows:

```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

----End

Deleting the Database and the User to Which the Database Belongs

If you have performed steps in **Clusters of Version Earlier than 8.2.0: (Optional) Creating a User and a Database and Granting Foreign Table Permissions**, perform the following steps to delete the database and the user to which the database belongs.

Step 1 Delete the customized database.

Connect to the default database gaussdb through the database client tool provided by DWS.

If you have logged in to the database using the gsql client, run the following command to switch the database and user:

Switch to the default database.

```
\c gaussdb
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:

```
DROP DATABASE
```

Step 2 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:


```
\c gaussdb dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:

```
REVOKE ALL ON FOREIGN DATA WRAPPER dfs_fdw FROM dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **dfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

2.1.3.6 Supported Data Types

In the big data field, the mainstream file formats are ORC and Parquet. You can use Hive to export data to an ORC or Parquet file and use DWS to query and analyze the data in the ORC or Parquet file through a read-only foreign table. Therefore, you need to map the data types supported by the ORC or Parquet file format with the data types supported by DWS. For details, see **Table 2-4**. Similarly, DWS exports data through a write-only foreign table, and stores the data in the ORC or Parquet format. Using Hive to read the ORC or Parquet file also requires matched data types. **Table 2-5** shows the matching relationship.

Table 2-4 Mapping between ORC or Parquet read-only foreign tables and Hive data types

Type	DWS Foreign Table Type	Hive Table Type
1-byte integer	TINYINT (not recommended)	TINYINT
	SMALLINT (recommended)	TINYINT
2-byte integer	SMALLINT	SMALLINT
4-byte integer	INTEGER	INT
8-byte integer	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE
Scientific data type	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	DECIMAL (The maximum precision can reach up to 38.) (HIVE 0.11)
Date type	DATE	DATE

Type	DWS Foreign Table Type	Hive Table Type
Time type	TIMESTAMP	TIMESTAMP
Boolean type	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)
VARCHAR type	VARCHAR(n)	VARCHAR (n)
String (large text object)	TEXT(CLOB)	STRING
Binary type	BYTEA	BINARY

NOTE

Binary types are available only for clusters of version 9.1.0.100 or later.

Table 2-5 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	*precision* ≤ **38**: DECIMAL; *precision* > **38**: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)][WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	*n* ≤ **255**: CHAR(n); *n* > **255**: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	*n* ≤ **65535**: VARCHAR(n); *n* > **65535**: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
Monetary type	MONEY	NUMERIC	BIGINT

NOTICE

1. The DWS foreign table supports the NULL definition, and the Hive data table supports and uses the corresponding NULL definition.
2. The TINYINT value range in a Hive data table is [-128, 127] while in DWS it is [0, 255]. To avoid discrepancies between the read and actual values, it is recommended to use the SMALLINT type when creating a DWS read-only foreign table for TINYINT in the Hive table. Similarly, when exporting data of the TINYINT type from DWS, you are advised to use the SMALLINT type for write-only foreign tables and Hive tables.
3. The time zone definition is not supported by the date and time types of the DWS foreign table, or by the Hive table.
4. The DATE type in Hive contains only date. The DATE type in DWS contains date and time.
5. In DWS, ORC files can be compressed in ZLIB, SNAPPY, LZ4, or NONE mode. The FLOAT4 format itself is not accurate, and the sum operation results in different effect in various environments. You are advised to use the DECIMAL type in the high-precision scenarios.
6. In Teradata-compatible mode, foreign tables do not support the DATE type.
7. Binary types are available only for clusters of version 9.1.0.100 or later.

2.2 Using GDS to Import Data from a Remote Server

2.2.1 Importing Data In Parallel Using GDS

INSERT and **COPY** statements can be used only for serially importing a small volume of data. To import a large volume of data to DWS, you can use GDS to import data in parallel using a foreign table.

In the current GDS version, you can import data to databases from pipe files.

- When the local disk space of the GDS user is insufficient, HDFS data can be directly written to the pipe file without occupying extra disk space.
- To clean data before importing, you can compile a program as needed and write the data to be processed into a pipe file.

NOTE

- The current version does not support data import through GDS in SSL mode. Do not use GDS in SSL mode.
- All pipe files mentioned in this section refer to named pipes on Linux.
- To ensure the correctness of data import or export using GDS, you need to import or export data in the same compatibility mode.

  If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

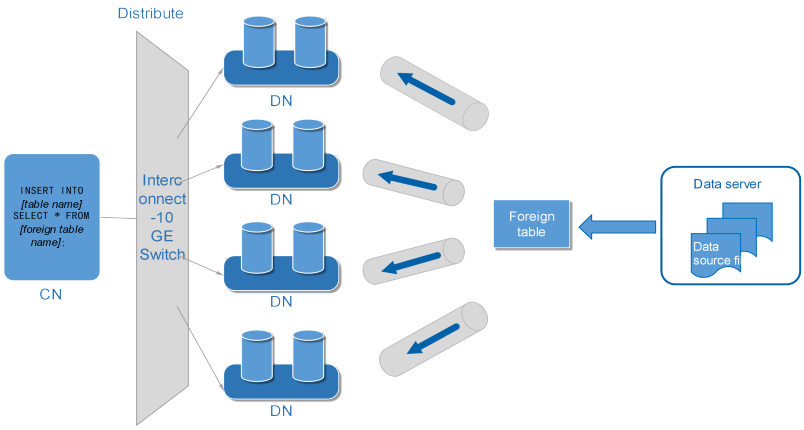
You can import data in parallel from the common file system (excluding HDFS) of a server to DWS.

Data files to be imported are specified based on the import policy and data formats set in a foreign table. Data is imported in parallel through multiple DNs from source data files to the database, which improves the overall data import performance. **Figure 2-3** shows an example.

- The CN only plans data import tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.
- In this way, the computing capacities and bandwidths of all the DNs are fully leveraged to import data, improving the import performance.

You can pre-process data (such as invalid character replacement and fault tolerance processing) by setting parameters in a foreign table.

Figure 2-3 Importing data in parallel



The concepts mentioned in the preceding figure are described as follows:

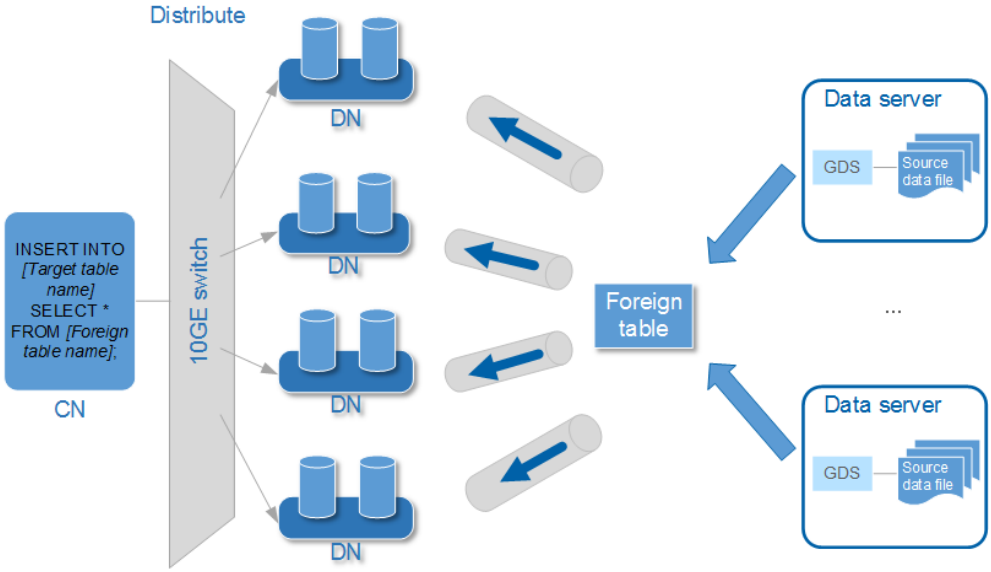
- **CN**: coordinator of DWS. After receiving import SQL requests from an application or client, the CN plans import tasks and delivers the tasks to DNs.
- **DN (Datanode)**: data node of DWS. After receiving the import tasks delivered by the CN, DNs import data from the source data file to the target table in the database through a foreign table.
- **Source data file**: a file that stores data to be imported.
- **Data server**: a server that stores source data files. For security purposes, it is recommended that the data server and DWS be on the same intranet.

- **Foreign table**: a table that stores information, such as the source location, format, destination location, encoding format, and data delimiter of a source data file. It is used to associate source data files with the target table.
- **Target table**: a table in the database. It can be a row-store table or column-store table. Data in the source data files will be imported to this table.

Parallel Import Using GDS

- If a large volume of data is stored on multiple servers, deploy, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel, as shown in **Figure 2-4**.

  Figure 2-4 Parallel import from multiple data servers



  NOTICE

  The number of GDS processes cannot exceed that of DNs. If multiple GDS processes are connected to one DN, some of the processes will probably become abnormal.

- If data is stored on data servers, and both DWS and the data servers have available I/O resources, you can use GDS for multi-thread concurrent import.

  GDS determines the number of threads based on the number of concurrent import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

Multi-thread concurrent import enables you to:

- Fully use resources and improve the concurrent import efficiency when you import multiple tables to the database.
- Speed up the import of a table with a large volume of data.

Table data is split into multiple data files, and multi-thread concurrent import is implemented by importing data using multiple foreign tables at the same time. Ensure that a data file can be read only by one foreign table.

Import Process

Figure 2-5 Concurrent import procedure of GDS

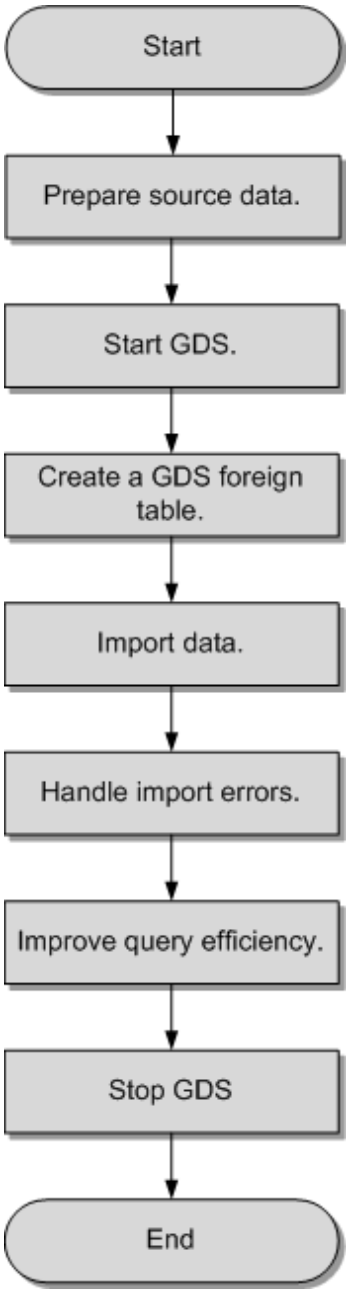


Table 2-6 Process description

Procedure	Description
Prepare source data.	Prepare the source data files to be imported to the database and upload the files to the data server. For details, see **Preparing Source Data**.
Start GDS.	Install, configure, and enable GDS on the data server. For details, see **Installing, Configuring, and Starting GDS**.
Create a foreign table.	A foreign table is used to identify source files. The foreign table stores information, such as the source location, format, destination location, encoding format, and inter-data delimiter of a source data file. For details, see **Creating a GDS Foreign Table**.
Import data.	After creating the foreign table, run the **INSERT** statement to quickly import data to the target table. For details, see **Importing Data**.
Handle the error table.	If errors occur during parallel data import, handle errors based on the error information to ensure data integrity. For details, see **Handling Import Errors**.
Improve query efficiency.	After data is imported, run the **ANALYZE** statement to generate table statistics. The **ANALYZE** statement stores the statistics in the **PG_STATISTIC** system catalog. The execution plan generator uses the statistics to generate the most efficient query execution plan.
Stop GDS.	After data import is complete, log in to each data server and stop GDS. For details, see **Stopping GDS**.

2.2.2 Preparing Source Data

Scenario

Generally, the data to be imported has been uploaded to the data server. In this case, you only need to check the communication between the data server and DWS, and record the data storage directory on the data server before the import.

If the data has not been uploaded to the data server, perform the following operations to upload it:

Procedure

Step 1 Log in to the data server as user **root**.

Step 2 Create the directory **/input_data**.

```
mkdir -p /input_data
```


Step 3 Upload the source data files to the created directory.

GDS parallel import supports source data only in CSV or TEXT format.

----End

2.2.3 Installing, Configuring, and Starting GDS

Scenario

DWS uses GDS to allocate the source data for parallel data import. Deploy GDS on the data server.

If a large volume of data is stored on multiple data servers, install, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel. The procedure for installing, configuring, and starting GDS is the same on each data server. This section describes how to perform this procedure on one data server.

Background

The GDS version must match the cluster version. For example, GDS V100R008C00 matches DWS 1.3.*X*. Otherwise, the import or export may fail, or the import or export process may fail to respond. Therefore, use the latest version of GDS.

After the database is upgraded, download the latest version of DWS GDS as instructed in **Procedure**. When the import or export starts, DWS checks the GDS versions. If the versions do not match, an error message is displayed and the import or export is terminated.

To obtain the version number of GDS, run the following command in the GDS decompression directory:

```
gds -V
```

To view the database version, run the following SQL statement after connecting to the database:

```
SELECT version();
```

Procedure

Step 1 Before using GDS to import or export data, see "Preparing an ECS as the GDS Server" and "Downloading the GDS Package" in **"Tutorial: Using GDS to Import Data from a Remote Server"**.

Step 2 Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

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

Step 3 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.***x.x*_**suse_x64.zip** to the directory created in the previous step.

Step 4 (Optional) If SSL is used, upload the SSL certificates to the directory created in **Step 2**.

Step 5 Go to the directory and decompress the package.

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

Step 6 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

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

Step 7 Change the owner of the GDS package directory and source data file directory to the GDS user.

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

Step 8 Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.*x* or earlier, skip **Step 9** and go to **Step 10**.

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

Step 9 Execute the script on which the environment depends (applicable only to version 8.1.*x*).

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

Step 10 Start GDS.

GDS is eco-friendly software that can be launched once it is decompressed. You can start GDS in two ways: by using the **gds** command to configure startup parameters or by writing the parameters into the **gds.conf** configuration file and running the **gds_ctl.py** command to initiate GDS.

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.
  - If data is transmitted in non-SSL mode, run the following command to start GDS:

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
    ```

    Example:

    ```
    /opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
    l /opt/bin/dws/gds/gds_log.txt -D -t 2
    ```

  - If data is transmitted in SSL mode, run the following command to start GDS:

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D
    -t worker_num --enable-ssl --ssl-dir Cert_file
    ```

    Example:

    Run the following command to upload the SSL certificate mentioned in **Step 4** to **/opt/bin**:

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

  Replace the information in italic as required.

  - **-d** *dir*: directory for storing data files that contain data to be imported. This tutorial uses **/input_data/** as an example.

- **-p** *ip:port*: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with DWS. The port number ranges from 1024 to 65535. The default port is **8098**. This tutorial uses **192.168.0.90:5000** as an example.
- **-H** *address_string*: specifies the hosts that are allowed to connect to and use GDS. The value must be in CIDR format. Configure this parameter to enable a DWS cluster to access GDS for data import. Ensure that the network segment covers all hosts in a DWS cluster.
- **-l** *log_file*: GDS log directory and log file name. This tutorial uses **/opt/bin/dws/gds/gds_log.txt** as an example.
- **-D**: GDS in daemon mode. This parameter is used only in Linux.
- **-t** *worker_num*: number of concurrent GDS threads. The default value is **8**. The value is a positive integer ranging between 0 and 200 (included). If the data server and DWS have available I/O resources, you can increase the number of concurrent threads.

  GDS determines the number of threads based on the number of concurrent import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.
- **--enable-ssl**: enables SSL for data transmission.
- **--ssl-dir** *Cert_file*: SSL certificate directory. Set this parameter to the certificate directory in **Step 4**.

For details about GDS parameters, see **gds**.

- Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

  a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see **Table 2-7**.

  ```
  vim /opt/bin/dws/gds/config/gds.conf
  ```

  Example:

  The **gds.conf** configuration file contains the following information:

  ```
  <?xml version="1.0"?>
  <config>
  <gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/" err_dir="/err"
  data_seg="100MB" err_seg="100MB" log_file="/log/gds_log.txt" host="10.10.0.1/24"
  daemon='true' recursive="true" parallel="32"></gds>
  </config>
  ```

  Information in the configuration file is described as follows:

  - The data server IP address is **192.168.0.90** and the GDS listening port is **5000**.
  - Data files are stored in the **/input_data/** directory.
  - Error log files are stored in the **/err** directory. The directory must be created by a user who has the GDS read and write permissions.
  - The size of a single data file is 100 MB.
  - The size of a single error log file is 100 MB.

- Logs are stored in the **/log/gds_log.txt** file. The directory must be created by a user who has the GDS read and write permissions.
- Only nodes with the IP address **10.10.0.***** can be connected.
- The GDS process is running in daemon mode.
- Recursive data file directories are used.
- The number of concurrent import threads is 2.

b. Start GDS and check whether it has been started.

```
python3 gds_ctl.py start
```

Example:

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1                  [OK]
gds [options]:
 -d dir            Set data directory.
 -p port           Set GDS listening port.
    ip:port        Set GDS listening ip address and port.
 -l log_file       Set log file.
 -H secure_ip_range
                   Set secure IP checklist in CIDR notation. Required for GDS to start.
 -e dir            Set error log directory.
 -E size           Set size of per error log segment.(0 < size < 1TB)
 -S size           Set size of data segment.(1MB < size < 100TB)
 -t worker_num     Set number of worker thread in multi-thread mode, the upper limit is 200. If
without setting, the default value is 8.
 -s status_file    Enable GDS status report.
 -D                Run the GDS as a daemon process.
 -r                Read the working directory recursively.
 -h                Display usage.
```

----End

gds.conf Parameter Description

Table 2-7 gds.conf configuration description

Attribute	Description	Value Range
name	Identifier	-
ip	Listening IP address	The IP address must be valid. Default value: **127.0.0.1**
port	Listening port	Value range: 1024 to 65535 (integer) Default value: **8098**
data_dir	Data file directory	-
err_dir	Error log file directory	Default value: data file directory
log_file	Log file Path	-

Attribute	Description	Value Range
host	Host IP address allowed to be connected to GDS (The value must in CIDR format and this parameter is available for the Linux OS only.)	-
recursive	Whether the data file directories are recursive Setting this to **true** will recursively read all files with the same name in the directory specified by **location**.	Value range: • **true**: recursive • **false**: not recursive Default value: **false**
daemon	Whether the process is running in daemon mode	Value range: • **true**: The process is running in daemon mode. • **false**: The process is not running in daemon mode. Default value: **false**
parallel	Number of concurrent data import threads	Value range: 0 to 200 (integer) Default value: **8**

2.2.4 Creating a GDS Foreign Table

The source data information and GDS access information are configured in a foreign table. Then, DWS can import data from a data server to a database table based on the configuration in the foreign table.

Procedure

Step 1 Collect source data information and GDS access information.

You need to collect the following source data information:

- **format**: format of the data to be imported. Only data in CSV, TEXT, or FIXED format can be imported using GDS in parallel.
- **header**: whether a source data file has a header. This parameter is set only for files in CSV or FIXED format.
- **delimiter**: delimiter in the source data file. For example, it can be a comma (,).
- **encoding**: encoding format of the data source file. Assume that the encoding format is UTF-8.
- **eol**: line break character in the data file. It can be a default character, such as 0x0D0A or 0X0A, or a customized line break character, such as a string: **!@#**. This parameter can be set only for TEXT import.

- For details about more source data information configured in a foreign table, see data format parameters.

You need to collect the following GDS access information:

location: GDS URL. GDS information in **Installing, Configuring, and Starting GDS** is used as an example. In non-SSL mode, **location** is set to **gsfs**://*192.168.0.90:5000*|*/input_data/*. In SSL mode, **location** is set to **gsfss**://*192.168.0.90:5000*|*/input_data/*. **192.168.0.90:5000** indicates the IP address and port number of GDS. **input_data** indicates the path of data source files managed by GDS. Replace the values as required.

Step 2 Design an error tolerance mechanism for data import.

DWS supports the following error tolerance in data import:

- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.

  NOTE

  Valid value: **true**, **on**, **false**, and **off**.

  - If this parameter is set to **true** or **on** and the last column of a data row in a source data file is lost, the column will be replaced with **null** and no error message will be generated.
  - If this parameter is set to **false** or **off** and the last column of a data row in a source data file is lost, the following error information will be displayed:
    ```
    missing data for column "tt"
    ```

  Default value: **false** or **off**

- **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.

  NOTE

  Value range: true/on, false/off.

  - When this parameter is **true** or **on** and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.
  - If the parameter is set to **false** or **off**, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:
    ```
    extra data after last expected column
    ```

  Default value: **false** or **off**

- **per node reject_limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import will fail and an error message will be reported. You can also set it to **unlimited**.
- **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

NOTE

Valid value: **true**, **on**, **false**, and **off**.

- When the parameter is **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
- If the parameter is **false** or **off**, and an error occurs when there are invalid characters, the import will be interrupted.

Default value: **false** or **off**

The following describes the rules for converting an invalid character:

- **\0** is converted to a space.
- Other invalid characters are converted to question marks (?).
- If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to adjust parameter settings to avoid import errors.

- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- **remote log 'name'**: This parameter specifies whether to store data format errors in files on the GDS server. **name** is the prefix of the error data file.
- For details about more error tolerance parameters, see error tolerance parameters.

Step 3 After connecting to the database using **gsql** or Data Studio, create a GDS foreign table based on the collected and design information.

For example:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server
 OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

The following describes information in the preceding command:

- The columns specified in the foreign table must be the same as those in the target table.
- Retain the value **gsmpp_server** for **SERVER**.
- Set **location** based on the GDS access information collected in **Step 1**. If SSL is used, replace **gsfs** with **gsfss**.
- Set **FORMAT**, **DELIMITER**, **ENCODING**, and **HEADER** based on the source data information collected in **Step 1**.

- Set **FILL_MISSING_FIELDS**, **IGNORE_EXTRA_DATA**, **LOG INTO**, and **PER NODE REJECT LIMIT** based on the error tolerance mechanism designed in **Step 2**. **LOG INTO** specifies the name of the error table.

----End

Example

For more examples, see **Example of Importing Data Using GDS**.

- Example 1: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is CSV.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 2: Create a GDS foreign table named **foreign_tpcds_reasons_SSL**. SSL is used and the data format is CSV.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons_SSL
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfss://192.168.0.90:5000/* | gsfss://192.168.0.91:5000/*',
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 3: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is TEXT.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'TEXT', delimiter E'\x08',  null '',reject_limit '2',EOL '0x0D') WITH err_foreign_tpcds_reasons;
```

- Example 4: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is FIXED.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk       integer      position(1,2),
  r_reason_id       char(16)     position(3,16),
  r_reason_desc     char(100)    position(19,100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/*', FORMAT 'FIXED', ENCODING
'utf8',FIX '119');
```

2.2.5 Importing Data

This section describes how to create tables in DWS and import data to the tables.

Before importing all the data from a table containing over 10 million records, you are advised to import some of the data, and check whether there is data skew and whether the distribution keys need to be changed (for details, see **Checking for Data Skew**). Troubleshoot the data skew if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Prerequisites

The GDS server can communicate with the DWS cluster.

- You need to create an ECS as the GDS server.
- The created ECS and DWS cluster must belong to the same region, VPC, and subnet.

Procedure

Step 1 Create a table in DWS to store imported data. For details, see CREATE TABLE.

Step 2 (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the indexes of related tables before importing data. If the data uniqueness cannot be ensured, you are not advised to delete the unique indexes. You can create indexes again after the import is complete.

1. Assume that the ordinary index **product_idx** exists in the **product_id** column of the target table **product_info**. Delete the index from the table before importing data.
   ```
   DROP INDEX product_idx;
   ```
2. After importing the data, create the **reasons_idx** index again.
   ```
   CREATE INDEX product_idx ON product_info(product_id);
   ```

Step 3 Import data.

```
INSERT INTO [Target table name] SELECT * FROM [Foreign table name]
```

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see **Handling Import Errors**.
  ```
  INSERT 0 9
  ```
- If data fails to be loaded, rectify the problem by following the instructions provided in **Handling Import Errors** and try again.

NOTE

- If a data loading error occurs, the entire data import task will fail.
- Compile a batch-processing task script to concurrently import data. The degree of parallelism (DOP) depends on the server resource usage. You can test-import several tables, monitor resource utilization, and increase or reduce concurrency accordingly. Common resource monitoring commands include **top** for monitoring memory and CPU usage, **iostat** for monitoring I/O usage, and **sar** for monitoring networks. For details about application cases, see **Data Import Using Multiple Threads**.
- If possible, more GDS servers can significantly improve the data import efficiency. For details about application cases, see **Parallel Import from Multiple Data Servers**.
- In a scenario where many GDS servers import data concurrently, you can increase the TCP Keepalive interval for connections between GDS servers and DNs to ensure connection stability. (The recommended interval is 5 minutes.) TCP Keepalive settings of the cluster affect its fault detection response time.

----End

Example:

1. Create a target table named **reasons**.
   ```
   CREATE TABLE reasons
   (
   ```



```
  r_reason_sk   integer  not null,
  r_reason_id   char(16) not null,
  r_reason_desc char(100)
)
DISTRIBUTE BY HASH (r_reason_sk);
```

2. You are advised to delete the indexes from the target table before the import.

   Assume that an ordinary table index **reasons_idx** exists on the **r_reason_id** column in the **reasons** table. Delete the index before the import. Delete the index from the table.

   ```
   DROP INDEX reasons_idx;
   ```

3. Import data from source data files through the **foreign_tpcds_reasons** foreign table to the **reasons** table.

   ```
   INSERT INTO reasons SELECT * FROM foreign_tpcds_reasons ;
   ```

4. You can create indexes again after the import is complete.

   ```
   CREATE INDEX reasons_idx ON reasons(r_reasons_id);
   ```

2.2.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that arise during data import are categorized as either data format errors or non-data format errors. The error table records only data format errors.

- Data format error

  When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

  ```
  SELECT * FROM error_table_name;
  ```

  Table 2-8 lists the columns of the *error_table_name* table.

  Table 2-8 Columns in the *error_table_name* table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where a data format error occurs If you use GDS for importing data, the error information includes the IP address and port number of the GDS server.

Column	Type	Description
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

  If a non-data format error occurs during data import, the entire process is halted. During data import, you can identify and correct errors by reviewing the displayed error information.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 2-9 Handling data import errors

Error Information	Error Type	Cause	Solution
missing data for column "r_reason_desc"	Format error	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter (|) into the value of the second column, causing loss of the value of the third column. `BE|Belgium\\|1`	1. If an error is reported due to missing columns, perform the following operations: • Add the **r_reason_desc** column to the source data file. • When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it is set to **NULL** and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.

Error Information	Error Type	Cause	Solution
extra data after last expected column	Format error	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.
invalid input syntax for type numeric: "a"	Format error	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	Non-format error	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	Non-format error	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. **INSERT INTO** reasons **SELECT DISTINCT * FROM** foreign_tpcds_reasons;

Error Information	Error Type	Cause	Solution
value too long for type character varying(16)	Format error	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes (VARCHAR2).

2.2.7 Stopping GDS

Scenarios

Stop GDS after data is imported successfully.

Procedure

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

Step 2 Select the mode of stopping GDS based on the mode of starting it.

- If GDS is started using the **gds** command, perform the following operations to stop GDS:
  a. Query the GDS process ID:
  ```
  ps -ef|grep gds
  ```
  For example, the GDS process ID is 128954.
  ```
  ps -ef|grep gds
  gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/gds_log.txt -D
  gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
  ```
  b. Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.
  ```
  kill -9 128954
  ```
- If GDS is started using the **gds_ctl.py** command, run the following commands to stop GDS:
  ```
  cd /opt/bin/dws/gds/bin
  python3 gds_ctl.py stop
  ```

----End

2.2.8 Example of Importing Data Using GDS

Parallel Import from Multiple Data Servers

The data servers and the cluster reside on the same intranet. The IP addresses are **192.168.0.90** and **192.168.0.91**. Source data files are in CSV format.

1. Log in to each GDS data server as user **root** and create the **/input_data** directory for storing data files on the servers. The following takes the data server whose IP address is **192.168.0.90** as an example. Operations on the other server are the same.
   ```
   mkdir -p /input_data
   ```
2. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group exist, skip this step.
   ```
   groupadd gdsgrp
   useradd -g gdsgrp gds_user
   ```
3. Evenly distribute CSV source data files to the **/input_data** directories on the data servers.
4. Change the owners of source data files and the **/input_data** directory on each data server to **gds_user**. The data server whose IP address is **192.168.0.90** is used as an example.
   ```
   chown -R gds_user:gdsgrp /input_data
   ```
5. Log in to each data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP addresses of the data servers are **192.168.0.90** and **192.168.0.91**. The GDS listening port is **5000**. GDS runs in daemon mode.

   Start GDS on the data server whose IP address is **192.168.0.90**.
   ```
   /opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
   ```

   Start GDS on the data server whose IP address is **192.168.0.91**.
   ```
   /opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24  -D
   ```
6. Use a tool to connect to the database.
7. Create the target table **tpcds.reasons**.
   ```
   CREATE TABLE tpcds.reasons
   (
     r_reason_sk integer not null,
     r_reason_id char(16) not null,
     r_reason_desc char(100)
   );
   ```
8. Create the foreign table **tpcds.foreign_tpcds_reasons** for receiving data from the data server.

   Data export mode settings are as follows:

   - Set the import mode to **Normal**.
   - When GDS is started, the source data file directory is **/input_data** and the GDS listening port is **5000**. Therefore, set **location** to **gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/***.

   Information about the data format is configured based on data format parameters specified during data export. The parameter configurations are as follows:

   - **format** is set to **CSV**.
   - **encoding** is set to **UTF-8**.
   - **delimiter** is set to **E'\x08'**.
   - **quote** is set to **E'\x1b'**.
   - **null** is set to an empty string without quotation marks.
   - **escape** defaults to the value of **quote**.
   - **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

Configure import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons** table.

Based on the above settings, the foreign table is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields
'false') LOG INTO err_tpcds_reasons PER NODE REJECT LIMIT 'unlimited';
```

9. Import data through the foreign table **tpcds.foreign_tpcds_reasons** to the target table **tpcds.reasons**.

   ```
   INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
   ```

10. Query data import errors in the **err_tpcds_reasons** table and rectify the errors (if any). For details, see **Handling Import Errors**.

    ```
    SELECT * FROM err_tpcds_reasons;
    ```

11. After data import is complete, log in to each data server as user **gds_user** and stop GDS.

    The data server whose IP address is **192.168.0.90** is used as an example. The GDS process ID is **128954**.

    ```
    ps -ef|grep gds
    gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D
    gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
    kill -9 128954
    ```

Data Import Using Multiple Threads

The data servers and the cluster reside on the same intranet. The server IP address is **192.168.0.90**. Source data files are in CSV format. Data will be imported to two tables using multiple threads in **Normal** mode.

1. Log in to the GDS data server as user **root**, and then create the data file directory **/input_data** and its sub-directories **/input_data/import1/** and **/input_data/import2/**.

   ```
   mkdir -p /input_data
   ```

2. Store the source data files of the target table **tpcds.reasons1** in **/input_data/import1/** and the source data files of the target table **tpcds.reasons2** in **/input_data/import2/**.
3. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.

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

4. Change the owners of source data files and the **/input_data** directory on the data server to **gds_user**.

   ```
   chown -R gds_user:gdsgrp /input_data
   ```

5. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP address of the data server is **192.168.0.90**. The GDS

listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2. A recursive directory is specified.

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24  -D -t 2 -r
```

6. Use a tool to connect to the database.
7. In the database, create the target tables **tpcds.reasons1** and **tpcds.reasons2**.

```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
CREATE TABLE tpcds.reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
```

8. In the database, create the foreign tables **tpcds.foreign_tpcds_reasons1** and **tpcds.foreign_tpcds_reasons2** for the source data.

   The foreign table **tpcds.foreign_tpcds_reasons1** is used as an example to describe how to configure parameters in a foreign table.

   Data export mode settings are as follows:

   - Set the import mode to **Normal**.
   - When GDS is started, the configured source data file directory is **/input_data** and the GDS listening port is **5000**. However, source data files are actually stored in **/input_data/import1/**. Therefore, set **location** to **gsfs://192.168.0.90:5000/import1/***.

   Information about the data format is configured based on data format parameters specified during data export. The parameter configurations are as follows:

   - **format** is set to **CSV**.
   - **encoding** is set to **UTF-8**.
   - **delimiter** is set to **E'\x08'**.
   - **quote** is set to **E'\x1b'**.
   - **null** is set to an empty string without quotation marks.
   - **escape** defaults to the value of **quote**.
   - **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

   Configure import error tolerance parameters as follows:

   - Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
   - Set **LOG INTO** to **err_tpcds_reasons1**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons1** table.
   - If the last column of a source data file is missing, the **fill_missing_fields** parameter is automatically set to **NULL**.

   Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons1
(
```



```
 r_reason_sk integer not null,
 r_reason_id char(16) not null,
 r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import1/*', format 'CSV',mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO
err_tpcds_reasons1 PER NODE REJECT LIMIT 'unlimited';
```

Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons2
(
 r_reason_sk integer not null,
 r_reason_id char(16) not null,
 r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import2/*', format 'CSV',mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO
err_tpcds_reasons2 PER NODE REJECT LIMIT 'unlimited';
```

9. Import data through the foreign table **tpcds.foreign_tpcds_reasons1** to **tpcds.reasons1** and through **tpcds.foreign_tpcds_reasons2** to **tpcds.reasons2**.

```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```

10. Query data import errors in the **err_tpcds_reasons1** and **err_tpcds_reasons2** tables and rectify the errors (if any). For details, see **Handling Import Errors**.

```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```

11. After data import is complete, log in to the data server as user **gds_user** and stop GDS.

    The GDS process ID is **128954**.

```
ps -ef|grep gds
gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D -t 2 -r
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Importing Data Through a Pipe File

Step 1 Start GDS.

```
/opt/bin/dws/gds/bin/gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

1. Log in to the database and create an internal table.

```
CREATE TABLE test_pipe_1( id integer not null, gender text not null, name  text );
```

2. Create a read-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',',  NULL '',
EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
```

3. Execute the import statement and the statement will be blocked.

```
INSERT INTO test_pipe_1 SELECT * FROM foreign_test_pipe_tr;
```

Step 3 Import data through the GDS pipes.

1. Log in to GDS and go to the GDS data directory.

```
cd /***/gds_data/
```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.

```
mkfifo foreign_test_pipe.pipe;
```


> **NOTE**
>
> A pipe will be automatically cleared after an operation is complete. To perform another operation, create a pipe file again.

3. Write data to the pipe.
   ```
   cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
   ```
4. To read the compressed file to the pipe, run the following command.
   ```
   gzip -d < out.gz > foreign_test_pipe.pipe
   ```
5. To read the HDFS file to the pipe, run the following command.
   ```
   hdfs dfs -cat - /user/hive/***/test_pipe.txt > foreign_test_pipe.pipe
   ```

Step 4 View the result returned by the import statement.

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | gender |      name
----+------+-----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----End

Importing Data Through Multi-Process Pipes

GDS also supports importing data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes importing a local file as an example.

Step 1 Start multiple GDSs. If the GDSs have been started, skip this step.

```
/opt/bin/dws/gds/bin/gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
/opt/bin/dws/gds/bin/gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log_1/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

1. Log in to the database and create an internal table.
   ```
   CREATE TABLE test_pipe( id integer not null, gender text not null, name  text );
   ```
2. Create a read-only foreign table.
   ```
   CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS
   (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe|gsfs://192.168.0.1:7790/
   foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' , file_type 'pipe',
   auto_create_pipe 'false');
   ```
3. Execute the export statement and the statement will be blocked.
   ```
   INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
   ```

Step 3 Import data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.
   ```
   cd /***/gds_data/
   cd /***/gds_data_1/
   ```
2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
   ```
   mkfifo foreign_test_pipe.pipe;
   ```
3. Read each pipe and write the new file to the pipes.


```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

Step 4 View the result returned by the import statement.

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | gender |      name
----+-----+----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----End

Direct Data Import Between Clusters

Step 1 Start the GDS. (If the process has been started, skip this step.)

```
gds -d /***/gds_data/ -D -p GDS_IP:GDS_PORT -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data from the source database.

1. Log in to the target database, create an internal table, and write data to the table.

   ```
   CREATE TABLE test_pipe( id integer not null, gender text not null, name  text );
   INSERT INTO test_pipe values(1,2,'11111111111111');
   INSERT INTO test_pipe values(2,2,'11111111111111');
   INSERT INTO test_pipe values(3,2,'11111111111111');
   INSERT INTO test_pipe values(4,2,'11111111111111');
   INSERT INTO test_pipe values(5,2,'11111111111111');
   ```

2. Create a write-only foreign table.

   ```
   CREATE FOREIGN TABLE foreign_test_pipe( id integer not null, age text not null, name  text ) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
   ```

3. Execute the import statement. The statement is blocked.

   ```
   INSERT INTO foreign_test_pipe SELECT * FROM test_pipe;
   ```

Step 3 Import data to the target cluster.

1. Create an internal table.

   ```
   CREATE TABLE test_pipe (id integer not null, gender text not null, name text);
   ```

2. Create a read-only foreign table.

   ```
   CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' , file_type 'pipe', auto_create_pipe 'false');
   ```

3. Run the following command to import data to the table.

   ```
   INSERT INTO test_pipe SELECT * FROM foreign_test_pipe;
   ```

Step 4 View the result returned by the import statement from the target cluster.

```
SELECT * FROM test_pipe;
 id | gender |      name
----+-----+----------------
  3 | 2   | 11111111111111
  6 | 2   | 11111111111111
  7 | 2   | 11111111111111
  1 | 2   | 11111111111111
  2 | 2   | 11111111111111
  4 | 2   | 11111111111111
  5 | 2   | 11111111111111
```



```
 8 | 2   | 11111111111111
 9 | 2   | 11111111111111
(9 rows)
```

----End

NOTE

By default, the pipeline file exported from or imported to GDS is named in the format of *Database name_Schema name_Foreign table name* **.pipe**. Therefore, the database name and schema name of the target cluster must be the same as those of the source cluster. If the database or schema is inconsistent, you can specify the same pipe file in the URL of the **location**.

Example:

- Pipe name specified by a write-only foreign table.
  ```
  CREATE FOREIGN TABLE foreign_test_pipe(id integer not null, age text not null, name  text)
  SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe',
  FORMAT 'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
  ```
- Pipe name specified by a read-only foreign table.
  ```
  CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS
  (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',',  NULL
  '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
  ```

2.3 Importing Data from MRS to a Cluster

2.3.1 Overview

MRS is a big data cluster running based on the open-source Hadoop ecosystem. It provides the industry's latest cutting-edge storage and analytical capabilities of massive volumes of data, satisfying your data storage and processing requirements. For details, see the **MapReduce Service User Guide**.

You can use Hive/Spark (analysis cluster of MRS) to store massive volumes of service data. Hive/Spark data files are stored on HDFS. On DWS, you can connect a data warehouse cluster to MRS clusters, read data from HDFS files, and write the data to DWS when the clusters are on the same network.

NOTICE

Ensure that MRS can communicate with DWS:

Scenario 1: If MRS and DWS are in the same region and VPC, they can communicate with each other by default.

Scenario 2: If MRS and DWS are in the same region but in different VPCs, you need to create a VPC peering connection. For details, see **VPC Peering Connection Overview**.

Scenario 3: If MRS and DWS are not in the same region, you need to use Cloud Connect (CC) to create network connections. For details, see the user guide of the corresponding service.

Scenario 4: If MRS is deployed on-premises, you need to use Direct Connect (DC) or Virtual Private Network (VPN) to connect the network. For details, see the User Guide of the corresponding service.

Importing Data from MRS to a DWS Cluster

1. **Preparing Data in an MRS Cluster**
2. (Optional) **Manually Creating a Foreign Server**
3. **Creating a Foreign Table**
4. **Importing Data**
5. **Deleting Resources**

2.3.2 Preparing Data in an MRS Cluster

Before importing data from MRS to a DWS cluster, you must have:

1. Created an MRS cluster.
2. Created a Hive/Spark ORC table in the MRS cluster and stored the table data to the HDFS path corresponding to the table.

If you have completed the preparations, skip this section.

In this tutorial, the Hive ORC table will be created in the MRS cluster as an example to complete the preparation work. The process and the SQL syntax for creating a Spark ORC table in the MRS cluster are similar to those in Hive.

Data File

The sample data of the **product_info.txt** data file is as follows:

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


Creating a Hive ORC Table in an MRS Cluster

1. Create an MRS cluster.

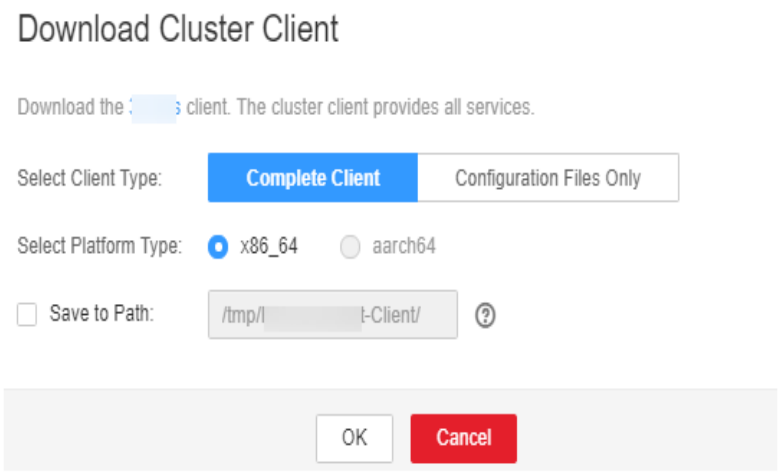
   For details, see section **Buying a Custom Cluster** in *MapReduce Service Management Guide*.

2. Download the client.

   a. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.

   b. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.

   c. Choose **Cluster** > *Name of the desired cluster* > **Dashboard**. On the page that is displayed, choose **More** > **Download Client**. The **Download Cluster Client** dialog box is displayed.



   NOTE

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

3. Log in to the Hive client of the MRS cluster.

   a. Log in to a Master node.

   For details, see **Logging In to an ECS** in the *MapReduce Service User Guide*.

b. Switch the user.
```
sudo su - omm
```

c. Run the following command to go to the client directory:
```
cd /opt/client
```

d. Run the following command to configure the environment variables:
```
source bigdata_env
```

e. If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create Hive tables.

For details, see **Creating a Role** in the *MapReduce Service User Guide*.

f. Configure roles with the corresponding permissions.

For details, see **Creating a User** in the *MapReduce Service User Guide*.

g. Bind roles to users. If the Kerberos authentication is disabled for the current cluster, skip this step.
```
kinit MRS cluster user
```
Example: **kinit hiveuser**

h. Run the following command to start the Hive client:
```
beeline
```

4. Create a database demo on Hive.

Run the following command to create the database demo:
```
CREATE DATABASE demo;
```

5. Create table **product_info** of the **Hive TEXTFILE** type in the database demo and import the **Data File** (**product_info.txt**) to the HDFS path corresponding to the table.

Run the following command to switch to the database demo:
```
USE demo;
```

Run the following command to create table **product_info** and define the table fields based on data in the **Data File**.
```
DROP TABLE product_info;

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

For details about how to import data to an MRS cluster, see section **Importing and Exporting Data** in the *MapReduce Service User Guide*.

6. Create a Hive ORC table named **product_info_orc** in the database demo.

Run the following command to create the Hive ORC table **product_info_orc**. The table fields are the same as those of the **product_info** table created in the previous step.

```
DROP TABLE product_info_orc;

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

7. Insert data in the **product_info** table to the Hive ORC table **product_info_orc**.

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

   Query table **product_info_orc**.

   ```
   SELECT * FROM product_info_orc;
   ```

   If data displayed in the **Data File** can be queried, the data has been successfully inserted to the ORC table.

2.3.3 Manually Creating a Foreign Server

In the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)** for creating a foreign table, you need to specify a foreign server associated with the MRS data source connection.

When you create an MRS data source connection on the DWS console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you want to create a foreign table in the default database **postgres** to read MRS data, skip this section.

To allow a common user to create a foreign table in a user-defined database to read MRS data, you must manually create a foreign server in the user-defined database. This section describes how does a common user create a foreign server in a user-defined database. The procedure is as follows:

1. Ensure that an MRS data source connection has been created for the DWS cluster.

   For details, see section **Creating an MRS Data Source Connection** in the Data Warehouse Service Management Guide.

2. **Creating a User and a Database and Granting the User Foreign Table Permissions**
3. **Manually Creating a Foreign Server**

> **NOTE**
>
> If you no longer need to read data from the MRS data source and have deleted the MRS data source on the DWS console, only the foreign server automatically created in the default database **postgres** will be deleted, and the manually created foreign server needs to be deleted manually. For details about how to delete a foreign server, see **Deleting the Manually Created Foreign Server**.

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

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

Switch to the created user.

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

Run the following command to create a database:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit |
datlastsysoid | datfrozenxid | dattablespace | datcompatibility |                    datacl

------------+--------+----------+------------+----------+---------------+--------------+--------------+---------------
+--------------+---------------+-----------------+-------------------------------------
--------------
 template1  |     10 |        0 | C          | C        | t             | t            |           -1 |        14146 |        1351
|         1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 template0  |     10 |        0 | C          | C        | t             | f            |           -1 |        14146 |        1350
|         1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 gaussdb    |     10 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1352 |
1663 | ORA             | {=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase |  17000 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1351
|         1663 | ORA             |
(4 rows)
```

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password as prompted.

NOTE

Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

```
 rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvalidbegin |
rolvaliduntil | memberof | rolreplication | rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+--------------+---------------
+---------------+----------+----------------+---------------+----------------+----------
 dbuser    | f        | t          | f             | t           | t           |           -1 |               |
| {}       | f
| f             | f             | t
 lily      | f        | t          | f             | f           | t           |           -1 |               |
| {}       | f             |
f             | f             | f
 Ruby      | t        | t          | t             | t           | t           |           -1 |               |
| {}       | t
| t             | t             | t
```

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  ```
  \c postgres dbadmin;
  ```
  Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:
  ```
  gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
  ```

Step 2 Run the following command to query the information about the foreign server that is automatically created:

```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                      srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                   |       10 |  13673 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(2 rows)
```

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in **Creating a User and a Database and Granting the User Foreign Table Permissions** to connect to **mydatabase** created by the user:

```
\c mydatabase dbuser;
```

Step 4 Create a foreign server.

For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*. For example:

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692cahdfs_server FOREIGN DATA
WRAPPER HDFS_FDW
OPTIONS
(
address '192.168.1.245:25000,192.168.1.218:25000',
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*

  You can customize a name.

  In this example, specify the name to the value of the **srvname** field recorded in **Step 2**, such as *hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*.

  Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.

- **FOREIGN DATA WRAPPER**

  This parameter can only be set to **HDFS_FDW**, which already exists in the database.

- **OPTIONS** parameters

  Set the following parameters to the values under **srvoptions** recorded in **Step 2**.

  – address

  Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.

  – hdfscfgpath

  Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.

  – type

  Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

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

The server is successfully created if the returned result is as follows:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                       srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |        |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----End

2.3.4 Creating a Foreign Table

This section describes how to create a Hadoop foreign table in the DWS database to access the Hadoop structured data stored on MRS HDFS. A Hadoop foreign table is read-only. It can only be queried using **SELECT**.

Prerequisites

- You have created an MRS cluster and imported data to the ORC table in the Hive/Spark database.

  For details, see **Preparing Data in an MRS Cluster**.

- You have created an MRS data source connection for the DWS cluster.

  For details, see section **Creating an MRS Data Source Connection** in the Data Warehouse Service Management Guide.

Obtaining the HDFS Path of the MRS Data Source

There are two methods for you to obtain the HDFS path.

- **Method 1**

  For Hive data, log in to the Hive client of MRS (see 2), run the following command to view the detailed information about the table, and record the data storage path in the **location** parameter:

  ```
  use <database_name>;
  desc formatted <table_name>;
  ```

  For example, if the value of the **location** parameter in the returned result is **hdfs://hacluster/user/hive/warehouse/demo.db/product_info_orc/**, the HDFS path is **/user/hive/warehouse/demo.db/product_info_orc/**.

- **Method 2**

  Perform the following steps to obtain the HDFS path:

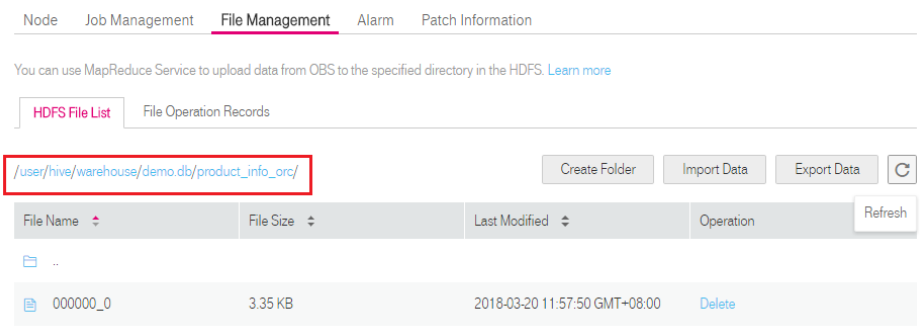
  a. Log in to the **MRS console**.

  b. Choose **Cluster** > **Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.

  c. Click **File Management** and select **HDFS File List**.

  d. Go to the storage directory of the data to be imported to the DWS cluster and record the path.

  Figure 2-6 Checking the data storage path on MRS

Obtaining Information About the Foreign Server Connected to the MRS Data Source

Step 1 Use the user who creates the foreign server to connect to the corresponding database.

Determine whether to use a common user to create a foreign table in the customized database based on requirements.

- **Yes**

  a. Ensure that you have created the common user **dbuser** and its database **mydatabase**, and manually created a foreign server in **mydatabase** by following steps in **Manually Creating a Foreign Server**.

  b. Connect to the database **mydatabase** as user **dbuser** through the database client tool provided by DWS.

If you have connected to the database using the gsql client, run the following command to switch the user and database:

```
\c mydatabase dbuser;
```

Enter your password as prompted.

- **No**

  When you create an MRS data source connection on the DWS console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you create a foreign table in the default database **postgres** as the database administrator **dbadmin**, you need to connect to the database using the database client tool provided by DWS. For example, use the **gsql** client to connect to the database by running the following command:

  ```
  gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
  ```

Step 2 Run the following command to view the information about the created foreign server connected to the MRS data source:

```
SELECT * FROM pg_foreign_server;
```

NOTE

You can also run the **\desc+** command to view the information about the foreign server.

The returned result is as follows:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                srvoptions
----------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                       |       10 |  13673 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(2 rows)
```

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

----End

Creating a Foreign Table

After **Obtaining Information About the Foreign Server Connected to the MRS Data Source** and **Obtaining the HDFS Path of the MRS Data Source** are completed, you can create a foreign table to read data from the MRS data source.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.


```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
   [ { [CONSTRAINT constraint_name] NULL |
   [CONSTRAINT constraint_name] NOT NULL |
     column_constraint [...]} ] |
     table_constraint [, ...]} [, ...] ] )
   SERVER dfs_server
   OPTIONS ( { option_name ' value ' } [, ...] )
   DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
   [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *foreign_product_info*, set parameters in the syntax as follows:

- **table_name**

  Mandatory. This parameter specifies the name of the foreign table to be created.

- Table column definitions
  - **column_name**: specifies the name of a column in the foreign table.
  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

  The number of columns and column types in the foreign table must be the same as those in the data stored on MRS. Learn **Data Type Conversion** before defining column data types.

- **SERVER dfs_server**

  This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign table can read data from an MRS cluster by configuring the foreign server and connecting to the MRS data source.

  Enter the value of the **srvname** field queried in **Obtaining Information About the Foreign Server Connected to the MRS Data Source**.

- **OPTIONS** parameters

  These are parameters associated with the foreign table. The key parameters are as follows:
  - **format**: This parameter is mandatory. The value can only be **orc**. It specifies the format of the source data file. Only Hive ORC files are supported.
  - **foldername**: This parameter is mandatory. It specifies the HDFS directory for storing data or data file path.

    If the MRS analysis cluster has enabled Kerberos authentication, ensure that the MRS user having the MRS data source connection has the read and write permissions for the directory.

    Follow the steps in **Obtaining the HDFS Path of the MRS Data Source** to obtain the HDFS path, which is the value of parameter **foldername**.
  - **encoding**: This parameter is optional. It specifies the encoding format of a source data file in the foreign table. Its default value is **utf8**.
  - **DISTRIBUTE BY**

    This parameter specifies the data read mode for the foreign table. There are two read modes supported. In this example, **ROUNDROBIN** is selected.

- **ROUNDROBIN**: When a foreign table reads data from the data source, each node in a DWS cluster randomly reads some data and integrates the random data to a complete data set.
- **REPLICATION**: When a foreign table reads data from the data source, each node in the DWS cluster reads a complete data set.

– Other parameters in the syntax

Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the above settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

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

Data Type Conversion

Data is imported to Hive/Spark and then stored on HDFS in ORC format. Actually, DWS reads ORC files on HDFS, and queries and analyzes data in these files.

Data types supported by Hive/Spark are different from those supported by DWS. Therefore, you need to learn the mapping between them. **Table 2-10** describes the mapping in detail.

Table 2-10 Data type mapping

Type	Column Type Supported by an HDFS/OBS Foreign Table of DWS	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in two bytes	SMALLINT	SMALLINT	SMALLINT
Integer in four bytes	INTEGER	INT	INT

Type	Column Type Supported by an HDFS/OBS Foreign Table of DWS	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in eight bytes	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE	FLOAT
Scientific data type	DECIMAL[p (,s)] The maximum precision can reach up to 38.	DECIMAL The maximum precision can reach up to 38 (Hive 0.11).	DECIMAL
Date type	DATE	DATE	DATE
Time type	TIMESTAMP	TIMESTAMP	TIMESTAMP
BOOLEAN type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)	STRING
VARCHAR type	VARCHAR(n)	VARCHAR (n)	VARCHAR (n)
String	TEXT(CLOB)	STRING	STRING

2.3.5 Importing Data

Viewing Data in the MRS Data Source by Directly Querying the Foreign Table

If the data amount is small, you can directly run SELECT to query the foreign table and view the data in the MRS data source.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM foreign_product_info;
```

If the query result is the same as the data in **Data File**, the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```


After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

You can query the MRS data after importing it to DWS.

Step 1 Create a table in DWS to store imported data.

The target table structure must be the same as the structure of the foreign table created in **Creating a Foreign Table**. That is, both tables must have the same number of columns and column types.

For example, create a table named **product_info**. The table example is as follows:

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

Step 2 Run the **INSERT INTO .. SELECT ..** command to import data from the foreign table to the target table.

Example:

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

If information similar to the following is displayed, the data has been imported.

```
INSERT 0 20
```

Step 3 Run the following **SELECT** command to view data imported from MRS to DWS:

```
SELECT * FROM product_info;
```

If the query result is the same as the data in **Data File**, the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

----End

Error Handling

During data import, the following error information indicates that DWS is to read an ORC data file but the actual file is in TXT format. Therefore, create a table of the Hive ORC type by referring to **Creating a Hive ORC Table in an MRS Cluster** and store the data to the table.


```
ERROR:  dn_6009_6010: Error occurs while creating an orc reader for file /user/hive/warehouse/
products_info.txt, detail can be found in dn log of dn_6009_6010.
```

2.3.6 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If operations in **Querying Data After Importing It** have been performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

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

```
DROP FOREIGN TABLE foreign_product_info;
```

----End

Deleting the Manually Created Foreign Server

If operations in **Manually Creating a Foreign Server** have been performed, perform the following steps to delete the foreign server, database, and user:

Step 1 Use the client provided by DWS to connect to the database where the foreign server is located as the user who created the foreign server.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  ```
  \c mydatabase dbuser;
  ```
  Enter the password as prompted.
- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:
  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```
  Enter the password as prompted.

Step 2 Delete the manually created foreign server.

Run the following command to delete the server. For details about the syntax, see DROP SERVER.

```
DROP SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca;
```

The foreign server is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

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

The server is successfully deleted if the returned result is as follows:


```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

Step 3 Delete the customized database.

Connect to the default database **postgres** through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:

```
DROP DATABASE
```

Step 4 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:

```
REVOKE ALL ON FOREIGN DATA WRAPPER hdfs_fdw FROM dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

2.4 GDS-based Cross-Cluster Interconnection

Function

With data processing based on foreign tables, GDS is used to transfer data and synchronize data between multiple clusters.

Scenarios

- Data is synchronized from one cluster to another. Full data synchronization and data synchronization based on filter criteria are supported.

- Currently, only the following syntax is supported.
  - INSERT INTO *inner table* SELECT... FROM *linked_external_table 1* [WHERE];
  - INSERT INTO *linked_external_table* SELECT * FROM *inner table 1* [JOIN *inner table 2* | WHERE];
  - SELECT ... FROM *linked_external_table*;

Important Notes

- The column name and type of the created foreign table must be the same as those of the corresponding source table, and the tables in the remote cluster must be row-store or column-store.
- Before running the synchronization statement, ensure that the tables to be synchronized exist in the local and remote clusters.
- The status of the two clusters is **Normal**.
- Both clusters must be able to connect to each other using GDS.
- The database code of both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.
- The compatible database types specified for both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.
- Ensure that the user performing table synchronization has the permission to access those tables.
- Foreign tables for interconnection can be used only for cross-cluster data synchronization. In other scenarios, errors may occur or the operation may be invalid.
- The foreign tables for interconnection do not support complex column expressions or complex syntax, including **join**, **sort**, **cursor**, **with**, and **set**.
- SQL statements that are not pushed down cannot use this feature to synchronize data.
- The explain plan and logical cluster are not supported.
- Only the simple JDBC mode is supported.
- If data is synchronized from the local cluster to a remote cluster, only internal table query is supported.
- The GDS specified by the **syncsrv** option of foreign servers does not support the SSL mode.
- After data synchronization is complete, only the number of data rows is verified.
- The maximum number of concurrent services cannot be greater than half of the value of the GDS startup parameter **-t** and cannot be greater than the value of **max_active_statements**. Otherwise, services may fail due to timeout.

Preparations

- Configure the interconnection between both clusters.
- Plan and deploy GDS servers. Ensure that all GDS servers can communicate with all nodes in the two clusters. That is, the security group of the GDS

servers must allow the corresponding GDS port (for example, 5000) and DWS port (8000 by default) in the inbound direction. For details about how to deploy GDS, see **Installing, Configuring, and Starting GDS**.

NOTE

When starting GDS, you can specify any directory as the data transit directory, for example, **/opt**. An example of the startup command is as follows:

/opt/gds/bin/gds -d /opt -p *192.168.0.2*:**5000 -H** *192.168.0.1/24* **-l /opt/gds/bin/gds_log.txt -D -t 2**

Procedure

Assume that the table **tbl_remote** in the remote cluster is to be synchronized with the table **tbl_local** in the local cluster and the user performing the synchronization is **user_remote**. Note that the user must have the permission to access the **tbl_remote** table.

Step 1 Create a server.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS(
  address '192.168.178.207:8000',
  dbname 'db_remote',
  username 'user_remote',
  password 'xxxxxxxx',
  syncsrv 'gsfs://192.168.178.129:5000|gsfs://192.168.178.129:5000'
);
```

- **server_remote** indicates the server name, which is used by the foreign table for interconnection.
- **address** indicates the IP address and port number of the CN in the remote cluster. Only one address is allowed.
- **dbname** indicates the database name of the remote cluster.
- **username** indicates the username used for connecting to the remote cluster. This user cannot be a system administrator.
- **password** indicates the password used for connecting to the remote cluster.
- **syncsrv** indicates the IP address and port number of the GDS server. If there are multiple addresses, use vertical bars (|) to separate them. The function of **syncsrv** is similar to that of **location** in the GDS foreign table.

  NOTE

  DWS tests the network connected to the GDS addresses set by **syncsrv**.

  - The test can only show the network status between the local cluster and the GDSs, but cannot show the network status between the remote cluster and GDS. You need to check the error message.
  - After removing the unavailable GDSs, select a proper number of GDSs that do not cause service suspension to synchronize data.

Step 2 Create a foreign table for interconnection.

```
CREATE FOREIGN TABLE ft_tbl(
  col_1    type_name,
  col_2    type_name,
  ...
) SERVER server_remote OPTIONS (
  schema_name 'schema_remote',
  table_name 'tbl_remote',
  encoding 'utf8'
);
```


- **schema_name** indicates the schema that the remote cluster table belongs to. If this option is not specified, **schema_name** is set to the schema of the foreign table.
- **table_name** indicates the remote cluster table name. If this option is not specified, **table_name** is set to the name of the foreign table.
- **encoding** indicates the encoding format of the remote cluster. If this option is not specified, the default encoding format of the source cluster database is used.

NOTE

- The values of **schema_name** and **table_name** are case sensitive and must be the same as those of the remote schema and table.
- The foreign table for interconnection cannot contain any constraints in its columns.
- The column names and column types of the foreign table must be the same as those of the **tbl_remote** table.
- **SERVER** must be set to the server created in **Step 1** and must contain the **syncsrv** attribute.

Step 3 Use the foreign table for interconnection to synchronize data.

- If the local cluster is the destination cluster, you can run the following statements:

  Full data synchronization of all columns:

  ```
  INSERT INTO tbl_local SELECT * FROM ft_tbl;
  ```

  Data synchronization of all columns based on filter criteria:

  ```
  INSERT INTO tbl_local SELECT * FROM ft_tbl WHERE col_2 = XX;
  ```

  Full data synchronization of some columns:

  ```
  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl;
  ```

  Data synchronization of some columns based on filter criteria:

  ```
  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl WHERE col_2 = XX;
  ```

- If the local cluster is the source cluster, you can run the following statements:

  Synchronization of unsharded tables:

  ```
  INSERT INTO ft_tbl SELECT * FROM tbl_local;
  ```

  Data synchronization of the **join** results:

  ```
  INSERT INTO ft_tbl SELECT * FROM tbl_local1 join tbl_local2 ON XXX;
  ```


NOTE

- If a connection failure is reported, check the server information and ensure that both clusters are connected.
- If an error indicating GDS connection failure is reported, check whether the GDS server specified by **syncsrv** has been started and whether it can communicate with all nodes in both clusters.
- If an error is reported indicating that the table does not exist, check whether the **option** information of the foreign table is correct.
- If an error is reported indicating that the column does not exist, check whether the column name of the foreign table is the same as that of the source table.
- If an error message is displayed indicating that a column is repeatedly defined, check whether the column name is too long. If yes, use the AS alias to simplify the column name.
- If an error is reported indicating that the column type cannot be parsed, check whether the statement contains a column expression.
- If a column mismatch error is reported, check whether the column information of the foreign table is the same as that of the corresponding table in the remote cluster.
- If an error is reported indicating that the syntax is not supported, check whether complex syntax is used, such as **join**, **distinct**, and **sort** .
- If garbled characters are displayed, check whether the encoding formats of both databases are the same.
- If the local cluster is the source cluster, there is a low probability that data is successfully synchronized to the remote cluster but the local cluster returns an execution failure. In this case, you are advised to check the number of synchronized data records.
- If the local cluster is the source cluster, data synchronization controlled by transaction blocks and sub-transactions can be queried only after the total transaction is committed.

Step 4 Delete the foreign table for interconnection.

```
DROP FOREIGN TABLE ft_tbl;
```

----End

2.5 Using a gsql Meta-Command to Import Data

The **gsql** tool of DWS provides the **\copy** meta-command to import data.

\copy Command

For details about the **\copy** command, see **Table 2-11**.

Table 2-11 \copy meta-command

Syntax	Description
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	You can run this command to import or export data after logging in to the database on any gsql client. Different from the **COPY** statement in SQL, this command performs read/write operations on local files rather than files on database servers. The accessibility and permissions of the local files are restricted to local users. **NOTE** **\copy** only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or **COPY** is preferred for data import.

Parameter Description

- table

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- column_list

  Specifies an optional list of columns to be copied.

  Value range: any field in the table. If the column list is not specified, all columns in the table will be copied.

- query

  Specifies that the results will be copied.

  Valid value: a **SELECT** or **VALUES** command in parentheses.

- filename

  Specifies the absolute path of a file. To run the **\copy** command, the user must have the write permission for this path.

- stdin

  Specifies that input comes from the client application.

- stdout

  Specifies that output goes to the client application.

- pstdin

  Specifies that input comes from the gsql client.

- pstdout
- Specifies that output goes to the gsql client.
- binary

  Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. After specifying BINARY, CSV, FIXED and TEXT cannot be specified through **option** or **copy_option**.

- oid

  Specifies the internal OID to be copied for each row.

  NOTE

  An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.

  Valid value: **true**, **on**, **false**, and **off**.

  Default value: **false/off**

- delimiter [as] 'character'

  Specifies the character that separates columns within each row (line) of the file.

  NOTE

  - A delimiter cannot be **\r** or **\n**.
  - A delimiter cannot be the same as the **null** value. The delimiter of CSV data cannot be same as the **quote** value.
  - The delimiter of TEXT data cannot contain any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789
  - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
  - You are advised to use multi-characters and invisible characters for delimiters. For example, you can use the multiple-character delimiter "$^&" and invisible delimiters, such as E'\x07', E'\x08', and E'\x1b'.

  Value range: a multi-character delimiter within 10 bytes.

  Default value:

  – A tab character in TEXT format
  – A comma (,) in CSV format
  – No delimiter in FIXED format

- null [as] 'string'

  Specifies that a string represents a null value in a data file.

  Value range:

  – A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
  – A null value cannot be the same as the **delimiter** or **quote** value.

  Default value:

  – An empty string without quotation marks in CSV format
  – **\N** in TEXT format

- header

  Specifies whether a data file contains a table header. **header** is available only for CSV and FIXED files.

  In data import scenarios, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

  If header is **on**, **fileheader** must be specified. **fileheader** specifies the content in the header. If header is **off**, the exported file does not contain a header.

  Valid value: **true**, **on**, **false**, and **off**.

  Default value: **false/off**

- quote [as] 'character'

  Specifies the quote character used when a data value is referenced in a CSV file.

  Default value: double quotation mark ("").

  NOTE

  - The **quote** value cannot be the same as the **delimiter** or **null** value.
  - The **quote** value must be a single-byte character.
  - You are advised to use invisible characters as quotes, for example, E'\x07', E'\x08', and E'\x1b'.

- escape [as] 'character'

  This option is allowed only when using CSV format. This must be a single one-byte character.

  Default value: double quotation mark (""). If the value is the same as the **quote** value, it will be replaced with **\0**.

- force quote column_list | *

  Quotes all not-null values in each declared column when **CSV COPY TO** is used. NULL values will not be quoted.

  Value range: an existing column.

- force not null column_list

  In CSV COPY FROM mode, processes each specified column as though it were quoted and hence not a null value.

  Value range: an existing column.

Example

Create the target table **copy_example**.

```
create table copy_example
(
    col_1 integer,
    col_2 text,
    col_3 varchar(12),
    col_4 date,
    col_5 time
);
```

- Example 1: Copy data from **stdin** to the target table **copy_example**.

  ```
  \copy copy_example from stdin csv;
  ```

  When you see the **>>** characters, you can start entering data. To finish your input, type a backslash and a period (\.).

  ```
  Enter data to be copied followed by a newline.
  End with a backslash and a period on a line by itself.
  >> 1,"iamtext","iamvarchar",2006-07-07,12:00:00
  >> \.
  ```

- Example 2: The **example.csv** file is in the local directory **/local/data/** and the file contains the header line. (|) is used as the delimiter, and the double quotation marks are used for **quote**. The content is as follows:

  iamheader

  1|"iamtext"|"iamvarchar"|2006-07-07|12:00:00

  2|"iamtext"|"iamvarchar"|2022-07-07|19:00:02

Import data from the local file **example.csv** to the target table **copy_example**. If the header option is **on**, the first row is automatically ignored. By default, quotation marks are used for **quote**.

```
\copy copy_example from  '/local/data/example.csv' with(header 'on', format 'csv', delimiter '|', date_format 'yyyy-mm-dd',  time_format 'hh24:mi:ss');
```

- Example 3: The **example.csv** file is in the local directory **/local/data/**. The comma (,) is used as the delimiter, and the quotation mark (") is used for **quote**. The last field is missing in the first line, and one extra field is added in the second line. The content is as follows:

  1,"iamtext","iamvarchar",2006-07-07

  2,"iamtext","iamvarchar",2022-07-07,19:00:02,12:00:00

  To import data from the local file **example.csv** to the target table **copy_example**, you don't need to specify the delimiter since the default delimiter is (,). Additionally, the fault tolerance parameters **IGNORE_EXTRA_DATA** and **FILL_MISSING_FIELDS** are set, which means that missing fields will be replaced with **NULL** and extra fields will be ignored.

  ```
  \copy copy_example from  '/local/data/example.csv' with( format 'csv', date_format 'yyyy-mm-dd', time_format 'hh24:mi:ss', IGNORE_EXTRA_DATA 'true', FILL_MISSING_FIELDS 'true');
  ```

- Example 4: Export the content of the **copy_example** table to **stdout** in CSV format, use double quotation marks as for **quote**, and use quotes to enclose the fourth and fifth columns.

  ```
  \copy copy_example to stdout CSV quote as '"' force quote col_4,col_5;
  ```

2.6 Using COPY FROM STDIN to Import Data

2.6.1 Data Import Using COPY FROM STDIN

This method is applicable to low-concurrency scenarios where a small volume of data is to be imported.

Use either of the following methods to write data to DWS using the **COPY FROM STDIN** statement:

- Write data into DWS by typing.
- Import data from a file or database to DWS through the CopyManager interface driven by JDBC. You can use any parameters in the **COPY** syntax.

2.6.2 Introduction to the CopyManager Class

CopyManager is an API interface class provided by the JDBC driver in DWS. It is used to import data to DWS in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **org.postgresql.copy** package class and is inherited from the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```


Constructor Method

public CopyManager(BaseConnection connection)

throws SQLException

Basic Methods

Table 2-12 Basic methods of CopyManager

Return Value	Method	Description	Throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOException
long	copyIn(String sql, Reader from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	Sends the result set of **COPY TO STDOUT** from a database to the OutputStream class.	SQLException,IOException

Return Value	Method	Description	Throws
long	copyOut(String sql, Writer to)	Sends the result set of **COPY TO STDOUT** from a database to the Writer class.	SQLException,IOException

2.7 Accessing Hive Metastore Across Clusters

To access MRS Hive data from storage-compute decoupled data warehouse, including when Hive is connected to HDFS or OBS, create an external schema. This function is supported only by 9.1.0 and later versions.

Preparing the Environment

- Create a storage-compute decoupled cluster and an MRS analysis cluster. Ensure that the MRS and DWS clusters are in the same region, AZ, and VPC subnet and that the clusters can communicate with each other.

Constraints and Limitations

- Currently, only the SELECT, INSERT, and INSERT OVERWRITE operations can be performed on tables in the Hive database through external schemas.
- MRS supports two types of data sources. For details, see **Table 2-13**.

Table 2-13 Operations supported by MRS data sources

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

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

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

Procedure

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

Preparing the ORC Table Data Source of MRS

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

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



```
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

Step 2 Log in to the OBS console, click **Create Parallel File System**, set the following parameters, and click **Create Now**.

Table 2-14 Bucket parameters

Parameter	Value
Region	Based on site requirements
Data Redundancy Policy	Single-AZ storage
Bucket Name	mrs-datasource
Default Storage Class	Standard
Bucket Policy	Private
Default Encryption	Disable
Direct Reading	Disable
Enterprise Project	default
Tags	N/A

Step 3 Switch back to the MRS console and click the name of the created MRS cluster. On the **Dashboard** page, click the Synchronize button next to **IAM User Sync**. The synchronization takes about 5 minutes.

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

Step 5 (Optional) Connect Hive to OBS.

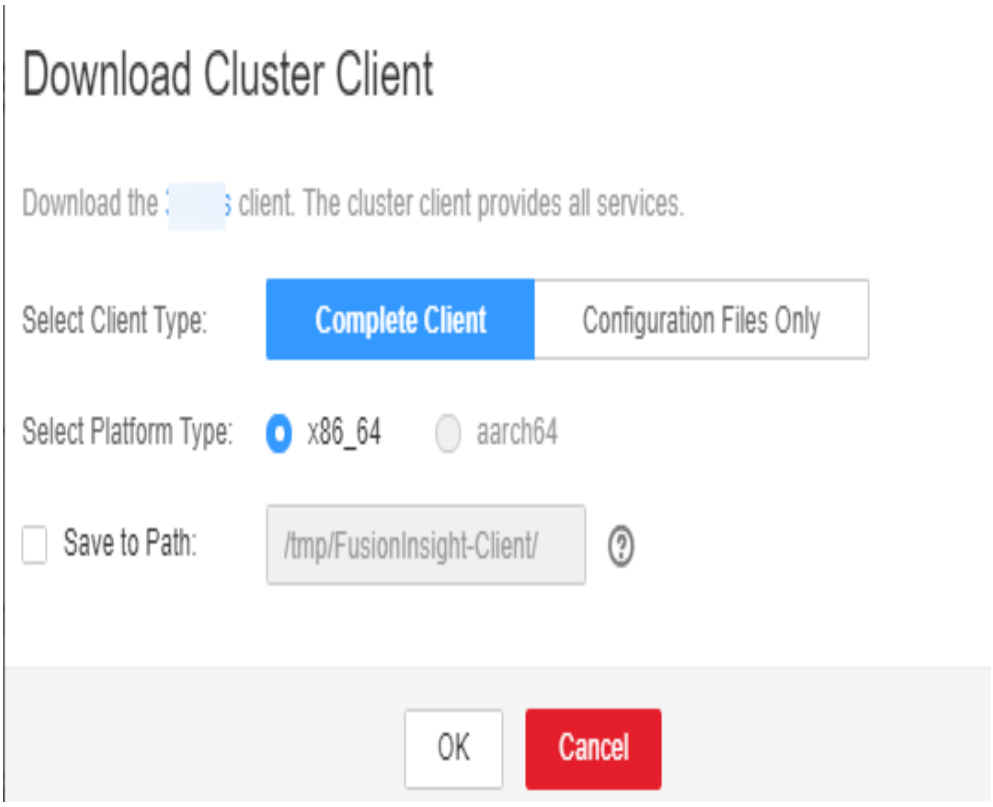
NOTE

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

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

Step 6 Download the client.

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



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

cd /opt/client

sh refreshConfig.sh /opt/client *Full_path_of_client_configuration_file_package*

In this tutorial, run the following command:

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

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

su - omm

cd /opt/client

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

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

   source bigdata_env

   NOTE

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

2. Log in to the Hive client.

   a. If Kerberos authentication is enabled for the current cluster, run the command below to authenticate the current user. The current user must have the permission to create Hive tables. For details, see **Creating a Role** in *MapReduce Service User Guide*.

   b. Configure roles with corresponding permissions. .For details, see **Creating a User** in *MapReduce Service User Guide*. Bind a role to the user. If Kerberos authentication is not enabled for the current cluster, you do not need to run the following command:

      kinit *MRS cluster user*

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

      beeline

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

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

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


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

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

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

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

```
DROP TABLE product_info_orc;

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

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

```
insert into product_info_orc select * from product_info;
```

3. Query whether the data import is successful.

```
select * from product_info_orc;
```

----End

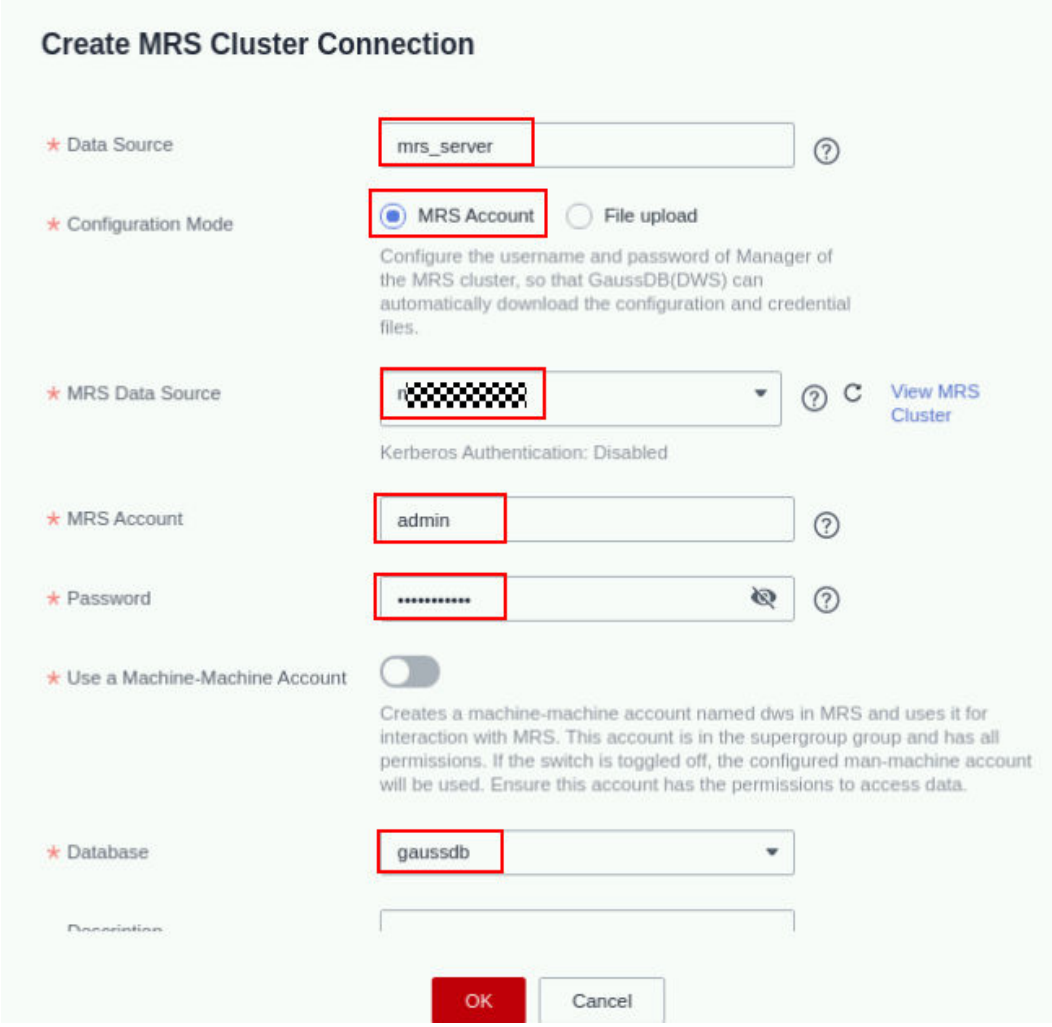
Creating an MRS Data Source Connection

Step 1 Log in to the **DWS console** and click the created DWS cluster. **Ensure that the DWS cluster and MRS are in the same region, AZ, and VPC subnet**.

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

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

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



----End

Creating a Foreign Server

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

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

Step 2 Run the following statement to create a foreign server.

NOTICE

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

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER DFS_FDW
OPTIONS
(
address 'obs.example.com:5443', //Address for accessing OBS
encrypt 'on',
access_key '{AK value}',
```



```
secret_access_key '{SK value}',
 type 'obs'
);
```

Step 3 Check the foreign server.

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

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

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

----End

Create an external schema.

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

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

Step 2 Create an external schema.

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

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

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

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


Step 3 View the created external schema.

```
SELECT * FROM pg_namespace WHERE nspname='ex1';
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE
nspname = 'ex1');
                  nspid                  | srvname | source | address | database | confpath
|                                          ensoptions   | catalog
---------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------
               16393                     |    obs_server |  hive | ***.***.***.***:***         |  demo          | ***
|                      |
(1 row)
```

----End

Importing Data

Step 1 Create a local table for data import.

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

Step 2 Import the target table from the Hive table.

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

Step 3 Query the import result.

```
SELECT * FROM product_info;
```

----End

Exporting Data

Step 1 Create a local source table.

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

Step 2 Create a target table on Hive.

```
DROP TABLE product_info_orc_export;
```



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

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

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

Step 4 Query the import result on Hive

```
SELECT * FROM product_info_orc_export;
```

----End

3 Full Database Migration

3.1 Using CDM to Migrate Data to DWS

You can use CDM to migrate data from other data sources (for example, MySQL) to the databases in DWS clusters.

CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources.

NOTE

CDM migrates data to DWS using the copy method or the GDS parallel import method.

For details about scenarios where CDM is used to migrate data to DWS, see the following sections of *Cloud Data Migration User Guide*:

- **Getting Started**: describes how to use CDM to migrate local MySQL databases to DWS.

Figure 3-1 Migrating data to DWS using CDM

3.2 Using DSC to Migrate SQL Scripts

The DSC is a CLI tool running on the Linux or Windows OS. It is dedicated to providing customers with simple, fast, and reliable application SQL script migration services. It parses the SQL scripts of source database applications using the built-in syntax migration logic, and converts them to SQL scripts applicable to DWS databases. You do not need to connect the DSC to a database. It can migrate data in offline mode without service interruption. In DWS, you can run the migrated SQL scripts to restore the database, thereby easily migrating offline databases to the cloud.

The DSC can migrate SQL scripts of Teradata, Oracle, Netezza, MySQL, and DB2 databases.

Downloading the DSC SQL Migration Tool

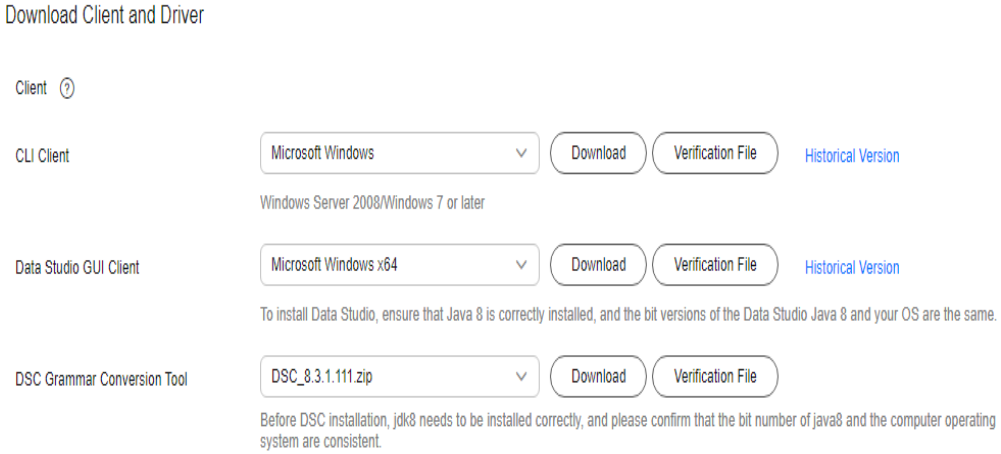
Step 1 Log in to the **DWS console**.

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

Step 3 In the **Download Client and Driver** area, click **here** to download the DSC migration tool.

If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the client corresponding to the cluster version. 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.

Figure 3-2 Downloading the tool



Step 4 Download the client.

Table 3-1 DSC download link

Download Link	Verification File
DSC_8.3.1.111.zip	DSC_8.3.1.111.zip.sha256

Step 5 After downloading the DSC tool to the local PC, use WinSCP to upload it to a Linux host.

The user who uploads the tool must have the full control permission on the target directory of the Linux host.

----End

Operation Guide for the DSC SQL Syntax Migration Tool

For details, see **DSC - SQL Syntax Migration Tool**.

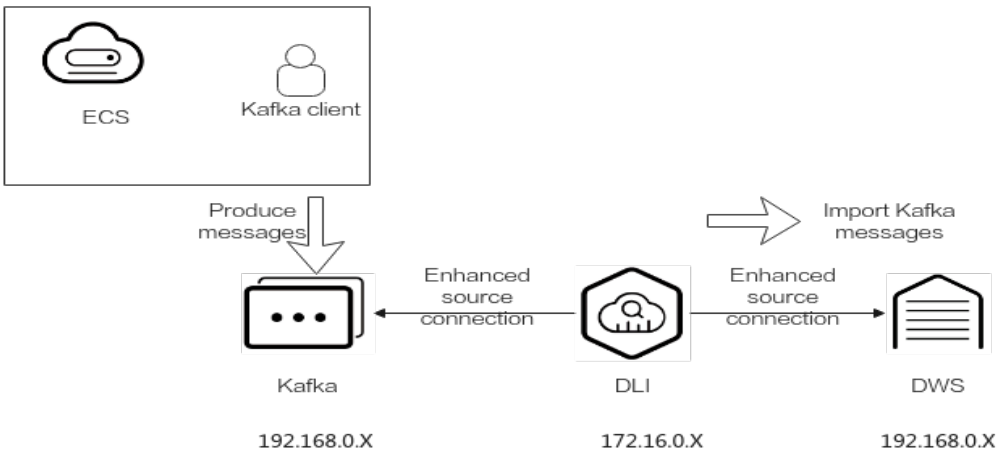
4 Real-time Import

4.1 Importing Kafka Data to DWS in Real Time

You can use **Data Lake Insight (DLI)** Flink jobs to synchronize Kafka consumption data to DWS in real time.

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

Figure 4-1 Importing Kafka data to DWS in real time



For details, see **Using DLI Flink Jobs to Write Kafka Data to DWS in Real Time**.

5 Metadata Migration

5.1 Using gs_dump and gs_dumpall to Export Metadata

5.1.1 Overview

DWS provides gs_dump and gs_dumpall to export required database objects and related information. To migrate database information, you can use a tool to import the exported metadata to a target database. gs_dump exports a single database or its objects. gs_dumpall exports all databases or global objects in a cluster. For details, see **Table 5-1**.

Table 5-1 Application scenarios

Application Scenario	Export Granularity	Export Format	Import Method
Exporting a single database	**Database-level export** • Export full information of a database. You can use the exported information to create a same database containing the same data as the current one. • Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same database as the current one, without data. • Export data of a database. **Schema-level export** • Export full information of a schema. • Export data of a schema. • Export all object definitions of a schema, including the definitions of tables, stored procedures, and indexes. **Table-level export** • Export full information of a table. • Export data of a table. • Export the definition of a table.	• Plain text • Custom • Directory • .tar	• For details about how to import data files in text format, see **Using a gsql Meta-Command to Import Data**. • For details about how to import data files in .tar, directory, or custom format, see **Using gs_restore to Import Data**.

Application Scenario	Export Granularity	Export Format	Import Method
Exporting all databases in a cluster	**Database-level export** • Export full information of a cluster. You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one. • Export all object definitions of a cluster, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data. • Export data of a cluster.	Plain text	For details about how to import data files, see **Using a gsql Meta-Command to Import Data**.
	Global object export • Export tablespaces. • Export roles. • Export tablespaces and roles.		

gs_dump and gs_dumpall use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, gs_dump or gs_dumpall uses the specified role to export data. See **Table 5-1** for application scenarios and **Data Export By a User Without Required Permissions** for operation details.

gs_dump and gs_dumpall encrypt the exported data files. These files are decrypted before being imported to prevent data disclosure for higher database security.

When gs_dump and gs_dumpall are used to export data, the exported tables are locked, which blocks read and write operations.

gs_dump and gs_dumpall can export complete, consistent data. For example, if gs_dump is used to export database A or gs_dumpall is used to export all databases from a cluster at T1, data of database A or all databases in the cluster at that time point will be exported, and modifications on the databases after that time point will not be exported.

Obtain gs_dump and gs_dumpall by decompressing the **gsql CLI client** package.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- For data consistency and integrity, gs_dump and gs_dumpall set a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, gs_dump and gs_dumpall lock the table after the lock is released. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.
- During an export, gs_dumpall reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When you use gsql to import scripts, cluster administrator permissions are also required to add users and user groups, and create databases.
- By default, the definitions of all views in the DWS database contain the prefix of table names or aliases (in **tab.col** format). Therefore, the definitions may be inconsistent with the original ones. As a result, the base table corresponding to the rebuilt view column is incorrect and an error is reported. However, this rarely happens. To prevent this problem, you are advised to set the GUC parameter **behavior_compat_options** to **compat_display_ref_table** when exporting view definitions, so the exported definitions are consistent with the original statements.

5.1.2 Exporting a Single Database

5.1.2.1 Exporting a Database

You can use gs_dump to export data and all object definitions of a database from DWS. You can specify the information to be exported as follows:

- Export full information of a database, including its data and all object definitions.

  You can use the exported information to create a same database containing the same data as the current one.
- Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures.

  You can use the exported object definitions to quickly create a same database as the current one, without data.
- Export data of a database.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- <*Path_for_storing_the_client*>: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dump to export data of the database **gaussdb**.

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.tar -p 8000 gaussdb -h 10.10.10.100 -F t
```

Table 5-2 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ *postgres*_backup.tar

Parameter	Description	Example Value
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	gaussdb
-F	Format of exported files. The values of **-F** are as follows: ● **p**: plain text ● **c**: custom ● **d**: directory ● **t**: .tar	-F t

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the database **gaussdb** and compress the exported files in SQL format.

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.sql -p 8000  -h 10.10.10.100 gaussdb -Z 8 -F p
gs_dump[port='']['gaussdb'][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port='']['gaussdb'][2017-07-21 15:36:13]: total time: 3793  ms
```

Example 2: Use gs_dump to run the following command to export data of the database **gaussdb**, excluding object definitions. The exported files are in a custom format.

```
gs_dump -W Password -U jack -f /home//backup/postgres_data_backup.dmp -p 8000 -h 10.10.10.100 gaussdb -a -F c
gs_dump[port='']['gaussdb'][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port='']['gaussdb'][2017-07-21 15:36:13]: total time: 3793  ms
```

Example 3: Use gs_dump to run the following command to export object definitions of the database **gaussdb**. The exported files are in SQL format.


```
--Before the export, the nation table contains data.
select n_nationkey,n_name,n_regionkey from nation limit 3;
 n_nationkey |          n_name           | n_regionkey
-------------+---------------------------+-------------
           0 | ALGERIA                   |           0
           3 | CANADA                    |           1
          11 | IRAQ                      |           4
(3 rows)

gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb -
s -F p
gs_dump[port='']['gaussdb'][2017-07-20 15:04:14]: dump database gaussdb successfully
gs_dump[port='']['gaussdb'][2017-07-20 15:04:14]: total time: 472 ms
```

Example 4: Use gs_dump to run the following command to export object definitions of the database **gaussdb**. The exported files are in text format and are encrypted.

```
gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb
--with-encryption AES128 --with-key 1234567812345678 -s -F p
gs_dump[port='']['gaussdb'][2018-11-14 11:25:18]: dump database gaussdb successfully
gs_dump[port='']['gaussdb'][2018-11-14 11:25:18]: total time: 1161  ms
```

5.1.2.2 Exporting a Schema

You can use gs_dump to export data and all object definitions of a schema from DWS. You can export one or more specified schemas as needed. You can specify the information to be exported as follows:

- Export full information of a schema, including its data and object definitions.
- Export data of a schema, excluding its object definitions.
- Export the object definitions of a schema, including the definitions of tables, stored procedures, and indexes.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dump to run the following command to export the **hr** and **public** schemas.

```
gs_dump -W Password -U jack -f /home//backup/MPPDB_schema_backup -p 8000 -h 10.10.10.100 human_resource -n hr -F d
```

Table 5-3 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/MPPDB_schema_backup
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100

Parameter	Description	Example Value
dbname	Name of the database to be exported.	human_resource
-n	Names of schemas to be exported. Data of the specified schemas will also be exported. • Single schema: Enter **-n** *schemaname*. • Multiple schemas: Enter **-n** *schemaname* for each schema.	• Single schema: **-n hr** • Multiple schemas: **-n hr -n public**
-F	Format of exported files. The values of **-F** are as follows: • **p**: plain text • **c**: custom • **d**: directory • **t**: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the **hr** schema. The exported files are compressed and stored in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.sql -p 8000 -h 10.10.10.100
human_resource -n hr -Z 6 -F p
gs_dump[port='']human_resource][2017-07-21 16:05:55]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 16:05:55]: total time: 2425  ms
```

Example 2: Use gs_dump to run the following command to export data of the **hr** schema. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -n hr -a -F t
gs_dump[port='']human_resource][2018-11-14 15:07:16]: dump database human_resource successfully
gs_dump[port='']human_resource][2018-11-14 15:07:16]: total time: 1865  ms
```

Example 3: Use gs_dump to run the following command to export the definition of the **hr** schema. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_def_backup -p 8000 -h 10.10.10.100
human_resource -n hr -s -F d
gs_dump[port='']human_resource][2018-11-14 15:11:34]: dump database human_resource successfully
gs_dump[port='']human_resource][2018-11-14 15:11:34]: total time: 1652  ms
```

Example 4: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -F c
gs_dump[port='']human_resource][2017-07-21 16:06:31]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 16:06:31]: total time: 2522  ms
```


Example 5: Use gs_dump to run the following command to export the object definitions of the **hr** and **public** schemas, encrypt the exported files, and store them in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup1.tar -p 8000 -h 10.10.10.100
human_resource -n hr -n public -s --with-encryption AES128 --with-key 1234567812345678 -F t
gs_dump[port='']['human_resource'][2017-07-21 16:07:16]: dump database human_resource successfully
gs_dump[port='']['human_resource'][2017-07-21 16:07:16]: total time: 2132  ms
```

Example 6: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr** and **public** schemas. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup2.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -N public -F c
gs_dump[port='']['human_resource'][2017-07-21 16:07:55]: dump database human_resource successfully
gs_dump[port='']['human_resource'][2017-07-21 16:07:55]: total time: 2296  ms
```

Example 7: Use gs_dump to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_backup3.dmp -p 8000 -h 10.10.10.100
human_resource -t public.* -t hr.staffs -F c
gs_dump[port='']['human_resource'][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port='']['human_resource'][2018-12-13 09:40:24]: total time: 896  ms
```

5.1.2.3 Exporting a Table

You can use gs_dump to export data and all object definitions of a table-level object from DWS. Views, sequences, and foreign tables are special tables. You can export one or more specified tables as needed. You can specify the information to be exported as follows:

- Export full information of a table, including its data and definition.
- Export data of a table.
- Export the definition of a table.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- <*Path_for_storing_the_client*>: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dump to run the following command to export the **hr.staffs** and **hr.employments** tables.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup -p 8000 -h 10.10.10.100 human_resource -t hr.staffs -F d
```

Table 5-4 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/MPPDB_table_backup
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000

Parameter	Description	Example Value
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
-t	Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters. When you use wildcard characters, quote wildcard patterns with single quotation marks ('') to prevent the shell from expanding the wildcard characters. • Single table: Enter **-t** *schema.table*. • Multiple tables: Enter **-t** *schema.table* for each table.	• Single table: **-t hr.staffs** • Multiple tables: **-t hr.staffs -t hr.employments**
-F	Format of exported files. The values of **-F** are as follows: • **p**: plain text • **c**: custom • **d**: directory • **t**: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the **hr.staffs** table. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -Z 6 -F p
gs_dump[port=''][human_resource][2017-07-21 17:05:10]: dump database human_resource successfully
gs_dump[port=''][human_resource][2017-07-21 17:05:10]: total time: 3116  ms
```


Example 2: Use gs_dump to run the following command to export data of the **hr.staffs** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -a -F t
gs_dump[port='']human_resource][2017-07-21 17:04:26]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 17:04:26]: total time: 2570  ms
```

Example 3: Use gs_dump to run the following command to export the definition of the **hr.staffs** table. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_def_backup -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -s -F d
gs_dump[port='']human_resource][2017-07-21 17:03:09]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 17:03:09]: total time: 2297  ms
```

Example 4: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr.staffs** table. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.dmp -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -F c
gs_dump[port='']human_resource][2017-07-21 17:14:11]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 17:14:11]: total time: 2450  ms
```

Example 5: Use gs_dump to run the following command to export the **hr.staffs** and **hr.employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup1.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments -F p
gs_dump[port='']human_resource][2017-07-21 17:19:42]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 17:19:42]: total time: 2414  ms
```

Example 6: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr.staffs** and **hr.employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup2.sql -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -T hr.employments -F p
gs_dump[port='']human_resource][2017-07-21 17:21:02]: dump database human_resource successfully
gs_dump[port='']human_resource][2017-07-21 17:21:02]: total time: 3165  ms
```

Example 7: Use gs_dump to run the following command to export data and definition of the **hr.staffs** table, and the definition of the **hr.employments** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup3.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
gs_dump[port='']human_resource][2018-11-14 11:32:02]: dump database human_resource successfully
gs_dump[port='']human_resource][2018-11-14 11:32:02]: total time: 1645  ms
```

Example 8: Use gs_dump to run the following command to export data and definition of the **hr.staffs** table, encrypt the exported files, and store them in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs --with-encryption AES128 --with-key 1212121212121212 -F p
gs_dump[port='']human_resource][2018-11-14 11:35:30]: dump database human_resource successfully
gs_dump[port='']human_resource][2018-11-14 11:35:30]: total time: 6708  ms
```

Example 9: Use gs_dump to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup5.dmp -p 8000 -h 10.10.10.100
human_resource -t public.* -t hr.staffs -F c
gs_dump[port='']human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port='']human_resource][2018-12-13 09:40:24]: total time: 896  ms
```


Example 10: Use gs_dump to run the following command to export the definition of the view referencing to the **test1** table in the **t1** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_view_backup6 -p 8000 -h 10.10.10.100
human_resource -t t1.test1 --include-depend-objs --exclude-self -F d
gs_dump[port='']['jack'][2018-11-14 17:21:18]: dump database human_resource successfully
gs_dump[port='']['jack'][2018-11-14 17:21:23]: total time: 4239  ms
```

5.1.3 Exporting All Databases

5.1.3.1 Exporting All Databases

You can use gs_dumpall to export full information of all databases in a cluster from DWS, including information about each database and global objects in the cluster. You can specify the information to be exported as follows:

- Export full information of all databases, including information about each database and global objects (such as roles and tablespaces) in the cluster.

  You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one.
- Export data of all databases, excluding all object definitions and global objects.
- Export all object definitions of all databases, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures.

  You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dumpall to run the following command to export information of all databases.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100
```

Table 5-5 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/MPPDB_backup.sql
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100
gs_dumpall[port='']2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port='']2017-07-21 15:57:31]: total time: 9627  ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export definitions of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100 -s
gs_dumpall[port='']2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port='']2018-11-14 11:28:14]: total time: 4147  ms
```

Example 3: Use gs_dumpall to run the following command export data of all databases in a cluster, encrypt the exported files, and store them in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100 -a --with-encryption AES128 --with-key 1234567812345678
gs_dumpall[port='']2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port='']2018-11-14 11:23:26]: total time: 4147  ms
```

5.1.3.2 Exporting Global Objects

You can use gs_dumpall to export global objects from DWS, including database users, user groups, tablespaces, and attributes (for example, global access permissions).

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- <*Path_for_storing_the_client*>: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dumpall to run the following command to export tablespace objects.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h 10.10.10.100 -t
```

Table 5-6 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ *MPPDB_tablespace*.sql

Parameter	Description	Example Value
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
-t	Dumps only tablespaces. You can also use **--tablespaces-only** alternatively.	-

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global tablespaces and users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_globals.sql -p 8000 -h
10.10.10.100 -g
gs_dumpall[port='']2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port='']2018-11-14 19:06:24]: total time: 1150  ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export global tablespaces in a cluster, encrypt the exported files, and store them in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h
10.10.10.100 -t --with-encryption AES128 --with-key 1212121212121212
gs_dumpall[port='']2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port='']2018-11-14 19:00:58]: total time: 186  ms
```

Example 3: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_user.sql -p 8000 -h 10.10.10.100 -
r
gs_dumpall[port='']2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port='']2018-11-14 19:03:18]: total time: 162  ms
```


5.1.4 Data Export By a User Without Required Permissions

gs_dump and gs_dumpall use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, gs_dump or gs_dumpall uses the specified role to export data.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

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

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

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

```
All things done.
```

Step 5 Use gs_dump to export data of the **human_resource** database.

User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
gs_dump -U jack -W password -f /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100
human_resource --role role1 --rolepassword password -F t
```

Table 5-7 Common parameters

Parameter	Description	Example Value (dbadmin)
-U	Username for database connection.	-U jack

Parameter	Description	Example Value (dbadmin)
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/MPPDB_backup.tar
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
--role	Role name for the export operation. After this parameter is set and gs_dump or gs_dumpall connects to the database, the **SET ROLE** command will be issued. When the user specified by **-U** does not have the permissions required by gs_dump or gs_dumpall, this parameter allows the user to switch to a role with the required permissions.	-r role1

Parameter	Description	Example Value (dbadmin)
--rolepassword	Role password.	--rolepassword password
-F	Format of exported files. The values of **-F** are as follows: • **p**: plain text • **c**: custom • **d**: directory • **t**: .tar	-F t

For details about other parameters, see "gs_dump" or "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup11.tar -p 8000 -h 10.10.10.100
human_resource --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 4239  ms
```

Example 2: User **jack** does not have the permission for exporting the **public** schema and the role **role1** has this permission. To export the **public** schema, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup12.tar -p 8000 -h 10.10.10.100
human_resource -n public --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 3278  ms
```

Example 3: User **jack** does not have the permission for exporting all databases in a cluster and the role **role1** has this permission. To export all databases, you can set **--role** to **role1** in the export command. The exported files are in text format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dumpall -U jack -W password -f /home//backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100 --role role1
--rolepassword password
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: dumpall operation successful
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: total time: 6437  ms
```


5.2 Using gs_restore to Import Data

Scenarios

gs_restore is an import tool provided by DWS. You can use **gs_restore** to import the files exported by **gs_dump** to a database. **gs_restore** can import the files in .tar, custom, or directory format.

gs_restore can:

- Import data to a database.

  If a database is specified, data is imported to the database. If multiple databases are specified, the password for connecting to each database also needs to be specified.
- Import data to a script.

  If no database is specified, a script containing the SQL statement to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output of **gs_dump**.

You can specify and sort the data to be imported.

Procedure

NOTE

gs_restore incrementally imports data by default. To avoid data exceptions caused by consecutive imports, use the **-e** and **-c** parameters for each import. This will delete existing data from the target database before each import and exit the import task with an error message (which is displayed after the import process is complete) before proceeding with the next import.

Step 1 Log in to the server as the **root** user and run the following command to go to the data storage path:

```
cd /opt/bin
```

Step 2 Use **gs_restore** to import all object definitions from the exported file of the whole **postgres** database to the **backupdb** database.

```
gs_restore -W password -U jack /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
```

Table 5-8 Common parameters

Parameters	Description	Example Value
-U	Username for database connection.	-U jack

Parameters	Description	Example Value
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-d	Database to which data will be imported.	-d backupdb
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
-e	Exits the current import task and performs the next if an error occurs when you send a SQL statement in the current import task. Error messages are displayed after the import process is complete.	-
-c	Cleans existing objects from the target database before the import.	-
-s	Imports only object definitions in schemas and does not import data. Sequence values will also not be imported.	-

For details about other parameters, see "Server Tools > gs_restore" in the *Tool Reference*.

----End

Examples

Example 1: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.dmp** file (custom format).

```
gs_restore -W password backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore: total time: 13053  ms
```

Example 2: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.tar** file.

```
gs_restore backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:21:32]: restore operation successful
gs_restore[2017-07-21 19:21:32]: total time: 21203  ms
```

Example 3: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup** directory.

```
gs_restore backup/MPPDB_backup -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:26:46]: restore operation successful
gs_restore[2017-07-21 19:26:46]: total time: 21003  ms
```

Example 4: Run **gs_restore** to import the definitions of all objects in the **postgres** database to the **backupdb** database. Table data is not imported.

```
gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
gs_restore[2017-07-21 19:46:27]: restore operation successful
gs_restore[2017-07-21 19:46:27]: total time: 32993  ms
```

Example 5: Run **gs_restore** to import data and all definitions in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Existing objects are deleted from the target database before the import. If an existing object references to an object in another schema, you need to manually delete the referenced object first.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR:  cannot drop table table1 because other objects depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

Manually delete the referenced object and create it again after the import is complete.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore[2017-07-21 19:52:26]: restore operation successful
gs_restore[2017-07-21 19:52:26]: total time: 2203  ms
```

Example 6: Run **gs_restore** to import the definition of the **hr.staffs** table in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table does not exist.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -s -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 19:56:29]: restore operation successful
gs_restore[2017-07-21 19:56:29]: total time: 21000  ms
```


Example 7: Run **gs_restore** to import data of the **hr.staffs** table in **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table is empty.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -a -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 20:12:32]: restore operation successful
gs_restore[2017-07-21 20:12:32]: total time: 20203  ms
```

Example 8: Run **gs_restore** to import the definition of the **hr.staffs** table. Before the import, the **hr.staffs** table already exists.

```
human_resource=# select * from hr.staffs;
 staff_id | first_name  |  last_name  |  email   |    phone_number    |      hire_date      | employment_id |
salary  | commission_pct | manager_id | section_id
----------+-------------+-------------+----------+--------------------+---------------------+---------------+----------
+----------------+------------+------------
      200 | Jennifer    | Whalen      | JWHALEN  | 515.123.4444       | 1987-09-17 00:00:00 | AD_ASST       |
4400.00 |                |        101 |         10
      201 | Michael     | Hartstein   | MHARTSTE | 515.123.5555       | 1996-02-17 00:00:00 | MK_MAN        |
13000.00 |                |        100 |         20

gsql -d human_resource -p 8000
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE;
NOTICE:  drop cascades to view hr.staff_details_view

gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100-d human_resource -n
hr -t staffs -s -e
restore operation successful
total time: 904  ms

human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email | phone_number | hire_date | employment_id | salary |
commission_pct | manager_id | section_id
----------+------------+-----------+-------+--------------+-----------+---------------+--------+----------------
+------------+------------
(0 rows)
```

Example 9: Run **gs_restore** to import data and definitions of the **staffs** and **areas** tables. Before the import, the **staffs** and **areas** tables do not exist.

```
human_resource=# \d
                        List of relations
 Schema |        Name        | Type  |  Owner   |             Storage
--------+--------------------+-------+----------+---------------------------------
 hr     | employment_history | table |  | {orientation=row,compression=no}
 hr     | employments        | table |  | {orientation=row,compression=no}
 hr     | places             | table |  | {orientation=row,compression=no}
 hr     | sections           | table |  | {orientation=row,compression=no}
 hr     | states             | table |  | {orientation=row,compression=no}
(5 rows)

gs_restore -W password /home/mppdb/backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d
human_resource -n hr -t staffs -n hr -t areas
restore operation successful
total time: 724  ms

human_resource=# \d
                        List of relations
 Schema |        Name        | Type  |  Owner   |             Storage
--------+--------------------+-------+----------+---------------------------------
 hr     | areas              | table |  | {orientation=row,compression=no}
 hr     | employment_history | table |  | {orientation=row,compression=no}
 hr     | employments        | table |  | {orientation=row,compression=no}
 hr     | places             | table |  | {orientation=row,compression=no}
 hr     | sections           | table |  | {orientation=row,compression=no}
 hr     | staffs             | table |  | {orientation=row,compression=no}
 hr     | states             | table |  | {orientation=row,compression=no}
(7 rows)
```



```
human_resource=# select * from hr.areas;
 area_id |       area_name
---------+------------------------
       4 | Iron
       1 | Wood
       2 | Lake
       3 | Desert
(4 rows)
```

Example 10: Run **gs_restore** to import data and all object definitions in the **hr** schema.

```
gs_restore -W password /home//backup/MPPDB_backup1.sql -p 8000 -h 10.10.10.100 -d backupdb -n hr -e -
c
restore operation successful
total time: 702  ms
```

Example 11: Run **gs_restore** to import all object definitions in the **hr** and **hr1** schemas to the **backupdb** database.

```
gs_restore -W password /home//backup/MPPDB_backup2.dmp -p 8000 -h 10.10.10.100 -d backupdb -n hr -
n hr1 -s
restore operation successful
total time: 665  ms
```

Example 12: Run **gs_restore** to decrypt the files exported from the **human_resource** database and import them to the **backupdb** database.

```
create database backupdb;


gs_restore /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb --with-
key=1234567812345678
restore operation successful
total time: 23472  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |       area_name
---------+------------------------
       4 | Iron
       1 | Wood
       2 | Lake
       3 | Desert
(4 rows)
```

Example 13: **user 1** does not have the permission to import data from an exported file to the **backupdb** database and **role1** has this permission. To import the exported data to the **backupdb** database, you can set **--role** to **role1** in the **gs_restore** command.

```
human_resource=# CREATE USER user1 IDENTIFIED BY 'password';

gs_restore -U user1 -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
--role role1 --rolepassword password
restore operation successful
total time: 554  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |       area_name
```



```
---------+-------------------------
       4 | Iron
       1 | Wood
       2 | Lake
       3 | Desert
(4 rows)
```


6 Exporting Data

6.1 Exporting Data to OBS

6.1.1 Parallel OBS Data Export

Overview

When you export data in parallel using OBS foreign tables, data files to be exported are specified based on the export mode and exported data format included in OBS foreign table settings. Data is exported in parallel through multiple DNs from DWS to the data files, which improves the overall data export performance.

- The CN only plans data export tasks and delivers the tasks to DNs for execution. In this case, the CN is released to process external requests.
- Every DN is involved in data export, and the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.
- You can concurrently export data using multiple OBS services, but the bucket and object paths specified for the export tasks must be different and cannot be null.
- The OBS server connects to DWS cluster nodes. The export rate is affected by the network bandwidth.
- The text, CSV, ORC, and Parquet formats are supported. The size of data in a single row must be less than 1 GB.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.

  For example, data imported or exported in MySQL compatibility mode can be exported or imported only in MySQL compatibility mode.

Currently, data can be exported using either of the following methods:

- Method 1: You do not need to create a server. Use the default server to create a foreign table. Data in TXT or CSV format is supported. For details, see **Exporting CSV/TXT Data to OBS**.

- Method 2: Create a server and use it to create a foreign table. ORC and Parquet data formats are supported. For details, see **Exporting ORC and Parquet Data to OBS**. Only 9.1.0 and later versions support data in Parquet format.

NOTICE

- Ensure that no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the DWS cluster are in the same region.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.

  If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Related Concepts

- **Source data file**: a text, CSV, ORC, or Parquet file that stores data.
- **OBS**: a cloud storage service used to store unstructured data, such as documents, images, and videos. Data objects concurrently exported from DWS are stored on the OBS server.
- **Bucket**: a container storing objects on OBS.
  - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
  - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.
  - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object**: a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.

  Generally, objects are managed as files. However, OBS has no file system–related concepts, such as files or folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key**: name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name*+*Object name*.
- **Metadata**: object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.

  - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
  - User metadata contains object descriptions specified by users for uploading objects.
- **Data**: object content, which is regarded by OBS as stateless binary data.
- **Foreign table**: A foreign table is used to identify data in a source data file. It stores information, such as the location, format, destination location, encoding format, and data delimiter of a source data file.

Principles

The following describes the principles of exporting data from a cluster to OBS by using a distributed hash table or a replication table.

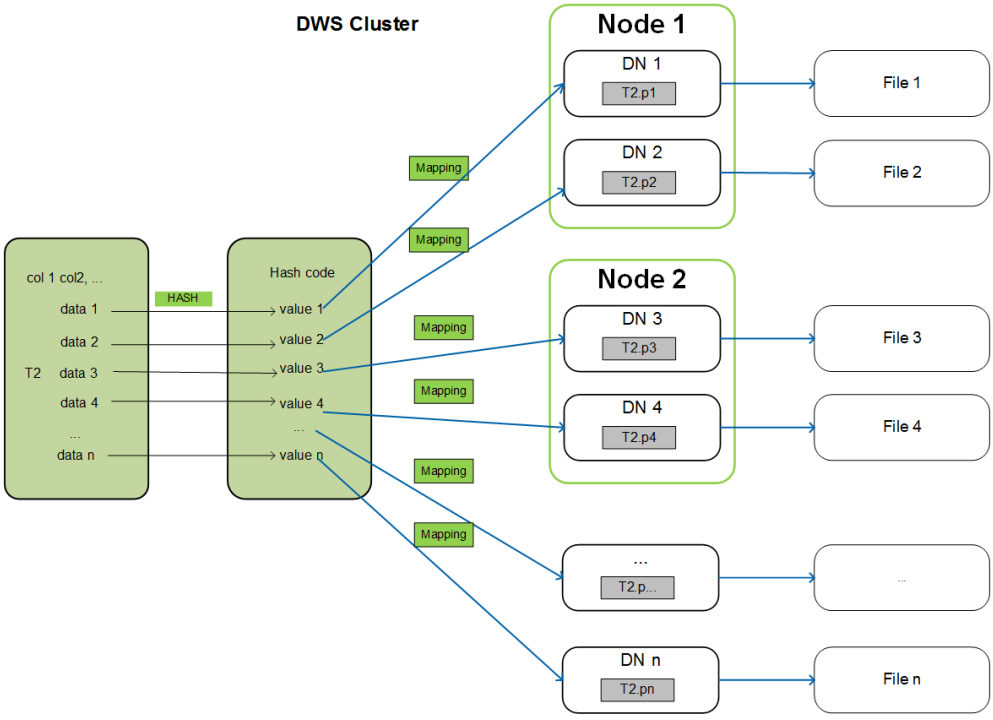
- Distributed hash table: the table for which **DISTRIBUTE BY HASH (Column_Name)** is specified in the table creation statement.

  A distributed hash table stores data in hash mode. **Figure 6-1** shows how to export data from table (**T2**) to OBS as an example.

  During table data storage, the **col2** hash column in table **T2** is hashed, and a hash value is generated. The tuple is distributed to corresponding DNs for storage according to the mapping between the DNs and the hash value.

  When data is exported to OBS, DNs that store the exported data of **T2** directly export their data files to OBS. Original data on multiple nodes will be exported in parallel.

  Figure 6-1 Hash distribution principle



- Replication table: the table for which **DISTRIBUTE BY REPLICATION** is specified in the table creation statement.

A replication table stores a package of complete table data on each DWS node. When exporting data to OBS, DWS randomly selects a DN for export.

Naming Rules of Exported Files

Rules for naming the files exported from DWS to OBS are as follows:

- Data exported from DNs is stored on OBS in segment format. The file is named as *Table name_Node name*_**segment.***n*. *n* is a natural number starting from 0, for example, 0, 1, 2, 3.

  For example, the data of table **t1** on **datanode3** will be exported as **t1_datanode3_segment.0**, **t1_datanode3_segment.1**, and so on.

  You are advised to export data from different clusters or databases to different OBS buckets or different paths of the same OBS bucket.

- Each segment can store a maximum of 1 GB data, with no tuples sliced. If data stored in a segment exceeds 1 GB, the excess data will be stored in the second segment.

  For example:

  A segment has already stored 100 pieces of tuples (1023 MB) when **datanode3** exports data from **t1** to OBS. If a 5 MB tuple is inserted to the segment, the data size becomes 1028 MB. In this case, file **t1_datanode3_segment.0** (1023 MB) is generated and stored on OBS, and the new tuple is stored on OBS as file **t1_datanode3_segment.1**.

- When data is exported from a distributed hash table, the number of segments generated on each DN depends on the data volume stored on a DN, not on the number of DNs in the cluster. Data stored in hash mode may not be evenly distributed on each DN.

  For example, a cluster has **DataNode1**, **DataNode2**, **DataNode3**, **DataNode4**, **DataNode5**, and **DataNode6**, which store 1.5 GB, 0.7 GB, 0.6 GB, 0.8 GB, 0.4 GB, and 0.5 GB data, respectively. Seven OBS segment files will be generated during data export because **DataNode1** will generate two segment files, which store 1 GB and 0.5 GB data, respectively.

Data Export Process

Figure 6-2 Concurrent data export

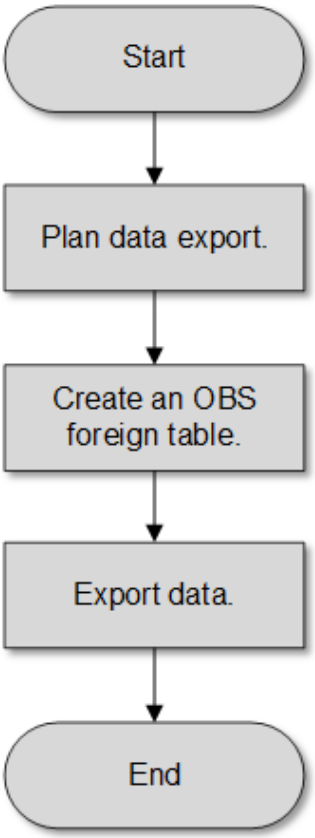


Table 6-1 Process description

Procedure	Description	Subtask
Plan data export.	Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data files. For details, see **Planning Data Export**.	-
Create an OBS foreign table.	Create a foreign table to help OBS specify information about data files to be exported. The foreign table stores information, such as the destination location, format, encoding, and data delimiter of a source data file. For details, see **Creating an OBS Foreign Table**.	-

Procedure	Description	Subtask
Export data.	After the foreign table is created, run the **INSERT** statement to efficiently export data to data files. For details, see **Exporting Data**.	-

6.1.2 Exporting CSV/TXT Data to OBS

6.1.2.1 Planning Data Export

Scenarios

Plan the storage location of exported data in OBS.

Planning OBS Save Path and File

You need to specify the OBS path (to directory) for storing data that you want to export. The exported data can be saved to a file in CSV format. The system also supports TEXT so that you can import the exported data to various applications.

The target directory cannot contain any files.

Planning OBS Bucket Permissions

The user used to export data must:

- Have OBS enabled.
- Have the write permission on the OBS bucket where the data export path is located.

  You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

  For details, see **Granting Write Permission to OBS Storage Location and OBS Bucket as Planned**.

Planning Data to Be Exported and Foreign Tables

You must prepare data to be exported in the database table, and the data volume per row must be less than 1 GB. Based on the data to be exported, plan foreign tables whose attributes such as columns, column types, and length match those of user data.

Granting Write Permission to OBS Storage Location and OBS Bucket as Planned

Step 1 Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data.

1. **Log in to the OBS console**.

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

2. Create a bucket.

   For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Buckets > Creating a Bucket" in the *Object Storage Service User Guide*.

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

3. Create a folder.

   For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Creating a Folder" in the *Object Storage Service User Guide*.

   Example:

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

Step 2 Determine the path of the created OBS folder.

Specify the OBS path for storing exported data files. This path is the value of the **location** parameter used for creating a foreign table.

The OBS folder path in the **location** parameter consists of **obs://**, a bucket name, and a file path. Example:

In this example, the OBS folder path is as follows:

```
obs://mybucket/output_data/
```

NOTE

The OBS directory to be used for storing data files must be empty.

Step 3 Grant the OBS bucket write permission to the user who wants to export data.

When exporting data, a user must have the write permission on the OBS bucket where the data export path is located. You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

----End

6.1.2.2 Creating an OBS Foreign Table

Procedure

Step 1 Based on the path planned in **Planning Data Export**, determine the value of the **location** parameter used for creating a foreign table.

Step 2 Obtain the access keys (AK and SK) to access OBS.

To obtain the keys, log in to the **DWS console**, click the username in the upper right corner, choose **My Credentials**, and click **Access Keys** in the left navigation tree. On the displayed page, view the existing **Access Key ID** (AK). To obtain both the AK and SK, click **Create Access Key** to create and download an access key.

Step 3 Examine the formats of data to be exported and determine the values of data format parameters used for creating a foreign table. For details, see data format parameters.

Step 4 Create an OBS table based on the parameter settings in the preceding steps.

----End

Example 1

For example, in the DWS database, create a write-only foreign table with the **format** parameter as **text** to export text files. Set parameters as follows:

- **location**

  The OBS path of the source data file has been obtained in **step 2** in **Planning Data Export**.

  For example, set **location** as follows:

  ```
  location 'obs://mybucket/output_data/',
  ```

- **Access keys (AK and SK)**
  - Set **access_key** to the AK you have obtained.
  - Set **secret_access_key** to the SK you have obtained.

  NOTE

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**
  - Set **format** to **TEXT**.
  - Set **encoding** to **UTF-8**.
  - Configure **encrypt**. Its default value is **off**.
  - Set **delimiter** to **|**.

Based on the preceding settings, the foreign table is created using the following statements:

NOTICE

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

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext1;
CREATE FOREIGN TABLE product_info_output_ext1
(
 c_bigint bigint,
 c_char char(30),
 c_varchar varchar(30),
```



```
c_nvarchar2 nvarchar2(30) ,
c_data date,
c_time time ,
c_test varchar(30))
server gsmpp_server
options (
LOCATION 'obs://mybucket/output_data/',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
format 'text',
delimiter '|',
encoding 'utf-8',
encrypt 'on'
 )
WRITE ONLY;
```

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

```
CREATE FOREIGN TABLE
```

Example 2:

For example, in the DWS database, create a write-only foreign table with the **format** parameter as **CSV** to export CSV files. Set parameters as follows:

- **location**

  The OBS path of the source data file has been obtained in **step 2** in **Planning Data Export**.

  For example, set **location** as follows:

  ```
  location 'obs://mybucket/output_data/',
  ```

- **Access keys (AK and SK)**
  - Set **access_key** to the AK you have obtained.
  - Set **secret_access_key** to the SK you have obtained.

  NOTE

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**
  - Set **format** to **CSV**.
  - Set **encoding** to **UTF-8**.
  - Configure **encrypt**. Its default value is **off**.
  - Set **delimiter** to **,**.
  - Set **header** (whether the exported data file contains the header row).

    Specifies whether a file contains a header with the names of each column in the file.

    When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**, indicating that the first row of the exported data file is not the header.

Based on the preceding settings, the foreign table is created using the following statements:

NOTICE

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

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext2;
CREATE FOREIGN TABLE product_info_output_ext2
(
    product_price               integer       not null,
    product_id                  char(30)      not null,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(
location 'obs://mybucket/output_data/',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)
WRITE ONLY ;
```

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

```
CREATE FOREIGN TABLE
```

6.1.2.3 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1**: Export data from table **product_info_output** to a data file through the **product_info_output_ext** foreign table.

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

  If information similar to the following is displayed, the data has been exported.

  ```
  INSERT 0 10
  ```

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
  ```

NOTE

- The directory to be used for data storage must be empty, or the export will fail.
- Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to the hexadecimal format before export.

6.1.2.4 Examples

Exporting a Table

Create two foreign tables and use them to export tables from a database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 On the DWS database, create the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS data server to receive data exported from the database.

OBS and the database are in the same region. The example DWS table to be exported is **tpcds.customer_address**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from a database. The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- Configure **encrypt**. Its default value is **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

  NOTE

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

NOTICE

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

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
ca_address_sk             integer                       ,
ca_address_id             char(16)                      ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk             integer                    ,
ca_address_id             char(16)                   ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In DWS, export the data table **tpcds.customer_address** to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** concurrently.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address;
```

NOTE

The design of OBS foreign tables does not allow exporting files to a non-empty path. However, in concurrent export scenarios, multiple files are exported to the same path, causing an error.

Assume that a user concurrently exports data from the same table to the same OBS foreign table, and that one SQL statement is executed to export data when another SQL statement is being executed and has not generated any file on the OBS server. In this case, certain data is overwritten although both SQL statements are successfully executed. Therefore, you are advised not to concurrently export data to the same OBS foreign table.

----End

Concurrently Exporting Tables

Use the two foreign tables to export tables from the database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 In DWS, create foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS server to receive exported data.

OBS and the database are in the same region. Tables to be exported are **tpcds.customer_address** and **tpcds.customer_demographics**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from DWS. The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- Configure **encrypt**. Its default value is **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

  NOTE

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

NOTICE

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

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
ca_address_sk             integer               ,
ca_address_id             char(16)              ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk             integer               ,
ca_address_id             char(16)              ,
ca_address_name             varchar(20)           ,
ca_address_code           integer               ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In DWS, export the data tables **tpcds.customer_address** and **tpcds.warehouse** in parallel to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2**, respectively.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.warehouse;
```

----End

6.1.3 Exporting ORC and Parquet Data to OBS

6.1.3.1 Planning Data Export

For details about exporting data to OBS, see **Planning Data Export**.

For the data types that can be exported to OBS, see **Table 6-2**.

Table 6-2 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	*precision* ≤ **38**: DECIMAL; *precision* > **38**: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)][WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	*n* ≤ **255**: CHAR(n); *n* > **255**: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	*n* ≤ **65535**: VARCHAR(n); *n* > **65535**: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY
Monetary type	MONEY	NUMERIC	BIGINT

For details about HDFS data export or MRS configuration, see the **MapReduce Service User Guide**.

NOTE

This section uses the ORC format as an example to describe how to export data. The method for exporting Parquet data is similar. Parquet data can be exported from clusters of version 9.1.0 or later.

6.1.3.2 Creating a Foreign Server

Create an OBS foreign server and an HDFS foreign server.

Create an OBS external server to define the OBS server information for the foreign table. The creation method varies by cluster version. For details, see **Creating a Foreign Server**.

- For 8.2.0 and later versions, delegate access to OBS bucket data by creating an OBS data source on the OBS console.
- For versions before 8.2.0, you will need to manually create an OBS server. For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

For how to create an HDFS foreign server, see **Manually Creating a Foreign Server**.

6.1.3.3 Creating a Foreign Table

After operations in **Creating a Foreign Server** are complete, create an OBS/HDFS write-only foreign table in the DWS database to access data stored in OBS/HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
   [ { [CONSTRAINT constraint_name] NULL |
   [CONSTRAINT constraint_name] NOT NULL |
     column_constraint [...]} ] |
     table_constraint [, ...]} [, ...] ] )
   SERVER dfs_server
   OPTIONS ( { option_name ' value ' } [, ...] )
   [ {WRITE ONLY }]
   DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
   [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *product_info_ext_obs*, set parameters in the syntax as follows:

- **table_name**

  Specifies the name of the foreign table to be created.
- **Table column definitions**
  - **column_name**: specifies the name of a column in the foreign table.
  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).
- **SERVER dfs_server**

  Specifies the foreign server name of the foreign table. This server must exist. The foreign table connects to OBS/HDFS to read data through the foreign server.

  Enter the name of the foreign server created by following steps in **Creating a Foreign Server**.
- **OPTIONS parameters**

  These are parameters associated with the foreign table. The key parameters are as follows:
  - **format**: specifies the format of the exported data file. The ORC and Parquet formats are supported.

- **foldername**: (mandatory) specifies the data source file directory in the foreign table. OBS: specifies the OBS path of the data source file. You only need to enter */Bucket name/Folder directory level/*. HDFS: specifies the path in the HDFS file system. This parameter is mandatory for the write-only foreign table.
- **encoding**: specifies the encoding of the data source file in the foreign table. The default value is **utf8**.
- **filesize**

  Specifies the file size of a write-only foreign table, in MB. If this parameter is not specified, the file size in the distributed file system configuration is used by default. This syntax is available only for the write-only foreign table.

  Value range: an integer ranging from 1 to 1024

  NOTE

  The **filesize** parameter is valid only for the ORC-formatted write-only HDFS foreign table.
- **compression**

  (Optional) Specifies the compression mode of files. This syntax is available only for the write-only foreign table.

  Value range: **zlib**, **snappy**, and **lz4**

  Specifies the compression mode of ORC files. The default value is **zlib**. Specifies the compression mode of Parquet files. The default value is **snappy**.
- **version**

  (Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.

  Value range: Only **0.12** is supported. The default value is **0.12**.
- **dataencoding**

  (Optional) Specifies the data code of the data table to be exported when the database code is different from the data code of the data table. For example, the database code is Latin-1, but the data in the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding format is used by default. This syntax is valid only for the write-only HDFS foreign table.

  Value range: data code types supported by the database encoding

  NOTE

  The **dataencoding** parameter is valid only for the ORC-formatted write-only HDFS foreign table.

- **Other parameters in the syntax**

  Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

  Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

-- Create an OBS foreign table that does not contain partition columns. The foreign server associated
```


with the table is **obs_server**, the file format on OBS corresponding to the table is ORC, and the data storage path on OBS is**/mybucket/data/**.

```
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
  compression 'snappy',
   version '0.12'
) Write Only;
```

6.1.3.4 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1**: Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

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

  If information similar to the following is displayed, the data has been exported.

  ```
  INSERT 0 10
  ```

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
  ```

  NOTE

  Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

6.2 Exporting ORC and Parquet Data to MRS

6.2.1 Overview

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

- The CN only plans data export tasks and delivers the tasks to DNs for execution. In this case, the CN is released to process external requests.
- Every DN is involved in data export, and the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.
- Multiple HDFS servers can export data concurrently. The export path can be empty. The naming rule of the path must be the same as that of the exported file.
- MRS connects to cluster nodes. The export rate is affected by the network bandwidth.
- Data files in the ORC or Parquet format are supported.

NOTE

This section uses the ORC format as an example to describe how to export data. The method for exporting Parquet data is similar. Parquet data can be exported from clusters of version 9.1.0 or later.

Naming Rules of Exported Files

The rules for naming ORC and Parquet data files exported from DWS are as follows:

1. Data exported to MRS (HDFS): When data is exported from a DN, the data is stored in HDFS in the segment format. The file is named in the format of **mpp**_*Database name*_*Schema name*_*Table name*_*Node name*_*n*_*UUID*.*Data format*. *n* is a natural number starting from 0 in ascending order, for example, 0, 1, 2, 3. The UUID should be a standard one, comprising of 32 hexadecimal characters and divided into five segments by hyphens (-). The format should be 8-4-4-4-12.
2. You are advised to export data from different clusters or databases to different paths. The maximum size of a single file in ORC or Parquet format is about 256 MB. (This is a soft constraint and may exceed a little in actual services.)
3. After the export is complete, the **_SUCCESS** file is generated.

6.2.2 Planning Data Export

For details about the data types that can be exported to MRS, see **Table 6-3**.

Table 6-3 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	*precision* ≤ **38**: DECIMAL; *precision* > **38**: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)][WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	*n* ≤ **255**: CHAR(n); *n* > **255**: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	*n* ≤ **65535**: VARCHAR(n); *n* > **65535**: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY
Monetary type	MONEY	NUMERIC	BIGINT

For details about HDFS data export or MRS configuration, see the **MapReduce Service User Guide**.

6.2.3 Creating a Foreign Server

To create a foreign server for HDFS, you need to create a user and database and provide the necessary permissions for the foreign table.

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the **gsql** client to connect to the database by running the following command:


```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

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

Switch to the created user.

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

Run the following command to create a database:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit |
datlastsysoid | datfrozenxid | dattablespace | datcompatibility |                    datacl

------------+--------+----------+------------+----------+---------------+--------------+--------------+---------------
+--------------+---------------+------------------+-------------------------------------
--------------
 template1  |     10 |        0 | C          | C        | t             | t            |           -1 |        14146 |        1351
|          1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 template0  |     10 |        0 | C          | C        | t             | f            |           -1 |        14146 |        1350
|          1663 | ORA             | {=c/Ruby,Ruby=CTc/Ruby}
 gaussdb    |     10 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1352 |
1663 | ORA             | {=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase |  17000 |        0 | C          | C        | f             | t            |           -1 |        14146 |        1351
|          1663 | ORA             |
(4 rows)
```

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password as prompted.

> **NOTE**
>
> Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:


```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

```
 rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvalidbegin |
rolvaliduntil | memberof | rolreplication | rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+--------------+---------------
+---------------+----------+----------------+---------------+----------------+----------
 dbuser    | f        | t          | f             | t           | t           |           -1 |               |               | {}       | f
| f             | f             | t
 lily      | f        | t          | f             | f           | t           |           -1 |               |               | {}       | f             |
f             | f             | f
 Ruby      | t        | t          | t             | t           | t           |           -1 |               |               | {}       | t
| t             | t             | t
```

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  ```
  \c postgres dbadmin;
  ```
  Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:
  ```
  gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
  ```

Step 2 Run the following command to query the information about the foreign server that is automatically created:

```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                srvoptions
----------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------
 gsmpp_server                           |       10 |  13673 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
```



```
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(2 rows)
```

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in **Creating a User and a Database and Granting the User Foreign Table Permissions** to connect to **mydatabase** created by the user:

```
\c mydatabase dbuser;
```

Step 4 Create a foreign server.

For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*. For example:

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692cahdfs_server FOREIGN DATA
WRAPPER HDFS_FDW
OPTIONS
(
address '192.168.1.245:25000,192.168.1.218:25000',
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*

  You can customize a name.

  In this example, specify the name to the value of the **srvname** field recorded in **Step 2**, such as *hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*.

  Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.

- **FOREIGN DATA WRAPPER**

  This parameter can only be set to **HDFS_FDW**, which already exists in the database.

- **OPTIONS** parameters

  Set the following parameters to the values under **srvoptions** recorded in **Step 2**.

  – address

    Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.

- hdfscfgpath

  Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.

- type

  Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

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

The server is successfully created if the returned result is as follows:

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                       srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |        |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----End

6.2.4 Creating a Foreign Table

After operations in **Creating a Foreign Server** are complete, create an HDFS write-only foreign table in the DWS database to access data stored in HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
   [ { [CONSTRAINT constraint_name] NULL |
   [CONSTRAINT constraint_name] NOT NULL |
     column_constraint [...]} ] |
     table_constraint [, ...]} [, ...] ] )
   SERVER dfs_server
   OPTIONS ( { option_name ' value ' } [, ...] )
   [ {WRITE ONLY }]
   DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
   [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table *product_info_ext_obs*, configure the parameters in the syntax as follows.

- **table_name**

  Specifies the name of the foreign table.

- **Table column definitions**

  - **column_name**: specifies the name of a column in the foreign table.
  - **type_name**: specifies the data type of the column.

  Multiple columns are separated by commas (,).

- **SERVER dfs_server**

  Specifies the foreign server name of the foreign table. This server must exist. The foreign table connects to OBS/HDFS to read data through the foreign server.

Enter the name of the foreign server created in **Creating a Foreign Server**.

- **OPTIONS parameters**

  These parameters are associated with the foreign table. The key parameters are as follows:

  - **format**: specifies the format of the exported data file. The ORC and Parquet formats are supported.
  - **foldername**: specifies the directory of the data source file in the foreign table, that is, the corresponding file directory in HDFS. This parameter is mandatory for write-only foreign tables and optional for read-only foreign tables.
  - **encoding**: specifies the encoding format of the data source file in the foreign table. The default value is **utf8**.
  - **filesize**

    (Optional) Specifies the file size of a write-only foreign table. If this parameter is not specified, the file size in the distributed file system is used by default. This syntax is available only for the write-only foreign table.

    Value range: an integer ranging from 1 to 1024

    NOTE

    The **filesize** parameter is valid only for the write-only HDFS foreign table in ORC format.
  - **compression**

    (Optional) Specifies the compression mode of files. This syntax is available only for the write-only foreign table.

    Value range: **zlib**, **snappy**, and **lz4**.

    Specifies the compression mode of ORC files. The default value is **zlib**. Specifies the compression mode of Parquet files. The default value is **snappy**.
  - **version**

    (Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.

    Value range: Only **0.12** is supported. The default value is **0.12**.
  - **dataencoding**

    (Optional) Specifies the data encoding of the data table to be exported when the database encoding is different from the data encoding of the data table. For example, the database encoding is Latin-1, but the data encoding of the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding is used by default. This syntax is valid only for the write-only HDFS foreign table.

    Value range: data encoding types supported by the database encoding

    NOTE

    The **dataencoding** parameter is valid only for the write-only HDFS foreign table in ORC format.

- **Other parameters in the syntax**

Other parameters are optional and can be configured as required. In this example, they do not need to be configured. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

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

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

6.2.5 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1**: Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

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

  If information similar to the following is displayed, the data has been exported.

  ```
  INSERT 0 10
  ```

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
  ```

  NOTE

  Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

6.3 Using GDS to Export Data to a Remote Server

6.3.1 Exporting Data In Parallel Using GDS

In high-concurrency scenarios, you can use GDS to export data from a database to a common file system.

In the current GDS version, data can be exported from a database to a pipe file.

- When the local disk space of the GDS user is insufficient:
  - The data exported from GDS is compressed using the pipe to occupy less disk space.
  - The exported data is transferred through the pipe to the HDFS server for storage.
- If you need to cleanse data before exporting data:
  - You can compile programs as needed and read streaming data from pipes in real time.

  NOTE

  - The current version does not support data export through GDS in SSL mode. Do not use GDS in SSL mode.
  - All pipe files mentioned in this section refer to named pipes on Linux.
- To ensure the correctness of data import or export using GDS, you need to import or export data in the same compatibility mode.

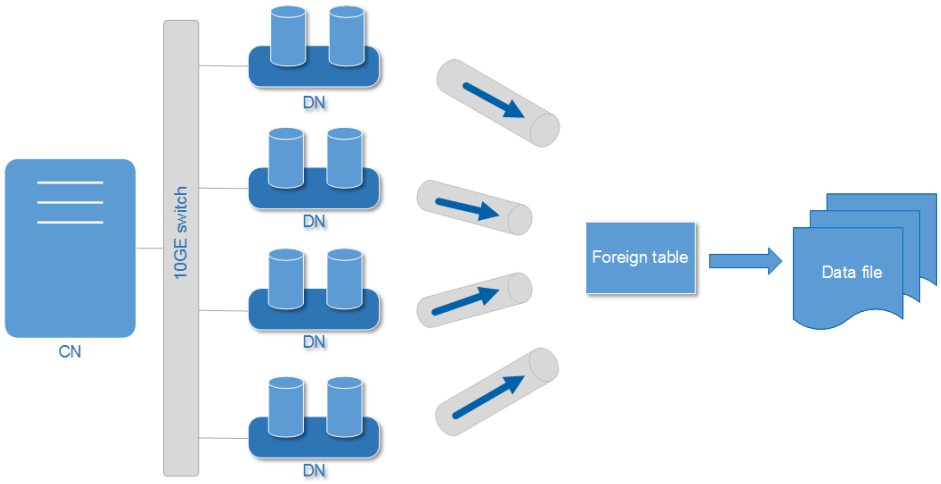
  If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

Using foreign tables: A GDS foreign table specifies the exported file format and export mode. Data is exported in parallel through multiple DNs from the database to data files, which improves the overall data export performance. The data files cannot be directly exported to HDFS.

- The CN only plans data export tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.

Figure 6-3 Exporting data using foreign tables



Related Concepts

- **Data file**: A TEXT, CSV, or FIXED file that stores data exported from the DWS database.
- **Foreign table**: A table that stores information, such as the format, location, and encoding format of a data file.
- **GDS**: A data service tool. To export data, deploy it on the server where data files are stored.
- **Table**: Tables in the database, including row-store tables and column-store tables. Data in the data files is exported from these tables.
- **Remote mode**: Service data in a cluster is exported to hosts outside the cluster.

Exporting a Schema

Data can be exported to DWS in **Remote** mode.

- **Remote mode**: Service data in a cluster is exported to hosts outside the cluster.
  - In this mode, multiple GDSs are used to concurrently export data. One GDS can export data for only one cluster at a time.
  - The data export rate of a GDS that resides on the same intranet as cluster nodes is limited by the network bandwidth. A 10GE configuration is recommended.
  - Data files in TEXT, FIXED, or CSV format are supported. The size of data in a single row must be less than 1 GB.

Data Export Process

Figure 6-4 Concurrent data export

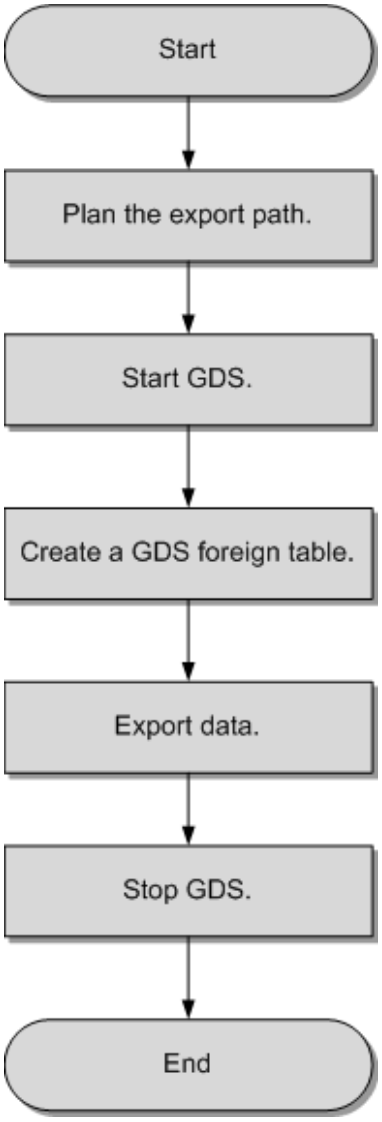


Table 6-4 Process description

Process	Description	Subtask
Plan data export.	Prepare data to be exported and plan the export path for the mode to be selected. For details, see **Planning Data Export**.	-

Process	Description	Subtask
Start GDS.	If the **Remote** mode is selected, install, configure, and start GDS on data servers. For details, see **Installing, Configuring, and Starting GDS**.	-
Create a foreign table,	Create a foreign table to help GDS specify information about a data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a data file. For details, see **Creating a GDS Foreign Table**.	-
Export data.	After the foreign table is created, run the **INSERT** statement to efficiently export data to data files. For details, see **Exporting Data**.	-
Stop GDS.	Stop GDS after data is exported. For details, see **Stopping GDS**.	-

6.3.2 Planning Data Export

Scenarios

Before you use GDS to export data from a cluster, prepare data to be exported and plan the export path.

Planning an Export Path

- **Remote** mode

Step 1 Log in to the GDS data server as user **root** and create the **/output_data** directory for storing data files.

```
mkdir -p /output_data
```

Step 2 (Optional) Create a user and the user group to which it belongs. This user is used to start GDS and must have the write permission on the directory for storing data files.

```
groupadd gdsgrp
useradd -g gdsgrp gdsuser
```

If the following information is displayed, the user and user group already exist. Skip this step.

```
useradd: Account 'gdsuser' already exists.
groupadd: Group 'gdsgrp' already exists.
```


Step 3 Change the directory owner to **gdsuser**.

```
chown -R gdsuser:gdsgrp /output_data
```

----End

6.3.3 Installing, Configuring, and Starting GDS

GDS is a data service tool provided by DWS. Using the foreign table mechanism, this tool helps export data at a high speed.

To install, configure, and start GDS, perform the following steps. For details, see **Installing, Configuring, and Starting GDS**.

Procedure

Step 1 Before using GDS to import or export data, see "Preparing an ECS as the GDS Server" and "Downloading the GDS Package" in **Tutorial: Using GDS to Import Data from a Remote Server**.

Step 2 Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

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

Step 3 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.***x.x*_**suse_x64.zip** to the directory created in the previous step.

Step 4 (Optional) If SSL is used, upload the SSL certificates to the directory created in **Step 2**.

Step 5 Go to the directory and extract the package.

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

Step 6 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

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

Step 7 Change the owner of the GDS package directory and source data file directory to the GDS user.

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

Step 8 Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.*x* or earlier, skip **Step 9** and go to **Step 10**.

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

Step 9 Execute the script on which the environment depends (applicable only to version 8.1.*x*).

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

Step 10 Start GDS.

GDS is green software and can be started after being decompressed. There are two ways to start GDS.

Method 1: Run the **gds** command to set startup parameters.

Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.
  - If data is transmitted in non-SSL mode, run the following command to start GDS:
    ```
    gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
    ```
    Example:
    ```
    /opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
    l /opt/bin/dws/gds/gds_log.txt -D -t 2
    ```
  - If data is transmitted in SSL mode, run the following command to start GDS:
    ```
    gds -d dir -p ip:port -H address_string -l log_file -D
    -t worker_num --enable-ssl --ssl-dir Cert_file
    ```
    Example:

    Run the following commands to upload the SSL certificates mentioned in **Step 4** to **/opt/bin**:
    ```
    /opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
    l /opt/bin/dws/gds/gds_log.txt -D --enable-ssl --ssl-dir /opt/bin/
    ```

  Replace the information in italic as required.

  - **-d** *dir*: directory to store data files that contain data to import. This tutorial uses **/input_data/** as an example.
  - **-p** *ip:port*: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with DWS. The port number ranges from 1024 to 65535. The default value is **8098**. This tutorial uses **192.168.0.90:5000** as an example.
  - **-H** *address_string*: servers that are allowed to connect to and use GDS. The value must be in CIDR format. Set this parameter to enable a DWS cluster to access GDS for data import. Ensure that the network segment covers all hosts in a DWS cluster.
  - **-l** *log_file*: GDS log directory and log file name. This tutorial uses **/opt/bin/dws/gds/gds_log.txt** as an example.
  - **-D**: GDS in daemon mode. This command can only be used in Linux.
  - **-t** *worker_num*: number of concurrent GDS threads. If the data server and DWS have available I/O resources, you can increase the number of concurrent GDS threads.

    GDS determines the number of threads based on the number of parallel import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.
  - **--enable-ssl**: enables SSL for data transmission.

  - **--ssl-dir** *Cert_file*: SSL certificate directory. Set this parameter to the certificate directory in **Step 4**.
  - For details about GDS parameters, see **gds**.

- Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

  a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see **Table 2-7**.

  ```
  vim /opt/bin/dws/gds/config/gds.conf
  ```

  Example:

  Configure the **gds.conf** file as follows:

  ```
  <?xml version="1.0"?>
  <config>
  <gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/" err_dir="/err"
  data_seg="100MB" err_seg="100MB" log_file="/log/gds_log.txt" host="10.10.0.1/24"
  daemon='true' recursive="true" parallel="32"></gds>
  </config>
  ```

  Information in the configuration file is as follows:

  - The data server IP address is **192.168.0.90**, and the GDS listening port is **5000**.
  - Data files are stored in **/input_data/**.
  - Error log files are stored in **/err**. This directory must be created by a user with GDS read and write permissions.
  - The size of a single data file is 100 MB.
  - The size of each error log file is 100 MB.
  - Logs are stored in the **/log/gds_log.txt** file. This directory must be created by a user with GDS read and write permissions.
  - Only nodes with the IP address **10.10.0.***** can be connected.
  - The GDS process runs in the background.
  - Data file directories are recursive.
  - The number of concurrent import threads is 2.

  b. Start GDS and check whether it has been started.

  ```
  python3 gds_ctl.py start
  ```

  Example:

  ```
  cd /opt/bin/dws/gds/bin
  python3 gds_ctl.py start
  Start GDS gds1                [OK]
  gds [options]:
   -d dir           Set data directory.
   -p port           Set GDS listening port.
     ip:port        Set GDS listening ip address and port.
   -l log_file       Set log file.
   -H secure_ip_range
                  Set secure IP checklist in CIDR notation. Required for GDS to start.
   -e dir           Set error log directory.
   -E size          Set size of per error log segment.(0 < size < 1TB)
   -S size          Set size of data segment.(1MB < size < 100TB)
  ```



```
-t worker_num     Set number of worker thread in multi-thread mode, the upper limit is 200. If
without setting, the default value is 8.
 -s status_file    Enable GDS status report.
 -D                Run the GDS as a daemon process.
 -r                Read the working directory recursively.
 -h                Display usage.
```

----End

6.3.4 Creating a GDS Foreign Table

Procedure

Step 1 Set the **location** parameter for the foreign table based on the path planned in **Planning Data Export**.

- **Remote** mode

  Set the **location** parameter to the URL of the directory that stores the data files.

  - You do not need to specify a file name in the URL.
  - When the number of data sources exported is fewer than the available paths, files are created for the surplus paths without writing any data.

  For example:

  The IP address of the GDS data server is 192.168.0.90. The listening port number set during GDS startup is 5000. The directory for storing data files is **/output_data**.

  In this case, set the **location** parameter to **gsfs://192.168.0.90:5000/**.

  NOTE

  - By setting **location** to a subdirectory, for example, **gsfs://192.168.0.90:5000/2019/11/**, you can export the same table to different directories based on the date.
  - In the current version, the system checks if the **/output_data/2019/11** directory exists when an export task is executed. If it does not exist, the system creates it. During the export, files are written to this directory. In this way, you do not need to manually run the **mkdir -p /output_data/2019/11** command after creating or modifying a foreign table.

Step 2 Set data format parameters in the foreign table based on the planned data file formats.

Step 3 Create a GDS foreign table based on the parameter settings in the preceding steps.

----End

Example

- **Example**: Create the GDS foreign table **foreign_tpcds_reasons** for the source data. Data is to be exported as CSV files.

  Data export mode settings are as follows:

  The data server resides on the same intranet as the cluster. The IP address of the data server is 192.168.0.90. Data is to be exported as CSV files. The **Remote** mode is selected for parallel data export.

Assume that the directory for storing data files is **/output_data/** and the GDS listening port is 5000 when GDS is started. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **E'\x1b'**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to the same value as that of **quote** by default.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.
- **EOL** is set to **0X0A**.

The foreign table is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk    integer        not null,
  r_reason_id    char(16)       not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL '',
EOL '0x0a'
)
WRITE ONLY;
```

6.3.5 Exporting Data

Prerequisites

Ensure that the IP addresses and ports of servers where CNs and DNs are deployed can connect to those of the GDS server.

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

NOTE

Create batch processing scripts to export data in parallel. The degree of parallelism depends on the server resource usage. You can test several tables and monitor resource usage to determine whether to increase or reduce the amount. Common resource monitoring commands include **top** for memory and CPU usage, **iostat** for I/O usage, and **sar** for networks. For details about application cases, see **Exporting Data Using Multiple Threads**.

Examples

- **Example 1**: Export data from the **reason** table to data files through the **foreign_tpcds_reasons** foreign table.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```


- **Example 2**: Export part of the data to data files by specifying the filter condition **r_reason_sk =1**.
  ```
  INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason WHERE r_reason_sk=1;
  ```
- **Example 3**: Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to hexadecimal the format before export.
  ```
  INSERT INTO foreign_tpcds_reasons SELECT RAWTOHEX(c) FROM tpcds.reason;
  ```

6.3.6 Stopping GDS

GDS is a data service tool provided by DWS that enables high-speed data export through coordination with external mechanisms.

If GDS is no longer used, perform the following steps to stop it:

Procedure

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

Step 2 Select the mode of stopping GDS based on the mode of starting it.

- If GDS is started using the **gds** command, perform the following operations to stop GDS:
  a. Query the GDS process ID:
     ```
     ps -ef|grep gds
     ```
     For example, the GDS process ID is 128954.
     ```
     ps -ef|grep gds
     gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/gds_log.txt -D
     gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
     ```
  b. Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.
     ```
     kill -9 128954
     ```
- If GDS is started using the **gds_ctl.py** command, run the following commands to stop GDS:
  ```
  cd /opt/bin/dws/gds/bin
  python3 gds_ctl.py stop
  ```

----End

6.3.7 Examples of Exporting Data Using GDS

Exporting Data in Remote Mode

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this scenario, data is exported in parallel in **Remote** mode.

To export data in parallel in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create user **gds_user** and its user group.
   ```
   mkdir -p /output_data
   ```
2. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group exist, skip this step.


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

3. Change the owner of the **/output_data** directory on the data server to **gds_user**.

   ```
   chown -R gds_user:gdsgrp /output_data
   ```

4. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode.

   ```
   /opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
   ```

5. In the database, create the foreign table **foreign_tpcds_reasons** for receiving data from the data server.

   Data export mode settings are as follows:

   - The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

   Data format parameter settings are as follows:

   - **format** is set to **CSV**.
   - **encoding** is set to **UTF-8**.
   - **delimiter** is set to **E'\x08'**.
   - **quote** is set to **E'\x1b'**.
   - **null** is set to an empty string without quotation marks.
   - **escape** defaults to the value of **quote**.
   - **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

   Based on the above settings, the foreign table is created using the following statement:

   ```
   CREATE FOREIGN TABLE foreign_tpcds_reasons
   (
     r_reason_sk    integer         not null,
     r_reason_id    char(16)        not null,
     r_reason_desc  char(100)
   )
   SERVER gsmpp_server
   OPTIONS
   (
   LOCATION 'gsfs://192.168.0.90:5000/',
   FORMAT 'CSV',
   ENCODING 'utf8',
   DELIMITER E'\x08',
   QUOTE E'\x1b',
   NULL ''
   )
   WRITE ONLY;
   ```

6. In the database, export data to data files through the foreign table **foreign_tpcds_reasons**.

   ```
   INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
   ```

7. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

   The GDS process ID is **128954**.

   ```
   ps -ef|grep gds
   gds_user 128954     1  0 15:03 ?        00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D
   ```



```
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Exporting Data Using Multiple Threads

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this scenario, data is concurrently exported to two target tables using multiple threads in **Remote** mode.

To concurrently export data using multiple threads in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create the database user and its user group.
   ```
   mkdir -p /output_data
   groupadd gdsgrp
   useradd -g gdsgrp gds_user
   ```
2. Change the owner of the **/output_data** directory on the data server to **gds_user**.
   ```
   chown -R gds_user:gdsgrp /output_data
   ```
3. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2.
   ```
   /opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2
   ```
4. In DWS, create the foreign tables **foreign_tpcds_reasons1** and **foreign_tpcds_reasons2** for receiving data from the data server.
   - Data export mode settings are as follows:
     - The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.
   - Data format parameter settings are as follows:
     - **format** is set to **CSV**.
     - **encoding** is set to **UTF-8**.
     - **delimiter** is set to **E'\x08'**.
     - **quote** is set to **E'\x1b'**.
     - **null** is set to an empty string without quotation marks.
     - **escape** defaults to the value of **quote**.
     - **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

   Based on the preceding settings, the foreign table **foreign_tpcds_reasons1** is created using the following statement:
   ```
   CREATE FOREIGN TABLE foreign_tpcds_reasons1
   (
   ```



```
 r_reason_sk    integer     not null,
 r_reason_id    char(16)    not null,
 r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

Based on the preceding settings, the foreign table **foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons2
(
 r_reason_sk    integer     not null,
 r_reason_id    char(16)    not null,
 r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

5. Export data from table **reasons1** to **foreign table foreign_tpcds_reasons1** and from table **reasons2** to **foreign table foreign_tpcds_reasons2** in the database, and save the exported data to the **/output_data** directory.

```
INSERT INTO foreign_tpcds_reasons1 SELECT * FROM tpcds.reason;
INSERT INTO foreign_tpcds_reasons2 SELECT * FROM tpcds.reason;
```

6. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

   The GDS process ID is **128954**.

```
ps -ef|grep gds
gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D -t 2
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Exporting Data Through a Pipe

Step 1 Start GDS.

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database, create an internal table, and write data to the table.

```
CREATE TABLE test_pipe( id integer not null, gender text not null, name text ) ;



INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
```



```
INSERT INTO test_pipe values(4,2,'11111111111111');
INSERT 0 1
```

2. Create a write-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name  text )
SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/', FORMAT 'text', DELIMITER ',',
NULL '', EOL '0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE ONLY;
```

3. Execute the export statement. In this case, the statements are blocked.

```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to the GDS data directory.

```
cd /***/gds_data/
```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.

```
mkfifo postgres_public_foreign_test_pipe_tw.pipe
```

> NOTE
>
> A pipe will be automatically cleared after an operation is complete. To perform another operation, create a pipe file again.

3. Read data from the pipe and write it to a new file.

```
cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt
```

4. To compress the exported files, run the following command:

```
gzip -9 -c < postgres_public_foreign_test_pipe_tw.pipe  > out.gz
```

5. To export the content from the pipe to the HDFS server, run the following command:

```
cat postgres_public_foreign_test_pipe_tw.pipe  | hdfs dfs -put -  /user/hive/***/test_pipe.txt
```

Step 4 Verify the exported data.

1. Check whether the exported file is correct.

```
cat postgres_public_foreign_test_pipe_tw.txt
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

2. View the compressed file.

```
vim out.gz
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

3. View the data exported to the HDFS server.

```
hdfs dfs -cat /user/hive/***/test_pipe.txt
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

----End

Exporting Data Through Multi-Process Pipes

GDS also supports importing and exporting data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes exporting a local file as an example.

Step 1 Start multiple GDSs.

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```


If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database and create an internal table.
```
CREATE TABLE test_pipe (id integer not null, gender text not null, name  text);
```
2. Write data.
```
INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
INSERT INTO test_pipe values(4,2,'11111111111111');
```
3. Create a write-only foreign table.
```
CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name  text )
SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/|gsfs://192.168.0.1:7790/',
FORMAT 'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE
ONLY;
```
4. Execute the export statement. In this case, the statements are blocked.
```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.
```
cd /***/gds_data/
cd /***/gds_data_1/
```
2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
```
mkfifo postgres_public_foreign_test_pipe_tw.pipe
```
3. Read each pipe and write the new file to the pipes.
```
cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt
```

Step 4 Verify the exported data.
```
cat /***/gds_data/postgres_public_foreign_test_pipe_tw.txt
3,2,11111111111111
cat /***/gds_data_1/postgres_public_foreign_test_pipe_tw.txt
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

----End

7 Other Operations

7.1 GDS Pipe FAQs

Precautions

- GDS supports concurrent import and export. The **gds -t** parameter is used to set the size of the thread pool and control the maximum number of concurrent working threads. But it does not accelerate a single SQL task. The default value of **gds -t** is **8**, and the upper limit is **200**. When using the pipe function to import and export data, ensure that the value of **-t** is greater than or equal to the number of concurrent services. In the dual-cluster interconnection scenario, the value of **-t** must be greater than or equal to twice the number of concurrent services.
- Data in pipes is deleted once read. Therefore, ensure that no other program except GDS reads data in the pipe during import or export. Otherwise, data may be lost, task errors may occur, or the exported files may be disordered.
- Concurrent import and export of foreign tables with the same location are not supported. That is, multiple threads of GDS cannot read or write pipe files at the same time.
- A single import or export task of GDS identifies only one pipe. Therefore, do not carry wildcard characters ({}[]?) in the location address set for the GDS foreign table. Example:

  ```
  CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_*', FORMAT 'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
  ```

- When the **-r** recursion parameter is enabled for GDS, only one pipe can be identified. That is, GDS identifies only one pipe in the current data directory and does not recursively search for it. Therefore, the **-r** parameter does not take effect in the pipe import and export scenarios.
- CN retry is not supported during the import and export through a pipe, because GDS cannot control the operations performed by peer users and programs on pipes.
- During the import, if the peer program does not write data into the pipe for more than one hour, the import task times out and an error is reported.

- During the export, if the peer program does not read data from the pipe for more than one hour by default, the export task times out and an error is reported.
- Ensure that the GDS version and kernel version support the function of importing and exporting data through pipes.
- If the **auto_create_pipe parameter** of the foreign table is set to **true**, a delay may occur when GDS automatically creates a pipe. Before any operation on a pipe, check whether the automatically created pipe exists and whether it is a pipe file.
- Once an import or export task through a GDS pipe is complete, the pipe is automatically deleted. However, the pipe deletion is delayed, if you manually terminate an import or export task. In this situation, the pipe is deleted after the timeout interval expires.

Common Troubleshooting Methods:

- Issue 1: **"/***/postgres_public_foreign_test_pipe_tr.pipe" must be named pipe.**

  Locating method: The type of the GDS foreign table **file_type** is pipe, but the operated file is a common file. Check whether the **postgres_public_foreign_test_pipe_tr.pipe** file is a pipe file.
- Issue 2: **could not open pipe "/***/ postgres_public_foreign_test_pipe_tw.pipe" cause by Permission denied**.

  Locating method: GDS does not have the permission to open the pipe file.
- Issue 3: **could not open source file /*****/ postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for WRITING**.

  Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not read by any program within 300 seconds.
- Issue 4: **could not open source file /*****/ postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for READING**.

  Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not written by any program within 300 seconds.
- Issue 5: **could not poll writing source pipe file "/****/ postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**

  Locating method: If the GDS does not receive any write event on the pipe within 300 seconds during data export, the pipe is not read for more than 300 seconds.
- Issue 6: **could not poll reading source pipe file "/****/ postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**

  Locating method: If the GDS does not receive any read event on the pipe within 300 seconds during data import, the pipe is not written for more than 300 seconds.

- Issue 7: **could not open pipe file "/***/ postgres_public_foreign_test_pipe_tw.pipe" for "WRITING" with error No such device or address**.

  Locating method: It indicates that the **/***/ **postgres_public_foreign_test_pipe_tw.pipe** file is not read by any program. As a result, GDS cannot open the pipe file by writing.

7.2 Checking for Data Skew

Scenarios

Data skew causes the query performance to deteriorate. Before importing all the data from a table consisting of over 10 million records, you are advised to import some of the data and check whether data skew occurs and whether the distribution keys need to be changed. Troubleshoot the problems if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Background

DWS uses a massively parallel processing (MPP) system of the shared-nothing architecture. The MPP performs horizontal partitioning to store tuples in service data tables on all DNs using proper distribution policies.

The following user table distribution policies are supported:

- Replication: stores a full table on each DN. You are advised to use the replication mode for tables containing a small volume of data.
- Hash: A distribution key must be specified for a user table. If a record is inserted, the system performs hash computing based on values in the distribute column and then stores data on the related DN. You are advised to use the hash distribution policy for tables with a large volume of data.
- Round-robin: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. If no proper distribution column can be found in a table with a large amount of data in hash mode, you are advised to use the round-robin distribution policy.

If an inappropriate distribution key is used, data skew may occur when you use the hash policy. Check for data skew when you use the hash distribution policy so that data can be evenly distributed to each DN. You are advised to use the column with few replicated values as the distribution key.

Procedure

Step 1 Analyze data source features and select candidate distribution columns that have more distinct values and evenly distributed data.

Step 2 Select a candidate column from **Step 1** to create a target table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    ({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint    | LIKE source_table [ like_option [...] ] }
    [, ... ])    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
```



```
[ COMPRESS | NOCOMPRESS ]    [ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION
                | ROUNDROBIN
                | { HASH ( column_name [,...] ) } } ];
```

Step 3 Import a small batch of data to the target table.

When importing a single data file, you can evenly split this file and import a part of it to check for the data skew in the target table.

Step 4 Check for data skew. (Replace *table_name* with the actual table name.)

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

Step 5 If the data distribution deviation is less than 10% across DNs, data is evenly distributed and an appropriate distribution key has been selected. Delete the small batch of imported data and import full data to complete data migration.

If data distribution deviation across DNs is greater than or equal to 10%, data skew occurs. Remove this distribution key from the candidates in **Step 1**, delete the target table, and repeat **Step 2** through **Step 5**.

NOTE

The data distribution deviation indicates the difference between the actual data volume on DNs and the average data volume on DNs. You can view the distribution difference in the **PGXC_GET_TABLE_SKEWNESS** view.

Step 6 (Optional) If you fail to select an appropriate distribution key after performing the preceding steps, select multiple columns from the candidates as distribution keys.

----End

Examples

Assume you want to select an appropriate distribution key for the **staffs** table.

1. Analyze the source data for the **staffs** table and select the **staff_ID**, **FIRST_NAME**, and **LAST_NAME** columns as candidate distribution keys.
2. Select the **staff_ID** column as the distribution key and create the target table **staffs**.

```
CREATE TABLE staffs
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(2,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID);
```

3. Import a small batch of data to the target table **staffs**.

   There are eight DNs in the cluster based on the following query, and you are advised to import 80,000 records.

```
SELECT count(*) FROM pgxc_node where node_type='D';
 count
```



```
-------
    8
(1 row)
```

4. Verify the data skew of the target table **staffs** whose distribution key is **staff_ID**:

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
------+-----------
11010 | datanode4
10000 | datanode3
12001 | datanode2
 8995 | datanode1
10000 | datanode5
 7999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

5. The preceding query result indicates that the distribution deviation across DNs is greater than 10%. The data skew occurs. Therefore, delete **staff_ID** from the distribution key candidates and delete the **staffs** table.

```
DROP TABLE staffs;
```

6. Use **staff_ID**, **FIRST_NAME**, and **LAST_NAME** as distribution keys and create the target table **staffs**.

```
CREATE TABLE staffs
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(2,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID,FIRST_NAME,LAST_NAME);
```

7. Verify the data skew of the target table **staffs** whose distribution keys are **staff_ID**, **FIRST_NAME**, and **LAST_NAME**.

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
------+-----------
10010 | datanode4
10000 | datanode3
10001 | datanode2
 9995 | datanode1
10000 | datanode5
 9999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

8. The preceding query result indicates that the data deviation across DNs is less than 10%. The data is evenly distributed and the appropriate distribution keys have been selected.
9. Delete the imported small-batch data.

```
TRUNCATE TABLE staffs;
```

10. Import the full data to complete data migration.

7.3 Analyzing a Table

The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the **ANALYZE** statement to update table statistics. The statistics are stored in the **PG_STATISTIC** system catalog.

Analyzing a Table

ANALYZE supports row-store, column-store, HDFS, and OBS tables in ORC or CARBONDATA format. **ANALYZE** can also collect statistics about specified columns of a local table.

Do **ANALYZE** to the **product_info** table.

```
ANALYZE product_info;
```

Automatically Analyzing a Table

DWS provides automatic table analysis for the following two scenarios.

- If **ANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold, and the execution plan does not use Fast Query Shipping (FQS), the GUC parameter autoanalyze is used to control the automatic collection of table statistics. In this case, a better execution plan is generated based on the collected statistics.
- When autovacuum is set to **on**, the system periodically starts the **autovacuum** thread to automatically collect statistics on tables whose amount of data modification reaches the threshold for triggering **ANALYZE** in the background.

Table 7-1 Automatically Analyzing a Table

Trigger Mode	Trigger Condition	Scaling Frequency	Control Parameter	Remarks
Synchronization	Statistics are completely missing.	At each query execution	autoanalyze	Statistics are cleared when the primary table is truncated.
Synchronization	The amount of modified data reaches the **ANALYZE** threshold.	At each query execution	autoanalyze	ANALYZE is triggered before the optimal plan is determined.

Trigger Mode	Trigger Condition	Scaling Frequency	Control Parameter	Remarks
Asynchronization	The amount of modified data reaches the **ANALYZE** threshold.	Autovacuum thread polling check	autovacuum_mode, autovacuum_naptime	The lock times out in 2 seconds, and the execution times out in 5 minutes.

NOTICE

- The autoanalyze function supports only the default sampling mode and not the percentage sampling mode.
- The autoanalyze function does not collect multi-column statistics, which only supports percentage sampling.
- **AUTOANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold. In this case, **AUTOANALYZE** cannot be triggered for foreign tables or temporary tables with the **ON COMMIT [DELETE ROWS | DROP]** option.
- If the amount of data modification reaches the threshold for triggering **ANALYZE**, the amount of data modification exceeds **autovacuum_analyze_threshold** + **autovacuum_analyze_scale_factor** * *reltuples*. *reltuples* indicates the estimated number of rows in the table recorded in **pg_class**.
- The autoanalyze function triggered by a scheduled **autovacuum** thread supports only row-store and column-store tables. It does not support foreign tables, HDFS tables, OBS foreign tables, temporary tables, unlogged tables, or toast tables.
- When **ANALYZE** is triggered during a query, a level-4 lock is added to all partitions in the partitioned table. The lock is released only after the transaction containing the query is committed. The level-4 lock does not block adding, deletion, modification, and query operations, but blocks partition modification operations such as **TRUNCATE**. You can set **object_mtime_record_mode** to **disable_partition** to release the partition locks in advance.
- The autovacuum function also depends on the following two GUC parameters in addition to **autovacuum**:
  - track_counts must be set to **on** to enable statistics collection about the database.
  - autovacuum_max_workers must be set to a value greater than 0 to specify the maximum number of concurrent autovacuum threads.

7.4 Managing Concurrent Write Operations

7.4.1 Write and Read/Write Operations

Transaction Isolation

DWS manages transactions based on MVCC and two-phase locks, avoiding conflicts between read and write operations. SELECT is a read-only operation, whereas UPDATE and DELETE are read/write operations.

- There is no conflict between read/write and read-only operations, or between read/write operations. Each concurrent transaction creates a snapshot when it starts. Concurrent transactions cannot detect updates made by each other.
  - At the **READ COMMITTED** isolation level, if transaction T1 is committed, transaction T2 can see changes made by T1.
  - At the **REPEATABLE READ** level, if T2 starts before T1 is committed, T2 will not see changes made by T1 even after T1 is committed. The query results in a transaction are consistent and unaffected by other transactions.
- Read/Write operations use row-level locks. Different transactions can concurrently update the same table but not the same row. A row update transaction will start only after the previous one is committed.
  - **READ COMMITTED**: At this level, a transaction can access only committed data. This is the default level.
  - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.

Write and Read/Write Operations

Statements for write-only and read/write operations are as follows:

- **INSERT**: used to insert one or more rows of data into a table.
- **UPDATE**: used to modify existing data in a table.
- **DELETE**: used to delete existing data from a table.
- **COPY**: used to import data.

INSERT and COPY are write-only operations. Only one of them can be performed at a time. If the INSERT or COPY of transaction T1 locks a table, the INSERT or COPY of transaction T2 needs to wait until T1 unlocks the table.

UPDATE and DELETE operations are read/write operations. They need to query for the target rows before modifying data. Concurrent transactions cannot see the changes made by each other, and UPDATE and DELETE operations read snapshots of data committed before their transactions started. Write operations use row-level locks. If T2 starts after T1 and is to update the same row as T1 does, T2 waits for T1 to finish update. If T1 is not complete within the specified timeout duration, T2 will time out. If T1 and T2 updates different rows in a table, they can be concurrently executed.

Potential Deadlocks During Concurrent Write

Whenever transactions involve updates of more than one table, there is always the possibility of concurrently running transactions becoming deadlocked when they

both try to write to the same set of tables. A transaction releases all of its locks at once when it either commits or rolls back; it does not relinquish locks one at a time. For example, suppose that transactions T1 and T2 start at roughly the same time.

- If T1 starts writing to table A and T2 starts writing to table B, both transactions can proceed without conflict; however, if T1 finishes writing to table A and needs to start writing to the same rows as T2 does in table B, it will not be able to proceed because T2 still holds the lock on B. Conversely, if T2 finishes writing to table B and needs to start writing to the same rows as T1 does in table A, it will not be able to proceed either because T1 still holds the lock on A. In this case, a deadlock occurs. If T1 is committed and releases the lock within the lock timeout duration, subsequent update will proceed. If a lock times out, an error will be reported and the corresponding transaction will exit.
- If T1 updates rows 1 to 5 and T2 updates rows 6 to 10 in the same table, the two transactions do not conflict. However, if T1 finishes the update and proceeds to update rows 6 to 10, and T2 proceeds to update rows 1 to 5, neither of them can continue. If either of the transactions is committed and releases the lock within the lock timeout duration, subsequent update will proceed. If a lock times out, an error will be reported and the corresponding transaction will exit.

7.4.2 Concurrent Write Examples

This section uses table **test** as an example to describe how to perform concurrent INSERT and DELETE in the same table, concurrent INSERT in the same table, concurrent UPDATE in the same table, and concurrent import and queries.

```
CREATE TABLE test(id int, name char(50), address varchar(255));
```

Concurrent INSERT and DELETE in the Same Table

Transaction T1:

```
START TRANSACTION;
INSERT INTO test VALUES(1,'test1','test123');
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
DELETE test WHERE NAME='test1';
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, the DELETE of T2 is performed. In this case, **DELETE 0** is displayed, because T1 is not committed and T2 cannot see the data inserted by T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed and the DELETE of T2 is

performed. In this case, **DELETE 1** is displayed, because T2 can see the data inserted by T1.

- **REPEATABLE READ** level

  T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed and the DELETE of T2 is performed. In this case, **DELETE 0** is displayed, because the data obtained in queries is consistent in a transaction.

Concurrent INSERT in the Same table

Transaction T1:

```
START TRANSACTION;
INSERT INTO test VALUES(2,'test2','test123');
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
INSERT INTO test VALUES(3,'test3','test123');
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, the INSERT of T2 is performed and succeeds. At the **READ COMMITTED** and **REPEATABLE READ** levels, the SELECT of T1 cannot see data inserted by T2, and a query in T2 cannot see the data inserted by T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed. In T2, a query performed after INSERT can see the data inserted by T1.

- **REPEATABLE READ** level

  T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed. In T2, a query performed after INSERT cannot see the data inserted by T1.

Concurrent UPDATE in the Same Table

Transaction T1:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test2';
COMMIT;
```

Transaction T3:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```


Scenario 1:

T1 is started but not committed. At the same time, T2 is started. The UPDATE of T1 and then T2 starts, and both of them succeed. This is because UPDATE operations use row-level locks and do not conflict when they update different rows.

Scenario 2:

T1 is started but not committed. At the same time, T3 is started. The UPDATE of T1 and then T3 starts, and the UPDATE of T1 succeeds. The UPDATE of T3 times out. This is because T1 and T3 update the same row, the lock on which is held by the uncommitted T1.

Concurrent Data Import and Queries

Transaction T1:

```
START TRANSACTION;
COPY test FROM '...';
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
SELECT * FROM test;
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. The COPY of T1 and then the SELECT of T2 starts, and both of them succeed. In this case, T2 cannot see the data added by the COPY of T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At the same time, T2 is started. The COPY of T1 is complete and T1 is committed. In this case, T2 can see the data added by the COPY of T1.
- **REPEATABLE READ** level

  T1 is started but not committed. At the same time, T2 is started. The COPY of T1 is complete and T1 is committed. In this case, T2 cannot see the data added by the COPY of T1.

7.5 Updating Data in a Table

7.5.1 Updating a Table by Using DML Statements

In DWS, you can update a table by running DML statements.

Procedure

Assume that there is a table **customer_t** and the table structure is as follows:


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

You can run the following DML statements to update data in the table.

- Run the **INSERT** statement to insert data into the table.
  - Insert a row to the **customer_t** table.
    ```
    INSERT INTO customer_t (c_customer_sk, c_customer_id, c_first_name,c_last_name) VALUES
    (3769, 5, 'Grace','White');
    ```
  - Insert multiple rows to the **customer_t** table.
    ```
    INSERT INTO customer_t (c_customer_sk, c_customer_id, c_first_name,c_last_name) VALUES
    (6885, 1, 'Joes', 'Hunter'),
    (4321, 2, 'Lily','Carter'),
    (9527, 3, 'James', 'Cook'),
    (9500, 4, 'Lucy', 'Baker');
    ```

    For details about how to use INSERT, see "TRUNCATE" in the *SQL Syntax*.
- Run the **UPDATE** statement to update data in the table. Change the value of the **c_customer_id** column to **0**.
  ```
  UPDATE customer_t SET c_customer_id = 0;
  ```

  For details about how to use UPDATE, see "UPDATE" in the *SQL Syntax*.
- Run the **DELETE** statement to delete rows from the table.

  You can use the **WHERE** clause to specify the rows whose data is to be deleted. If you do not specify it, all rows in the table are deleted and only the data structure is retained.
  ```
  DELETE FROM customer_t WHERE c_last_name = 'Baker';
  ```

  For details about how to use DELETE, see "DELETE" in the *SQL Syntax*.
- Run the **TRUNCATE** statement to delete all rows from the table.
  ```
  TRUNCATE TABLE customer_t;
  ```

  For details about how to use TRUNCATE, see "TRUNCATE" in the *SQL Syntax*.

  When deleting data from a table, the **DELETE** statement deletes a row of data each time while the **TRUNCATE** statement deletes data by releasing the data page stored in the table. Therefore, data can be deleted more quickly by using **TRUNCATE** than using **DELETE**.

  DELETE deletes table data but does not release table storage space. **TRUNCATE** deletes table data and releases table storage space.

7.5.2 Updating and Inserting Data by Using the MERGE INTO Statement

To add all or a large amount of data in a table to an existing table, you can run the **MERGE INTO** statement in DWS to merge the two tables so that data can be quickly added to the existing table.

The **MERGE INTO** statement matches data in a source table with that in a target table based on a join condition. If data matches, **UPDATE** will be executed on the target table. Otherwise, **INSERT** will be executed. This statement is a convenient way to combine multiple operations and avoids multiple **INSERT** or **UPDATE** statements.

Prerequisites

You must have the **INSERT** and **UPDATE** permissions for the target table and the **SELECT** permission for the source table.

Procedure

Step 1 Create a source table **products** and insert data.

```
CREATE TABLE products
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

INSERT INTO products VALUES
(1502, 'olympus camera', 'electrncs'),
(1601, 'lamaze', 'toys'),
(1666, 'harry potter', 'toys'),
(1700, 'wait interface', 'books');
```

Step 2 Create a target table **newproducts** and insert data.

```
CREATE TABLE newproducts
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

INSERT INTO newproducts VALUES
(1501, 'vivitar 35mm', 'electrncs'),
(1502, 'olympus ', 'electrncs'),
(1600, 'play gym', 'toys'),
(1601, 'lamaze', 'toys'),
(1666, 'harry potter', 'dvd');
```

Step 3 Run the **MERGE INTO** statement to merge data in the source table **products** into the target table **newproducts**.

```
MERGE INTO newproducts np
USING products p
ON (np.product_id = p.product_id )
WHEN MATCHED THEN
  UPDATE SET np.product_name = p.product_name, np.category = p.category
WHEN NOT MATCHED THEN
  INSERT VALUES (p.product_id, p.product_name, p.category) ;
```

For details about parameters in the statement, see **Table 7-2**. For details, see "MERGE INTO" in the *SQL Syntax*.

Table 7-2 Parameters in the MERGE INTO statement

Parameter	Description	Example Value
INTO clause	Specifies a target table that is to be updated or has data to be inserted. A table alias is supported.	Value: **newproducts np** The table name is **newproducts** and the alias is **np**.

Parameter	Description	Example Value
USING clause	Specifies a source table. A table alias is supported.	Value: **products p** The table name is **products** and the alias is **p**.
ON clause	Specifies a join condition between a target table and a source table. Columns in the join condition cannot be updated.	Value: **np.product_id = p.product_id** The join condition is that the **product_id** column in the target table **newproducts** has equivalent values with the **product_id** column in the source table **products**.
WHEN MATCHED clause	Performs **UPDATE** if data in the source table matches that in the target table based on the condition. • Only one **WHEN MATCHED** clause can be specified. • The **WHEN MATCHED** clause can be omitted. If it is omitted, no operation will be performed on the rows that meet the condition in the **ON** clause. • Columns involved in the distribution key of the target table cannot be updated.	Value: **WHEN MATCHED THEN UPDATE SET np.product_name = p.product_name, np.category = p.category** When the condition in the **ON** clause is met, the values of the **product_name** and **category** columns in the target table **newproducts** are replaced with the values in the corresponding columns in the source table **products**.

Parameter	Description	Example Value
WHEN NOT MATCHED clause	Performs **INSERT** if data in the source table does not match that in the target table based on the condition. • Only one **WHEN NOT MATCHED** clause can be specified. • The **WHEN NOT MATCHED** clause can be omitted. • An **INSERT** clause can contain only one **VALUES**. • The **WHEN NOT MATCHED** and **WHEN NOT MATCHED** clauses can be exchanged in sequence. One of them can be omitted, but they cannot be omitted at the same time.	Value: **WHEN NOT MATCHED THEN** INSERT VALUES (p.product_id, p.product_name, p.category) Insert rows in the source table **products** that do not meet the condition in the **ON** clause into the target table **products**.

Step 4 Query the target table **newproducts** after the merge.

```
SELECT * FROM newproducts;
```

The following information is displayed:

```
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1666 | harry potter   | toys
       1600 | play gym       | toys
       1601 | lamaze         | toys
       1700 | wait interface | books
(6 rows)
```

----End

7.6 Performing a Deep Copy

After data is imported, you can perform a deep copy to modify a distribution key or partition key, change a row-store table to a column-store table, or add a partial cluster key. A deep copy refers to the process of recreating a table and then batch inserting data into the table.

DWS provides three deep copy modes: **CREATE TABLE**, **CREATE TABLE LIKE**, and creating a temporary table and truncating the original table.

Performing a Deep Copy by Using the CREATE TABLE Statement

Run the **CREATE TABLE** statement to create a copy of the original table, batch insert data of the original table into the copy, and rename the copy to the name of the original table.

When creating the copy, you can specify table and column attributes, including the primary key and foreign key.

Perform the following steps to perform a deep copy for the **customer_t** table:

Step 1 Run the **CREATE TABLE** statement to create the copy **customer_t_copy** of the **customer_t** table.

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

Step 2 Run the **INSERT INTO...SELECT** statement to batch insert data of the original table into the copy.

```
INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

Step 3 Delete the original table.

```
DROP TABLE customer_t;
```

Step 4 Run the **ALTER TABLE** statement to rename the copy to the name of the original table.

```
ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----End

Performing a Deep Copy by Using the CREATE TABLE LIKE Statement

Run the **CREATE TABLE LIKE** statement to create a copy of the original table, batch insert data of the original table into the copy, and rename the copy to the name of the original table. This method does not inherit the primary key and foreign key of the original table. You can use the **ALTER TABLE** statement to add them.

Procedure

Step 1 Run the **CREATE TABLE LIKE** statement to create the copy **customer_t_copy** of the **customer_t** table.

```
CREATE TABLE customer_t_copy (LIKE customer_t);
```

Step 2 Run the **INSERT INTO...SELECT** statement to batch insert data of the original table into the copy.

```
INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

Step 3 Delete the original table.

```
DROP TABLE customer_t;
```

Step 4 Run the **ALTER TABLE** statement to rename the copy to the name of the original table.

```
ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----End

Performing a Deep Copy by Creating a Temporary Table and Truncating the Original Table

Run the **CREATE TABLE ... AS** statement to create a temporary table for the original table, truncate the original table, and batch insert data of the temporary data into the original table.

When creating the temporary table, retain the primary key and foreign key of the original table. This method is recommended if the original table has dependency items.

NOTE

Compared with the use of permanent tables, the use of temporary tables can improve performance but may incur data loss. Temporary tables appear only during your current session and delete themselves when you close it. To keep your data safe, choose a permanent table instead.

Procedure

Step 1 Run the **CREATE TABLE AS** statement to create a temporary table **customer_t_temp** for the **customer_t** table.

```
CREATE TABLE customer_t_temp AS SELECT * FROM customer_t;
```

Step 2 Truncate the original table **customer_t**.

```
TRUNCATE customer_t;
```

Step 3 Run the **INSERT INTO...SELECT** statement to batch insert data of the temporary table into the original table.

```
INSERT INTO customer_t (SELECT * FROM customer_t_temp);
```

Step 4 Delete the temporary table **customer_t_temp**.

```
DROP TABLE customer_t_temp;
```

----End

