Huawei HCIA Certification Training

HCIA-openGauss

Lab Guide

Issue: 1.0

Huawei Technologies Co., Ltd.

Huawei Technologies Co., Ltd.

Address:	Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China
Website:	http://e.huawei.com

Huawei Certification System

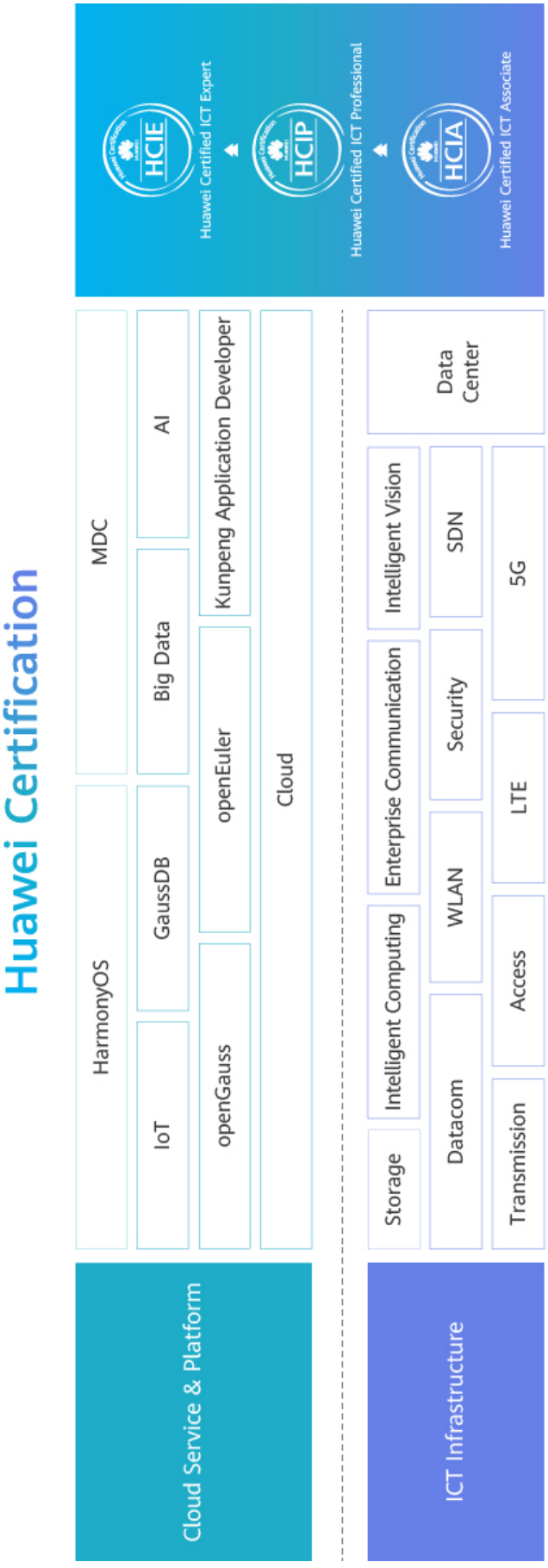
Huawei Certification, which adheres to the "platform + ecosystem" development strategy based on a new "Cloud-Pipe-Terminal" collaborative ICT architecture, is a complete certification system covering two categories: ICT infrastructure and application certification, and cloud service and platform certification.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

In keeping with the ICT convergence trend, Huawei Certification covers all ICT fields. Through its leading talent development system and certification standards, Huawei is committed to cultivating new ICT professionals in the digital era and building a healthy ICT talent ecosystem.

HCIA-openGauss certification is particular for openGauss engineers. It is intended to improve the skills of database development engineers and personnel dedicated to database development. The certification provides rich in-class experiments and industry cases to improve trainees' practical capabilities and promote the cultivation of database talent.

The Huawei certification system introduces the industry, fosters innovation, and imparts cutting-edge WLAN knowledge.



About This Document

Overview

This document is designed for the HCIA-openGauss certification training. It is intended for trainees who are going to take the HCIA-openGauss exam or readers who want to understand the basic knowledge of openGauss and GaussDB(for openGauss) as well as classification and application scenarios of SQL syntax.

Description

This document consists of five parts, including openGauss installation and deployment, database and object management, SQL syntax in openGauss, GaussDB(for openGauss), and scenario-based comprehensive experiment.

- Experiment 1 is about openGauss environment setup and operations. It aims to help trainees understand how to deploy and connect to openGauss by purchasing Elastic Cloud Servers (ECSs) on HUAWEI CLOUD.
- Experiment 2 is about openGauss and object management. It aims to help trainees manage objects of openGauss by managing tablespaces, databases, and users in openGauss.
- Experiment 3 is about SQL syntax basics. It aims to help trainees master SQL syntax basics through data definition language (DDL) and data manipulation language (DML) operations.
- Experiment 4 is about GaussDB(for openGauss). It aims to describe how to add, delete, modify, and query data in the database by purchasing GaussDB(for openGauss) on HUAWEI CLOUD and connecting to the database by using Data Admin Service (DAS).
- Experiment 5 is a comprehensive experiment. It shows related operations performed on openGauss in the financial industry, including object management, connection management, user management, SQL statements, indexes, and views, with an aim to help trainees master openGauss through practices in real scenarios.

Background Knowledge Required

This course is for Huawei's basic certification. To better understand this course, familiarize yourself with the following:

Computer basics, HUAWEI CLOUD console, and Linux basics.

Lab Environment

Networking Description

This lab environment is intended for database engineers who are preparing for the HCIA-openGauss exam. One GaussDB(for openGauss), one DAS, one elastic IP address (EIP), and one ECS are provided for each lab environment.

Devices

To meet the HCIA-openGauss lab requirements, you are advised to use the following configurations in each lab environment:

The mapping between device, model, and version is as follows.

Table 1-1 Mapping relationships

Device	Model	Software Version
GaussDB	GaussDB(for openGauss)	Version 1.4
Data Admin Service (DAS)	DAS	-
Elastic IP address service	EIP	-
ECS	4 vCPUs | 8 GiB	-
openGauss database	openGauss database	2.0.0

Contents

1 openGauss Installation and Operation

1.1 Overview

1.1.1 About This Experiment

This experiment describes how to install and deploy an openGauss database on an openEuler elastic cloud server (ECS).

1.1.2 Objectives

- Understand openGauss database deployment modes.
- Master how to install and deploy openGauss databases.

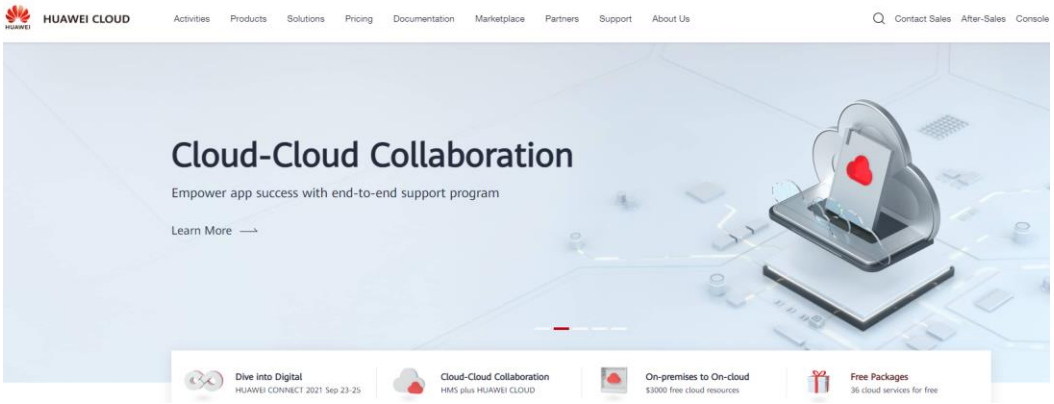
1.2 Database Installation

1.2.1 Purchasing an ECS (Arm-based openEuler)

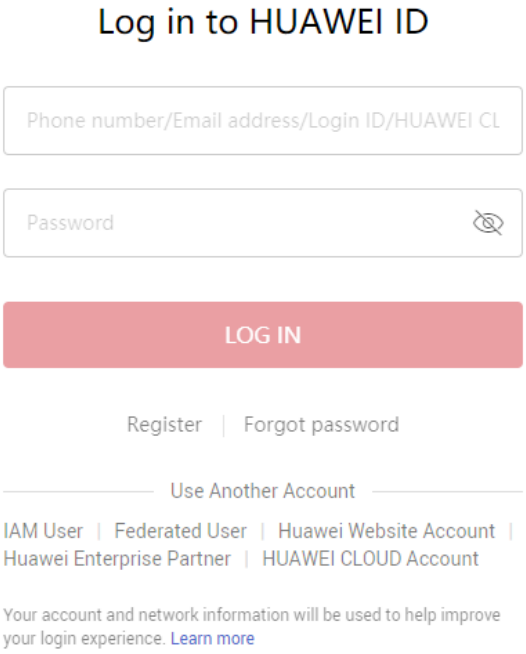
1.2.1.1 Logging In to HUAWEI CLOUD

Step 1 Log in to the HUAWEI CLOUD website.

HUAWEI CLOUD official website: https://www.huaweicloud.com/en-us/.



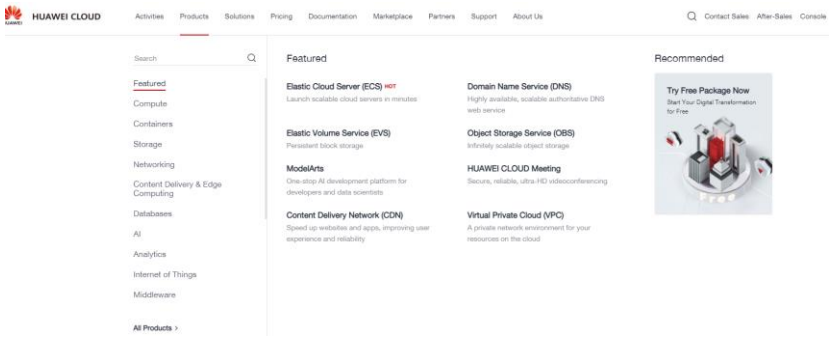
Step 2 Enter the account name and password, and click **LOG IN**.



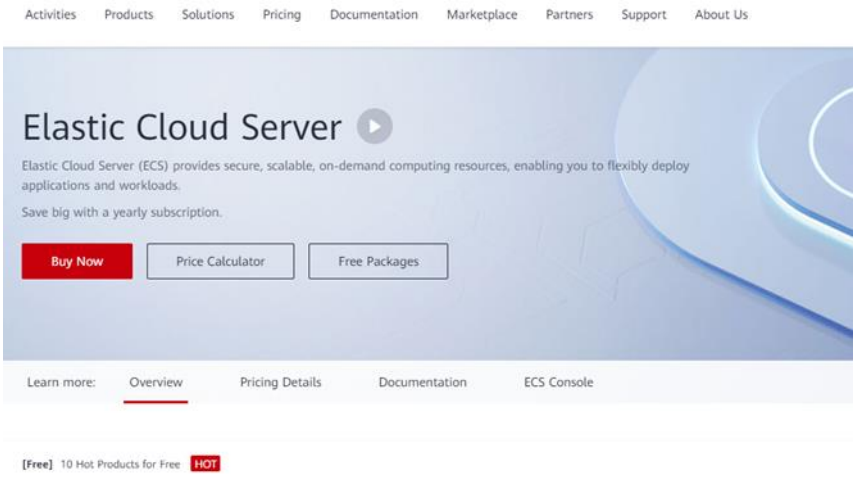
If you have not registered with HUAWEI CLOUD, click **Register** to register a HUAWEI CLOUD account first.

1.2.1.2 Purchasing an ECS

Step 1 On the HUAWEI CLOUD homepage (https://www.huaweicloud.com/), choose **Products** > **Featured** > **Elastic Cloud Server (ESC)**.



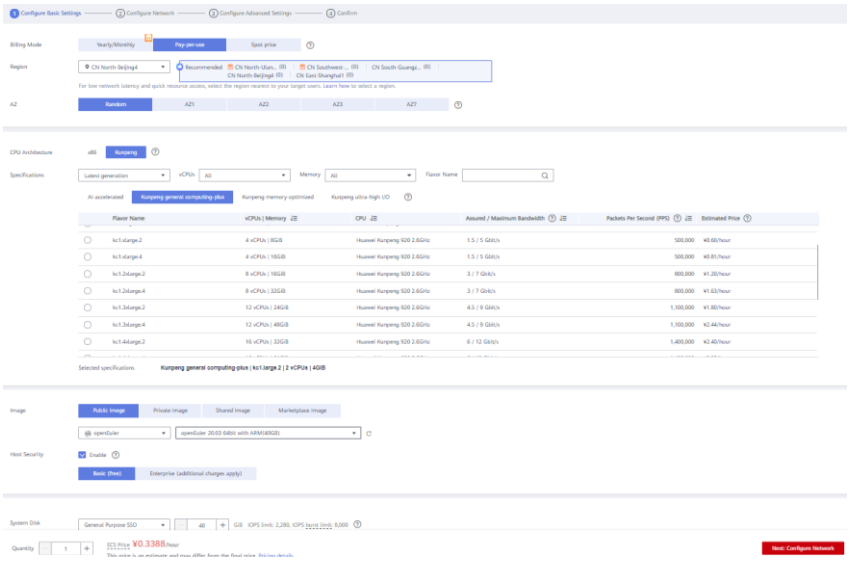
Step 2 Access the Elastic Cloud Server page.



Step 3 Configure the basic settings for your purchase.

Table 1-1 Basic ECS configurations

Item	Value
Billing Mode	**Pay-per-use** (Mandatory. Then, you need to configure the fee.)
Region	**CN North-Beijing4** (Recommended. openEuler public images may not be available if you select other regions.)
CPU Architecture	**Kunpeng**
Specifications	**2 vCPUs | 4 GB**
Image	**Public image** openEuler openEuler 20.03 64bit with ARM(40GB)

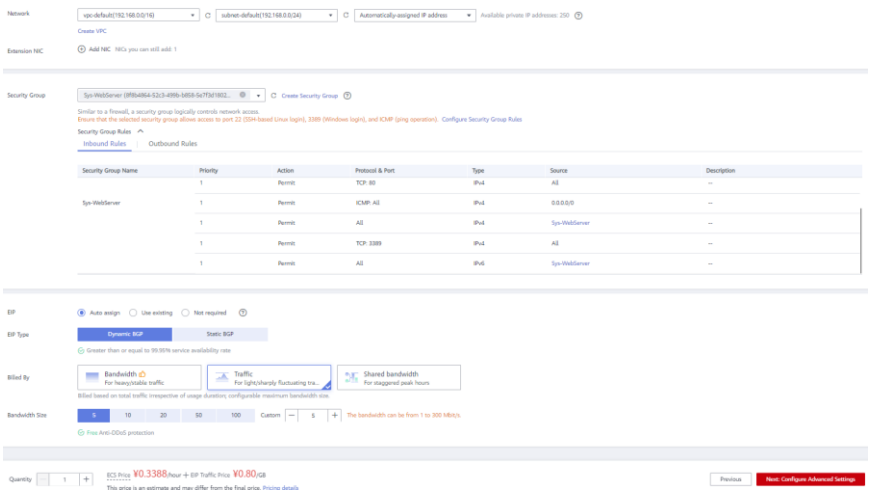


Retain the default settings for the other items and click **Next: Configure Network**.

Step 4 Configure the network for your purchase.

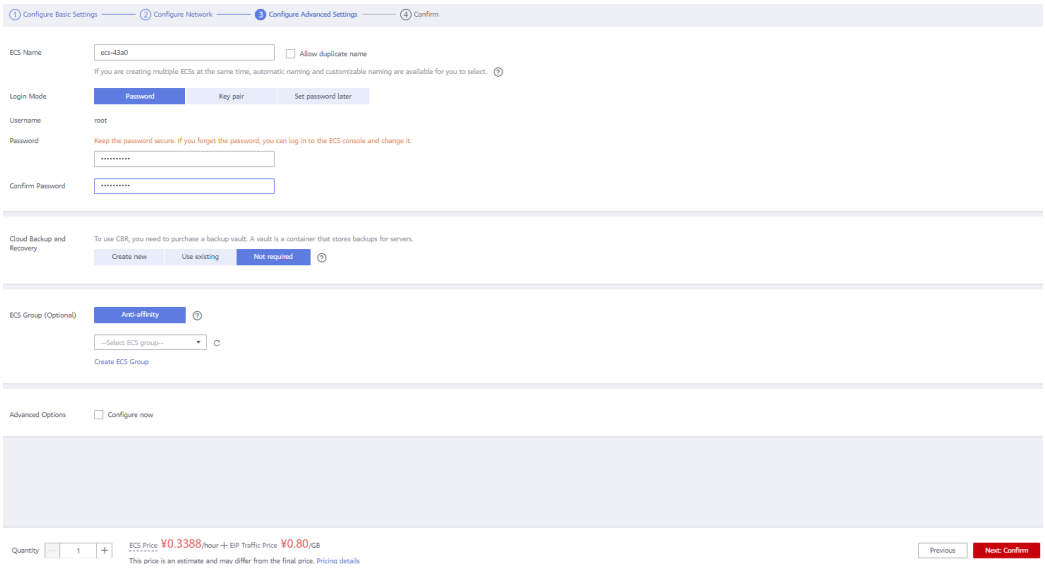
Table 1-2 ECS network configuration

Item	Value
Network	**vpc-default(192.168.0.0/16)** (existing default network)
EIP	**Auto assign**
Billed By	**Traffic**
Bandwidth Size	5



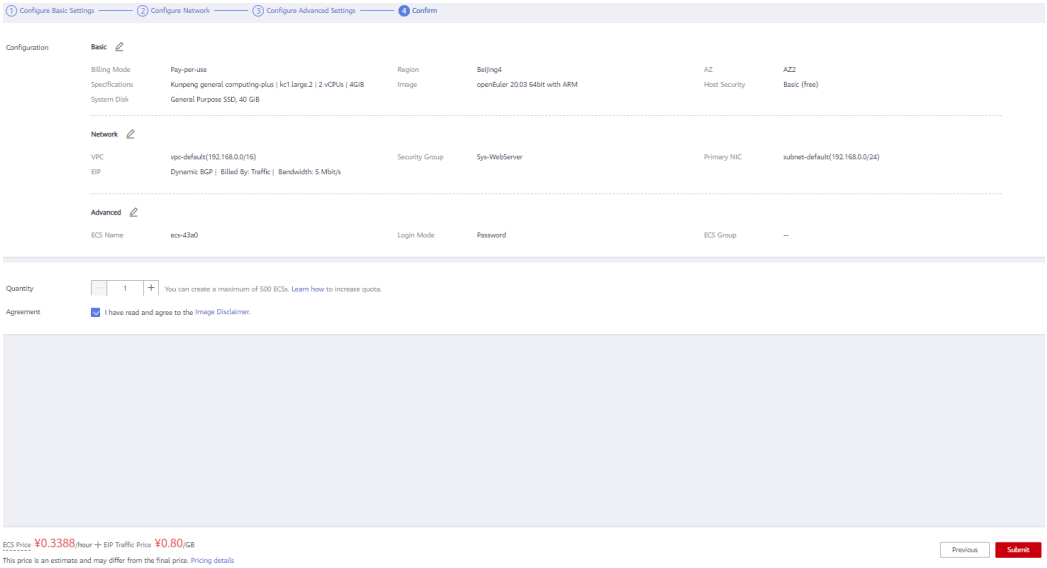
Retain the default settings for the other items and click **Next: Configure Advanced Settings**.

Step 5 Configure the advanced settings for your purchase.

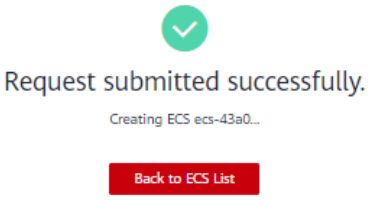


Note that **Username** is set to **root**, enter a custom password, and confirm the password. Retain the default settings for the other items, and click **Next: Confirm**.

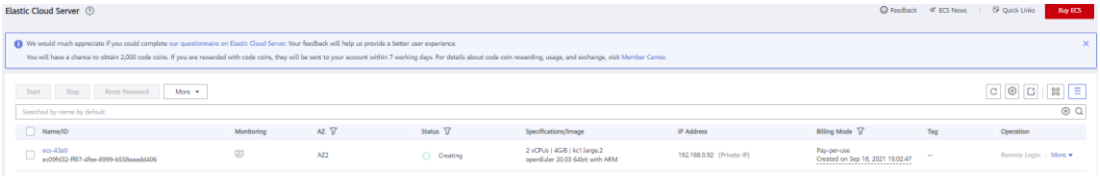
Step 6 Verify the configurations for your purchase.



Confirm the configurations, especially about the fee, select **I have read and agree to the Image Disclaimer**, and click **Submit**.



View the ECS list.

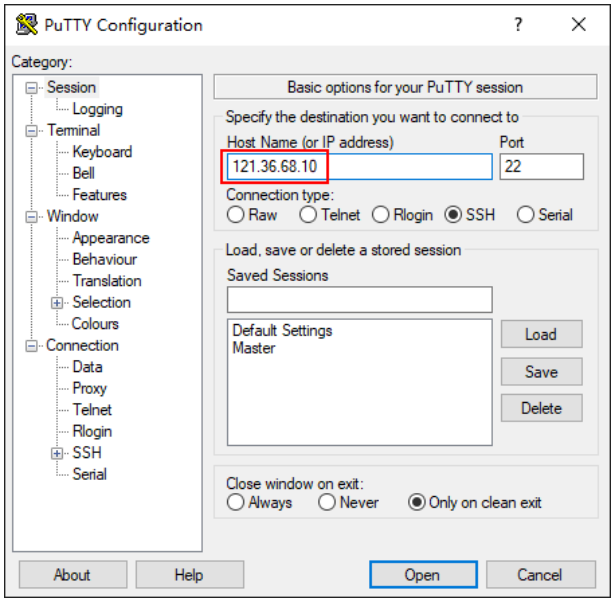


The preceding figure shows that the purchase is complete.

Note: In this experiment, the price of the Kunpeng server is the open beta price. For details, see pricing details on the HUAWEI CLOUD official website.

1.2.2 Modifying OS Configuration

To facilitate operations, you can use an SSH tool (such as PuTTY) to connect to an ECS by configuring an EIP (such as 121.36.68.10) of the ECS on the local PC, and then log in to the ECS as the **root** user.



1.2.2.1 Setting Character Set Parameters

Set the same character set for all database nodes. You can add **export LANG=***Unicode* to the **/etc/profile** file.

Step 1 Add export LANG= en_US.UTF-8 to the /etc/profile file.

```
[root@ecs-f239 ~]# cat >>/etc/profile<<EOF
> export LANG=en_US.UTF-8
> EOF
```

Step 2 Run the following command to make the change take effect:

```
[root@ecs-f239 ~]# source /etc/profile
```

1.2.2.2 Modifying a Python Version and Installing the libaio Package

In the previous installation process, openGauss users trust each other. You need to switch the default Python-2.7.x version to Python-3.7.x required on openEuler servers.

Step 1 Open the **/usr/bin** file and back up the **python** file.

```
[root@ecs-f239 ~]# cd /usr/bin
```

Back up the **python** file.

```
[root@ecs-f239 bin] # mv python python.bak
```

Step 2 Create the **python3** soft link.

```
[root@ecs-f239 bin] # ln -s python3 /usr/bin/python
```

Step 3 Verify the Python version.

```
[root@ecs-f239 bin] # python -V
```

If the following information is displayed, the version is switched:

```
Python 3.7.4
```

Step 4 After the Python version is switched, download the **libaio** package and install it for subsequent operations.

```
[root@ecs-f239 ~]# yum install libaio* -y
```

1.2.3 Installing the openGauss Database

1.2.3.1 Downloading the Database Installation Package

Step 1 Log in to a host where openGauss is to be installed as user **root** and create a directory for storing the installation package as planned.

```
[root@ecs-f239 bin]# mkdir -p /opt/software/openGauss
[root@ecs-f239 bin]# chmod 755 -R /opt/software
```

Note: Do not create the directory in the home directory or subdirectory of any openGauss user because you may lack permissions for such directories. The openGauss user must have the read and write permissions on the **/opt/software/openGauss** directory.

Step 2 Run the **wget** command to download the database installation package to the installation package directory.

Go to the installation package directory.

```
[root@ecs-f239 bin]# cd /opt/software/openGauss
```

Run the **wget** command to download the installation package.

```
[root@ecs-f239 openGauss]# wget https://openGauss.obs.cn-south-1.myhuaweicloud.com/2.0.0/arm/openGauss-2.0.0-openEuler-64bit-all.tar.gz
```

Note: The download link is **https://openGauss.obs.cn-south-1.myhuaweicloud.com/2.0.0/arm/openGauss-2.0.0-openEuler-64bit-all.tar.gz**. No line break is required.

If the download is successful, the following information is displayed:

```
......
2021-06-18 21:49:34 (1.20 MB/s) - 'openGauss-2.0.0-openEuler-64bit-all.tar.gz' saved [108840859/108840859]
```

1.2.3.2 Creating an XML Configuration File

- Before installing openGauss, create an XML configuration file. The configuration file contains information about the server where openGauss will be deployed, the installation path, IP

address, and port. This file guides how to deploy the openGauss. You need to configure the XML configuration file based on deployment requirements.

- The following takes the single-node configuration scheme as an example.

Step 1 Log in to a host where openGauss is to be installed as user **root** and switch to a directory where the installation package is stored.

```
[root@ecs-f239 bin]# cd /opt/software/openGauss
```

Step 2 Create an XML configuration file for installing the database.

```
[root@ecs-f239 openGauss]# vi clusterconfig.xml
```

Step 3 Enter **i** to enter the INSERT mode and add the following content. The content in bold is an example and can be replaced as required. **ecs-f239** indicates the ECS name, and **192.168.0.193** indicates the private IP address of the ECS. Other values do not need to be changed.

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
    <CLUSTER>
        <PARAM name="clusterName" value="dbCluster" />
        <PARAM name="nodeNames" value="ecs-f239" />
        <PARAM name="backIp1s" value="192.168.0.193"/>
        <PARAM name="gaussdbAppPath" value="/opt/gaussdb/app" />
        <PARAM name="gaussdbLogPath" value="/var/log/gaussdb" />
        <PARAM name="gaussdbToolPath" value="/opt/huawei/wisequery" />
        <PARAM name="corePath" value="/opt/opengauss/corefile"/>
        <PARAM name="clusterType" value="single-inst"/>
    </CLUSTER>

    <DEVICELIST>

        <DEVICE sn="1000001">
            <PARAM name="name" value="ecs-f239"/>
            <PARAM name="azName" value="AZ1"/>
            <PARAM name="azPriority" value="1"/>
            <PARAM name="backIp1" value="192.168.0.193"/>
            <PARAM name="sshIp1" value="192.168.0.193"/>

      <!--dbnode-->
      <PARAM name="dataNum" value="1"/>
      <PARAM name="dataPortBase" value="26000"/>
      <PARAM name="dataNode1" value="/gaussdb/data/db1"/>
        </DEVICE>
    </DEVICELIST>
</ROOT>
```

View the ECS name and private IP address.

Step 4 Press **Esc** to exit the INSERT mode, enter **:wq**, and press **Enter** to save the configuration and exit.

Table 1-3 Parameter description

Instance Type	Parameter	Description
Sorting information	clusterName	openGauss name.
	nodeNames	Host name in openGauss.
	backIp1s	Intranet IP address of the host in the backend storage network. All the openGauss hosts communicate with each other on this network.
	gaussdbAppPath	Installation directory of the openGauss program. This directory must meet the following requirements: ● The disk space is greater than 1 GB. ● This directory is independent of other directories required by the database.
	gaussdbLogPath	Directory that stores run logs and operation logs of the openGauss. This directory must meet the following requirements: ● You are advised to plan the disk space based on the number of database nodes on the host. Reserve 1 GB space for database nodes and reserve redundant space. ● This directory is independent of other directories required by openGauss. This path can be customized. If the directory is not specified, **$GAUSSLOG/***Installation user account* will be specified as the log directory by default during openGauss installation.
	tmpdbPath	Directory for storing temporary database files. If **tmpdbPath** is not set, the file is stored in **/opt/huawei/wisequery/perfadm_db** by default.
	gaussdbToolPath	Directory for storing openGauss system tools. This directory is used to store tools for mutual trust. This directory must meet the following requirements: ● Disk space: > 100 MB

Instance Type	Parameter	Description
		• This directory cannot be changed and is independent of other directories required by the database. This directory is optional. If this parameter is not specified, **/opt/huawei/wisequery** is specified as the database system tool directory by default during openGauss installation.
	corePath	Directory for storing the openGauss core file.

Note:

- The **/opt/huawei/newsql/tools** directory is used to store tools for mutual trust. To avoid permission problems, do not store instance data in the directory.
- The installation and data directories must be empty or do not exist. Otherwise, the installation may fail.
- When configuring database instances, ensure that the configured directories are not coupled with each other. This means that the configured directories must not be associated with each other. If any directory is deleted, the other directories will not be deleted accordingly. For example, when **gaussdbAppPath** is **/opt/gaussdb/app**, and **gaussdbLogPath** is **/opt/gaussdb/app/omm**, deleting the directory specified by **gaussdbAppPath** will also delete that specified by **gaussdbLogPath**, causing unexpected problems.
- To automatically create installation users, ensure that the configured directories are not coupled with the default user directories created by the system.
- The openGauss and instance paths cannot contain double backslashes (\\) or the following characters: |;&$<>`'\"{}()[]~*?

1.2.3.3 Initializing the Installation Environment

To ensure the correct installation of openGauss, you need to configure the host environment first.

1.2.3.3.1 Creating the Required User Account and Configuring the Installation Environment

After the openGauss configuration file is created, you need to run the **gs_preinstall** script to prepare the account and environment so that you can perform openGauss installation and management operations with the minimum permission, ensuring system security.

1.2.3.3.2 Prerequisites

All installation preparations have been completed.

1.2.3.3.3 Precautions

- You must check the upper-layer directory permissions to ensure that the user has the read, write, and execution permissions on the installation package and configuration file directory.
- Each host name and IP address mapping must be correctly configured in the XML file.
- Only user **root** is authorized to run the **gs_preinstall** command.

1.2.3.3.4 Procedure

Step 1 Modify the **performance.sh** file.

Run the **vi** command to open the **/etc/profile.d/performance.sh** file.

```
[root@ecs-f239 openGauss]# vi /etc/profile.d/performance.sh
```

Enter **i** to enter the INSERT mode. Use the comment tag (#) to comment out the **sysctl -w vm.min_free_kbytes=112640 &> /dev/null** line.

```
CPUNO=`cat /proc/cpuinfo|grep processor|wc -l`
export GOMP_CPU_AFFINITY=0-$[CPUNO - 1]

#sysctl -w vm.min_free_kbytes=112640 &> /dev/null
sysctl -w vm.dirty_ratio=60 &> /dev/null
sysctl -w kernel.sched_autogroup_enabled=0 &> /dev/null
```

Press **Esc** to exit the INSERT mode. Enter **:wq** and press **Enter** to save the settings and exit.

Step 2 To ensure that the OpenSSL version is correct, load the **lib** library in the installation package before preinstallation.

Run the following command. **{packagePath}** indicates the path where the installation package is stored. In this example, the path is **/opt/software/openGauss**.

```
[root@ecs-f239 openGauss]# vi /etc/profile
```

Enter **i** to enter the INSERT mode. Add the following code at the bottom of the file to load the library in the installation package: Press **Esc** to exit the INSERT mode. Enter **:wq** and press **Enter** to save the settings and exit.

```
export packagePath=/opt/software/openGauss
export LD_LIBRARY_PATH=$packagePath/script/gspylib/clib:$LD_LIBRARY_PATH
```

After the configuration is complete, run the following command to make the settings take effect:

```
[root@ecs-f239 openGauss]# source /etc/profile
```

Step 3 Go to the directory for storing the uploaded software package and decompress the package.

```
[root@ecs-f239 openGauss]# cd /opt/software/openGauss
```

Decompress the installation package.

Decompress the **openGauss-2.0.0-openEuler-64bit-all.tar.gz** package.

```
[root@ecs-f239 openGauss]# tar -zxvf openGauss-2.0.0-openEuler-64bit-all.tar.gz
```

Decompress the **openGauss-2.0.0-openEuler-64bit-om.tar.gz** package.

```
[root@ecs-f239 openGauss]# tar -zxvf openGauss-2.0.0-openEuler-64bit-om.tar.gz
```

After the decompression, run the **ls** command to view the following information:

```
[root@ecs-f239 openGauss]# ls
clusterconfig.xml                          openGauss-2.0.0-openEuler-64bit-om.sha256  openGauss-2.0.0-
openEuler-64bit.tar.bz2  simpleInstall          version.cfg
lib                                        openGauss-2.0.0-openEuler-64bit-om.tar.gz  openGauss-
Package-bak_78689da9.tar.gz      upgrade_sql.sha256
openGauss-2.0.0-openEuler-64bit-all.tar.gz  openGauss-2.0.0-openEuler-64bit.sha256      script
upgrade_sql.tar.gz
```

After the installation package is decompressed, the **script** subdirectory is automatically generated in **/opt/software/openGauss**. OM tool scripts such as **gs_preinstall** are generated in the **script** subdirectory.

Step 4 Use **gs_preinstall** to prepare the installation environment and switch to the directory where the **gs_preinstall** command is located.

```
[root@ecs-f239 openGauss]# cd /opt/software/openGauss/script/
```

The content in the script is as follows:

```
[root@ecs-f239 script]# ls
__init__.py  gs_backup  gs_checkos     gs_collector  gs_expansion  gs_om          gs_preinstall
gs_sshexkey  gs_upgradectl  impl       local
config       gs_check   gs_checkperf  gs_dropnode   gs_install    gs_postuninstall  gs_ssh
gs_uninstall  gspylib       killall  transfer.py
```

Step 5 Execute **gs_preinstall** in interactive mode. During the execution, the mutual trust between users **root** and between openGauss users is automatically established.

```
[root@ecs-f239 script]# python gs_preinstall -U omm -G dbgrp -X /opt/software/openGauss/clusterconfig.xml
```

In the preceding command, **omm** is the OS user (**omm** is also the openGauss database administrator account, which is created in section 1.4.4), **dbgrp** is the group name of the OS user who runs openGauss, and **/opt/software/openGauss/clusterconfig.xml** is the path of the openGauss configuration file. During the execution, you need to determine whether to establish mutual trust as prompted and enter the password of user **root** or the openGauss user.

Create a trust relationship for the **root** user and enter the password of the **root** user. The password is customized when the ECS is purchased.

```
Are you sure you want to create trust for root (yes/no)? yes
Please enter password for root.
Password:-Note: When you enter the password, no information is displayed on the screen. Do not worry about it. The Linux operating system projects your password.
Creating SSH trust for the root permission user.
```

Create OS user **omm**, create a trust relationship for user **omm**, and set the password to **Admin@123**. (You are advised to customize the password.)

```
Are you sure you want to create the user[omm] and create trust for it (yes/no)? yes
Please enter password for cluster user.
Password:-Note: When you enter the password, no information is displayed on the screen. Do not worry about it. The Linux operating system projects your password.
Please enter password for cluster user again.
Password:-Note: When you enter the password, no information is displayed on the screen. Do not worry about it. The Linux operating system projects your password.
Successfully created [omm] user on all nodes.
```

If the operation is successful, the following information is displayed:

```
.....
Setting finish flag.
Successfully set finish flag.
Preinstallation succeeded.
```

1.2.3.4 Executing Installation

After the openGauss installation environment is prepared by executing the pre-installation script, deploy openGauss based on the installation process.

1.2.3.4.1 Prerequisites

- You have successfully executed the **gs_preinstall** script.
- Server OSs and networks are functioning properly.

1.2.3.4.2 Procedure

Step 1 Modify the file permissions.

```
[root@ecs-f239 script]# chmod -R 755 /opt/software/openGauss/script/
```

Step 2 Log in to the openGauss host and switch to user **omm**.

```
[root@ecs-f239 script]# su - omm
```

Note:

- **omm** indicates the user specified by the **-U** parameter in the **gs_preinstall** script.
- You need to execute the **gs_install** script as user **omm** specified in the **gs_preinstall** script. Otherwise, an execution error occurs.

Step 3 You need to use **gs_install** to install openGauss.

Run the following commands to start the installation:

```
[omm@ecs- f239 ~]$ gs_install -X /opt/software/openGauss/clusterconfig.xml --gsinit-parameter="--encoding=UTF8"
--dn-guc="max_process_memory=4GB"    --dn-guc="shared_buffers=256MB" --dn-
guc="bulk_write_ring_size=256MB" --dn-guc="cstore_buffers=16MB"
```

/opt/software/ openGauss/clusterconfig.xml is the path of the openGauss configuration file. During the execution, you need to enter the database administrator **omm** as prompted. The password must meet complexity requirements. To ensure that you can use the database properly, remember the entered database password.

Set the password based on requirements. (You are advised to customize the password.)

```
encrypt cipher and rand files for database.
Please enter password for database:        --Note: When you enter the password, no information is displayed on the screen.
Please repeat for database:                --Note: When you enter the password, no information is displayed on the screen.
begin to create CA cert files
```

The password must:

- Contain at least eight characters.
- Cannot be the same as the username, the current password (ALTER), or the current password in an inverted sequence.
- Contain at least three of the following: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and other characters (limited to ~!@#$%^&*()-_=+\|[{}];:,<.>/?).

If the installation is successful, the following information is displayed:

```
......
Successfully deleted instances from all nodes.
Checking node configuration on all nodes.
Initializing instances on all nodes.
Updating instance configuration on all nodes.
Check consistence of memCheck and coresCheck on database nodes.
```

```
Configuring pg_hba on all nodes.
Configuration is completed.
Successfully started cluster.
Successfully installed application.
end deploy..
```

1.2.3.5 Directories Generated After the Installation

The following table describes the directories generated after the installation and the files in the directories.

Table 1-4 Directories and files in each directory after the installation

No.	Directory Description	Directory	Subdirectory	Description
1	Cluster openGauss installation directory	/opt/gaussdb/app	etc	Stores the configuration file of the Cgroup tool.
			include	Stores the header file required for running the database.
			lib	Saves library files of the database.
			share	Stores common files required for database running, such as the configuration file template.
2	Cluster openGauss data directory	/gaussdb/data	data_dnxxx	Stores database node data. For the primary node, the directory name is **data_dnxxx**. For the standby node, the directory name is **data_dnS***xxx*. **xxx** indicates the database node number.
3	Cluster openGauss log directory	**/var/log/gaussdb/***username*	bin	Saves logs of binary programs.
			gs_profile	Stores database kernel performance logs.
			om	Stores OM logs. For example: It contains logs for executing some local scripts, as well as adding and deleting database node interfaces, gs_om interfaces, pre-installation interfaces, and node replacement interfaces.
			pg_audit	Stores database audit logs.

No.	Directory Description	Directory	Subdirectory	Description
			pg_log	Stores the run logs of database node instances.
4	Cluster openGauss system tool directory	/opt/huawei/wisequery	script	Stores script file used by openGauss users to manage openGauss.
			lib	Stores the library files that the binaries in the **bin** directory depend on.

1.3 Database Connection — gsql

gsql is a CLI database connection tool provided by openGauss. Users can use gsql to connect to a server and perform operations and maintenance on the server. In addition to basic functions for performing operations on a database, gsql provides several other more advanced features.

1.3.1 Database Connection — gsql

gsql is a client tool provided by openGauss. You can use gsql to connect to the database and enter, edit, and execute SQL statements in an interactive manner.

1.3.1.1 Determining Connection Information

The client tool connects to the database through the primary database node. Before connecting to the database, you need the IP address and port for the server where the primary database node is located.

Step 1 Log in to the primary database node as user **omm**.

```
[root@ecs-opengauss ~]# su - omm
```

Step 2 Run the **gs_om -t status --detail** command to query instances in openGauss.

```
[omm@ecs-opengauss ~]$ gs_om -t status --detail
```

The following information is displayed:

```
 [     Cluster State     ]

cluster_state     : Normal
redistributing    : No
current_az          : AZ_ALL

[    Datanode State     ]

node                    node_ip             instance                    state
--------------------------------------------------------------------------------
```

```
1    ecs-opengauss 192.168.0.203      6001 /gaussdb/data P Primary Normal
```

Step 3 Check the port number of the primary database node.

Check the port in the **postgresql.conf** file in the data directory of the primary database node obtained in step 2. For example:

```
[omm@ecs-opengauss ~]$ cat /gaussdb/data/db1/postgresql.conf | grep port
```

The information is displayed as follows:

```
port = 26000                                    # (change requires restart)
#ssl_renegotiation_limit = 0                    # amount of data between renegotiations, no longer supported
#comm_sctp_port = 1024                          # Assigned by installation (change requires restart)
#comm_control_port = 10001                      # Assigned by installation (change requires restart)
                                                # supported by the operating system:
                                                # The heartbeat thread will not start if not set
localheartbeatport and remoteheartbeatport.
                                                # e.g. 'localhost=10.145.130.2 localport=12211
localheartbeatport=12214 remotehost=10.145.130.3 remoteport=12212 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12213 remotehost=10.145.133.3 remoteport=12214'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.4 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.4 remoteport=12314'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.5 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.5 remoteport=12314'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.6 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.6 remoteport=12314'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.7 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.7 remoteport=12314'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.8 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.8 remoteport=12314'
                                                # e.g. 'localhost=10.145.130.2 localport=12311
localheartbeatport=12214 remotehost=10.145.130.9 remoteport=12312 remoteheartbeatport=12215,
localhost=10.145.133.2 localport=12313 remotehost=10.145.133.9 remoteport=12314'
                                                #      %r = remote host and port
alarm_report_interval = 10
#gtm_port = 6666                                # Port of GTM
#gtm_port1 = 6665                               # Port1 of GTM
#gtm_port2 = 6664                               # Port2 of GTM
#gtm_port3 = 6663                               # Port3 of GTM
#gtm_port4 = 6662                               # Port4 of GTM
#gtm_port5 = 6661                               # Port5 of GTM
#gtm_port6 = 6660                               # Port6 of GTM
#gtm_port7 = 6659                               # Port7 of GTM
#streaming_router_port = 5438                   # Port the streaming router listens on, please keep the value
(streaming_router_port - port) not change.
                                                # default port is (port + 6), setting by kernel, range 0 ~ 65535,
                                                # value 0 means use default value (port + 6).
```

26000 is the port number of the primary database node.

Record the IP address, data path, and port number of the server where the primary database instance is located, and replace them as required.

1.3.1.2 Connecting a Database Locally

Step 1 Log in to the primary database node as user **omm**.

```
[root@ecs-opengauss ~]# su - omm
```

Step 2 Start the database service.

```
[root@ecs-opengauss ~]# gs_om -t start
```

If the following information is displayed, the startup is successful:

```
Starting cluster.
=========================================
[SUCCESS] ecs-opengauss:
[2021-06-18 22:24:37.101][64478][][gs_ctl]: gs_ctl started,datadir is /gaussdb/data
[2021-06-18 22:24:37.104][64478][][gs_ctl]:  another server might be running; Please use the restart command
=========================================
Successfully started.
```

Step 3 Connect to the database.

Run the following command to connect to the database:

```
[omm@ecs-opengauss /]$ gsql -d postgres -p 26000 -r
```

postgres is the name of the database to be connected, and **26000** is the port of the primary database node. Replace the values as required.

If information similar to the following is displayed, the connection is successful:

```
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.


postgres=#
```

If you log in and connect to the database as administrator **omm**, **DBNAME=#** is displayed. If you have logged in as a common user and connected to the database, **DBNAME=>** is displayed.

Non-SSL connection indicates that the database is not connected in SSL mode. For improved security, use SSL for secure TCP/IP connections.

Step 4 Exit the database.

```
postgres=# \q
```

1.3.1.3 Obtaining gsql Help

1.3.1.3.1 Prerequisites

The following operations are performed on the host where the primary openGauss database node is located (you have connected to a database locally) as user **omm**.

```
su - omm
```

1.3.1.3.2 When connecting to the database, run the following command to obtain the help information:

```
gsql --help
```

The following information is displayed:

```
[omm@ecs-opengauss /]$ gsql --help
gsql is the openGauss interactive terminal.

Usage:
  gsql [OPTION]... [DBNAME [USERNAME]]

General options:
  -c, --command=COMMAND        run only single command (SQL or internal) and exit
  -d, --dbname=DBNAME          database name to connect to (default: "omm")
  -f, --file=FILENAME          execute commands from file, then exit
  -l, --list                   list available databases, then exit
  -v, --set=, --variable=NAME=VALUE
                               set gsql variable NAME to VALUE
  -V, --version                output version information, then exit
  -X, --no-gsqlrc              do not read startup file (~/.gsqlrc)
  -1 ("one"), --single-transaction
                               execute command file as a single transaction
  -?, --help                   show this help, then exit

Input and output options:
  -a, --echo-all               echo all input from script
  -e, --echo-queries           echo commands sent to server
  -E, --echo-hidden            display queries that internal commands generate
  -k, --with-key=KEY           the key for decrypting the encrypted file
  -L, --log-file=FILENAME      send session log to file
  -m, --maintenance            can connect to cluster during 2-pc transaction recovery
  -n, --no-libedit             disable enhanced command line editing (libedit)
  -o, --output=FILENAME        send query results to file (or |pipe)
  -q, --quiet                  run quietly (no messages, only query output)
  -C, --enable-client-encryption                enable client encryption feature
  -s, --single-step            single-step mode (confirm each query)
  -S, --single-line            single-line mode (end of line terminates SQL command)

Output format options:
  -A, --no-align               unaligned table output mode
  -F, --field-separator=STRING
                               set field separator (default: "|")
  -H, --html                   HTML table output mode
  -P, --pset=VAR[=ARG]         set printing option VAR to ARG (see \pset command)
  -R, --record-separator=STRING
                               set record separator (default: newline)
  -r                           if this parameter is set,use libedit
  -t, --tuples-only            print rows only
```

```
  -T, --table-attr=TEXT       set HTML table tag attributes (e.g., width, border)
  -x, --expanded              turn on expanded table output
  -z, --field-separator-zero
                              set field separator to zero byte
  -0, --record-separator-zero
                              set record separator to zero byte

Connection options:
  -h, --host=HOSTNAME         database server host or socket directory (default:
"/opt/opengauss/wisequery/omm_mppdb")
  -p, --port=PORT             database server port (default: "5432")
  -U, --username=USERNAME     database user name (default: "omm")
  -W, --password=PASSWORD     the password of specified database user

For more information, type "\?" (for internal commands) or "\help" (for SQL
commands) from within gsql, or consult the gsql section in the openGauss documentation.
```

1.3.1.3.3 After connecting to the database, run the following command to obtain the help information:

Step 1 Run the following command to connect to the database:

```
gsql -d postgres -p 26000 -r
```

Step 2 Run the **help** command.

```
postgres=# help
```

The following information is displayed:

```
You are using gsql, the command-line interface to gaussdb.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with gsql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

Step 3 View the copyright information.

```
postgres=# \copyright
```

The following version information is displayed:

```
GaussDB Kernel Database Management System
Copyright (c) Huawei Technologies Co., Ltd. 2018. All rights reserved.
```

Step 4 Query all SQL statements supported by openGauss.

```
postgres=# \h
```

The following information is displayed:

```
Available help:
  ABORT                              ALTER TEXT SEARCH DICTIONARY       CREATE MASKING POLICY
DROP CLIENT MASTER KEY               DROP WORKLOAD GROUP
  ALTER APP WORKLOAD GROUP
```

```
......
```

Step 5 View parameters of the **CREATE DATABASE** command:

```
postgres=# \help CREATE DATABASE
```

The following information is displayed:

```
Command:     CREATE DATABASE
Description: create a new database
Syntax:
CREATE DATABASE database_name
    [ [ WITH ] {[ OWNER [=] user_name ]|
                [ TEMPLATE [=] template ]|
                [ ENCODING [=] encoding ]|
                [ LC_COLLATE [=] lc_collate ]|
                [ LC_CTYPE [=] lc_ctype ]|
                [ DBCOMPATIBILITY [=] compatibility_type ]|
                [ TABLESPACE [=] tablespace_name ]|
                [ CONNECTION LIMIT [=] connlimit ]}[...] ];
```

Step 6 Check the meta-commands supported by gsql.

```
postgres=# \?
```

The following information is displayed:

```
General
  \copyright                show openGauss usage and distribution terms
  \g [FILE] or ;            execute query (and send results to file or |pipe)
  \h(\help) [NAME]                  help on syntax of SQL commands, * for all commands
  \parallel [on [num]|off] toggle status of execute (currently off)
  \q                                quit gsql
......
```

Step 7 Exit the database.

```
postgres=# \q
```

1.4 Using Database Tools

This section explains how to use databases, including creating databases and tables, inserting data to tables, and querying data in tables.

1.4.1 Prerequisites

- The openGauss is running properly.
- To use openGauss databases, you need to master the basic operations and SQL syntax of openGauss databases. The openGauss database supports the SQL2003 standard syntax. For details about the basic database operations, see Appendix 2.

1.4.2 Basic Operation Procedure

Step 1 On the primary database node, switch to user **omm**.

```
[root@ecs-f239 script]# su - omm
```

If you are not sure which server the primary database node is deployed on, check the connection information.

Step 2 If the database service is not started, perform this step to start it.

Start the service.

```
[omm@ecs-f239 ~]$ gs_om -t start
Starting cluster.
=========================================
=========================================
Successfully started.
```

Check whether the service is started.

```
[omm@ ecs-f239 ~]$ gs_om -t status
------------------------------------------------------------------------


cluster_state   : Normal
redistributing  : No


------------------------------------------------------------------------
```

Step 3 Connect to the database.

```
[omm@ecs-f239 ~]$ gsql -d postgres -p 26000 -r
```

If the following information is displayed, the connection has been established:

```
gsql ((openGauss 1.1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

postgres is the database generated by default after openGauss installation is complete. You can connect to this database to create a database. **26000** is the port number of the primary database node. Replace it with the actual port number of the openGauss node.

Note:

- You need to use a client program or tool to connect to the database and to deliver SQL statements.
- **gsql** is a command-line interface (CLI) tool provided for connecting to a database.

Step 4 Create a database user.

Only administrators that are created during openGauss installation can access the initial database by default. You can also create other database users.

```
postgres=# CREATE USER joe WITH PASSWORD "Bigdata@123";
```

If the following information is displayed, the creation is successful:

```
CREATE ROLE
```

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

Step 5 Create a database.

```
postgres=# CREATE DATABASE db_tpcc OWNER joe;
```

If the following information is displayed, the creation is successful:

```
CREATE DATABASE
```

Step 6 Connect to the database as a new user and perform subsequent operations such as creating tables. You can also continue using the default **postgres** database for more operations.

Exit the **postgres** database.

```
postgres=#    \q
```

Connect to the database as the new user.

```
[omm@ecs-f239 ~]$ gsql -d db_tpcc -p 26000 -U joe -W Bigdata@123    -r
```

If the following information is displayed, the connection has been established:

```
gsql ((openGauss 1.1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

db_tpcc=>
```

Step 7 Create a schema.

```
db_tpcc=> CREATE SCHEMA joe AUTHORIZATION joe;
```

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

```
CREATE SCHEMA
```

Step 8 Create a table.

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

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

Step 9 Insert data to the table.

```
db_tpcc=> INSERT INTO mytable values (100);
```

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

```
INSERT 0 1
```

The value **0** indicates the OID, and the value **1** indicates the number of inserted records.

View data in the table.

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

1.4.3 Procedure for Using Tools

1.4.3.1 gs_checkos

Step 1 Log in to the operating system where the openGauss database service is installed as the **root** user. Run the **gs_checkos** command as user **root** to check system parameters.

```
[root@ecs-opengauss bin]# gs_checkos -i A
Checking items:
    A1. [ OS version status ]                                    : Normal
    A2. [ Kernel version status ]                                : Normal
    A3. [ Unicode status ]                                       : Normal
    A4. [ Time zone status ]                                     : Normal
    A5. [ Swap memory status ]                                   : Normal
    A6. [ System control parameters status ]                     : Warning
    A7. [ File system configuration status ]                     : Normal
    A8. [ Disk configuration status ]                            : Normal
    A9. [ Pre-read block size status ]                           : Normal
    A10.[ IO scheduler status ]                                  : Normal
BondMode Null
    A11.[ Network card configuration status ]                    : Warning
    A12.[ Time consistency status ]                              : Warning
    A13.[ Firewall service status ]                              : Normal
    A14.[ THP service status ]                                   : Normal
Total numbers:14. Abnormal numbers:0. Warning numbers:3.
```

Notes:

Normal indicates that the check item is normal. **Abnormal** indicates that the check item must be handled. **Warning** indicates that the check item does not need to be handled.

Total numbers:14. Abnormal numbers:0. Warning numbers:3.

The preceding information indicates that a total of 14 items are checked, where there are 0 abnormal items and 3 warning items.

Step 2 Adjust system parameters.

In the parameter configuration file **/etc/sysctl.conf**, set the following parameters:

Change the value of **vm.min_free_kbytes** to **348844**. **vm.min_free_kbytes** indicates the size of the memory reserved for kernel memory allocation.

Change the value of **net.ipv4.tcp_retries1** to **5**.

Change the value of **net.ipv4.tcp_syn_retries** to **5**.

Change the value of **net.sctp.path_max_retrans** to **10**.

Change the value of **net.sctp.max_init_retransmits** to **10**.

The detailed configuration is as follows:

vm.min_free_kbytes = 348844

net.ipv4.tcp_retries1 = 5

net.ipv4.tcp_syn_retries = 5

net.sctp.path_max_retrans = 10

net.sctp.max_init_retransmits = 10

Enter **i** to enter the INSERT mode.

```
[root@ecs-opengauss bin]# vi /etc/sysctl.conf
kernel.sysrq = 0
net.ipv4.ip_forward = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.tcp_syncookies = 1
kernel.dmesg_restrict = 1
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
vm.swappiness = 0
net.ipv4.tcp_max_tw_buckets = 10000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_keepalive_time = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_retries2 = 12
net.ipv4.ip_local_reserved_ports = 20050-20057,26000-26007
net.core.wmem_max = 21299200
net.core.rmem_max = 21299200
net.core.wmem_default = 21299200
net.core.rmem_default = 21299200
net.sctp.sctp_mem = 94500000 915000000 927000000
net.sctp.sctp_rmem = 8192 250000 16777216
net.sctp.sctp_wmem = 8192 250000 16777216
kernel.sem = 250 6400000 1000 25600
net.ipv4.tcp_rmem = 8192 250000 16777216
net.ipv4.tcp_wmem = 8192 250000 16777216
vm.min_free_kbytes = 348844
net.core.netdev_max_backlog = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.core.somaxconn = 65535
kernel.shmall = 1152921504606846720
kernel.shmmax = 18446744073709551615
net.ipv4.tcp_retries1 = 5
net.ipv4.tcp_syn_retries = 5
net.sctp.path_max_retrans = 10
net.sctp.max_init_retransmits = 10
```

After modifying the parameters, press **Esc** to exit the INSERT mode, enter **:wq**, and press **Enter** to save the modification. Run the **sysctl -p** command for the modification to take effect. The details are as follows:

```
[root@ecs-opengauss bin]# sysctl -p
kernel.sysrq = 0
net.ipv4.ip_forward = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.tcp_syncookies = 1
kernel.dmesg_restrict = 1
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
vm.swappiness = 0
net.ipv4.tcp_max_tw_buckets = 10000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_keepalive_time = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_retries2 = 12
net.ipv4.ip_local_reserved_ports = 20050-20057,26000-26007
net.core.wmem_max = 21299200
net.core.rmem_max = 21299200
net.core.wmem_default = 21299200
net.core.rmem_default = 21299200
net.sctp.sctp_mem = 94500000 915000000 927000000
net.sctp.sctp_rmem = 8192 250000 16777216
net.sctp.sctp_wmem = 8192 250000 16777216
kernel.sem = 250 6400000 1000 25600
net.ipv4.tcp_rmem = 8192 250000 16777216
net.ipv4.tcp_wmem = 8192 250000 16777216
vm.min_free_kbytes = 348844
net.core.netdev_max_backlog = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.core.somaxconn = 65535
kernel.shmall = 1152921504606846720
kernel.shmmax = 18446744073709551615
net.ipv4.tcp_retries1 = 5
net.ipv4.tcp_syn_retries = 5
```

```
net.sctp.path_max_retrans = 10
net.sctp.max_init_retransmits = 10
```

Step 3 Run the **gs_checkos -i A** command to check whether the system parameters pass the check.

```
[root@ecs-e1b3 ~]# gs_checkos -i A
Checking items:
      A1. [ OS version status ]                                    : Normal
      A2. [ Kernel version status ]                                : Normal
      A3. [ Unicode status ]                                       : Normal
      A4. [ Time zone status ]                                     : Normal
      A5. [ Swap memory status ]                                   : Normal
      A6. [ System control parameters status ]                     : Normal
      A7. [ File system configuration status ]                     : Normal
      A8. [ Disk configuration status ]                            : Normal
      A9. [ Pre-read block size status ]                           : Normal
      A10.[ IO scheduler status ]                                  : Normal
BondMode Null
      A11.[ Network card configuration status ]                    : Warning
      A12.[ Time consistency status ]                              : Warning
      A13.[ Firewall service status ]                              : Normal
      A14.[ THP service status ]                                   : Normal
Total numbers:14. Abnormal numbers:0. Warning numbers:2.
```

The check result indicates that the system parameters pass the check.

1.4.3.2 gs_check

gs_check fully checks the openGauss operation environment, OS environment, network environment, and database execution environment during openGauss running and before major operations, ensuring operation success.

In this experiment, the gs_check tool is used to check the running status of the openGauss database. Set the scenario, and then adjust the database according to the check result.

The syntax is as follows:

Check a single-item.

```
gs_check -i ITEM [...] [-U USER] [-L] [-l LOGFILE] [-o OUTPUTDIR] [--skip-root-items][--set][--routing]
```

Check a scenario.

```
gs_check -e SCENE_NAME [-U USER] [-L] [-l LOGFILE] [-o OUTPUTDIR] [--hosts] [--skip-root-items] [--time-out=SECS][--set][--routing][--skip-items]
```

Scenario-specific check items. Default scenarios include **inspect** (routine inspection), **upgrade** (pre-upgrade inspection), **binary_upgrade** (local pre-upgrade inspection), **health** (health check inspection), and **install** (installation). You can also define scenarios as required.

Display help information.

```
gs_check -? | --help
```

1.4.3.2.1 Procedure

Step 1 Log in to the operating system where the openGauss database service is installed as the **root** user and run the **su – omm** command to switch to the **omm** user environment. The following information is displayed after the login:

```
[root@ecs-opengauss ~]# su - omm
Last login: Fri Jun 18 19:56:55 CST 2021 on pts/1


Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time:    Fri Jun 18 21:09:12 CST 2021

System load:       0.46
Processes:         148
Memory used:       13.1%
Swap used:         0.0%
Usage On:          14%
IP address:        192.168.0.203
Users online:      3
```

Step 2 Check whether the openGauss database service is started.

```
[omm@ecs-opengauss ~]$ gs_om -t status;
-----------------------------------------------------------------------

cluster_name       : dbCluster
cluster_state      : Normal
redistributing     : No

-----------------------------------------------------------------------
```

cluster_state: Normal indicates that the cluster is started and can be used normally. If the status is not **Normal**, the service is unavailable.

If the database service has been started, perform step 3 to stop the service.

Step 3 Stop the openGauss database service.

```
[omm@ecs-opengauss ~]$ gs_om -t stop;
Stopping cluster.
=========================================
Successfully stopped cluster.
=========================================
End stop cluster.
```

Step 4 Check the connection of the openGauss instance.

```
[omm@ecs-opengauss ~]$ gs_check -i CheckDBConnection
Parsing the check items config file successfully
Distribute the context file to remote hosts successfully
Start to health check for the cluster. Total Items:1 Nodes:1

Checking...                          [=========================] 1/1
Start to analysis the check result
CheckDBConnection.............................NG
The item run on 1 nodes.    ng: 1
```

```
The ng[ecs-e1b3] value:
The database can not be connected.

Analysis the check result successfully
Failed.    All check items run completed. Total:1        NG:1
For more information please refer to
/opt/huawei/wisequery/script/gspylib/inspection/output/CheckReport_2020072139449163171.tar.gz
```

Notes:

CheckDBConnection.....NG indicates that the connection check item is useless.

The database can not be connected. indicates that the instance cannot be connected.

Failed. All check items run completed. Total:1 NG:1 indicates that one item is checked and the check fails.

Step 5 Start the openGauss database service.

```
[omm@ecs-opengauss ~]$ gs_om -t start;
Starting cluster.
=========================================
[SUCCESS] ecs-opengauss
2021-06-18 21:19:28.470 60cc9d60.1 [unknown] 281464957239312 [unknown] 0 dn_6001 01000    0 [BACKEND]
WARNING:    could not create any HA TCP/IP sockets
2021-06-18 21:19:28.478 60cc9d60.1 [unknown] 281464957239312 [unknown] 0 dn_6001 01000    0 [BACKEND]
WARNING:    No explicit IP is configured for listen_addresses GUC.
2021-06-18 21:19:28.478 60cc9d60.1 [unknown] 281464957239312 [unknown] 0 dn_6001 01000    0 [BACKEND]
WARNING:    Failed to initialize the memory protect for g_instance.attr.attr_storage.cstore_buffers (1024 Mbytes) or
shared memory (4361 Mbytes) is larger.
=========================================
Successfully started.
```

Step 6 Check whether the openGauss database service is started.

```
 [omm@ecs-opengauss ~]$ gs_om -t status;
-----------------------------------------------------------------------


cluster_name       : dbCluster
cluster_state      : Normal
redistributing     : No


-----------------------------------------------------------------------
```

Step 7 Check the connection of the openGauss instance again.

```
[omm@ecs-opengauss ~]$ gs_check -i CheckDBConnection
Parsing the check items config file successfully
Distribute the context file to remote hosts successfully
Start to health check for the cluster. Total Items:1 Nodes:1

Checking...                          [=========================] 1/1
Start to analysis the check result
CheckDBConnection.....................OK
```

```
The item run on 1 nodes.    success: 1

Analysis the check result successfully
Success. All check items run completed. Total:1      Success:1
For more information please refer to
/opt/huawei/wisequery/script/gspylib/inspection/output/CheckReport_2020072140672174672.tar.gz
```

Notes:

CheckDBConnection.....OK indicates that the connection is normal.

Success. All check items run completed. Total:1 Success:1 indicates that one item is checked and the check is successful.

1.4.3.3 gs_checkperf

1.4.3.3.1 Using the gs_checkperf Tool to Check the Database Performance

Notes:

gs_checkperf can check the following items:

- openGauss-level metrics (such as host CPU usage, Gauss CPU usage, and I/O usage)
- Node-level metrics (such as CPU usage, memory usage, and I/O usage)
- Session/Process-level metrics (such as CPU usage, memory usage, and I/O usage)
- SSD performance metrics (read and write performance)

You need to check the SSD performance as the **root** user and check the openGauss performance as the openGauss installation user.

This experiment is to check the openGauss performance.

Step 1 Log in to the operating system where the openGauss database service is installed as the **root** user and run the **su – omm** command to switch to the **omm** user environment. The following information is displayed after the login:

```
[root@ecs-opengauss ~]# su - omm
Last login: Fri Jun 18 21:09:12 CST 2021 on pts/2
Last failed login: Fri Jun 18 21:26:19 CST 2021 on pts/2
There was 1 failed login attempt since the last successful login.


Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time:    Fri Jun 18 21:27:35 CST 2021

System load:       0.51
Processes:         155
Memory used:       13.1%
Swap used:         0.0%
Usage On:          14%
IP address:        192.168.0.203
Users online:      4
```

Step 2 Start the database service, and then use gs_checkperf to check whether the database service is started. Use the gsql client to connect to the **postgres** database as an administrator. Assume that the port number is 26000.

Start the database service.

```
[omm@ecs-opengauss ~]$ gs_om -t start;
Starting cluster.
=========================================
[SUCCESS] ecs-opengauss:
[2021-06-18 21:29:14.292][35641][][gs_ctl]: gs_ctl started,datadir is /gaussdb/data
[2021-06-18 21:29:14.297][35641][][gs_ctl]:    another server might be running; Please use the restart command
=========================================
Successfully started.
```

Run **gs_checkperf**.

```
[omm@ecs-opengauss ~]$ gs_checkperf
Cluster statistics information:
      Host CPU busy time ratio                    :      11.76         %
      MPPDB CPU time % in busy time               :       .81          %
      Shared Buffer Hit ratio                     :      99.50         %
      In-memory sort ratio                        :      0
      Physical Reads                              :      358
      Physical Writes                             :      0
      DB size                                     :      38            MB
      Total Physical writes                       :      0
      Active SQL count                            :      4
Session count                                     :      6
```

Check whether the openGauss database service is normal.

```
[omm@ecs-opengauss ~]$ gs_om -t status;
-----------------------------------------------------------------------


cluster_state     : Unavailable
redistributing    : No


-----------------------------------------------------------------------
```

cluster_state: Normal indicates that the cluster is started and can be used normally. If the status is **Unavailable**, the service is unavailable.

To continue the experiment, start the database service first.

Start the database service. (If the database service is normal, skip this step.)

```
[omm@ecs-opengauss ~]$ gs_om -t start;
Starting cluster.
=========================================
=========================================
Successfully started.
```

Connect to the **postgres** database.

```
[omm@ecs-opengauss ~]$ gsql -d postgres -p 26000 -r
```

```
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr   )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

Step 3 This command is used to collect statistics about tables in PMK mode.

```
postgres=# analyze pmk.pmk_configuration;
ANALYZE
postgres=# analyze pmk.pmk_meta_data;
ANALYZE
postgres=# analyze pmk.pmk_snapshot;
ANALYZE
postgres=# analyze pmk.pmk_snapshot_datanode_stat;
ANALYZE
postgres=#
```

Notes:

- The monitoring information of gs_checkperf depends on the data of tables in pmk mode. If the analyze operation is not performed on tables in pmk mode, gs_checkperf may fail to be executed.

Step 4 Perform a brief performance check.

Run the **\q** command to exit the **postgres** database, and then run the gs_checkperf tool in the **omm** user environment.

```
postgres=#
postgres=# \q
[omm@ecs-opengauss ~]$ gs_checkperf
Cluster statistics information:
        Host CPU busy time ratio                    :       11.76          % -----Host CPU usage
        MPPDB CPU time % in busy time               :       1.40           % ----Gauss CPU usage
        Shared Buffer Hit ratio                     :       99.50          % ----Shared memory hit ratio
        In-memory sort ratio                        :       0                ---Sorting ratio in the memory
        Physical Reads                              :       398              ---Number of physical reads
        Physical Writes                             :       98               ---Number of physical writes
        DB size                                     :       38               MB---Database size
        Total Physical writes                       :       98               ---Total number of physical writes
        Active SQL count                            :       4                ---Number of executed SQL statements
Session count                                       :       6              ---Number of sessions
```

Step 5 Perform a detailed performance check.

```
[omm@ecs-opengauss ~]$ gs_checkperf --detail
Cluster statistics information:
Host CPU usage rate:
        Host total CPU time                         :       348558610.000 Jiffies
        Host CPU busy time                          :       41004320.000 Jiffies
        Host CPU iowait time                        :       11310.000    Jiffies
        Host CPU busy time ratio                    :       11.76          %
```

```
        Host CPU iowait time ratio                  :       0.00          %
MPPDB CPU usage rate:
        MPPDB CPU time % in busy time               :       1.62          %
        MPPDB CPU time % in total time              :       .19           %
Shared buffer hit rate:
        Shared Buffer Reads                         :       3213
        Shared Buffer Hits                          :       636030
        Shared Buffer Hit ratio                     :       99.50         %
In memory sort rate:
        In-memory sort count                        :       0
        In-disk sort count                          :       0
        In-memory sort ratio                        :       0
I/O usage:
        Number of files                             :       95
        Physical Reads                              :       531
        Physical Writes                             :       98
        Read Time                                   :       30980         ms
        Write Time                                  :       3175          ms
Disk usage:
        DB size                                     :       38            MB
        Total Physical writes                       :       98
        Average Physical write                      :       30866.14
        Maximum Physical write                      :       98
Activity statistics:
        Active SQL count                            :       4
        Session count                               :       6
Node statistics information:
dn_6001:
        MPPDB CPU Time                              :       662800        Jiffies
        Host CPU Busy Time                          :       41004320      Jiffies
        Host CPU Total Time                         :       348558610     Jiffies
        MPPDB CPU Time % in Busy Time               :       1.62          %
        MPPDB CPU Time % in Total Time              :       .19           %
        Physical memory                             :       7143948288 Bytes
        DB Memory usage                             :       6266421248 Bytes
        Shared buffer size                          :       1073741824 Bytes
        Shared buffer hit ratio                     :       99.50         %
        Sorts in memory                             :       0
        Sorts in disk                               :       0
        In-memory sort ratio                        :       0
        Number of files                             :       95
        Physical Reads                              :       531
        Physical Writes                             :       98
        Read Time                                   :       30980
        Write Time                                  :       3175
Session statistics information(Top 10):
Session CPU statistics:
1 dn_6001-postgres-omm:
```

```
        Session CPU time                        :       2
        Database CPU time                       :       663070
        Session CPU time %                      :       0.00            %
2 dn_6001-postgres-omm:
        Session CPU time                        :       0
        Database CPU time                       :       663070
        Session CPU time %                      :       0.00            %
3 dn_6001-postgres-omm:
        Session CPU time                        :       0
        Database CPU time                       :       663070
        Session CPU time %                      :       0.00            %
4 dn_6001-postgres-omm:
        Session CPU time                        :       0
        Database CPU time                       :       663070
        Session CPU time %                      :       0.00            %

Session Memory statistics:
1 dn_6001-postgres-omm:
        Buffer Reads                    :       1010
        Shared Buffer Hit ratio         :       100.00
        In Memory sorts                 :       1
        In Disk sorts                   :       0
        In Memory sorts ratio           :       100.00
        Total Memory Size               :       8879168
        Used Memory Size                :       6787952
2 dn_6001-postgres-omm:
        Buffer Reads                    :       1514
        Shared Buffer Hit ratio         :       93.17
        In Memory sorts                 :       0
        In Disk sorts                   :       0
        In Memory sorts ratio           :       0
        Total Memory Size               :       5881992
        Used Memory Size                :       4563032
3 dn_6001-postgres-omm:
        Buffer Reads                    :       284
        Shared Buffer Hit ratio         :       100.00
        In Memory sorts                 :       0
        In Disk sorts                   :       0
        In Memory sorts ratio           :       0
        Total Memory Size               :       5878536
        Used Memory Size                :       4504656
4 dn_6001-postgres-omm:
        Buffer Reads                    :       284
        Shared Buffer Hit ratio         :       100.00
        In Memory sorts                 :       0
        In Disk sorts                   :       0
        In Memory sorts ratio           :       0
        Total Memory Size               :       5857416
```

```
      Used Memory Size                    :      4509632

Session IO statistics:
1 dn_6001-postgres-omm:
      Physical Reads                      :      111
      Read Time                           :      2100
2 dn_6001-postgres-omm:
      Physical Reads                      :      0
      Read Time                           :      0
3 dn_6001-postgres-omm:
      Physical Reads                      :      0
      Read Time                           :      0
4 dn_6001-postgres-omm:
      Physical Reads                      :      0
      Read Time                           :      0
[omm@ecs-opengauss ~]$
```

2 Database and Object Management

2.1 Overview

2.1.1 About This Experiment

This experiment describes how to create, query, modify, and delete tablespaces and databases, and how to create, query, modify, and delete row-store tables, column-store tables, and MOTs. It also describes how to manage users, roles, and schemas, and how to import and export data.

2.1.2 Objectives

- Master how to create, query, modify, and delete tablespaces.
- Master how to create, query, modify, and delete databases.
- Master how to create, query, modify, and delete row-store tables, column-store tables, and MOTs.
- Master how to manage users, roles, and schemas.
- Master how to import and export data in different methods.

2.2 Tasks

2.2.1 Logging In to a Database

Step 1 Use the Shell tool to log in to the ECS.

Step 2 Switch from the **root** user to the **omm** database user.

```
[root@opengauss ~]# su - omm
```

Step 3 Log in to a database.

```
[omm@opengauss ~]$ gsql -d postgres -p 26000 -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

2.2.2 Creating, Querying, Modifying, and Deleting Tablespaces

The administrator can use tablespaces to control the layout of disks where a database is installed. The following are the advantages:

1. Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance. A frequently used index can be placed in a disk having stable performance and high computing speed, such as a solid-state device. A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.

2. The administrator can use tablespaces to set the maximum available disk space. In this way, when a partition is shared with other data, tablespaces will not occupy excessive space in the partition.

3. You can use tablespaces to control the disk space occupied by data in a database. If the usage of a disk where a tablespace resides reaches 90%, the database switches to read-only mode. It switches back to read/write mode when the disk usage becomes less than 90%.

4. A tablespace corresponds to a file system directory, and you must have the read and write permissions on an empty directory.

5. If the tablespace quota management is used, the performance may deteriorate by about 30%. **MAXSIZE** specifies the maximum quota for each database node. The deviation must be within 500 MB. Determine whether to set a tablespace to its maximum size as required.

Step 1 Create a tablespace.

```
postgres=# create tablespace test_tbs LOCATION '/opt/opengauss/tablespace/test_tbs1';
CREATE TABLESPACE
```

If **CREATE TABLESPACE** is displayed, the tablespace is successfully created.

In the preceding command, **test_tbs** is the newly created tablespace, **/opt/opengauss/tablespace/test_tbs1** is the empty directory on which the user has the read and write permissions, and is also the directory for storing data files.

Step 2 Create a user and grant the user the permissions to access the **test_tbs** tablespace.

```
postgres=# create user jack IDENTIFIED BY 'xxxxxxxx';
CREATE ROLE
postgres=# GRANT CREATE ON TABLESPACE test_tbs TO jack;
GRANT
```

In the preceding command, **'xxxxxxxx'** indicates the user password, which must be customized and meet the password complexity requirements. After the permission is granted, user **jack** has the permissions to create database objects under **test_tbs**.

Step 3 Create a table in the **test_tbs** tablespace.

Method 1:

```
postgres=# create table test (id int) tablespace test_tbs;
CREATE TABLE
```

Method 2:

```
postgres=# SET default_tablespace = test_tbs;
SET
postgres=# create table test1 (id int);
CREATE TABLE
```

Step 4 Query a tablespace.

Method 1: Check the **pg_tablespace** system catalog. View all the tablespaces defined by the system and users.

```
postgres=# SELECT spcname FROM pg_tablespace;
   spcname
------------
 pg_default
 pg_global
 test_tbs
(3 rows)
```

Method 2: Run the following meta-command of the gsql program to query the tablespaces:

```
postgres=# \db
                              List of tablespaces
      Name       | Owner |                      Location
-----------+-------+-----------------------------------
 pg_default | omm      |
 pg_global    | omm      |
 test_tbs      | omm      | /opt/opengauss/tablespace/test_tbs1
(3 rows)
```

Step 5 Query the tablespace usage.

Query the current usage of the tablespace. The result indicates the size of the tablespace, in bytes.

```
postgres=# SELECT PG_TABLESPACE_SIZE('test_tbs');
 pg_tablespace_size
--------------------
                8192
(1 row)
```

Calculate the tablespace usage.

Tablespace usage rate = Value of **PG_TABLESPACE_SIZE**/Size of the disk where the tablespace resides

Step 6 Modify the tablespace.

Run the following command to rename the tablespace:

```
postgres=# ALTER TABLESPACE test_tbs RENAME TO test_tbs1;
ALTER TABLESPACE
```

Step 7 Delete a tablespace.

Run the following command to delete the tablespace: Only the tablespace owner or system administrator can delete a tablespace.

```
postgres=# drop tablespace test_tbs1;
ERROR:    tablespace "test_tbs1" is not empty
```

An error is reported, indicating that the tablespace is not empty. You need to delete the objects created in the tablespace and then delete the tablespace.

```
postgres=# drop table test;
DROP TABLE
postgres=# drop table test1;
DROP TABLE
postgres=#    drop tablespace test_tbs1;
```

```
DROP TABLESPACE
```

2.2.3 Creating, Viewing, Modifying, and Deleting a Database

The user must have the permission to create a database or the permission that the system administrator owns.

Step 1 Create two tablespaces **db_tbs** and **db_tbs1**.

```
postgres=#    CREATE TABLESPACE db_tbs LOCATION '/opt/opengauss/tablespace/db_tbs1';
postgres=#    CREATE TABLESPACE db_tbs1 LOCATION '/opt/opengauss/tablespace/db_tbs2';
CREATE TABLESPACE
CREATE TABLESPACE
```

Step 2 Create the **testdb** database in the **db_tbs** tablespace.

```
postgres=# CREATE DATABASE testdb with    TABLESPACE = db_tbs;
CREATE DATABASE
```

Step 3 View a database.

Method 1: Query the database list in the **pg_database** system catalog.

```
postgres=# SELECT datname FROM pg_database;
   datname
-----------
 template1
testdb
 template0
 postgres
(4 rows)
```

Method 2: Run the **\l** meta-command to view the database list of the database system.

```
postgres=# \l
                                  List of databases
    Name      | Owner | Encoding |     Collate     |      Ctype      | Access privileges
-----------+-------+----------+-------------+-------------+-------------------
 postgres  | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/omm               +
           |       |          |             |             | omm=CTc/omm
 template1 | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/omm               +
           |       |          |             |             | omm=CTc/omm
 testdb    | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 (4 rows)
```

Step 4 Modify a database.

Modify the default search path of the database.

```
postgres=# ALTER DATABASE testdb SET search_path TO pa_catalog,public;
ALTER DATABASE
```

Change the database tablespace.

```
postgres=#    ALTER DATABASE testdb SET tablespace db_tbs1;
```

```
ALTER DATABASE
```

Rename the database.

```
postgres=# ALTER DATABASE testdb RENAME TO testdb1;
ALTER DATABASE
```

Step 5 Delete a database.

```
postgres=# DROP DATABASE testdb1;
DROP DATABASE
```

2.2.4 Creating, Viewing, Modifying, and Deleting Row-store Tables, Column-store Tables, and MOTs

Application scenarios of row-store tables:

Point queries (simple index-based queries that only return a few records).

Scenarios requiring frequent addition, deletion, and modification.

Application scenarios of column-store tables:

Statistical analysis queries (requiring a large number of association and grouping operations).

Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables).

Step 1 Create a row-store table. By default, the table is a row-store table, that is, **WITH (ORIENTATION = ROW)** can be omitted.

```
postgres=# CREATE TABLE PART
(
      P_PARTKEY        BIGINT NOT NULL,
  P_NAME              VARCHAR(55) NOT NULL,
  P_MFGR              CHAR(25) NOT NULL,
  P_BRAND             CHAR(10) NOT NULL,
  P_TYPE             VARCHAR(25) NOT NULL,
  P_SIZE             BIGINT NOT NULL,
  P_CONTAINER      CHAR(10) NOT NULL,
  P_RETAILPRICE DECIMAL(15,2) NOT NULL,
  P_COMMENT         VARCHAR(23) NOT NULL
)
WITH (ORIENTATION = ROW);
```

Step 2 Create a column-store table.

```
postgres=# CREATE TABLE PART1
(
      P_PARTKEY        BIGINT NOT NULL,
  P_NAME              VARCHAR(55) NOT NULL,
  P_MFGR              CHAR(25) NOT NULL,
  P_BRAND             CHAR(10) NOT NULL,
  P_TYPE             VARCHAR(25) NOT NULL,
  P_SIZE             BIGINT NOT NULL,
  P_CONTAINER      CHAR(10) NOT NULL,
```

```
  P_RETAILPRICE DECIMAL(15,2) NOT NULL,
  P_COMMENT         VARCHAR(23) NOT NULL
)
WITH (ORIENTATION = COLUMN);
```

Step 3 View the table information.

Run the **\d** meta-command to view the table list.

```
postgres=# \d
                                  List of relations
 Schema | Name  | Type  | Owner |                      Storage
--------+-------+-------+-------+-----------------------------------
 public | part  | table | omm   | {orientation=row,compression=no}
 public | part1 | table | omm   | {orientation=column,compression=low}
(2 rows)
```

Step 4 Modify table attributes.

Adds a column.

```
postgres=# ALTER TABLE part ADD COLUMN p_col1 bigint;
ALTER TABLE
```

Add the default value in the column.

```
postgres=# ALTER TABLE part ALTER COLUMN p_col1 SET DEFAULT 1;
ALTER TABLE
```

Delete the default value in the column.

```
postgres=# ALTER TABLE part ALTER COLUMN p_col1 drop DEFAULT ;
ALTER TABLE
```

Change the data type of a column.

```
postgres=# ALTER TABLE part MODIFY p_col1 INT;
ALTER TABLE
```

Rename a column.

```
postgres=# ALTER TABLE part RENAME p_col1 to p_col;
ALTER TABLE
```

Deletes a column.

```
postgres=# ALTER TABLE part DROP COLUMN p_col;
ALTER TABLE
```

Step 5 Create an MOT.

```
postgres=#    create FOREIGN table test(x int) ;
CREATE TABLE
```

If an error message "ERROR: Cannot create MOT tables while incremental checkpoint is enabled" is displayed, modify the **enable_incremental_checkpoint** parameter to create an MOT after the database is restarted.

```
[omm@opengauss ~]$    gs_guc set -N all -I all -c "enable_incremental_chec=off"
Begin to perform the total nodes: 1.
```

```
Popen count is 1, Popen success count is 1, Popen failure count is 0.
Begin to perform gs_guc for datanodes.
Command count is 1, Command success count is 1, Command failure count is 0.

Total instances: 1. Failed instances: 0.
ALL: Success to perform gs_guc!
[omm@opengauss ~]$ gs_om -t restart
[omm@opengauss ~]$ gsql -d postgres -p 26000 -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commi
Non-SSL connection (SSL connection is recommended when requiring high-securi
Type "help" for help.

postgres=# create foreign table abc_test(id int);
CREATE FOREIGN TABLE
```

Step 6 Create an index for the MOT.

```
postgres=#    create index    text_index1 on test(x) ;
CREATE INDEX
```

Step 7 Delete a table.

```
postgres=# DROP TABLE PART;
DROP TABLE
postgres=# DROP TABLE PART1;
DROP TABLE
postgres=# DROP TABLE test;
DROP TABLE
```

2.2.5 User, Role, and Schema Management

2.2.5.1 User Management

A user created using the **CREATE USER** statement has the **LOGIN** permission by default.

When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed. In other databases, the schema with the same name is not automatically created.

You can run the **CREATE SCHEMA** command to create a schema with the same name in other databases.

Step 1 Access the SQL command page after connecting to the database. Create the **jim** user and set its login password to **Bigdata@123**.

```
postgres=# CREATE USER jim PASSWORD 'Bigdata@123';
CREATE ROLE
```

A password must:

Contain at least eight characters. This is the default length.

Differ from the user name or the user name spelled backward.

Contain at least three types of the following four types of characters: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and special characters, including: ~!@#$%^&*()-_=+\|[{}];:,<.>/?

Be enclosed by single or double quotation marks.

Step 2 Check the user list.

```
postgres=# SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |    passwd    | valbegin | valuntil |      respool
| parent | spacelimit | useconfig | nodegroup |
 tempspacelimit | spillspacelimit
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+-------------+--------+------------+----------
-+-----------+
----------------+-----------------
 omm         |           10 | t               | t            | t             | t          | ******** |              |
| default_pool |          0 |                 |              |               |
                       |
 jim         |       16389 | f               | f            | f             | f          | ******** |              |              |
default_pool |          0 |                 |              |               |
                       |
(2 rows)
```

Step 3 For a user having the **Create Database** permission, add the **CREATEDB** keyword:

```
postgres=# CREATE USER dim CREATEDB PASSWORD 'Bigdata@123';
CREATE ROLE
```

Step 4 Change the login password of **jim** from **Bigdata@** to **Abcd@123**.

```
postgres=# ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'Bigdata@123';
ALTER ROLE
```

Step 5 Grant the **CREATEROLE** permission to user **jim**.

```
postgres=# ALTER USER jim CREATEROLE;
ALTER ROLE
```

Step 6 Lock the **jim** account:

```
postgres=# ALTER USER jim ACCOUNT LOCK;
ALTER ROLE
```

Step 7 Unlock the **jim** account.

```
postgres=# ALTER USER jim ACCOUNT UNLOCK;
ALTER ROLE
```

Step 8 Delete a user.

```
postgres=# DROP USER jim CASCADE;
DROP ROLE
```

2.2.5.2 Role Management

A role is an entity that has own database objects and permissions.

In different environments, a role can be considered a user, a group, or both.

Add a role to the database. The role does not have the login permission.

Only the user who has the **CREATE ROLE** permission or a system administrator is allowed to create roles.

Step 1 Create role **manager** whose password is **Bigdata123@**.

```
postgres=# CREATE ROLE manager IDENTIFIED BY 'Bigdata@123';
CREATE ROLE
```

Step 2 Create a role. The role takes effect from July 1, 2020 and expires on December 1, 2020.

```
postgres=# CREATE ROLE miriam WITH LOGIN PASSWORD 'Bigdata@123' VALID BEGIN '2020-07-01' VALID UNTIL
'2020-12-01';
CREATE ROLE
```

Step 3 Change role **manager** to the system administrator:

```
postgres=# ALTER ROLE manager SYSADMIN;
ALTER ROLE
```

Step 4 Delete the role **manager**.

```
postgres=# DROP ROLE manager;
DROP ROLE
```

Step 5 View the roles.

```
postgres=# SELECT * FROM PG_ROLES;
  rolname | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit | rol
password |         rolvalidbegin          |         rolvaliduntil          |    rolrespool    | rolparentid | roltabspace |
rolconfig |    oid    | roluseft | rolkind | nodegroup | rolte
mpspace | rolspillspace
---------+----------+-----------+---------------+-------------+-------------+------------+---------------+--------------+---------------+------
-------+----
---------+-----------------------+-----------------------+-------------+------------+------------+----------+-------+---------+---------+------
-----+------
--------+---------------
 omm       | t           | t             | t               | t             | t               | t              | t
| t               | t                 |             -1 | ***
*****     |                               |                                 | default_pool |             0 |
|              |     10 | t          | n          |             |
          |
 dim       | f           | t             | f               | t             | f               | t              | f
| f               | f                 |             -1 | ***
*****     |                               |                                 | default_pool |             0 |
|              | 16393 | f          | n          |             |
          |
 miriam    | f           | t             | f               | f             | f               | t              | f
| f               | f                 |             -1 | ***
*****     | 2020-07-01 00:00:00+08 | 2020-12-01 00:00:00+08 | default_pool |             0 |                |
| 16401 | f           | n          |             |
```

```
              |
(3 rows)
```

2.2.5.3 Schema Management

Schemas function as models.

Schema management allows multiple users to use the same database without interfering with each other.

Each database has one or more schemas.

When a user is created in the database, the system automatically creates a schema with the same name as the user.

Step 1 Create the **ds** schema.

```
postgres=# CREATE SCHEMA ds;
CREATE SCHEMA
```

Step 2 Rename the **ds** schema to **ds_new**.

```
postgres=# ALTER SCHEMA ds RENAME TO ds_new;
ALTER SCHEMA
```

Step 3 Create a user named **jack**.

```
postgres=# CREATE USER jack PASSWORD 'Bigdata@123';
CREATE ROLE
```

Step 4 Change the owner of **ds_new** to **jack**.

```
postgres=# ALTER SCHEMA ds_new OWNER TO jack;
ALTER SCHEMA
```

Step 5 View the schema owner.

```
postgres=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE s.nspowner =
u.usesysid;
        nspname          | nspowner
--------------------+----------
 pg_toast                | omm
 cstore                  | omm
 dbe_perf                | omm
 snapshot                | omm
 pg_catalog              | omm
 public                  | omm
 information_schema | omm
 dim                       | dim
 jack                     | jack
 ds_new                  | jack
(10 rows)
```

Step 6 Delete user **jack** and the **ds_new** schema.

```
postgres=# DROP SCHEMA ds_new;
```

```
DROP SCHEMA
postgres=# DROP USER jack;
DROP ROLE
```

2.2.5.4 Granting System Permissions to Users or Roles

Step 1 Create user **joe** and grant the **sysadmin** permissions to the user.

```
postgres=# CREATE USER joe PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# ALTER USER joe with sysadmin;
ALTER ROLE
```

2.2.5.5 Granting Database Object Permissions to Roles or Users

Step 1 Revoke the **sysadmin** permission of user **joe**, create the **tpcds** schema, and create a **reason** table for the **tpcds** schema.

```
postgres=# ALTER USER joe with nosysadmin;
ALTER ROLE
postgres=# CREATE SCHEMA tpcds;
CREATE SCHEMA
postgres=# CREATE TABLE tpcds.reason
(
      r_reason_sk               INTEGER                   NOT NULL,
      r_reason_id               CHAR(16)                  NOT NULL,
      r_reason_desc                   VARCHAR(20)
);
CREATE TABLE
```

Step 2 Grant the usage permission on the **tpcds** schema and all the permissions on the **tpcds.reason** table to the user **joe**.

```
postgres=# GRANT USAGE ON SCHEMA tpcds TO joe;
GRANT
postgres=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
GRANT
```

After the granting succeeds, user **joe** has all the permissions on the **tpcds.reason** table, including addition, deletion, modification, and query permissions.

Step 3 Grant the query permission on **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns and the update permission on the **r_reason_desc** column in the **tpcds.reason** table to user **joe**.

```
postgres=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO joe;
GRANT
```

Step 4 Grant the **postgres** database connection permission and schema creation permission to user **joe**, and enable this user to grant these permissions to other users.

```
postgres=# GRANT create,connect on database postgres TO joe WITH GRANT OPTION;
GRANT
```

Step 5 Create role **tpcds_manager**, grant the **tpcds** schema access permission and object creation permission to this role, but do not enable this role to grant these permissions to others.

```
postgres=# CREATE ROLE tpcds_manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
GRANT
```

2.2.5.6 Granting the Permissions of One User or Role to Others

Step 1 Create the **manager** role, grant **joe**'s permissions to **manager**, and allow **manager** to grant these permissions to others.

```
postgres=# CREATE ROLE manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT joe TO manager WITH ADMIN OPTION;
GRANT ROLE
```

Step 2 Create role **senior_manager** and grant **manager**'s permissions to **senior_manager**.

```
postgres=# CREATE USER senior_manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT manager TO senior_manager;
GRANT ROLE
```

2.2.5.7 Revoking Permissions

Step 1 Revoke permissions and delete users.

```
postgres=# REVOKE manager FROM joe;
REVOKE ROLE
postgres=# REVOKE manager FROM joe;
REVOKE ROLE
postgres=# REVOKE senior_manager FROM manager;
REVOKE ROLE
postgres=# DROP USER manager;
DROP ROLE
postgres=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
REVOKE
postgres=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
REVOKE
postgres=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
REVOKE
postgres=# DROP ROLE tpcds_manager;
DROP ROLE
postgres=# DROP ROLE senior_manager;
DROP ROLE
postgres=# DROP USER joe CASCADE;
DROP ROLE
```

2.2.6 Data Import and Export

gs_dump, provided by openGauss, is used to export database information. You can export a database or its objects (such as schemas, tables, and views). The database can be the default **postgres** database or a user-specified database.

2.2.6.1 Creating a Backup Directory

Create a backup directory as user **omm**.

```
[omm@opengauss]$ mkdir /gaussdb/backup
[omm@opengauss]$ cd /gaussdb/backup
```

2.2.6.2 Creating a Table and Inserting Data

Step 1 Log in to a database.

```
[omm@opengauss backup]$ gsql -d postgres -p 26000 -r
```

Step 2 Create the **customer_t1** table.

```
postgres=# drop table if exists customer_t1;
NOTICE:  table "customer_t1" does not exist, skipping
DROP TABLE
postgres=# CREATE TABLE customer_t1
postgres-# (
postgres(#        c_customer_sk             integer,
postgres(#        c_customer_id             char(5),
postgres(#        c_first_name              char(6),
postgres(#        c_last_name               char(8)
postgres(# );
CREATE TABLE
```

Step 3 Insert data into the **customer_t1** table.

```
postgres=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
postgres-# (3769, 'hello', DEFAULT) ,
postgres-#        (6885, 'maps', 'Joes'),
postgres-#        (4321, 'tpcds', 'Lily'),
postgres-# (9527, 'world', 'James');
INSERT 0 4
postgres=# INSERT 0 4
```

Step 4 View data in the **customer_t1** table.

```
postgres=# select * from customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3769 | hello         |              |
          6885 | maps          | Joes         |
          4321 | tpcds         | Lily         |
          9527 | world         | James        |
(4 rows)
```

Step 5 Create the user **joe**.

```
postgres=#    drop user if exists joe;
NOTICE:    role "joe" does not exist, skipping
DROP ROLE
postgres=#
postgres=# CREATE USER joe    WITH PASSWORD "Bigdata@123";
CREATE ROLE
```

Step 6 Create the **mytable** table as user **joe**.

```
postgres=# \q
[omm@ opengauss ~]$ gsql -d postgres -U joe -W Bigdata@123 -p 26000 -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr    )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=> drop table if exists mytable;
DROP TABLE
postgres=> CREATE TABLE mytable (firstcol int);
CREATE TABLE
```

Step 7 Insert data into the **mytable** table.

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

Step 8 View the **mytable** file.

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

Step 9 Display information about the current table.

```
postgres=> \d
                                List of relations
 Schema |       Name        | Type  | Owner |                Storage
--------+-------------------+-------+-------+----------------------------------
 joe    | mytable           | table | joe   | {orientation=row,compression=no}
 public | customer_t1 | table |             | {orientation=row,compression=no}
(2 rows)
```

2.2.6.3 Exporting Data Using the copy Command

Step 1 Switch to the default user **omm**.

```
postgres=> \c - omm
```

Step 2 Run the **copy** command to export data.

```
postgres=#    copy customer_t1 to '/gaussdb/backup/copy_cost.txt' delimiter '^';
COPY 4
postgres=# \q
```

Step 3 View the exported file.

```
[omm@opengauss ~]$ cd /gaussdb/backup/
[omm@opengauss backup]$ ll
total 4.0K
-rw------- 1 omm dbgrp 80 Jun 17 14:39 copy_cost.txt
[omm@opengauss backup]$ more copy_cost.txt
3769^hello^\N^\N
6885^maps ^Joes    ^\N
4321^tpcds^Lily    ^\N
9527^world^James ^\N
```

2.2.6.4 Exporting Data Using the gs_dump Command

Step 1 Run the **gs_dump** command to export the **postgres** database data to a plain-text file.

Some important **gs_dump** commands are described as follows:

-f: Folder to store exported files.

-F: Format of the file to be exported. The values of **-F** are as follows:

p: plaintext

c: custom

d: directory

t: .tar

-n: Names of schemas to be exported. Data of the specified schemas will also be exported.

-t: Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters.

-T: A list of tables, views, sequences, or foreign tables cannot be dumped. You can use multiple **-T** options or wildcard characters to specify tables.

```
[omm@opengauss backup]$ gs_dump -U omm -W Bigdata@123 -f /gaussdb/backup/gsdump_post.sql -p 26000
postgres -F p
gs_dump[port='26000'][postgres][2021-06-17 14:45:16]: The total objects number is 391.
gs_dump[port='26000'][postgres][2021-06-17 14:45:16]: [100.00%] 391 objects have been dumped.
gs_dump[port='26000'][postgres][2021-06-17 14:45:16]: dump database postgres successfully
gs_dump[port='26000'][postgres][2021-06-17 14:45:16]: total time: 254    ms
```

Step 2 View the exported file.

```
[omm@opengauss backup]$ cd /gaussdb/backup/
[omm@opengauss backup]$ ll
total 8.0K
-rw------- 1 omm dbgrp      80 Jun 17 14:39 copy_cost.txt
-rw------- 1 omm dbgrp 1.8K Jun 17 14:45 gsdump_post.sql
[omm@opengauss backup]$ cat test.sql|grep -v ^$
--
-- PostgreSQL database dump
```

```
--
SET statement_timeout = 0;
SET xmloption = content;
SET client_encoding = 'SQL_ASCII';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;
--
-- Name: postgres; Type: COMMENT; Schema: -; Owner: omm
--
COMMENT ON DATABASE postgres IS 'default administrative connection database';
--
-- Name: joe; Type: SCHEMA; Schema: -; Owner: joe
--
CREATE SCHEMA joe;
ALTER SCHEMA joe OWNER TO joe;
SET search_path = joe;
SET default_tablespace = '';
SET default_with_oids = false;
--
-- Name: mytable; Type: TABLE; Schema: joe; Owner: joe; Tablespace:
--
CREATE TABLE mytable (
    firstcol integer
)
WITH (orientation=row, compression=no);
ALTER TABLE joe.mytable OWNER TO joe;
SET search_path = public;
--
-- Name: customer_t1; Type: TABLE; Schema: public; Owner: omm; Tablespace:
--
CREATE TABLE customer_t1 (
    c_customer_sk integer,
    c_customer_id character(5),
    c_first_name character(6),
    c_last_name character(8)
)
WITH (orientation=row, compression=no);
ALTER TABLE public.customer_t1 OWNER TO omm;
SET search_path = joe;
--
-- Data for Name: mytable; Type: TABLE DATA; Schema: joe; Owner: joe
--
COPY mytable (firstcol) FROM stdin;
100
\.
;
SET search_path = public;
```

```
--
-- Data for Name: customer_t1; Type: TABLE DATA; Schema: public; Owner: omm
--
COPY customer_t1 (c_customer_sk, c_customer_id, c_first_name, c_last_name) FROM
stdin;
3769    hello   \N      \N
6885    maps    Joes    \N
4321    tpcds   Lily    \N
9527    world   James   \N
\.
;
--
-- Name: public; Type: ACL; Schema: -; Owner: omm
--
REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM omm;
GRANT CREATE,USAGE ON SCHEMA public TO omm;
GRANT USAGE ON SCHEMA public TO PUBLIC;
--
-- PostgreSQL database dump complete
--
```

Step 3 Use **gs_dump** to export the **postgres** database data in .tar format.

```
[omm@opengauss backup]$ gs_dump -U omm -W Bigdata@123 -f /gaussdb/backup/gsdump_post.tar -p 26000
postgres -F t
gs_dump[port='26000'][postgres][2021-06-17 14:56:50]: The total objects number is 391.
gs_dump[port='26000'][postgres][2021-06-17 14:56:50]: [100.00%] 391 objects have been dumped.
gs_dump[port='26000'][postgres][2021-06-17 14:56:50]: dump database postgres successfully
gs_dump[port='26000'][postgres][2021-06-17 14:56:50]: total time: 219  ms
```

Step 4 View the exported file.

```
[omm@opengauss backup]$ ll
total 20K
-rw------- 1 omm dbgrp    80 Jun 17 14:39 copy_cost.txt
-rw------- 1 omm dbgrp 1.8K Jun 17 14:45 gsdump_post.sql
-rw------- 1 omm dbgrp 9.5K Jun 17 14:56 gsdump_post.tar
```

Step 5 Use **gs_dump** to export tables-dependent views.

Connect to the database, create the **vw_customer_t1** view that depends on the **customer_t1** table, and exit.

```
[omm@opengauss backup]$ gsql -d postgres -p 26000 -r
postgres=#  create view vw_customer_t1 as select * from customer_t1 limit 2;
CREATE VIEW
postgres=# \q
```

Export the **vw_customer_t1** view that depends on the **customer_t1** table.

```
[omm@opengauss backup]$ gs_dump -s -p 26000 postgres -t PUBLIC.customer_t1 --include-depend-objs --exclude-
self -f /gaussdb/backup/view_cust.sql -F p
```

```
gs_dump[port='26000'][postgres][2021-06-17 15:05:36]: The total objects number is 379.
gs_dump[port='26000'][postgres][2021-06-17 15:05:36]: [100.00%] 379 objects have been dumped.
gs_dump[port='26000'][postgres][2021-06-17 15:05:36]: dump database postgres successfully
gs_dump[port='26000'][postgres][2021-06-17 15:05:36]: total time: 215  ms
```

View the exported file.

```
[omm@opengauss backup]$ cd /gaussdb/backup/
[omm@opengauss backup]$ ll
total 24K
-rw------- 1 omm dbgrp      80 Jun 17 14:39 copy_cost.txt
-rw------- 1 omm dbgrp 1.8K Jun 17 14:45 gsdump_post.sql
-rw------- 1 omm dbgrp 9.5K Jun 17 14:56 gsdump_post.tar
-rw------- 1 omm dbgrp    598 Jun 17 15:05 view_cust.sql
```

2.2.6.5 Exporting Data Using the gs_dumpall Command

gs_dumpall, provided by openGauss, is used to export all openGauss database information, including data of the default database **postgres**, user-defined databases, and common global objects of all openGauss databases.

gs_dumpall exports all databases in two parts:

- gs_dumpall exports all global objects, including information about database users and groups, tablespaces, and attributes (for example, global access permissions).
- gs_dumpall calls gs_dump to export SQL scripts from each openGauss database, which contain all the SQL statements required to restore databases.

The exported files are both plain-text SQL scripts. Use gsql to execute them to restore openGauss databases.

Step 1 Use **gs_dumpall** to export all openGauss databases at a time.

```
[omm@opengauss backup]$ gs_dumpall -f  /gaussdb/backup/gsdumpall.sql -p 26000
gs_dump[port='26000'][dbname='postgres'][2021-06-17 15:27:35]: The total objects number is 393.
gs_dump[port='26000'][dbname='postgres'][2021-06-17 15:27:35]: [100.00%] 393 objects have been dumped.
gs_dump[port='26000'][dbname='postgres'][2021-06-17 15:27:35]: dump database dbname='postgres' successfully
gs_dump[port='26000'][dbname='postgres'][2021-06-17 15:27:35]: total time: 223  ms
gs_dumpall[port='26000'][2021-06-17 15:27:35]: dumpall operation successful
gs_dumpall[port='26000'][2021-06-17 15:27:35]: total time: 257  ms
```

Step 2 View the exported file.

```
[omm@opengauss backup]$ cd /gaussdb/backup/
[omm@opengauss backup]$ ll
total 28K
-rw------- 1 omm dbgrp      80 Jun 17 14:39 copy_cost.txt
-rw------- 1 omm dbgrp 3.2K Jun 17 15:27 gsdumpall.sql
-rw------- 1 omm dbgrp 1.8K Jun 17 14:45 gsdump_post.sql
-rw------- 1 omm dbgrp 9.5K Jun 17 14:56 gsdump_post.tar
-rw------- 1 omm dbgrp    598 Jun 17 15:05 view_cust.sql
```

2.2.6.6 Importing Data Using the gsql Command

You can use **gsql** to import .sql files.

Step 1 Delete the exported tables.

```
[omm@opengauss backup]$ gsql -p 26000 postgres -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=# drop table joe.mytable;
DROP TABLE
postgres=# drop table public.customer_t1 cascade;
NOTICE:  drop cascades to view vw_customer_t1
DROP TABLE
postgres=# \q
```

Step 2 Use **gsql** to restore the exported table. (Ignore "ERROR: schema "joe" already exists".)

```
[omm@opengauss backup]$ gsql -p 26000 postgres -r -f /gaussdb/backup/gsdump_post.sql
```

Step 3 Check the restored table.

```
[omm@opengauss backup]$ gsql -p 26000 postgres -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=# select * from joe.mytable;
 firstcol
----------
        100
(1 row)

postgres=# select * from public.customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3769 | hello         |              |
          6885 | maps          | Joes         |
          4321 | tpcds         | Lily         |
          9527 | world         | James        |
(4 rows)

postgres=#\q
```

2.2.6.7 Importing Data Using the copy Command

Data exported using **copy** can also be imported using **copy**.

Step 1 Delete the tables that have been exported by running the **copy** command.

```
[omm@opengauss backup]$ gsql -p 26000 postgres -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
```

```
postgres=# truncate table public.customer_t1;
TRUNCATE TABLE
```

Step 2 Run the **copy** command to import data.

```
postgres=# copy customer_t1 from '/gaussdb/backup/copy_cost.txt' delimiter '^';
COPY 4
```

Step 3 View the imported table data.

```
postgres=# select * from customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3769 | hello         |              |
          6885 | maps          | Joes         |
          4321 | tpcds         | Lily         |
          9527 | world         | James        |
(4 rows)

postgres=# \q
```

2.2.6.8 Importing Data Using the gs_restore Command

gs_restore, provided by openGauss, is used to import data or files that were exported using gs_dump. gs_restore is executed by OS user **omm**.

Step 1 Delete the tables and views exported using **gs_dump**.

```
[omm@ecs-opengauss backup]$ gsql -p 26000 postgres -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=# drop table joe.mytable;
DROP TABLE
postgres=# drop table public.customer_t1;
DROP TABLE
```

Step 2 Use **gs_restore** to import data from the **gsdump_post.tar** file to the **tpcc** database.

Create a **tpcc** database.

```
postgres=# create database tpcc;
CREATE DATABASE
postgres=# \q
```

Import data from the **gsdump_post.tar** file to the **tpcc** database.

```
[omm@opengauss backup]$ gs_restore /gaussdb/backup/gsdump_post.tar -p 26000 -d tpcc
start restore operation ...
table mytable complete data imported !
table customer_t1 complete data imported !
Finish reading 12 SQL statements!
```

```
end restore operation ...
restore operation successful
total time: 22  ms
```

Step 3 Log in to the **tpcc** database and view the restored data.

```
[omm@opengauss backup]$ gsql -p 26000 tpcc -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

tpcc=# select * from public.customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3769 | hello         |              |
          6885 | maps          | Joes         |
          4321 | tpcds         | Lily         |
          9527 | world         | James        |
(4 rows)

tpcc=# select * from joe.mytable;
 firstcol
----------
       100
(1 row)

tpcc=#\q
```

2.3 Clearing the Lab Environment

Delete the created database user.

```
[omm@opengauss backup]$ gsql -p 26000 postgres -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
postgres=# \l
                                  List of databases
   Name    | Owner | Encoding  | Collate | Ctype | Access privileges
-----------+-------+-----------+---------+-------+-------------------
 postgres  | omm   | SQL_ASCII | C       | C     |
 template0 | omm   | SQL_ASCII | C       | C     | =c/omm               +
           |       |           |         |       | omm=CTc/omm
 template1 | omm   | SQL_ASCII | C       | C     | =c/omm               +
           |       |           |         |       | omm=CTc/omm
 tpcc      | omm   | SQL_ASCII | C       | C     |
(4 rows)
```

```
postgres=# drop database tpcc;
DROP DATABASE
postgres=# \du
                                                                    List of roles
 Role name |                                                         Attributes
| Member
of
-----------+-----------------------------------------------------------------------------------------------------------------+--------
---
 joe            |
| {}
 omm            | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Operatoradmin,
Policyadmin, UseFT | {}

postgres=# drop user joe;
DROP ROLE
postgres=#\q
```

2.4 Summary

This chapter describes how to log in to the openGauss database, maintain the tablespaces, databases, tables, and user roles of the openGauss database, and use the import and export tool to import data to the openGauss database, be familiar with the management of the openGauss database and objects.

2.5 Quiz

1. Use **COPY** to export the query result dataset (that is, the **where** condition statement).
2. Assume that the following operations are performed:
   a. Create a table **customer_t1** and import four data records to the table.
   b. Create the **vw_customer_t1** view that depends on the **customer_t1** table.

```
create view vw_customer_t1 as select * from customer_t1 limit 2;
```

Two data records are displayed in the **vw_customer_t1** view.

   c. Export the view **vw_customer_t1** of the **customer_t1** table to the **vw_customer.sql** file.
   d. Rename the **customer_t1** table to **customer_t1_bak**.
   e. Create the **customer_t1** table and do not insert data into the table.
   f. Import the **vw_customer.sql** file.

How many data records are displayed in the **vw_customer** view?

Reference answer: No data exists in the **vw_customer** view.

3 Basic SQL Syntax Experiment

3.1 Overview

3.1.1 About This Experiment

Starting from data preparation, this experiment describes the SQL syntax in the openGauss database, including data query, data update, data definition, and common functions and operators.

3.1.2 Objectives

Learn SQL-related operations, such as common DQL, DML, DDL, operators, and functions, and master the basic syntax and usage of these operations.

3.1.3 Description

- Sub-experiment 1: Prepare experiment data for the preset SQL script.
- Sub-experiment 2: Query data. The basic DQL is introduced to help you understand how to retrieve data from one or more tables or views.
- Sub-experiment 3: Update data The basic DML syntax and usage are introduced to help you understand how to update data in database tables, including data insertion, data modification, and data deletion.
- Sub-experiment 4: Define data. The DDL types, syntax formats, and application scenarios are introduced to help you master how to use DDL to define or modify objects in databases.
- Sub-experiment 5: Common operators and functions and their application scenarios are introduced to help you understand how to use the built-in operators and functions of the database.

3.2 Preparing Data

This section describes how to create a financial data model instance using preset SQL scripts to prepare data for SQL syntax learning in subsequent sections.

Insert data into related tables in the financial database to provide data support for subsequent experiments.

3.2.1 Tasks

openGauss provides a financial sample library for users to learn and verify databases. The database contains six tables. This section uses the following three tables:

- Customer (customer ID, name, email address, ID card number, mobile number, and login password)
- Bank card (bank card No., bank card type, and customer ID)

- Insurance (insurance name, insurance No., insurance amount, target people, insurance period, and insurance item)

The codes of the preceding attributes are as follows:

- client (c_id, c_name, c_mail, c_id_card, c_phone, and c_password)
- bank_card (b_number, b_type, and b_c_id)
- insurance (i_name, i_id, i_amount, i_person, i_year, and i_project)

Relationships:

- One customer can apply for multiple bank cards.
- One customer can purchase multiple insurances. One insurance can be purchased by multiple customers.

Step 1 Create a table.

Log in to openGauss as user **omm**.

```
[gaussdba@localhost ~]$ su - omm
```

Start the database server. If the database has been started, skip this step.

```
[omm@kwephicprd12118 bin]$ gs_ctl start –D /opt/opengauss/data/ -Z single_node –l logfile
```

Use the **postgres** account to connect to the database. **5432** is the connection port, which may vary depending on the environment. Set it as required.

```
[omm@kwephicprd12118 bin]$ gsql –d postgres –p 5432
```

Create the **client** table.

```
--Delete the client table.
POSTGRES=# DROP TABLE IF EXISTS client;

Succeed.

--Create the client table.
POSTGRES=# CREATE TABLE client
(
        c_id INT PRIMARY KEY,
        c_name NVARCHAR2(100) NOT NULL,
        c_mail NCHAR(30) UNIQUE,
        c_id_card NCHAR(20) UNIQUE NOT NULL,
        c_phone NCHAR(20) UNIQUE NOT NULL,
        c_password NCHAR(20) NOT NULL
);

Succeed.
```

Create the **bank_card** table.

```
--Delete the bank_card table.
POSTGRES=# DROP TABLE IF EXISTS bank_card;

Succeed.

-- Create the bank_card table.
```

```
POSTGRES=# CREATE TABLE bank_card
(
        b_number NCHAR(30) PRIMARY KEY,
        b_type NCHAR(20),
        b_c_id INT NOT NULL
);

Succeed.
--Add a foreign key constraint to the bank_card table.
POSTGRES=# ALTER TABLE bank_card ADD CONSTRAINT fk_c_id FOREIGN KEY (b_c_id) REFERENCES client(c_id) ON DELETE CASCADE;

Succeed.
```

Create the **insurance** table.

```
--Delete the insurance table.
POSTGRES=# DROP TABLE IF EXISTS insurance;

Succeed.

--Create the insurance table.
POSTGRES=# CREATE TABLE insurance
(
        i_name NVARCHAR2(100) NOT NULL,
        i_id INT PRIMARY KEY,
        i_amount INT,
        i_person NCHAR(20),
        i_year INT,
        i_project NVARCHAR2(200)
);

Succeed.
```

Step 2 Edit the **client.sql** file.

```
[gaussdba@kwephicprd12118 bin]$ vi client.sql
```

The **client.sql** script is as follows:

```
INSERT INTO client(c_id, c_name, c_mail, c_id_card, c_phone, c_password) VALUES(1,'Zhang Yi','zhangyi@huawei.com', '340211199301010001', '18815650001','gaussdb_001');
INSERT INTO client(c_id, c_name, c_mail, c_id_card, c_phone, c_password) VALUES(2,'Zhang Er','zhanger@huawei.com', '340211199301010002', '18815650002','gaussdb_002');
INSERT INTO client(c_id, c_name, c_mail, c_id_card, c_phone, c_password) VALUES (3,'Zhang San', 'zhangsan@huawei.com', '340211199301010003', '18815650003', 'gaussdb_003');
INSERT INTO client(c_id, c_name, c_mail, c_id_card, c_phone, c_password) VALUES (4,'Zhang Si', 'zhangsi@huawei.com', '340211199301010004', '18815650004', 'gaussdb_004');
```

Step 3 Edit the **bank_card.sql** file.

```
[gaussdba@kwephicprd12118 bin]$ vi bank_card.sql
```

The **bank_card.sql** script is as follows:

```
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000001','Credit Card', 1);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62222022302020000002','Credit Card', 3);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000003','Credit Card',5);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000004','Credit Card', 7);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000005','Credit Card', 9);
```

Step 4 Edit the **insurance.sql** script.

```
[gaussdba@kwephicprd12118 bin]$ vi insurance.sql
```

The **insurance.sql** script is as follows:

```
INSERT INTO insurance(i_name, i_id, i_amount, i_person, i_year, i_project) VALUES ('Health Insurance', 1, 2000,'Old People', 30,'Ping An Insurance');
INSERT INTO insurance(i_name, i_id, i_amount, i_person, i_year, i_project) VALUES ('Life Insurance', 2,3000, 'Seniors', 30,'Ping An Insurance');
INSERT INTO insurance(i_name, i_id, i_amount, i_person, i_year, i_project) VALUES ('Accident Insurance', 3,5000, 'All', 30,'Ping An Insurance');
INSERT INTO insurance(i_name, i_id, i_amount, i_person, i_year, i_project) VALUES ('Medical Insurance', 4, 2000,'All', 30,'Ping An Insurance');
INSERT INTO insurance(i_name, i_id, i_amount, i_person, i_year, i_project) VALUES ('Loss of Property Insurance', 5, 1500, 'Middle-aged', 30,'Ping An Insurance');
```

Step 5 Run the preceding scrips on the SQL UI.

```
--Run the scripts after starting the database:
[gaussdba@kwephicprd12118 bin]$ gsql –d postgres –p 5432

connected.

POSTGRES=# \i client.sql
POSTGRES=# \i bank_card.sql
```

3.3 Querying Data

DQL is used to demonstrate how to query data from tables, including simple query, conditional query, join query, sorting, and restriction.

- Understand the DQL syntax structure of data query.
- Master how to use the DQL language in different scenarios.

3.3.1 Tasks

3.3.1.1 Simple Query

Generally, queries are implemented through the **FROM** clause.

Use an asterisk (*) after **SELECT** to query all columns in the **bank_card** table.

```
SELECT * FROM bank_card;
b_number                    |   b_type              |    b_c_id
----------------------------+-----------------------+-------------------
```

```
6222022302020000001   | Credit Card        |    1
6222022302020000002   | Credit Card        |    3
6222022302020000003   | Credit Card        |    5
```

Check the bank card number and card type in the **bank_card** table.

```
SELECT b_number,b_type FROM bank_card;
b_number                    |      b_type
----------------------------+--------------------
6222022302020000001 | Credit Card
6222022302020000002 | Credit Card
6222022302020000003 | Credit Card
```

View the ID, name, email address, and bank card number of customer 1.

```
SELECT a.c_id,a.c_name, a.c_mail, b.b_number FROM client a, bank_card b where a.c_id= 1 and b.b_c_id = 1;
c_id          |   c_name       |         c_mail                    |          b_number
------------+-----------------+-------------------------------+--------------------
1               |    Zhang Yi   |    zhangyi@Huawei.com     | 6222022302020000001
```

Use an alias.

```
SELECT b_c_id AS CardID, b_type CardType FROM bank_card;
Cardid      |             cardtype
------------+--------------------------------
1             |      Credit Card
3             |      Credit Card
5             |      Savings Card
7             |      Savings Card
9             |      Savings Card

SELECT a.c_id CID ,a.c_name Name, a.c_mail Email, b.b_number "Card Number" FROM client a, bank_card b where
a.c_id= 1 and b.b_c_id = 1;
cid        | name          |    email                        |    Card Number
---------+--------------+-------------------------------+-----------------
1           | Zhang Yi     |   zhangyi@Huawei.com     | 6222022302020000001
```

3.3.1.2 Conditional Query

In a SELECT statement, you can specify conditions for a more accurate query.

Use the comparison operators >, <, >=, <=, !=, <>, and = to specify query conditions.

Query information about a credit card in the **bank_card** table.

```
SELECT * FROM bank_card WHERE b_type= 'Credit Card';
b_number                       |      b_type             |    b_c_id
-------------------------------+----------------------+----------------
6222022302020000001    |   Credit Card          |    1
6222022302020000002    |   Credit Card          |    3
6222022302020000003    |   Credit Card          |    5
6222022302020000004    |   Credit Card          |    7
6222022302020000005    |   Credit Card          |    9
```

Query information about a credit card whose customer ID is 1 from the **bank_card** table.

```
SELECT * FROM bank_card where b_c_id= 1and b_type='Credit Card';
```

```
b_number                    |      b_type             |    b_c_id
----------------------------+---------------------+----------
6222022302020000001    |   Credit Card        |    1
```

3.3.1.3 Join Query

Generally, data to be queried is distributed in multiple tables. A query across tables or views is called a join query. In most cases, a join query is executed between a table and its child tables.

- Inner Join

The keyword is **INNER JOIN**, where **INNER** can be omitted. If an inner join is used, the join execution sequence will follow the sequence of tables in the statement.

Use an inner join to query tables. Query the customer ID, bank card number, and bank card type. Use the **(c_id)** column of the **client** and **bank_card** tables to perform the query operation.

```
SELECT c.c_id, b.b_number, b.b_type FROM client c JOIN bank_card b ON (b.b_c_id = c.c_id);
c_id    |              b_numer            |       b_type
-------+---------------------------------+-------------------
1      | 6222022302020000001      |   Credit Card
3       | 6222022302020000002      |   Credit Card
```

- Outer Join

Use the left outer join to query the customer ID, user name, bank card number, and bank card type.

```
SELECT c.c_id,c.c_name, b.b_number,b.b_type FROM client c left join bank_card b on c.c_id=b.b_c_id;
 c_id    |      c_name      |   b_numer                      |       b_type
---------+------------------+--------------------------------+-------------------
1         | Zhang Yi         |6222022302020000001      |   Credit Card
3         | Zhang San        |62220223020200000002    |   Credit Card
2         | Zhang Er         |6222022302020000003      |
4         | Zhang Si         |6222022302020000004      |
```

Use the right outer join to query data in the **client** and **bank_card** tables.

```
SELECT c.c_id,c.c_name, b.b_number,b.b_type FROM client c right join bank_card b on c.c_id =b.b_c_id;
c_id    |      c_name      |   b_numer                      |       b_type
---------+------------------+--------------------------------+-------------------
1         | Zhang Yi         |6222022302020000001      |   Credit Card
3         | Zhang San        |62220223020200000002    |   Credit Card
          |                  |6222022302020000003      |   Credit Card
          |                  |6222022302020000004      |   Credit Card
          |                  |6222022302020000005      |   Credit Card
```

Obtain the full join data of the **client** and **bank_card** tables through a full join.

```
SELECT c.c_id,c.c_name, b.b_number,b.b_type FROM client c FULL JOIN bank_card b ON (b.b_c_id = c.c_id);
c_id    |      c_name      |   b_numer                      |       b_type
---------+------------------+--------------------------------+-------------------
1         | Zhang Yi         |6222022302020000001      |   Credit Card
3         | Zhang San        |62220223020200000002    |   Credit Card
          |                  |6222022302020000003      |   Credit Card
          |                  |6222022302020000004      |   Credit Card
          |                  |6222022302020000005      |   Credit Card
```

```
2          | Zhang Er       |                              |
4          | Zhang Si       |                              |
```

3.3.1.4 Data Sorting

The **ORDER BY** clause is used to sort rows returned by a query by a specified column.

Query the names, amounts, and target people of insurances with insurance numbers higher than 2 in descending order.

```
SELECT i_name,i_amount,i_person FROM insurance WHERE i_id>2 ORDER BY i_amount DESC;
i_name                          |       i_amount    |       i_person
--------------------------------+-------------------+-----------------
Accident Insurance              |          5000     |     all
Medical Insurance               |          2000     |     all
Loss of Property Insurance |          1500     |     Middle-aged
```

3.3.1.5 Data Limit

The **LIMIT** clause allows users to limit the rows returned by a query. You can specify an offset and the number or percentage of rows to be returned. This clause can be used to generate top-N reports. To obtain consistent results, specify the **ORDER BY** clause to ensure a deterministic sort order.

Make two queries with the first excluding **LIMIT** and the second including **LIMIT**, and then compare the query results.

Query the insurance information in the following table. Add **LIMIT 2 OFFSET 1** to skip the first row and query a total of two rows.

```
SELECT i_name, i_id, i_amount, i_person FROM insurance LIMIT 2 OFFSET 1;
i_name                  |       i_id          | i_amount    |    i_person
------------------------+-----------------+-------------+----------------
Life Insurance          |          2          |    3000     | Seniors
Accident Insurance      |          3          |    5000     |    All
```

3.4 Updating Data

This section describes the data manipulation language (DML) and demonstrates how to use DML to update data in database tables, including data insertion, modification, and deletion.

- Understand the DML syntax structure.
- Master how to use DML in different scenarios.

3.4.1 Tasks

3.4.1.1 Inserting Data

INSERT inserts data into a table.

You can use **INSERT** to insert data in following ways:

Insert values. Create a row of records, and insert them into a table.

Insert a query. Create one or more rows of records based on the result set returned by **SELECT** and insert the records into a table.

Insert records into a table. If primary key conflicts are reported, values in the specified column are updated.

Insert data into the **bank_card** table.

```
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000030','Savings Card', 30);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000031','Savings Card', 31);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222022302020000032','Savings Card', 32);
```

Insert all data from the **bank_card** table into the **bank_card1** table using a subquery.

```
CREATE TABLE bank_card1(b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20), b_c_id INT NOT NULL);
INSERT INTO bank_card1 SELECT * FROM bank_card;
```

Insert a record and update a value for columns in which this primary key is a duplicate.

```
INSERT INTO bank_card VALUES ('6222022302020000001', 'Credit Card', 1) ON DUPLICATE KEY UPDATE b_type =
'Savings Card';
```

Data Modification

Query records in the **bank_card** table.

```
SELECT * FROM bank_card;
b_number                    |    b_type          |     b_c_id
--------------------------- +-----------------+--------------------
6222022302020000001 | Savings Card     |     1
6222022302020000002| Credit Card     |     3
6222022302020000003 | Credit Card     |     5
6222022302020000004| Credit Card     |     7
6222022302020000005 | Credit Card     |     9
6222022302020000030| Savings Card     |     30
6222022302020000031 | Savings Card     |     31
6222022302020000032 | Savings Card     |     32
```

Update the record whose customer ID is 1 in the **bank_card** table and change the value of **b_type** to **Credit Card**.

```
UPDATE bank_card SET bank_card.b_type = 'Credit Card' where b_c_id=1;
```

3.4.1.2 Deleting Data

DELETE deletes records from a table.

Query records in the **bank_card** table.

```
SELECT * FROM bank_card;
```

Delete the records, whose **b_type** is **Credit Card** and **b_c_id** is **1**, from the **bank_card** table.

```
DELETE FROM bank_card WHERE b_type='Credit Card' AND b_c_id=1;
```

Query records in the **bank_card** table.

```
SELECT * FROM bank_card;
```

Delete all data from the **bank_card** table.

```
DELETE FROM bank_card;
```

Query records in the **bank_card** table.

```
SELECT * FROM bank_card;
```

3.5 Defining Data

This section describes the Data Definition Language (DDL) and demonstrates how to use DDL to define or modify objects in a database, such as databases, schemas, tables, indexes, views, databases, sequences, and users.

- Understand the DDL syntax structure.
- Master how to use DDL in different scenarios.

3.5.1 Tasks

3.5.1.1 Defining a Database

A database is the warehouse for organizing, storing, and managing data. Defining a database includes creating a database, modifying database attributes, and deleting a database. Database-related DDL operations include creating a database, modifying database attributes, and deleting a database.

3.5.1.1.1 Creating a Database

Create a database user named **jim**.

```
CREATE USER jim PASSWORD 'Bigdata@123';
CREATE USER jim01 PASSWORD 'Bigdata@123';
```

Create database **music** using **template0** and specify user **jim** as its owner.

```
CREATE DATABASE music OWNER jim TEMPLATE template0;
```

3.5.1.1.2 Modifying Database Attributes

After a database is created, you can run the **ALTER Database** command to change the database definition if the service scenario changes.

Change the maximum number of connections to database **music**.

```
ALTER DATABASE music WITH CONNECTION LIMIT 30;
```

Rename the database.

```
ALTER DATABASE music RENAME TO music01;
```

Change the default tablespace of the database.

```
CREATE TABLESPACE tablespace01 RELATIVE LOCATION 'tablespace01/tablespace_01';
ALTER DATABASE music01 SET TABLESPACE tablespace01;
```

Change the database owner.

```
ALTER DATABASE music01 OWNER TO jim01;
```

3.5.1.1.3 Deleting a Database

If a database is no longer needed, you can run the **DROP DATABASE** command to delete it.

Delete a database.

```
DROP DATABASE music01;
```

Delete users.

```
DROP USER jim;
DROP USER jim01;
```

3.5.1.2 Defining a Schema

A schema is a set of database objects and is used to control the access to the database objects.

3.5.1.2.1 Creating a Schema

Create a role named **role1**.

```
CREATE ROLE role1 IDENTIFIED BY 'Bigdata@123';
```

Create a schema named **role1** for the **role1** role. The owner of the **films** and **winners** tables is **role1**, as they are created by using sub-commands under the role1 schema.

```
CREATE SCHEMA AUTHORIZATION role1
CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;
```

3.5.1.2.2 Modifying Schema Attributes

After a schema is created, you can run the **ALTER SCHEMA** command to change the schema definition if the service scenario changes.

Rename the schema.

```
ALTER SCHEMA role1 RENAME TO role1_new;
```

Change the owner of the schema.

```
CREATE USER jack PASSWORD 'Bigdata@123';
ALTER SCHEMA role1_new OWNER TO jack;
```

3.5.1.2.3 Deleting a Schema

When a schema is no longer needed, you can run the **DROP SCHEMA** command to delete it.

Delete a schema.

```
DROP SCHEMA role1_new;
```

Delete a user.

```
DROP USER jack;
```

3.5.1.3 Defining a Table

A table is a special data structure in a database and is used to store data objects and relationships between data objects. Table-related DDL operations include creating a table, modifying table attributes, deleting a table, and deleting all data from a table.

3.5.1.3.1 Creating a Table

Common table

Create a table named **bank_card**.

```
CREATE TABLE bank_card( b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);
```

Temporary table

```
CREATE TEMPORARY TABLE bank_card2(b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);
```

3.5.1.3.2 Modifying Table Attributes

After a table is created, you can run the **ALTER TABLE** statement to change the table definition if the service scenario changes.

Rename the **bank_card** table to **bank_card1**.

```
ALTER TABLE bank_card RENAME TO bank_card1;
```

Add a column to the **bank_card1** table. The data type is Integer.

```
ALTER TABLE bank_card1 ADD full_masks INTEGER;
```

Change the data type of a column.

```
ALTER TABLE bank_card1 MODIFY full_masks NVARCHAR2(20);
```

Add a constraint.

```
ALTER TABLE bank_card1 ADD CONSTRAINT ck_bank_card CHECK(b_c_id>0);
ALTER TABLE bank_card1 ADD CONSTRAINT uk_bank_card UNIQUE(full_masks);
```

Delete a column.

```
ALTER TABLE bank_card1 DROP full_masks;
```

Insert data into a table. If the constraint is not met, an error is reported.

```
INSERT INTO bank_card1(b_number, b_type, b_c_id) VALUES ('6222022302020000001','Credit Card', 0);
GS-01222, Check constraint violated
```

3.5.1.3.3 Deleting Data from a Table

When table data is no longer used, delete all rows of the table, that is, clear the table.

Delete all rows from the **bank_card1** table.

```
DELETE FROM bank_card1;
```

3.5.1.3.4 Deleting a Table

When the data or definition of a table is no longer needed, you can run the **DROP TABLE** statement to delete the table.

```
DROP TABLE IF EXISTS bank_card1;
```

3.5.1.4 Defining a Partitioned Table

openGauss supports range partitioning, list partitioning, and hash partitioning.

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

High query performance: Users can specify partitions when querying a partitioned table, improving query efficiency.

High availability: If a partition in a partitioned table is faulty, data in the other partitions is still available.

Easy maintenance: If a partition in a partitioned table is faulty, only the faulty partition needs to be repaired.

Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

Partitioned table-related DDL operations include creating a table, modifying table attributes, deleting a table, and deleting all data from a table.

3.5.1.4.1 Creating a Partitioned Table

Create a tablespace and a partitioned table.

```
CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

```
CREATE TABLE bank_card(b_number NCHAR(30),b_type NCHAR(20),b_c_id INT NOT NULL)
TABLESPACE example1
PARTITION BY RANGE(b_c_id)
(
    PARTITION bank_card1 VALUES LESS THAN(100),
    PARTITION bank_card2 VALUES LESS THAN(200),
    PARTITION bank_card3 VALUES LESS THAN(300),
    PARTITION bank_card4 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

3.5.1.4.2 Modifying a Partitioned Table

After a partitioned table is created, you can run the **ALTER TABLE** statement to change the table definition if the service scenario changes.

Modify a tablespace of a partitioned table.

```
ALTER TABLE bank_card MOVE PARTITION bank_card1 TABLESPACE example3;
```

Rename the partitioned table.

```
ALTER TABLE bank_card RENAME PARTITION bank_card1 TO bank_card5;
```

Combine partitions.

```
ALTER TABLE bank_card MERGE PARTITION bank_card1, bank_card2 INTO PARTITION bank_card3;
```

Delete a partition.

```
ALTER TABLE bank_card DROP PARTITION bank_card4;
```

3.5.1.4.3 Deleting a Partitioned Table

If a partition table is no longer needed, you can run the **DROP TABLE** command to delete it.

```
DROP TABLE bank_card;
```

Delete a tablespace.

```
DROP TABLESPACE example1;
DROP TABLESPACE example2;
DROP TABLESPACE example3;
DROP TABLESPACE example4;
```

3.5.1.5 Defining an Index

Indexes accelerate data access but increase the processing time of table insertion, update, and deletion operations. Therefore, before creating an index, you need to consider whether to add an index for a table and determine the columns for which indexes are to be created. You can determine whether to add an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be sorted. Index-related DDL operations include creating indexes, deleting index attributes, and deleting indexes.

3.5.1.5.1 Principles for Creating Indexes

After creation, using indexes properly during queries can improve database performance. However, it takes time to create indexes and maintain indexes. Index files occupy physical space. In addition, indexes need to be maintained when **INSERT**, **UPDATE**, and **DELETE** statements are executed on tables, which reduces data maintenance efficiency. Therefore, you are advised to create indexes on the following columns and use the indexes accordingly:

- Columns that are often queried

- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**

3.5.1.5.2 Creating an Index

Run the **CREATE INDEX** statement to create an index.

Create an index on **b_number, b_type** in the **bank_card** table.

```
CREATE INDEX idx_bank_card ON bank_card(b_number ASC,b_type);
```

Create a partitioned index in the partitioned table **education**.

```
-- Create the education partitioned table.
postgres=# CREATE TABLE education(staff_id INTEGER NOT NULL, higest_degree CHAR(8), graduate_school VARCHAR(64),graduate_date DATETIME, education_note VARCHAR(70))
PARTITION BY LIST(higest_degree)
(
PARTITION doctor VALUES ('Doctor'),
PARTITION master VALUES ('Master'),
PARTITION undergraduate VALUES ('Bachelor')
);
-- Create a partitioned index with the name of the index partition specified.
postgres=# CREATE INDEX idx_education ON education(staff_id ASC, higest_degree) LOCAL (PARTITION doctor, PARTITION master, PARTITION undergraduate);
```

3.5.1.5.3 Modifying an Index

- Rebuilding an index

After a large number of tables are added, deleted, or modified, the table data may be fragmented on the physical file of the disk, affecting the access speed. Using the **ALTER INDEX** statement to rebuild an index can reassemble index data and release unused space, improving data access efficiency and space usage.

Indexes cannot be rebuilt during database restart or rollback.

Rebuild the index.

```
ALTER INDEX idx_bank_card REBUILD;
```

- Renaming an index

If you need to generalize about the naming style of indexes, use the **RENAME** syntax to rename indexes, without changing other index attributes.

Indexes cannot be renamed during database restart or rollback.

Rename an index.

```
ALTER INDEX idx_bank_card RENAME TO idx_bank_card_temp;
```

Set an index as unusable.

```
ALTER INDEX idx_bank_card UNUSABLE;
```

3.5.1.5.4 Deleting an Index

Run the **DROP INDEX** statement to delete an index. Restrictions on deleting an index are as follows:

- Indexes cannot be deleted during database restart or rollback.

- After **ENABLE_IDX_CONFS_NAME_DUPL** is enabled, duplicate index names are allowed between different tables. Therefore, you must specify the table name when deleting an index.
- Indexes related to existing **UNIQUE KEY** or **PRIMARY KEY** constraints cannot be deleted.
- When a table is deleted, its indexes are also deleted.

Delete the **idx_bank_card** index from the **bank_card** table.

```
DROP INDEX idx_bank_card;
```

3.5.1.6 Defining a View

- If certain columns in one or more tables in a database are frequently searched for, an administrator can define a view for these columns, and then users can directly access these columns in the view without entering search criteria. View-related DDL operations include creating and deleting views.

3.5.1.6.1 Concepts

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

3.5.1.6.2 Creating a View

Run the **CREATE VIEW** command to create a view.

```
CREATE OR REPLACE VIEW privilege_view AS SELECT b_number, b_type FROM bank_card;
```

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

Query the data in the view. The syntax is the same as that for querying a table.

```
SELECT * FROM privilege_view;
```

Query the view structure.

```
\d privilege_view;
```

3.5.1.6.3 Deleting a View

Run the **DROP VIEW** command to delete a view.

Delete the view.

```
DROP VIEW privilige_view;
```

3.5.1.7 Defining a Sequence

A sequence can generate numbers with the same interval, and these are used as primary key values. When a sequence value is generated, the sequence is incremented. Sequence-related DDL operations include creating, modifying, and deleting sequences.

3.5.1.7.1 Creating a Sequence

Add a sequencer to the current database. The current user is the owner of the sequencer.

Create the **seq_auto_extend** sequence starting with **10**, and with **increment** set to **2**, **MAXVALUE** set to **200**, and **CYCLE** specified.

```
CREATE SEQUENCE seq_auto_extend START WITH 10 MAXVALUE 200 INCREMENT BY 2 CYCLE;
```

#Query the next value of **seq_auto_extend**.

```
SELECT seq_auto_extend.NEXTVAL FROM DUAL;
```

```
NEXTVAL
--------------------
10
```

3.5.1.7.2 Modifying a Sequence

Set **INCREMENT BY** to **4** and **MAXVALUE** to **400**.

```
ALTER SEQUENCE seq_auto_extend MAXVALUE 400 INCREMENT BY 4 CYCLE;
```

3.5.1.7.3 Deleting a Sequence

Delete the **seq_auto_extend** sequence.

```
DROP SEQUENCE IF EXISTS seq_auto_extend;
```

3.6 Common Functions and Operators

This section describes common database functions and operators, such as character processing functions and operators, arithmetic operation functions and operators, and time and date processing functions and operators.

3.6.1 Tasks

3.6.1.1 Character Processing Functions

String processing functions and operators provided by openGauss are for concatenating strings with each other, concatenating strings with non-strings, and matching the patterns of strings.

instr(string1,string2,int1,int2) returns the text from **int1** to **int2** in **string1**. The first **int** indicates the start position for matching, and the second **int** indicates the number of matching times.

```
SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
Instr
-------
6
```

overlay(string placing string FROM int [for int]) replaces a substring. **FROM int** indicates the start position of the replacement in the first string. **for int** indicates the number of characters replaced in the first string.

```
SELECT overlay('hello' placing 'world' FROM 2 for 3 );
overlay
---------
hworldo
```

position(substring in string) specifies the position of a substring. Parameters are case-sensitive.

```
SELECT position('ing' in 'string');
position
----------
4
```

substring_inner(string [FROM int] [for int]) extracts a substring. **FROM int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

```
SELECT substring_inner('adcde', 2,3);
```

```
 substring_inner
-----------------
 dcd
```

replace(string text, FROM text, to text) replaces all occurrences in **string** of substring **FROM** with substring **to**.

```
SELECT replace('abcdefabcdef', 'cd', 'XXX');
       replace
----------------
 abXXXefabXXXef
```

substring(string [FROM int] [for int]) extracts a substring. **FROM int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

```
SELECT substring('Thomas' FROM 2 for 3);
Substring
-----------
hom
(1 row)
```

3.6.1.2 Arithmetic Functions and Operators

Arithmetic Operators

Arithmetic Operator	Description	Arithmetic Operator	Description
+	Addition	!!	Factorial (prefix operator)
-	Subtraction	|	Binary OR
*	Multiplication	&	Binary AND
/	Division (The result is not rounded.)	#	Binary XOR
%	Modulo	~	Binary NOT
@	Absolute value	^	Power (exponent calculation)
|/	Square root	<<	Left shift
||/	Cubic root	>>	Right shift

```
SELECT 2+3,2*3, @ -5.0, 2.0^3.0, |/ 25.0, 91&15, 17#5,1<<4 AS RESULT;
2+3       |     2*3    | @ -5.0    | 2.0^3.0        | |/ 25.0|    91&15     | 17#5    | 1<<4
-------+-------+---------+-----------+--------+---------+-------+-------
5         |       6    |      5.0       |     8.00000    |      5       |      11      |      20    | 16
```

abs(exp), **cos(exp)**, **sin(exp)**, **acos(exp)**, and **asin(exp)** return the absolute value, cosine value, sine value, arc cosine value, and arc sine value of an expression, respectively.

```
SELECT abs(-10),cos(0),sin(0),acos(1),asin(0);
ABS            |          COS        |          SIN          |          ACOS        |       ASIN
---------+--------------+--------------+-------------+------------
      10      |            1          |            0          |            0          |         0
```

bitand(exp1,exp2) performs an AND (&) operation on two digits.

```
SELECT bitand(29,15);
BITAND(29,15)
--------------------
    13
```

round(number[, decimals]) truncates a number value to the left or right of a specified decimal point.

```
SELECT round(1234.5678,-2),round(1234.5678,2);
ROUND(1234.5678,-2)             ROUND(1234.5678,2)
------------------------ ---------------------------------
1200                                                  1234.57
```

3.6.1.3 Date and Time Processing Functions and Operators

Time and date operators (–)

```
SELECT date '2022-5-28' + integer '7' AS RESULT;
            result
---------------------------
2022-06-04 00:00:00

SELECT date '2022-05-28' + interval '1 hour' AS RESULT;
            result
---------------------
2022-05-28 01:00:00

SELECT date '2022-05-28' + time '03:00' AS RESULT;
            result
---------------------
2022-05-28 03:00:00

SELECT interval '1 day' + interval '1 hour' AS RESULT;
            result
----------------
1 day 01:00:00
```

Time and date operators (–)

```
SELECT date '2022-05-01' - date '2022-04-28' AS RESULT;
    result
------------
 3 days

SELECT date '2022-05-01' - integer '7' AS RESULT;
    result
```

```
---------------------
2022-04-24 00:00:00

SELECT date '2022-05-28' - interval '1 hour' AS RESULT;
     result
---------------------
2022-05-27 23:00:00

SELECT time '05:00' - interval '2 hours' AS RESULT;
     result
--------------
03:00:00
```

age(timestamp, timestamp) subtracts parameters, producing a result in YYYY-MM-DD format. If the result is negative, the returned result is also negative. The input parameters can also contain the timezone if required.

```
SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
                  age
-------------------------
 43 years 9 mons 27 days
```

current_time indicates the current time.

```
SELECT current_time;
           timetz
--------------------
 16:58:07.086215+08
```

3.6.1.4 Type Conversion Functions

to_char(int, text), **to_clob(str)**, **to_date(exp[, fmt])**, and **to_number(n[, fmt])** convert the specified input parameter to the CHAR, CLOB, DATE, and NUMBER types, respectively.

```
SELECT to_char(125, '999'),to_clob('hello111'::CHAR(15)),to_date('05 Dec 2000', 'DD Mon YYYY'),
to_number('12,454.8-', '99G999D9S');
TO_CHAR   |   TO_CLOB        |      TO_DATE                  |   TO_NUMBER
--------+-------------+---------------------+---------------------
125          |     hello111     | 2000-12-05 00:00:00 |     -12454.8
```

cast(x as y) converts **x** to the type specified by **y**.

```
SELECT cast('22-oct-1997' as timestamp);
         timestamp
---------------------
 1997-10-22 00:00:00
```

hextoraw(string) converts a hexadecimal string to RAW.

```
SELECT hextoraw('7D');
 hextoraw
----------
 7D
```

3.7 Summary

Starting from data preparation, this experiment describes the SQL syntax in the openGauss database, including data query, data update, data definition, data control, and common functions and operators.

3.8 Quiz

1. What are the SQL statements? Briefly describe the function of each SQL statement.

Reference answer:

Data definition language (DDL) is used to define or modify database objects, for example, tables, indexes, views, databases, sequences, users, roles, tablespaces, and stored procedures.

Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, and deleting data. No data exists in the **vw_customer** view.

Data control language (DCL) is used to set or change database transactions, perform authorization operations (such as granting permissions to users or roles, revoking permissions, creating roles, and deleting roles), lock tables (supporting shared and exclusive locks), and stop services.

Data query language (DQL) is used to query data in the database, for example, querying data and combining the result sets of multiple SELECT statements.

2. What is a function? What is the purpose of designing functions? List common functions.

Reference answer: As an object of a database, a function is an independent program unit designed based on SQL.

Based on the data types of objects processed by functions, functions are classified into character processing functions, binary string functions, bit string functions, arithmetic operation functions, time and date processing functions, geometric functions, and type conversion functions.

4 GaussDB(for openGauss)

4.1 Overview

4.1.1 About This Experiment

This experiment shows how to purchase GaussDB(for openGauss) and use the Data Admin Service (DAS) client tool to connect to the database. In addition, it describes how to create databases and tables, and add, delete, modify, and query data in GaussDB(for openGauss).

4.1.2 Objectives

- Master the process of purchasing GaussDB(for openGauss).

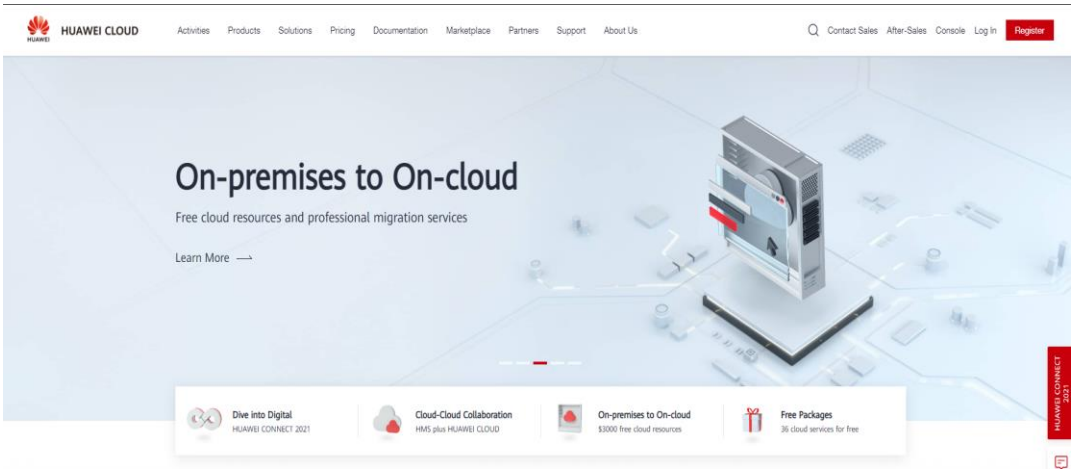
- Understand how to use the DAS client tool.
- Master the usage of basic SQL statements in GaussDB(for openGauss).

4.2 Purchasing a GaussDB(for openGauss) Instance

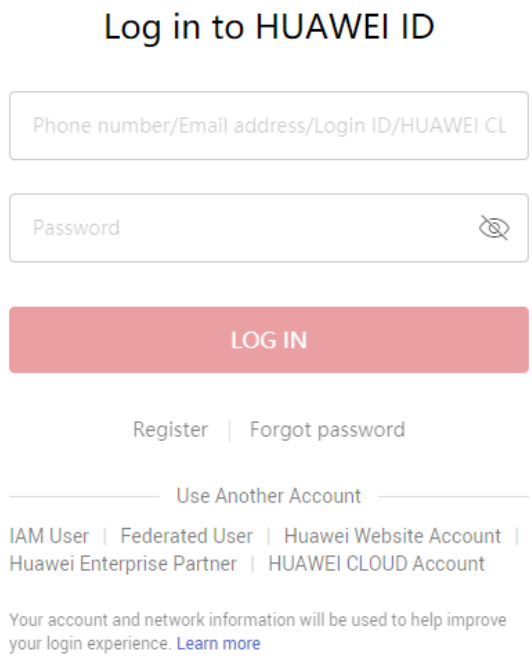
GaussDB(for openGauss) is an enterprise-level distributed relational database developed by Huawei and based on the openGauss ecosystem. It features Hybrid Transactional/Analytical Processing (HTAP) capabilities and supports intra-city deployment across AZs, scale-out of more than 1,000 nodes, and storage for petabytes of data to ensure zero data loss. It is highly available, reliable, secure, and scalable, and provides key capabilities including quick deployment, backup, restoration, monitoring, and alarm reporting for enterprises.

4.2.1 Logging In to the HUAWEI CLOUD Official Website

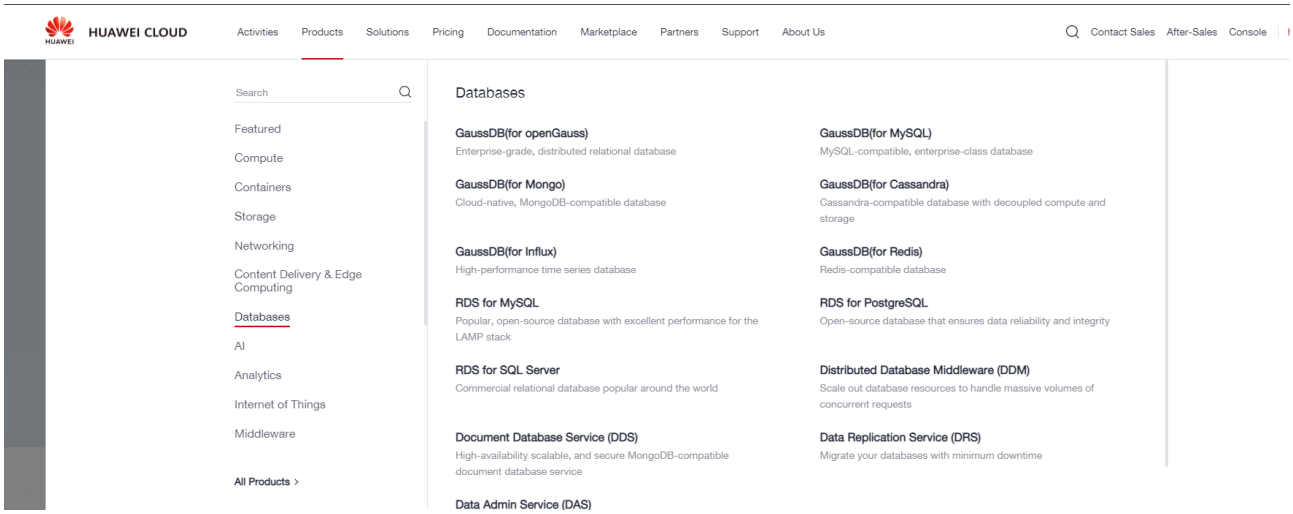
Step 1 Open the HUAWEI CLOUD official website at https://www.huaweicloud.com/intl/en-us/ and click **Log In** in the upper right corner of the page to access the HUAWEI CLOUD login page.



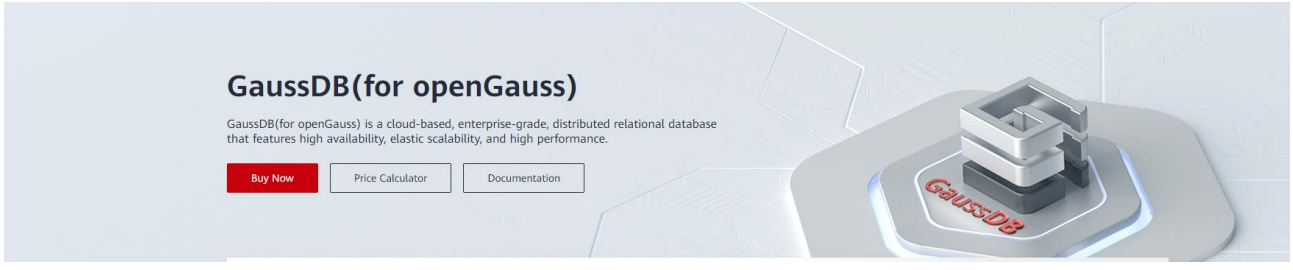
Step 2 Enter the username and password of the HUAWEI CLOUD account and click **LOG IN** to log in to the HUAWEI CLOUD official website.



Step 3 Choose Products > Databases > GaussDB(for openGauss).



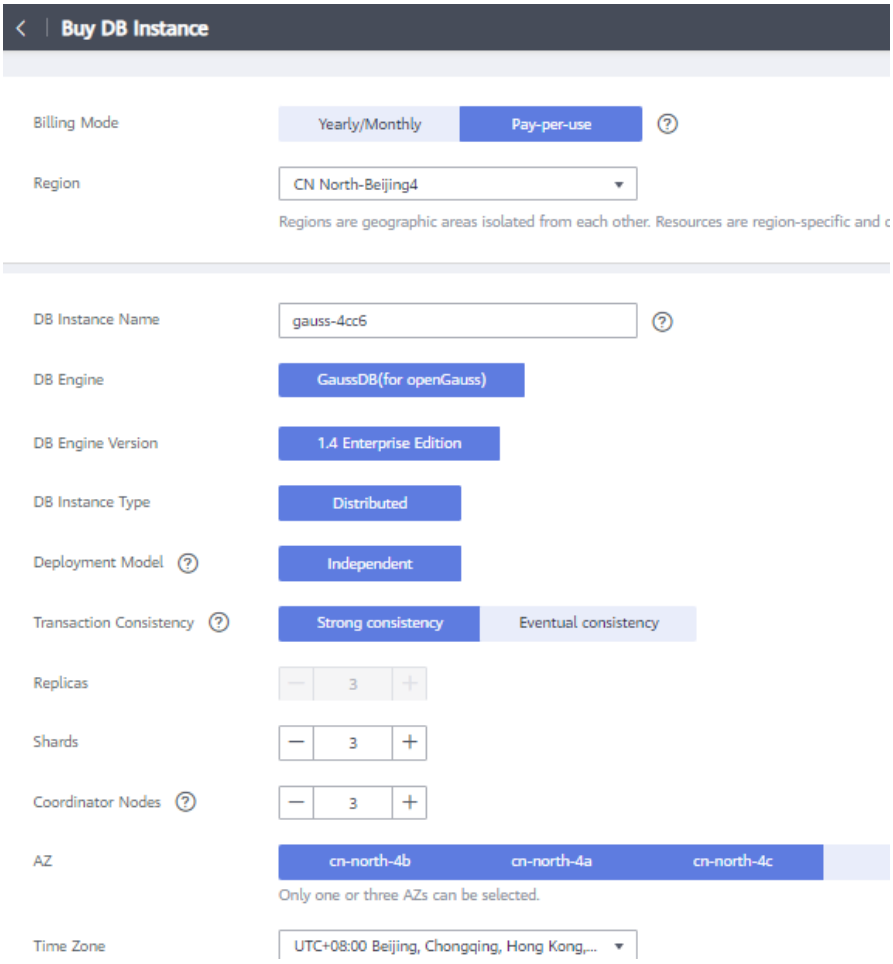
Step 4 Click **Buy Now**.



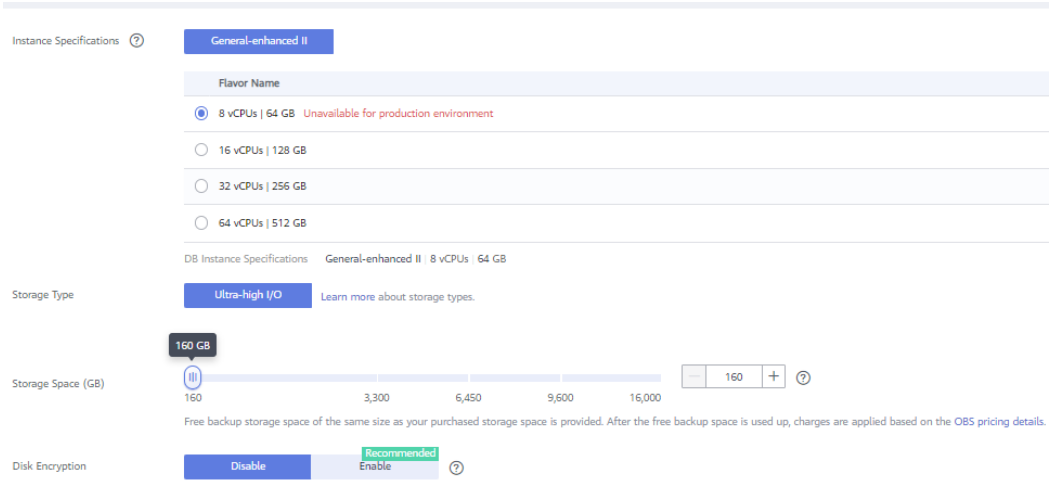
4.2.2 Purchasing a Database Instance

Step 1 Set **Billing Mode** to **Pay-per-use**, **DB Instance Name** to a custom name, and **Transaction Consistency** to **Strong consistency**. In this experiment, high availability is not considered to

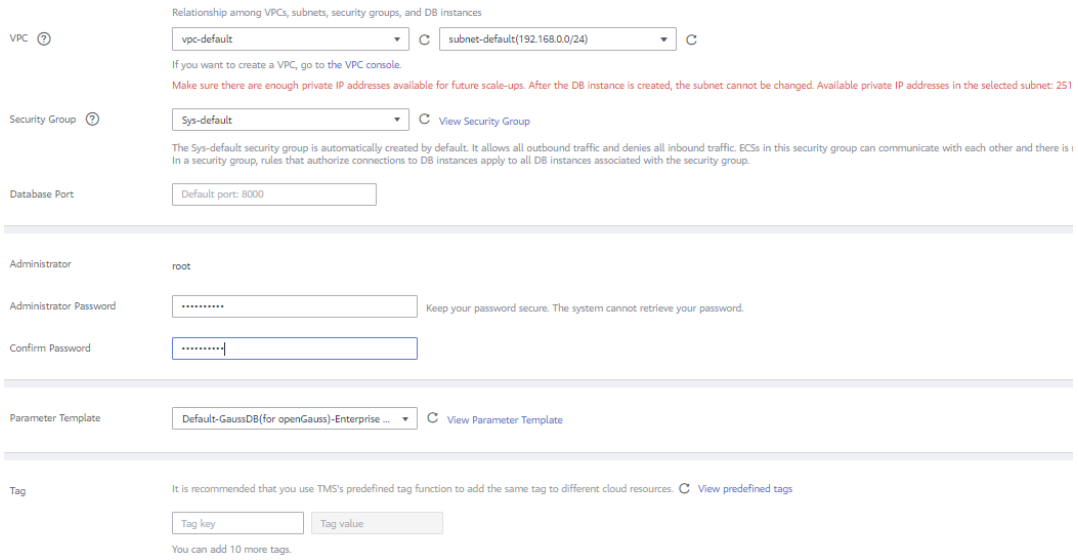
reduce the experiment cost. Therefore, set both **Shards** and **Coordinator Nodes** to **1**. For details, see the following figure.



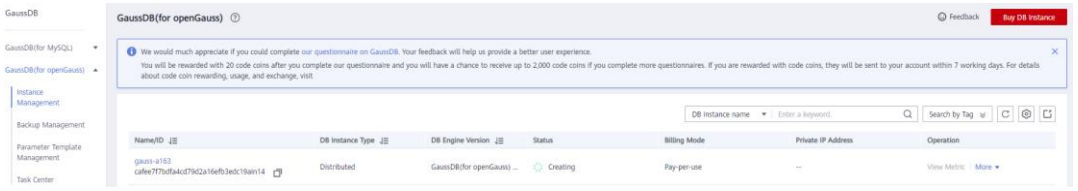
Step 2 Retain the default setting for **Instance Specifications**. For details, see the following figure.



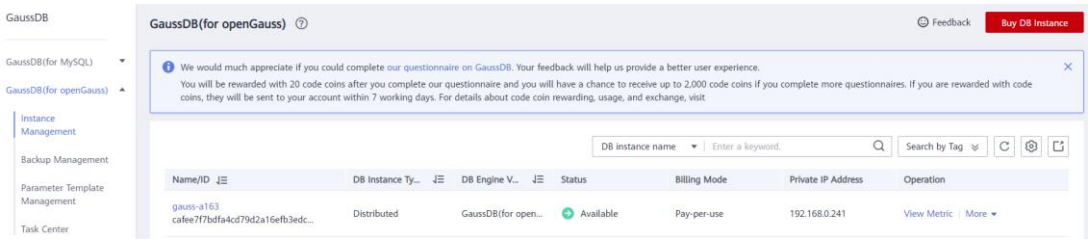
Step 3 Set **Administrator Password** and **Confirm Password**. Retain the default settings for the other items. Note that the configuration fee displayed in the lower left corner is about CNY84. For details, see the following figure.



Step 4 Click **Submit**. The configuration confirmation page is displayed. After confirmation, the GaussDB(for openGauss) instance is purchased. The console is displayed, where you can see that the database is being created.



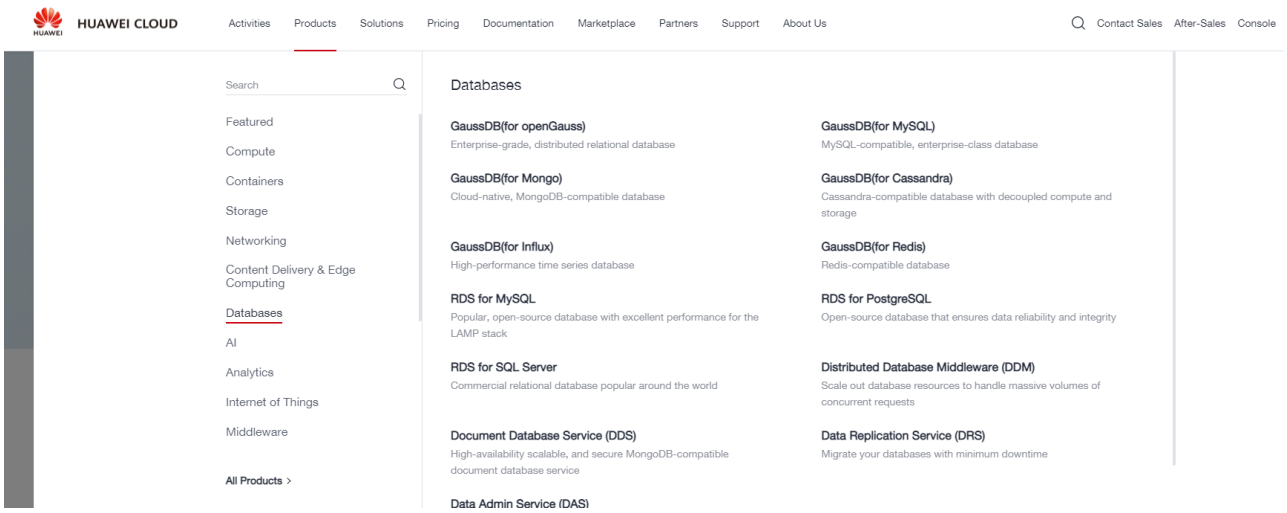
Step 5 Verify that the instance is successfully created and the instance state is **Available**, as shown in the following figure.



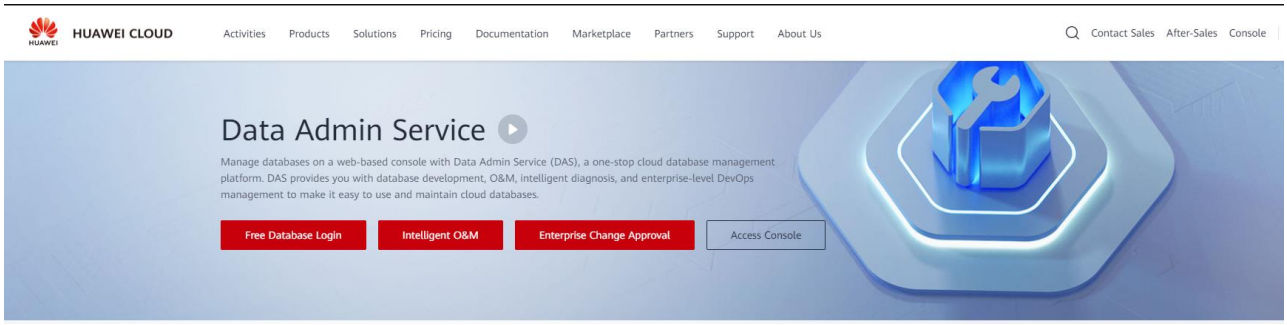
4.3 Using DAS

4.3.1 Connecting to the Database by Using DAS

Step 1 Log in to the HUAWEI CLOUD official website at https://www.huaweicloud.com/, and choose **Products** > **Databases** > **Data Admin Service (DAS)**.

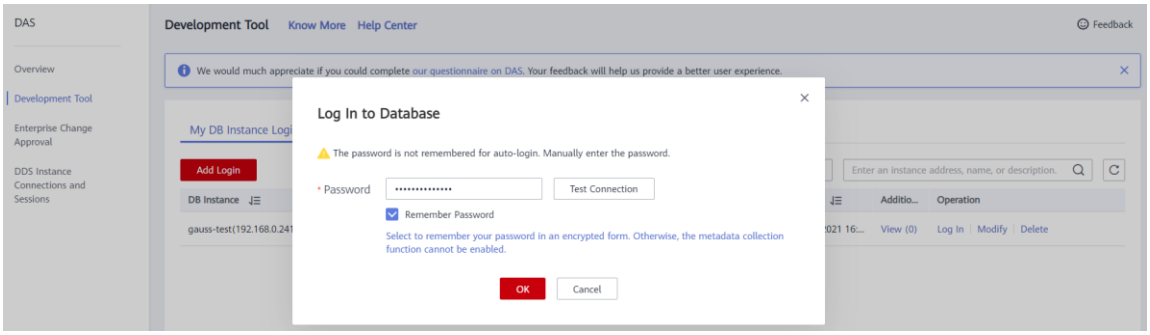


Step 2 Click Free Database Login.



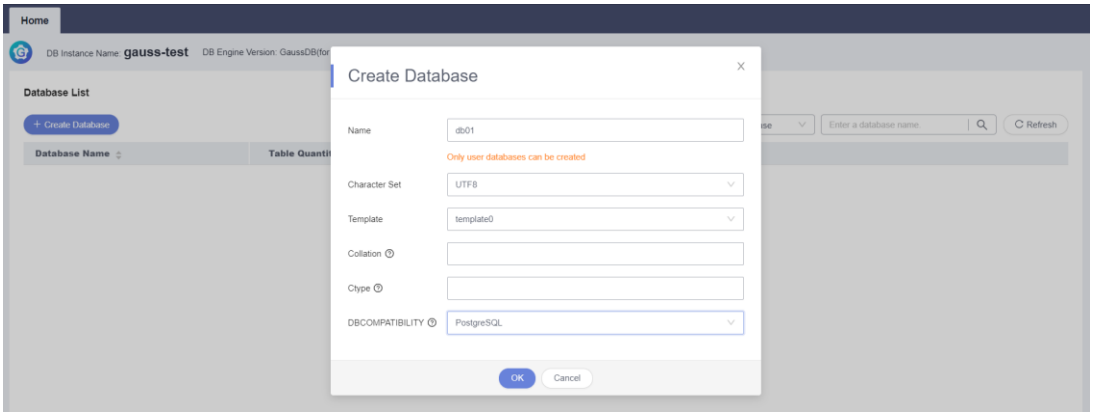
Step 3 Click **Development Tool**, locate GaussDB(for openGauss), and click **Log In**.

Enter the password set during database instance creation, select **Remember Password**, and click **Test Connection**. After the connection test is successful, click **OK**.



4.3.2 Creating a Database

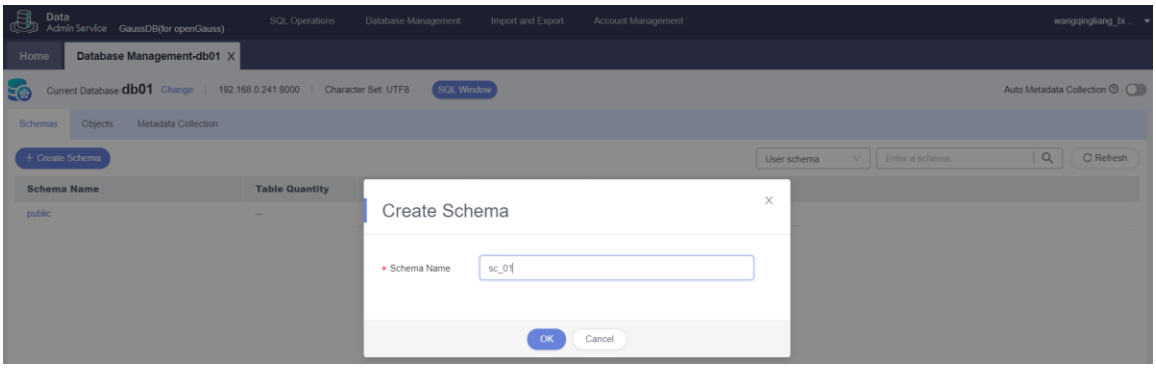
Step 1 Click **Create Database** to create a database. Set **Name**, select **PostgreSQL** for **DBCOMPATIBILITY**, and click **OK**.



Step 2 Click **Manage** to perform operations on the newly created **db01** database.

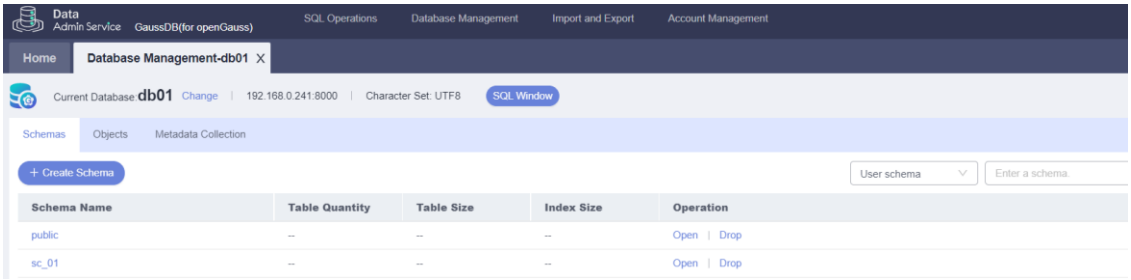
4.3.3 Creating a Schema

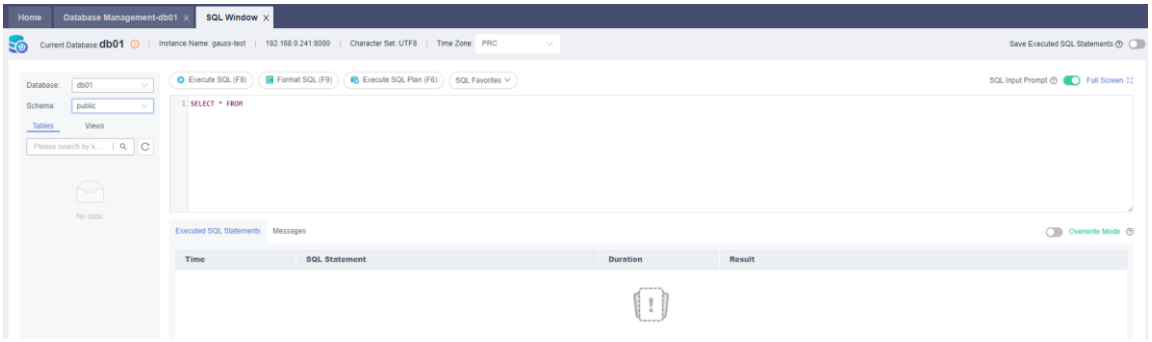
Step 1 Click **Create Schema** to create a schema, enter the schema name, and click **OK**.



4.3.4 Creating a Table

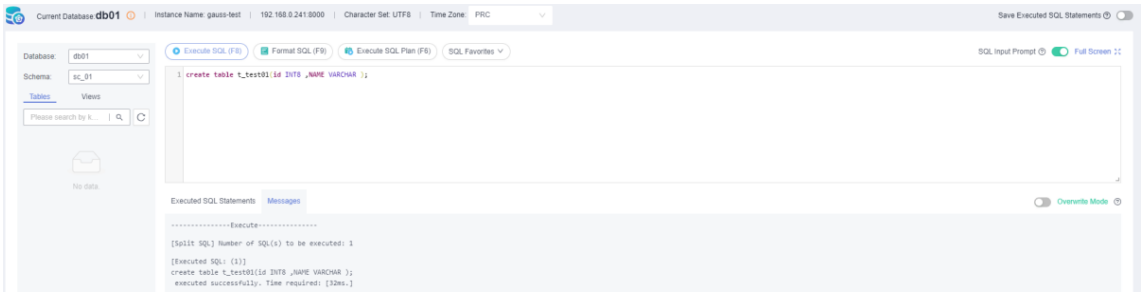
Step 1 Click **SQL Window**. The SQL query tab page is displayed, as shown in the following figure.





Step 2 On the left of the page, select the created **sc_01** schema and enter the SQL statement for creating a table. Click **Execute SQL**. The message "executed successfully" is displayed.

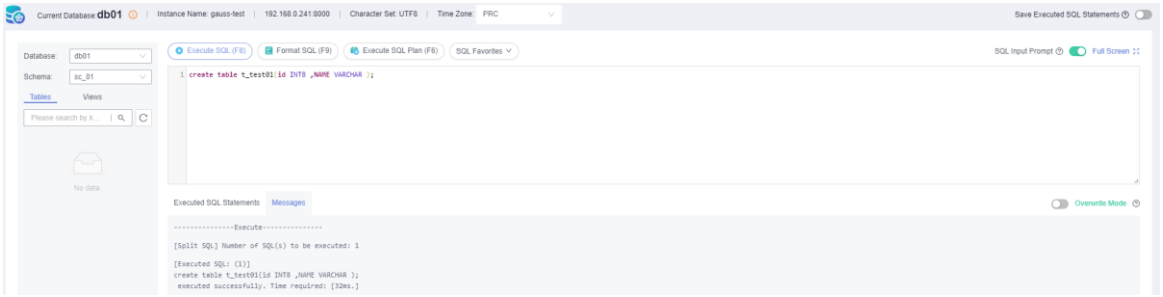
```
create table t_test01(id INT8 ,NAME VARCHAR );
```



4.3.5 Inserting Data

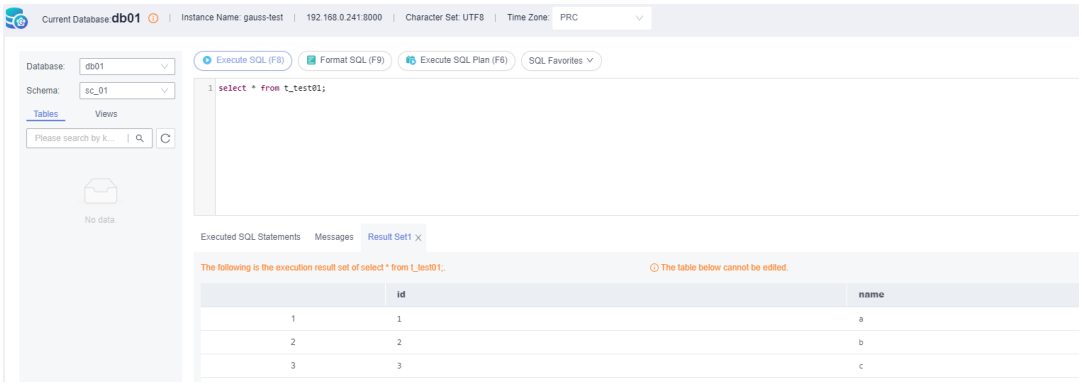
Step 1 Insert data to the created table on the current page and click **Execute SQL**.

```
insert into t_test01 values (1,'a');
insert into t_test01 values (2,'b');
insert into t_test01 values (3,'c');
```



Step 2 Query the inserted data.

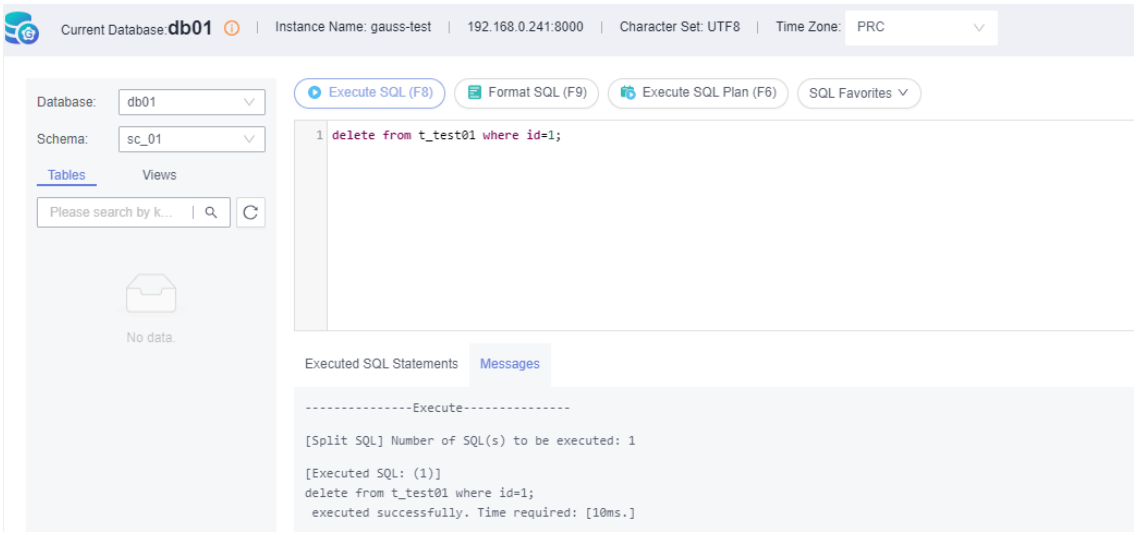
```
select * from t_test01;
```



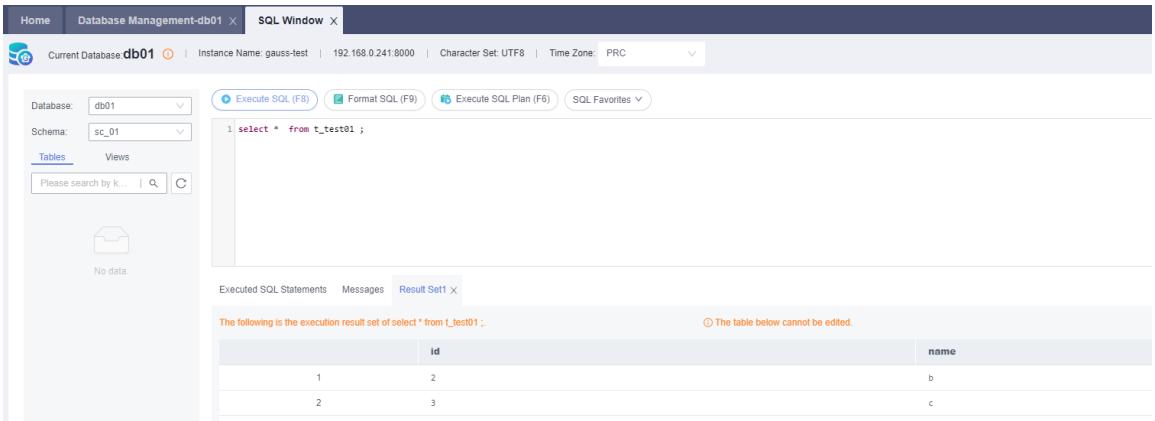
4.3.6 Deleting Data

Step 1 Enter the DELETE statement to delete the record whose **id** is set to **1**.

```
delete from t_test01 where id=1;
```



Step 2 View all data in the original table and verify that the record whose **id** is set to **1** has been deleted.

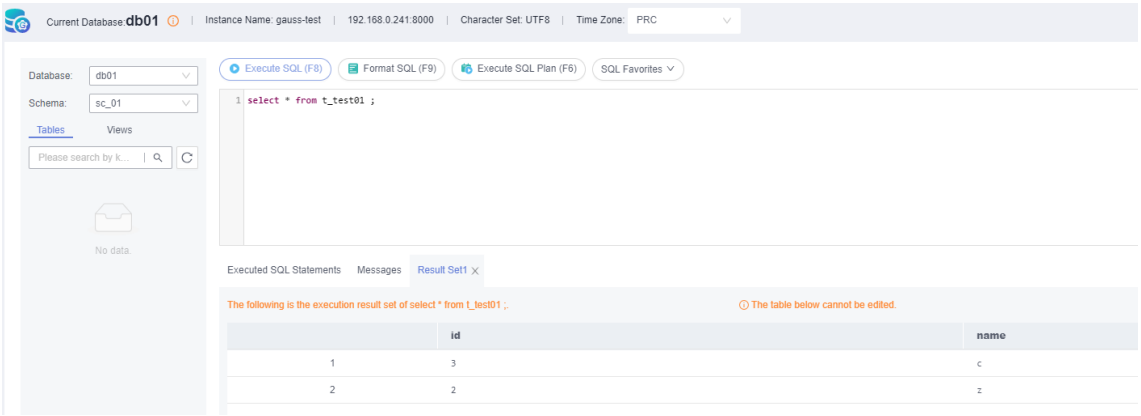


4.3.7 Updating Data

Step 1 Change the value of **name** of the record whose **id** is set to **2** to **z** in the **t_test01** table by using the UPDATE statement.

```
update t_test01 set name='z' where id=2;
```

Step 2 View the result.



4.4 Summary

This experiment shows how to purchase a GaussDB(for openGauss) instance and connect to it by using DAS. By creating databases, schemas, and tables, and adding, deleting, updating, and querying data in tables through DAS, you can understand the usage of basic SQL statements and DAS.

4.5 Quiz

How do I view the execution plan of SQL statements on DAS?

Use the function of viewing an execution plan to simulate a scenario where an index is created on a table to optimize the execution plan.

5 Comprehensive Experiment

5.1 Financial Data Model

Assume that bank C in city A uses openGauss to facilitate bank data management and operation. In this experiment, the objects are classified into customers, bank cards, wealth management products, insurance, and funds. For these database objects, this experiment assumes that the **finance** database of bank C has the following relationships: A customer can apply for a bank card, and the customer can purchase different bank products, such as wealth management products, funds, and insurance. As such, according to the relationships, the corresponding relationship mode and entity relationship (ER) diagram are given in this experiment, and complex database operations are performed.

5.1.1 ER Diagram

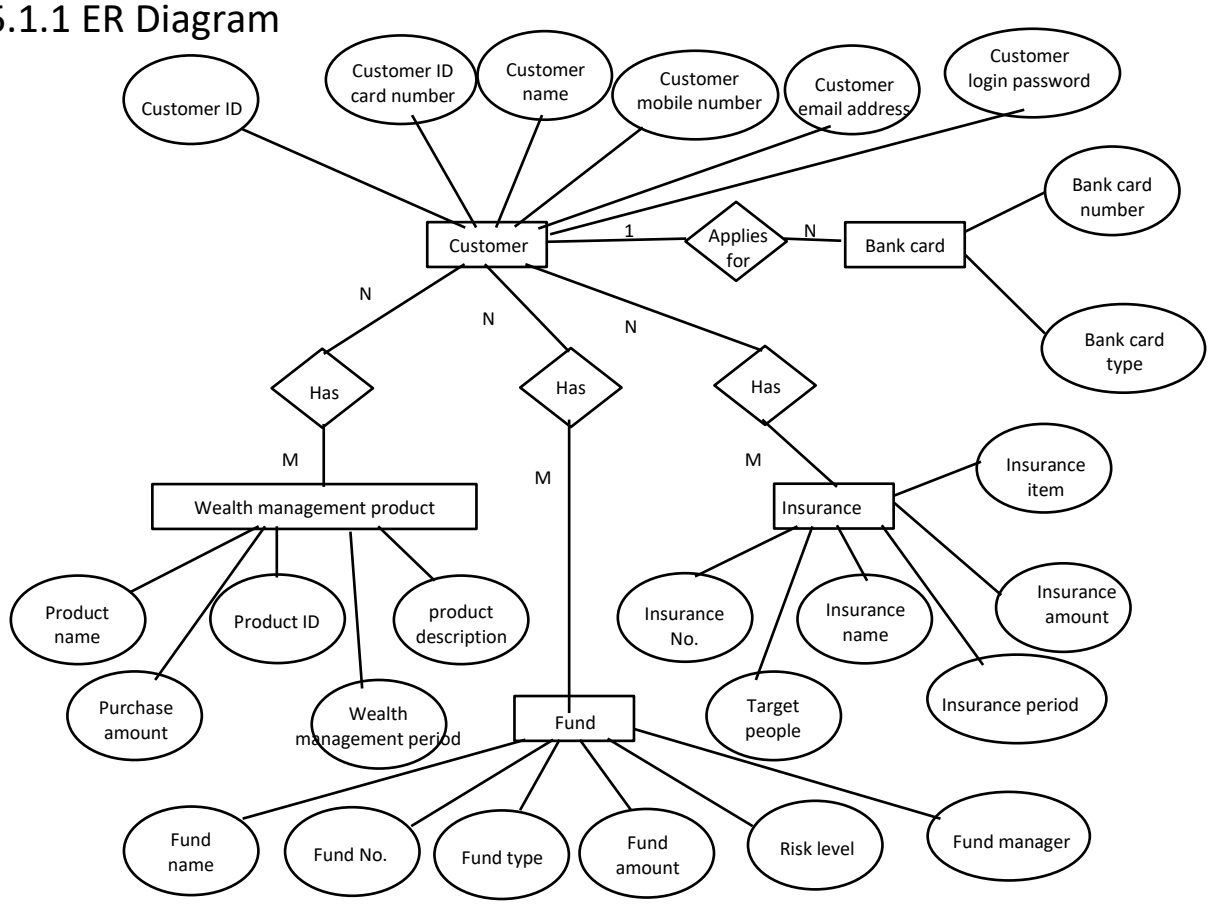


Figure 5-1 ER diagram

5.1.2 Relationship Mode

- For the five objects in bank C, an attribute set is created for each object. The attributes are listed as follows:
- Customer (customer ID, name, email address, ID card number, mobile number, and login password)
- Bank card (bank card number and bank card type)
- Wealth management product (product name, product ID, product description, purchase amount, and wealth management period)
- Insurance (insurance name, insurance No., insurance amount, target people, insurance period, and insurance item)
- Fund (fund name, fund No., fund type, fund amount, risk level, and fund manager)

Relationships:

- One customer can apply for multiple bank cards.
- One customer can purchase multiple wealth management products. One wealth management product can be purchased by multiple customers.
- One customer can purchase multiple fund products. One fund product can be purchased by multiple customers.
- One customer can purchase multiple insurances. One insurance can be purchased by multiple customers.

Based on the relationship analysis, the relationship mode is designed as follows:

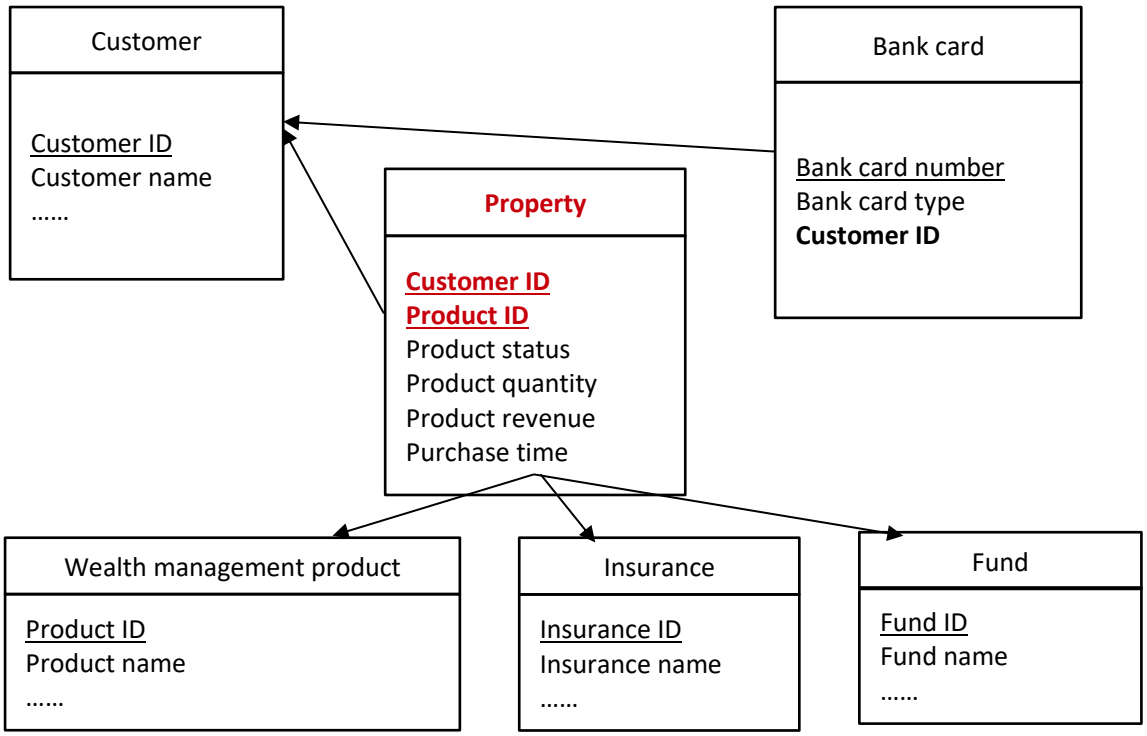


Figure 5-2 Design diagram of a financial data relationship model

Note:

- Because one customer can apply for multiple bank cards, the **bank_card** table uses the customer ID in the **client** table as the foreign key.
- Because one customer can purchase multiple wealth management products and one wealth management product can be purchased by multiple customers, a relationship table, that is, **property** table, is generated. The **property** table uses the product ID in the **client** table as the foreign key and the product ID in the **finances_product** table as the foreign key. Attributes such as product status, quantity, revenue, and purchase time are added to the **property** table.
- The relationship between the customer and the insurance, and that between the customer and the fund are similar. As such, the **property** table is also used as the generated relationship table. The wealth management product ID in the **property** table is changed to the product ID, which is referenced from the **finances_product** table, **insurance** table, and **fund** table.

5.1.3 Physical Model

The IDs of the objects and attributes are as follows:

- client (c_id, c_name, c_mail, c_id_card, c_phone, and c_password)
- bank_card (b_number, b_type, and **b_c_id**)
- finances_product (p_name, p_id, p_description, p_amount, and p_year)
- insurance (i_name, i_id, i_amount, i_person, i_year, and i_project)
- fund (f_name, f_id, f_type, f_amount, risk_level, and f_manager)
- property (**pro_id**, pro_c_id, pro_pif_id, **pro_type**, pro_status, pro_quantity, pro_income, and pro_purchase_time)

Note:

- In the **property** table, the **pro_pif_id** column references the **id** column in the **finances_product** table, **insurance** table, and **fund** table. To prevent the **id** columns of the three products from conflicting with each other, the **pro_type** column is added to distinguish the three products. In addition, the **pro_id** column is added to the **property** table as the primary key.

Table 5-1 client table

Column	Type	Constraint	Description
c_id	INTEGER	PRIMARY KEY	Customer ID
c_name	VARCHAR(100)	NOT NULL	Customer name
c_mail	CHAR(30)	UNIQUE	Customer's email address
c_id_card	CHAR(20)	UNIQUE NOT NULL	Customer's ID card number
c_phone	CHAR(20)	UNIQUE NOT NULL	Customer's mobile number
c_password	CHAR(20)	NOT NULL	Customer's login password

Table 5-2 bank_card table

Column	Type	Constraint	Description
b_number	CHAR(30)	PRIMARY KEY	Bank card number

b_type	CHAR(20)		Bank card type
b_c_id	INTEGER	NOT NULL FOREIGN KEY	Customer ID Note: This column is referenced from the **c_id** column in the **client** table.

Table 5-3 finances_product table

Column	Type	Constraint	
p_name	VARCHAR(100)	NOT NULL	Product name
p_id	INTEGER	PRIMARY KEY	Product ID
p_description	VARCHAR(4000)		Product description
p_amount	INTEGER		Purchase amount
p_year	INTEGER		Wealth management period

Table 5-4 insurance table

Column	Type	Constraint	Description
i_name	VARCHAR(100)	NOT NULL	Insurance name
i_id	INTEGER	PRIMARY KEY	Insurance No.
i_amount	INTEGER		Insurance amount
i_person	CHAR(20)		Target people
i_year	INTEGER		Insurance period
i_project	VARCHAR(200)		Insurance project

Table 5-5 fund table

Column	Type	Constraint	Description
f_name	VARCHAR(100)	NOT NULL	Fund name
f_id	INTEGER	PRIMARY KEY	Fund No.
f_type	CHAR(20)		Fund type
f_amount	INTEGER		Fund amount
risk_level	CHAR(20)	NOT NULL	Fund risk level
f_manager	INTEGER	NOT NULL	Fund manager

Table 5-6 property table

Column	Type	Constraint	
pro_id	INTEGER	PRIMARY KEY	Property No.
pro_c_id	VARCHAR(100)	NOT NULL FOREIGN KEY	Customer ID Note: This column is referenced from the **c_id** column in the **client** table.
pro_pif_id	INTEGER	NOT NULL FOREIGN KEY	Product ID Note: This column is referenced from the **id** columns in the **finances_product**, **insurance**, and **fund** tables.
pro_type	INTEGER	NOT NULL	Product type Note: The value **1** indicates wealth management products, the value **2** indicates insurance, and the value **3** indicates funds.
pro_status	CHAR(20)		Product status
pro_quantity	INTEGER		Product quantity
pro_income	INTEGER		Product revenue
pro_purchase_time	DATE		Purchase time

Next, operations will be performed on the openGauss model table.

5.1.4 Database Connection Settings

After a database is created, you need to configure the database listener and whitelist so that you can use a client tool, such as Data Studio, JDBC, or ODBC, to connect to the database.

Step 1 Modify the **pg_hba.conf** file of the database.

Search for the **pg_hba.conf** file in *GS_HOME*. In this experiment, *GS_HOME* of the database is set to **/gaussdb/data**. In actual operations, the value of *GS_HOME* can be obtained in the **<PARAM name="dataNode1" value="/gaussdb/data"/>** configuration file during the installation.

```
cd /gaussdb/data
vi pg_hba.conf
```

Find the corresponding location, enter **i** to switch to the **INSERT** mode, add the following content to the **pg_hba.conf** file, press **ESC** to exit the **INSERT** mode, enter **:wq**, and press **Enter** to save the change. **host all all 0.0.0.0/0 sha256** corresponds to **host DATABASE USER ADDRESS METHOD**, that is, the host mode. **all** is set for the database and user, **0.0.0.0/0** is set for the address, indicating all network segments, and **sha256** is set for the encryption mode.

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
```

```
host    all    all    192.168.0.19/32    trust
host all all 0.0.0.0/0 sha256
# IPv6 local connections:
host    all                all                ::1/128                trust
```

Step 2 Change the database listening address.

The *GS_HOME* of the database is set to **/gaussdb/data** in this experiment.

```
cd /gaussdb/data
vi postgresql.conf
```

Find the corresponding location, enter **i** to switch to the **INSERT** mode, change the value of **listen_addresses** to *****, press **ESC** to exit the **INSERT** mode, enter **:wq**, and press **Enter** to save the change. The value ***** indicates that all IP address requests are listened.

```
#listen_addresses = '192.168.0.19'                # what IP address(es) to listen on;
listen_addresses = '*'
```

You can also run the **gs_guc** command to modify the parameters. If you directly modify the **postgresql.conf** file, go to step 3.

```
gs_guc set -N all -I all -c "listen_addresses='*'"
```

The result is as follows:

Begin to perform the total nodes: 1.

Popen count is 1, Popen success count is 1, Popen failure count is 0.

Begin to perform gs_guc for datanodes.

Command count is 1, Command success count is 1, Command failure count is 0.

Total instances: 1. Failed instances: 0.

ALL: Success to perform gs_guc!

Step 3 Restart the database to make the modification take effect.

After the modification, restart the database for the modification to take effect. (Change the default database path following **-D** as required. In this experiment, *GS_HOME* of the database is set to **/gaussdb/data**.)

```
gs_ctl restart -D /gaussdb/data/
```

5.1.5 Creating a Database and a Schema

According to the scenario description of bank C, create a database named **finance** and a schema named **finance**. The procedure is as follows:

Step 1 Create a database named **finance**.

Switch to the **omm** user (if the current user is not **omm**), and log in to the primary database node as the **omm** user.

```
su - omm
```

Step 2 Use the **gsql** tool to connect to the database.

```
gsql -d postgres -p 26000 -r
```

Step 3 Create a database named **finance**.

```
CREATE DATABASE finance ENCODING 'UTF8' template = template0;
```

Step 4 Connect to the **finance** database.

```
\connect finance
```

Step 5 Create a schema named **finance** and set **finance** as the current schema.

```
CREATE SCHEMA finance;
```

Step 6 Run the meta-command to view the database.

```
\l
```

The result is as follows:

```
                                  List of databases
    Name     | Owner | Encoding  | Collate | Ctype | Access privileges
-------------+-------+-----------+---------+-------+-------------------
 finance     |       | UTF8      | C       | C     |
 postgres    |       | SQL_ASCII | C       | C     |
 template0 |       | SQL_ASCII | C       | C     | =c/omm               +
             |       |           |         |       | omm=CTc/omm
 template1 |       | SQL_ASCII | C       | C     | =c/omm               +
             |       |           |         |       | omm=CTc/omm
(4 rows)
```

Step 7 Change the default schema search path of the **finance** database to **public** and **finance**.

```
ALTER DATABASE finance SET search_path TO finance,public;
```

Step 8 View the default schema search path of the **finance** database.

```
show search_path
```

The result is as follows:

```
search_path
-----------------
 finance, public
(1 row)
```

Step 9 Exit the database.

```
\q
```

5.1.6 Creating and Configuring a Tablespace

Tablespaces are configured to logically manage tables in a database. Create the **finance_tbs** tablespace for the **finance** database, and set the **finance_tbs** tablespace as the default tablespace of the **finance** database.

Step 1 Connect to the database.

```
gsql -d postgres -p 26000 –r
```

Step 2 Create a tablespace.

```
CREATE TABLESPACE finance_tbs LOCATION '/opt/opengauss/tablespace/finance_tbs1';
```

Step 3 Run the meta-command to view the tablespace.

```
\db
```

The result is as follows:

```
                              List of tablespaces
    Name         | Owner |                      Location
-------------+-------+-----------------------------------------
 finance_tbs | omm     | /opt/opengauss/tablespace/finance_tbs1
 pg_default  | omm     |
 pg_global   | omm     |
(3 rows)
```

Step 4 Query the tablespace usage.

Query the current usage of the tablespace. The result indicates the size of the tablespace, in bytes.

```
SELECT PG_TABLESPACE_SIZE('finance_tbs');
```

The result is as follows:

```
pg_tablespace_size
--------------------
                4096
(1 row)
```

Step 5 Set the created tablespace to the default tablespace of the **finance** database.

```
ALTER DATABASE finance SET tablespace finance_tbs;
```

The following error information is displayed:

```
ERROR:  cannot change the tablespace of the currently open database
```

The reason for the error is that the session connected to the **finance** database cannot modify the default tablespace of the database. Therefore, you need to switch the session to the default **postgres** database and then modify the tablespace.

```
\connect postgres
```

Modify the default tablespace of the **finance** database again.

```
ALTER DATABASE finance SET tablespace finance_tbs;
```

If the following information is displayed, the modification is successful:

ALTER DATABASE

Step 6 Exit the database.

```
\q
```

5.1.7 Creating a User and Granting Permissions to It

Assume that an application of bank C needs to access a database. In this case, you need to create the **bank_app** user for the business and grant certain permissions to the user. The procedure is as follows:

Step 1 Connect to the database.

```
gsql -d postgres -p 26000 -r
```

Step 2 Access the SQL command page after connecting to the database. Create the **bank_app** user and set the password to **Gauss#3demo**.

```
CREATE USER bank_app IDENTIFIED BY 'Gauss#3demo';
```

Step 3 Switch to the **finance** database.

```
\connect finance;
```

Step 4 Grant the **SCHEMA** permission to the **bank_app** user.

```
GRANT ALL ON SCHEMA finance to bank_app;
```

Step 5 Exit the database.

```
\q
```

5.1.8 Connecting to a Database as a New User

Step 1 Log in to the database using the **gsql** tool and connect to the database as the new user.

In this experiment, you can use the **gsql** tool to log in to the database. You can also use other client tools, such as Data Studio, to connect to the database.

Open a new window for login as the **omm** user, run the following command, and enter the password (for example, **Gauss#3demo**):

```
gsql -d finance -U bank_app -p 26000 -r
```

Step 2 Access the **bank_card** table in the **finance** database.

```
\du;
```

The result is as follows:

```
                    List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 bank_app  |            | {}
```

Step 3 Set the default search path to **finance**.

(You do not need to set the parameters that have been set in section 1.1.5.)

```
SET search_path TO finance;
```

Step 4 Check search_path.

```
show search_path;
```

The result is as follows:

```
search_path
-----------------
 finance
(1 row)
```

5.1.9 Creating a Data Table

Based on the scenario description of bank C, **client**, **bank_card**, **finances_product**, **insurance**, **fund**, and **property** tables are created in this experiment. The detailed experiment procedure is as follows:

Step 1 Create a table named **client**.

In the previous experiment, if the connection is not disconnected, you can run the SQL statement for creating a table. If the connection is disconnected, you can perform the steps in section 1.1.7 again.

Enter the following statement in the SQL editing box to create the **client** table:

Delete the **client** table.

```
DROP TABLE IF EXISTS finance.client;
```

Create the **client** table.

```
CREATE TABLE finance.client
(
        c_id INT PRIMARY KEY,
        c_name VARCHAR(100) NOT NULL,
        c_mail CHAR(30) UNIQUE,
        c_id_card CHAR(20) UNIQUE NOT NULL,
        c_phone CHAR(20) UNIQUE NOT NULL,
        c_password CHAR(20) NOT NULL
```

```
);
```

Add annotations.

```
COMMENT ON TABLE finance.client IS 'client table';
```

Step 2 Create a table named **bank_card**.

Enter the following statement in the SQL editing box to create the **bank_card** table:

Delete the **bank_card** table.

```
DROP TABLE IF EXISTS finance.bank_card;
```

Create the **bank_card** table and add annotations.

```
CREATE TABLE finance.bank_card
(
        b_number CHAR(30) PRIMARY KEY,
        b_type CHAR(20),
        b_c_id INT NOT NULL
);
COMMENT ON TABLE finance.bank_card IS 'bank_card table';
```

Step 3 Create a table named **finances_product**.

Create the **finances_product** table.

Delete the **finances_product** table.

```
DROP TABLE IF EXISTS finance.finances_product;
```

Create the **finances_product** table.

```
CREATE TABLE finance.finances_product
(
        p_name VARCHAR(100) NOT NULL,
        p_id INT PRIMARY KEY,
        p_description VARCHAR(4000),
        p_amount INT,
        p_year INT
);
COMMENT ON TABLE finance.finances_product IS 'finances_product table';
```

Step 4 Create a table named **insurance**.

Enter the following statement in the SQL editing box to create the **insurance** table:

Delete the **insurance** table.

```
DROP TABLE IF EXISTS finance.insurance;
```

Create the **insurance** table.

```
CREATE TABLE finance.insurance
```

```
(
        i_name VARCHAR(100) NOT NULL,
        i_id INT PRIMARY KEY,
        i_amount INT,
        i_person CHAR(20),
        i_year INT,
        i_project VARCHAR(200)
);
COMMENT ON TABLE finance.insurance IS 'insurance table';
```

Step 5 Create a table named **fund**.

Enter the following statement in the SQL editing box to create the **fund** table:

Delete the **fund** table.

```
DROP TABLE IF EXISTS finance.fund;
```

Create the **fund** table.

```
CREATE TABLE finance.fund
(
        f_name VARCHAR(100) NOT NULL,
        f_id INT PRIMARY KEY,
        f_type CHAR(20),
        f_amount INT,
        risk_level CHAR(20) NOT NULL,
        f_manager INT NOT NULL
);
COMMENT ON TABLE finance.fund IS 'fund table';
```

Step 6 Create a table named **property**.

Enter the following statement in the SQL editing box to create the **property** table:

Delete the **property** table.

```
DROP TABLE IF EXISTS finance.property;
```

Create the **property** table.

```
CREATE TABLE finance.property
(
        pro_id INT PRIMARY KEY,
pro_c_id INT NOT NULL,
        pro_pif_id INT NOT NULL,
        pro_type INT NOT NULL,
        pro_status CHAR(20),
        pro_quantity INT,
        pro_income INT,
        pro_purchase_time DATE
);
COMMENT ON TABLE finance.property IS 'property table';
```

Step 7 View the created tables.

```
\d+;
```

The result is as follows:

```
                                                    List of relations
 Schema |          Name          | Type  |  Owner   |    Size    |                Storage                 | Description
--------+------------------------+-------+----------+------------+----------------------------------------+-------------
 finance | bank_card             | table | bank_app | 8192 bytes | {orientation=row,compression=no}       | bank_card table
 finance | client                | table | bank_app | 8192 bytes | {orientation=row,compression=no}       | client table
 finance | finances_product | table | bank_app | 16 kB      | {orientation=row,compression=no}       | finances_product table
 finance | fund                  | table | bank_app | 8192 bytes | {orientation=row,compression=no}       | fund table
 finance | insurance             | table | bank_app | 8192 bytes | {orientation=row,compression=no}       | insurance table
 finance | property              | table | bank_app | 8192 bytes | {orientation=row,compression=no}       | property table
(6 rows)
```

5.1.10 Inserting Data Into a Table

This experiment shows how to run SQL statements to insert some data into related tables in the **finance** database, aiming to describe how to perform operations on data in tables.

Step 1 Initialize data in the **client** table.

Insert data.

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (1,'Zuo Xiaoting','zuoxiaoting@huawei.com','340211199301010001','18815650001','gaussdb_001');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (2,'Yu Chenggang','yuchenggang@huawei.com','340211199301010002','18815650002','gaussdb_002');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (3,'Zhu Changgang','zhuchanggang@huawei.com','340211199301010003','18815650003','gaussdb_003');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (4,'Ren Gaofeng','rengaofeng@huawei.com','340211199301010004','18815650004','gaussdb_004');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (5,'An Yanfang','anyanfang@huawei.com','340211199301010005','18815650005','gaussdb_005');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (6,'Teng Changli','tengchangli@huawei.com','340211199301010006','18815650006','gaussdb_006');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (7,'Fu Xiaofang','fuxiaofang@huawei.com','340211199301010007','18815650007','gaussdb_007');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (8,'Bian Lanjuan','bianlanjuan@huawei.com','340211199301010008','18815650008','gaussdb_008');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (9,'Shao Xiaoting,'shaoxiaoting@huawei.com','340211199301010009','18815650009','gaussdb_009');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (10,'Zhang Xiaofeng','zhangxiaofeng@huawei.com','340211199301010010','18815650010','gaussdb_010');
```

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (11,'Kang
Xiaopeng','kangxiaopeng@huawei.com','340211199301010011','18815650011','gaussdb_011');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (12,'Feng
Changqiang','fengchangqiang@huawei.com','340211199301010012','18815650012','gaussdb_012');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (13,'Ying
Xiaopeng','yingxiaopeng@huawei.com','340211199301010013','18815650013','gaussdb_013');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (14,'Zhao
Xiaopeng','zhaoxiaopeng@huawei.com','340211199301010014','18815650014','gaussdb_014');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (15,'Su
Xiaogang','suxiaogang@huawei.com','340211199301010015','18815650015','gaussdb_015');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (16,'Yin
Gaofeng','yingaofeng@huawei.com','340211199301010016','18815650016','gaussdb_016');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (17,'Meng
Xiaoxuan','mengxiaoxuan@huawei.com','340211199301010017','18815650017','gaussdb_017');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (18,'Tang
Gaowei','tanggaowei@huawei.com','340211199301010018','18815650018','gaussdb_018');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (19,'Hong
Gaopeng','gaohongpeng@huawei.com','340211199301010019','18815650019','gaussdb_019');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (20,'Song
Chengfeng','songchengfeng@huawei.com','340211199301010020','18815650020','gaussdb_020');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (21,'Qiu
Changqiang','qiuchangqiang@huawei.com','340211199301010021','18815650021','gaussdb_021');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (22,'Wei
Gaopeng','weigaopeng@huawei.com','340211199301010022','18815650022','gaussdb_022');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (23,'Pan
Chengxuan','panchengxuan@huawei.com','340211199301010023','18815650023','gaussdb_023');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (24,'Ji
Xiaomei','jixiaomei@huawei.com','340211199301010024','18815650024','gaussdb_024');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (25,'Li
Chunting','lichunting@huawei.com','340211199301010025','18815650025','gaussdb_025');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (26,'Ni
Lanfang','nilanfang@huawei.com','340211199301010026','18815650026','gaussdb_026');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (27,'Jiang
Xiaofang','jiangxiaofang@huawei.com','340211199301010027','18815650027','gaussdb_027');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (28,'Li
Lanfang','lilanfang@huawei.com','340211199301010028','18815650028','gaussdb_028');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (29,'Lou
Xiaoting','louxiaoting@huawei.com','340211199301010029','18815650029','gaussdb_029');
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (30,'Fu
Xiaoting','fuxiaoting@huawei.com','340211199301010030','18815650030','gaussdb_030');
```

Query the insertion result.

```
select count(*) from finance.client;
```

The result is as follows:

```
count(*)
----------
30
```

Step 2 Initialize data in the **bank_card** table.

Insert data.

```
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000001','credit card',1);
```

```
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000002','credit card',3);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000003','credit card',5);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000004','credit card',7);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000005','credit card',9);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000006','credit card',10);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000007','credit card',12);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000008','credit card',14);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000009','credit card',16);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000010','credit card',18);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000011','debit card',19);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000012','debit card',21);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000013','debit card',7);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000014','debit card',23);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000015','debit card',24);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000016','debit card',3);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000017','debit card',26);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000018','debit card',27);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000019','debit card',12);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222022302020000020','debit card',29);
```

Query the insertion result.

```
select count(*) from finance.bank_card;
```

The result is as follows:

```
count(*)
----------
20
```

Step 3 Initialize data in the **finances_product** table.

Insert data.

```
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('bonds',1,'wealth management products with government bonds, financial bonds, central bank bills, and enterprise bonds as the main investment direction. ',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('credit-related property',2,'It generally refers to the situation where a bank, as the trustor, entrusts a trust company with fund raising through the issuance of wealth management products, and the trust company, as the trustee, makes a trust plan and purchases wealth management products from the trust company and sells the wealth management products to the bank or third-party credit property.',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('stocks',3,'wealth management products related to stocks. At present, Hong Kong stocks are in the majority in the market',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES('commodities',4,'wealth management products related to commodity futures. At present, wealth management products, such as gold, oil, agricultural products, are in the majority in the market',50000,6); ',50000,6);
```

Query the insertion result.

```
select count(*) from finance.finances_product;
```

The result is as follows:

```
count(*)
----------
4
```

Step 4 Initialize data in the **insurance** table.

Insert data.

```
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('health insurance',1,2000,'elderly people',30,'Ping An Insurance');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('life insurance',2,3000,'elderly people',30,'Ping An Insurance');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('accident insurance',35,000,'all people',30,'Ping An Insurance');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('medical insurance',4,2000,'all people',30,'Ping An Insurance');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('property loss insurance',5,1500,'middle-aged people',30,'Ping An Insurance');
```

Query the insertion result.

```
select count(*) from finance.insurance;
```

The result is as follows:

```
count(*)
-------------
5
```

Step 5 Initialize data in the **fund** table.

Insert data.

```
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('stock',1,'stock',10000,'high',1);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('investment',2,'bond',10000,'medium',2);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('government bond',3,'currency',10000,'low',3);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('CSI 300 index',4,'exponential',10000,'medium',4);
```

Query the insertion result.

```
select count(*) from finance.fund;
```

The result is as follows:

```
count(*)
-------------
4
```

Step 6 Initialize data in the **property** table.

Insert data.

```
INSERT INTO finance.property(pro_id,pro_c_id,pro_pif_id,pro_type,pro_status,pro_quantity,pro_income,pro_purchase_time) VALUES (1,5,1,1,'available',4,8000,'2018-07-01');
INSERT INTO finance.property(pro_id,pro_c_id,pro_pif_id,pro_type,pro_status,pro_quantity,pro_income,pro_purchase_time) VALUES (2,10,2,2,'available',4,8000,'2018-07-01');
INSERT INTO finance.property(pro_id,pro_c_id,pro_pif_id,pro_type,pro_status,pro_quantity,pro_income,pro_purchase_time) VALUES (3,15,3,3,'available',4,8000,'2018-07-01');
```

```
INSERT INTO
finance.property(pro_id,pro_c_id,pro_pif_id,pro_type,pro_status,pro_quantity,pro_income,pro_purchase_time) VALUES
(4,20,4,1,'freeze',4,8000,'2018-07-01');
```

Query the insertion result.

```
select count(*) from finance.property;
```

The result is as follows:

```
count(*)
----------
4
```

5.1.11 Inserting a Data Record Manually

When bank C needs to add new information to the database, a new data record needs to be added to the corresponding data table. As such, this section describes how to manually insert a data record in a scenario where a primary key attribute is defined.

Step 1 Add information about a customer to the **client** table in the **finance** database (in a scenario where the attributes conflict with each other).

Values of **c_id_card** and **c_phone** are not unique.

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (31,'Li
Li','lili@huawei.com','340211199301010005','18815650005','gaussdb_005')
```

The error information is as follows:

```
ERROR:  duplicate key value violates unique constraint "client_c_id_card_key"
DETAIL:  Key (c_id_card)=(340211199301010005  ) already exists.
```

Note: During table creation,the values of **c_id_card** and **c_phone** defined in this experiment are unique and not empty (**UNIQUE NOT NULL**). Therefore, when the same record exists in the table, data fails to be inserted.

Step 2 Add information about a customer to the **client** table in the **finance** database to meet the constraints in the table (in a scenario where the insertion succeeds).

The following example shows a successful data insertion.

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (31,'Li
Li','lili@huawei.com','340211199301010031','18815650031','gaussdb_031')
```

Step 3 Verify the insertion result.

```
select * from finance.client where c_id=31;
```

The result is as follows:

```
c_id | c_name |          c_mail          |       c_id_card       |       c_phone
|       c_password
------+--------+--------------------------+-----------------------+-----------------------
    31 | Li Li     | lili@huawei.com                  | 340211199301010031     | 18815650031
| gaussdb_031
```

5.1.12 Adding a Constraint

Step 1 Add foreign key constraints to tables. In the **bank_card** table and **property** table, each bank card must have a holder and each property must have an owner. As such, foreign key constraints are created.

Add a foreign key constraint to the **bank_card** table.

```
ALTER TABLE finance.bank_card ADD CONSTRAINT fk_c_id FOREIGN KEY (b_c_id) REFERENCES finance.client(c_id)
ON DELETE CASCADE;
```

Add a foreign key constraint to the **property** table.

```
ALTER TABLE finance.property ADD CONSTRAINT fk_pro_c_id FOREIGN KEY (pro_c_id) REFERENCES
finance.client(c_id) ON DELETE CASCADE;
```

Note:

- The value of **b_c_id** in the **bank_card** table is the same as that of **c_id** in the **client** table. Each bank card must have a holder.
- Before deleting the **client** table, delete the **bank_card** table because the two tables have constraints.
- The value of **pro_c_id** in the **property** table is the same as that of **c_id** in the **client** table. Each property must have an owner.
- Before deleting the **client** table, delete the **property** table because the two tables have constraints.
- In an actual application system, foreign key constraints are hardly created in database tables. All concepts about foreign keys are involved at the application layer.

Step 2 The amount attribute exists in the **finances_product**, **insurance**, and **fund** tables. In actual operations, the amount is not negative. As such, a constraint that the amount must be greater than 0 is added.

Add a constraint that the value of the **p_amount** column is greater than or equal to 0 to the **finances_product** table.

```
ALTER table finance.finances_product ADD CONSTRAINT c_p_mount CHECK (p_amount >=0);
```

Step 3 Manually insert a data record whose amount is less than 0.

```
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('credit-related
property',10,'It generally refers to the situation where a bank, as the trustor, entrusts a trust company with fund
raising through the issuance of wealth management products, and the trust company, as the trustee, makes a
trust plan and purchases wealth management products from the trust company and sells the wealth management
products to the bank or third-party credit property. ',-10,6);
```

The execution failed. The reason is as follows:

ERROR: new row for relation "finances_product" violates check constraint "c_p_mount"

DETAIL: Failed row contains (credit-related property, 10, It generally refer to the situation where a bank, as the trustor, raises funds by issuing wealth management products..., -10, 6).

Step 4 Add a constraint to the **fund** table.

Add a constraint that the value of the **f_amount** column is greater than or equal to 0 to the **fund** table.

```
ALTER table finance.fund ADD CONSTRAINT c_f_mount CHECK (f_amount >=0);
```

Step 5 Add a constraint to the **insurance** table.

Add a constraint that the value of the **i_amount** column is greater than or equal to 0 to the **insurance** table.

```
ALTER table finance.insurance ADD CONSTRAINT c_i_mount CHECK (i_amount >=0);
```

5.1.13 Querying Data

In the experiment on the **finance** database in this section, you can learn more about complex operations on openGauss and more about query operations.

Step 1 Query a single table.

- Query wealth management products.

```
SELECT * from finance.finances_product;
```

The result is as follows:

```
p_name   | p_id |
p_description                                                                                          |
p_amount | p_year
----------+------+------------------------------------------------------------------------------------------------------------------
-----------------------------+----------+--------
 Bonds         |        1 | Wealth management products with government bonds, financial bonds,
central bank bills, and enterprise bonds as the main investment direction
|        50000 |           6
 Credit-related property |        2 | It generally refers to the situation where a bank, as the trustor,
entrusts a trust company with fund raising through the issuance of wealth management products,
and the trust company, as the trustee, makes a trust plan and purchases wealth management
products from the trust company and sells the wealth management products to the bank or third-
party credit property.   |        50000 |           6
 Stocks        |        3 | Wealth management products related to stocks At present, Hong Kong stocks
are in the majority in the market.
|        50000 |           6
 Commodities |        4 | Wealth management products related to commodities futures. At present,
wealth management products, such as gold, oil, agricultural products, are in the majority in the
market.                                                                           |        50000 |
6
(4 rows)
```

Step 2 Perform a conditional query.

- Query available property data in the **property** table.

```
SELECT * from finance.property where pro_status='available';
```

The result is as follows:

```
pro_id | pro_c_id | pro_pif_id | pro_type |        pro_status        | pro_quantity | pro_income |
pro_purchase_time
--------+----------+------------+----------+--------------------------+--------------+------------+---------------------
       1 |        5 |          1 |        1 | Available                |            4 |
8000 | 2018-07-01 00:00:00
       2 |       10 |          2 |        2 | Available                |            4 |
8000 | 2018-07-01 00:00:00
       3 |       15 |          3 |        3 | Available                |            4 |
8000 | 2018-07-01 00:00:00
(3 rows)
```

Step 3 Perform an aggregation query.

- Check the number of customers in the **client** table.

```
SELECT count(*) FROM finance.client;
```

The result is as follows:

```
count
-------
    31
```

- Query the numbers of debit cards and credit cards in the **bank_card** table.

```
SELECT b_type,COUNT(*) FROM finance.bank_card GROUP BY b_type;
```

The result is as follows:

```
      b_type          | count
--------------------+-------
 Debit card          |    10
 Credit card         |    10
```

- Query the average insurance amount in the **insurance** table.

```
SELECT AVG(i_amount) FROM finance.insurance;
```

The result is as follows:

```
          avg
-----------------------
 2700.0000000000000000
```

- Query the insurance type and amount corresponding to the maximum and minimum insurance amounts in the **insurance** table.

```
SELECT i_name,i_amount from finance.insurance where i_amount in (select max(i_amount) from
finance.insurance)
union
```

```
SELECT i_name,i_amount from finance.insurance where i_amount in (select min(i_amount) from
finance.insurance);
```

The result is as follows:

```
i_name        | i_amount
---------------+----------
 Property insurance |        1500
 Accident insurance        |        5000
```

Step 4 Perform a join query.

(1) Perform a semi-join query.

- The semi-join query is performed to query the customer ID, name, and ID card number in the **bank_card** table.

```
SELECT c_id,c_name,c_id_card FROM finance.client WHERE EXISTS (SELECT * FROM finance.bank_card WHERE
client.c_id = bank_card.b_c_id);
```

The result is as follows:

```
c_id | c_name |          c_id_card
------+--------+---------------------
     1 | Zuo Xiaoting | 340211199301010001
     3 | Zhu Changgang | 340211199301010003
     5 | An Yanfang | 340211199301010005
     7 | Fu Xiaofang | 340211199301010007
     9 | Shao Xiaoting | 340211199301010009
    10 | Zhang Xiaofeng | 340211199301010010
    12 | Feng Changqiang | 340211199301010012
    14 | Zhao Xiaopeng | 340211199301010014
    16 | Yin Gaofeng | 340211199301010016
    18 | Tang Gaowei | 340211199301010018
    19 | Hong Gaopeng | 340211199301010019
    21 | Qiu Changqiang | 340211199301010021
    23 | Pan Chengxuan | 340211199301010023
    24 | Ji Xiaomei | 340211199301010024
    26 | Ni Lanfang | 340211199301010026
    27 | Jiang Xiaofang | 340211199301010027
    29 | Lou Xiaoting | 340211199301010029
(17 rows)
```

Note: Semi-join is a special type of join, and there is no keyword specified in the SQL statement. It is implemented by a subquery with **IN** or **EXISTS** following **WHERE**. If multiple rows in the **IN** or **EXISTS** subquery match the conditions, the query returns only one row instead of all the matched rows.

(2) Perform an anti-join query.

- The anti-join query is performed to query the customer ID, name, and ID card number of a customer whose bank card number is not **622202230202000001*** (* means unknown).

```
SELECT c_id,c_name,c_id_card FROM finance.client WHERE c_id NOT IN (SELECT b_c_id FROM finance.bank_card
WHERE b_number LIKE '622202230202000001_');
```

The result is as follows:

```
c_id | c_name |          c_id_card
------+--------+---------------------
     1 | Zuo Xiaoting | 340211199301010001
     2 | Yu Chenggang | 340211199301010002
     3 | Zhu Changgang | 340211199301010003
     4 | Ren Gaofeng | 340211199301010004
     5 | An Yanfang | 340211199301010005
     6 | Teng Changli | 340211199301010006
     7 | Fu Xiaofang | 340211199301010007
     8 | Bian Lanjuan | 340211199301010008
     9 | Shao Xiaoting | 340211199301010009
    10 | Zhang Xiaofeng | 340211199301010010
    11 | Kang Xiaopeng | 340211199301010011
    12 | Feng Changqiang | 340211199301010012
    13 | Ying Xiaopeng | 340211199301010013
    14 | Zhao Xiaopeng | 340211199301010014
    15 | Su Xiaogang | 340211199301010015
    16 | Yin Gaofeng | 340211199301010016
    17 | Meng Xiaoxuan | 340211199301010017
    18 | Tang Gaowei | 340211199301010018
    19 | Hong Gaopeng | 340211199301010019
    20 | Song Chengfeng | 340211199301010020
    21 | Qiu Changqiang | 340211199301010021
    22 | Wei Gaopeng | 340211199301010022
    23 | Pan Chengxuan | 340211199301010023
    24 | Ji Xiaomei | 340211199301010024
    25 | Li Chunting | 340211199301010025
    26 | Ni Lanfang | 340211199301010026
    27 | Jiang Xiaofang | 340211199301010027
    28 | Li Lanfang | 340211199301010028
    29 | Lou Xiaoting | 340211199301010029
    30 | Fu Xiaoting | 340211199301010030
    31 | Li Li      | 340211199301010031
(31 rows)
```

Note: Anti-join is a special type of join, and there is no keyword specified in the SQL statement. It is implemented by a subquery with **NOT IN** or **NOT EXISTS** following **WHERE**. It returns all rows that do not meet the condition. The concept of anti-join is opposite to that of semi-join.

Step 5 Perform a subquery.

- The subquery is performed to query the names of insurance products whose insurance amount is greater than the average amount and the target people.

```
SELECT i1.i_name,i1.i_amount,i1.i_person FROM finance.insurance i1 WHERE i_amount > (SELECT avg(i_amount)
FROM finance.insurance i2);
```

The result is as follows:

```
i_name    | i_amount |          i_person
----------+----------+-------------------
 Life insurance |        3000 | Elderly people
 Accident insurance |        5000 | All
(2 rows)
```

Step 6 Perform queries by using the **ORDER BY** and **GROUP BY** clauses.

(1) Perform a query by using the **ORDER BY** clause.

- Query the names, amounts, and target people of insurances with insurance numbers higher than 2 in descending order.

```
SELECT i_name,i_amount,i_person FROM finance.insurance WHERE i_id>2 ORDER BY i_amount DESC;
```

The result is as follows:

```
i_name       | i_amount |         i_person
-------------+----------+------------------
 Accident insurance         |        5000 | All
 Medical insurance          |        2000 | All
 Property insurance | 1500 | Middle-aged people
(3 rows)
```

(2) Perform a query by using the **GROUP BY** clause.

- Query the total number of wealth management products and group them by **p_year**.

```
SELECT p_year,count(p_id) FROM finance.finances_product GROUP BY p_year;
```

The result is as follows:

```
p_year | count
--------+-------
          6 |         4
```

Step 7 Perform queries by using the **HAVING** and **WITH AS** clauses.

(1) Perform a query by using the **HAVING** clause.

- Query the number of customers whose insurance amount is 2.

```
SELECT i_person,count(i_amount) FROM finance.insurance GROUP BY i_person HAVING count(i_amount)=2;
```

The result is as follows:

```
        i_person          | count
--------------------+-------
 All                    |     2
 Elderly people         |     2
```

Note: The **HAVING** clause depends on the **GROUP BY** clause.

(2) Perform a query by using the **WITH AS** clause.

- **WITH AS** is used to query the **fund** table.

```
WITH temp AS (SELECT f_name,ln(f_amount) FROM finance.fund ORDER BY f_manager DESC) SELECT * FROM
temp;
```

The result is as follows:

```
    f_name      |          ln
-------------+------------------
 CSI 300 index | 9.21034037197618
 Government bond          | 9.21034037197618
 Investment          | 9.21034037197618
 Stock          | 9.21034037197618
```

Note: This statement is used to define an SQL segment, which will be used by the entire SQL statement.

This makes SQL statements more readable. Different from a base table, the table storing SQL segments is a virtual table. The definition and data are not stored in the database but still in the base table. If data in the base table changes, the data queried from the table that stores SQL fragments changes accordingly.

5.1.14 View

A view is a virtual table and is the result of an SQL query. The content of a view is defined by a query. Complex SQL statements are used for complex queries for multiple associated tables, resulting in poor user experience. Using a view can greatly simplify operations. You do not need to pay attention to the structure and association conditions of the corresponding table.

Step 1 Create a view.

Create a view for querying the ID, name, and ID card number of a customer in the **bank_card** table.

```
CREATE VIEW finance.v_client as SELECT c_id,c_name,c_id_card FROM finance.client WHERE EXISTS (SELECT *
FROM finance.bank_card WHERE client.c_id = bank_card.b_c_id);
```

Perform a query by using the view.

```
SELECT * FROM finance.v_client;
```

The result is as follows:

```
c_id | c_name |          c_id_card
------+--------+----------------------
     1 | Zuo Xiaoting | 340211199301010001
     3 | Zhu Changgang | 340211199301010003
     5 | An Yanfang | 340211199301010005
     7 | Fu Xiaofang | 340211199301010007
     9 | Shao Xiaoting | 340211199301010009
    10 | Zhang Xiaofeng | 340211199301010010
    12 | Feng Changqiang | 340211199301010012
    14 | Zhao Xiaopeng | 340211199301010014
    16 | Yin Gaofeng | 340211199301010016
    18 | Tang Gaowei | 340211199301010018
    19 | Hong Gaopeng | 340211199301010019
    21 | Qiu Changqiang | 340211199301010021
    23 | Pan Chengxuan | 340211199301010023
    24 | Ji Xiaomei | 340211199301010024
    26 | Ni Lanfang | 340211199301010026
    27 | Jiang Xiaofang | 340211199301010027
    29 | Lou Xiaoting | 340211199301010029
(17 rows)
```

Step 2 Modify the view.

The view is modified to filter out credit card holders based on the original query result.

```
CREATE OR REPLACE VIEW finance.v_client as SELECT c_id,c_name,c_id_card FROM finance.client WHERE EXISTS
(SELECT * FROM finance.bank_card WHERE client.c_id = bank_card.b_c_id and bank_card.b_type='credit card');
```

Perform a query by using the view.

```
select * from finance.v_client;
```

The result is as follows:

```
c_id | c_name |          c_id_card
------+--------+----------------------
     1 | Zuo Xiaoting | 340211199301010001
     3 | Zhu Changgang | 340211199301010003
     5 | An Yanfang | 340211199301010005
     7 | Fu Xiaofang | 340211199301010007
     9 | Shao Xiaoting | 340211199301010009
    10 | Zhang Xiaofeng | 340211199301010010
    12 | Feng Changqiang | 340211199301010012
```

14 | Zhao Xiaopeng | 340211199301010014
16 | Yin Gaofeng | 340211199301010016
18 | Tang Gaowei | 340211199301010018
(10 rows)

Step 3 Change the view name.

```
ALTER VIEW finance.v_client RENAME TO v_client_new;
```

Step 4 Delete the view.

Delete the **v_client** view. Deleting the view does not affect the base table.

```
DROP VIEW finance.v_client_new;
```

5.1.15 Index

Step 1 Create an index.

- Create an index in the **property** table.

```
CREATE INDEX finance.idx_property ON finance.property(pro_c_id DESC,pro_income,pro_purchase_time);
```

The result is as follows:
CREATE INDEX

Step 2 Run the meta-command to view the created index.

```
\di idx_property
```

The result is as follows:

```
                              List of relations
 Schema   |       Name       | Type  |   Owner    |   Table    | Storage
----------+------------------+-------+------------+------------+---------
 finance | idx_property | index | bank_app | property |
(1 row)
```

Step 3 Rename the index.

- Rebuild the index in the **property** table and rename the index.

Rebuild the index.

```
DROP INDEX finance.idx_property;
CREATE INDEX finance.idx_property ON finance.property(pro_c_id DESC,pro_income,pro_purchase_time);
```

Rename the index.

```
ALTER INDEX finance.idx_property RENAME TO idx_property_temp;
```

Step 4 Run the meta-command to view the created index.

```
\di idx_property_temp
```

The result is as follows:

```
                          List of relations
 Schema  |          Name          | Type  |  Owner   |  Table   | Storage
---------+------------------------+-------+----------+----------+---------
 finance | idx_property_temp | index | bank_app | property |
(1 row)
```

Step 5 Delete the index.

- Delete the **idx_property_temp** index.

```
DROP INDEX finance.idx_property_temp;
```

5.1.16 Updating and Deleting Data

Step 1 Update data.

- Modify or update the value of the **b_c_id** column in the **bank_card** table to a value less than 10 and ensure the value is the same as that of the **b_type** column in the **client** table.

View the table data.

```
SELECT * FROM finance.bank_card where b_c_id<10 ORDER BY b_c_id;
```

The result is as follows:

```
        b_number         |      b_type      | b_c_id
-------------------------+------------------+--------
 6222022302020000001     | Credit card      |      1
 6222022302020000016     | Debit card       |      3
 6222022302020000002     | Credit card      |      3
 6222022302020000003     | Credit card      |      5
 6222022302020000004     | Credit card      |      7
 6222022302020000013     | Debit card       |      7
 6222022302020000005     | Credit card      |      9
(7 rows)
```

Update the data.

```
UPDATE finance.bank_card SET bank_card.b_type='debit card' from finance.client where bank_card.b_c_id =
client.c_id and bank_card.b_c_id<10;
```

Query the data again.

```
SELECT * FROM finance.bank_card ORDER BY b_c_id;
```

The result is as follows:

```
        b_number         |      b_type      | b_c_id
-------------------------+------------------+--------
 6222022302020000001     |  Debit card      |      1
 6222022302020000002     | Debit card       |      3
 6222022302020000016     | Debit card       |      3
 6222022302020000003     | Debit card       |      5
 6222022302020000013     | Debit card       |      7
 6222022302020000004     | Debit card       |      7
 6222022302020000005     | Debit card       |      9
 6222022302020000006     | Credit card      |     10
 6222022302020000007     | Credit card      |     12
 6222022302020000019     | Debit card       |     12
 6222022302020000008     | Credit card      |     14
 6222022302020000009     | Credit card      |     16
 6222022302020000010     | Credit card      |     18
 6222022302020000011     | Debit card       |     19
 6222022302020000012     | Debit card       |     21
 6222022302020000014     | Debit card       |     23
 6222022302020000015     | Debit card       |     24
 6222022302020000017     | Debit card       |     26
 6222022302020000018     | Debit card       |     27
 6222022302020000020     | Debit card       |     29
```

Step 2 Delete the specified data.

- Delete the rows where IDs are less than 3 from the **fund** table.

Query the result before deletion.

```
SELECT * FROM finance.fund;
```

The result is as follows:

```
   f_name      | f_id |       f_type       | f_amount |       risk_level        | f_manager
---------------+------+--------------------+----------+--------------------------+-----------
 Stock         |    1 | Stock              |    10000 | High                     |         1
 Investment    |    2 | Bond               |    10000 | Medium                   |
2
 Government bond |    3 | Currency         |    10000 | Low
|         3
 CSI 300 index |    4 | Exponential        |    10000 | Medium                   |
4
```

Delete the data.

```
DELETE FROM finance.fund WHERE f_id<3;
```

Query the deletion result.

```
SELECT * FROM finance.fund;
```

The result is as follows:

```
    f_name       | f_id |         f_type          | f_amount |        risk_level         | f_manager
-------------+------+-------------------+----------+---------------------+-----------
 Government bond             |      3 | Currency                  |      10000 | Low
|               3
 CSI 300 index |       4 | Exponential                  |       10000 | Medium                          |
4
```

Step 3 Exit the database.

```
\q
```

5.1.17 Deleting a Schema and a Database

Before performing this operation, ensure that the system is brought offline and is not reserved. Do not perform this operation in the production environment.

Step 1 Log in to the **finance** database as the administrator.

Use the **gsql** tool to log in to the **finance** database and create a session.

```
gsql -d finance -p 26000 -r
```

Step 2 Run the **\dn** command to view the schema in the database.

```
\dn
```

The query result is as follows:

```
      List of schemas
      Name         |    Owner
-------------+----------
 bank_app       | bank_app
 cstore         | omm
 dbe_perf       | omm
 finance        | omm
 pkg_service | omm
 public         | omm
 snapshot       | omm
(7 rows)
```

Step 3 Run the **\dt** command to view the objects in the **finance** database.

```
\dt
```

The query result is as follows:

```
List of relations
 Schema    |          Name          | Type  | Owner |                 Storage
-----------+------------------------+-------+-------+---------------------------------
 finance | bank_card            | table | omm     | {orientation=row,compression=no}
 finance | client               | table | omm     | {orientation=row,compression=no}
 finance | finances_product | table | omm     | {orientation=row,compression=no}
 finance | fund                 | table | omm     | {orientation=row,compression=no}
 finance | insurance            | table | omm     | {orientation=row,compression=no}
 finance | property             | table | omm     | {orientation=row,compression=no}
(6 rows)
```

Step 4 Run the **DROP SCHEMA** command to delete the **finance** database. An error is reported because objects exist in the **finance** database.

```
DROP SCHEMA finance;
```

The following error is reported:

```
ERROR:  cannot drop schema finance because other objects depend on it
DETAIL:  table finance.client depends on schema finance
table finance.bank_card depends on schema finance
table finance.insurance depends on schema finance
table finance.fund depends on schema finance
table finance.property depends on schema finance
table finance.finances_product depends on schema finance
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

Step 5 Run the **DROP SCHEMA....CASCADE** command to delete the **finance** database, which will also delete the objects in the **finance** database.

```
DROP SCHEMA finance CASCADE;
```

The result is as follows:

```
NOTICE:  drop cascades to 6 other objects
DETAIL:  drop cascades to table client
drop cascades to table bank_card
drop cascades to table insurance
drop cascades to table fund
drop cascades to table property
drop cascades to table finances_product
DROP SCHEMA
```

Step 6 Run the **\dt** command to view the objects in **finance** and **public**.

```
\dt
```

The result is as follows:

No relations found.

Step 7 Delete the database.

DROP DATABASE finance;

The result is as follows:

DROP DATABASE

Step 8 Run the meta-command to view the database.

```
\l
```

The result shows that the **finance** database has been deleted.

```
                                  List of databases
    Name     | Owner | Encoding  | Collate | Ctype | Access privileges
-------------+-------+-----------+---------+-------+-------------------
 postgres    | omm   | SQL_ASCII | C       | C     |
 template0   | omm   | SQL_ASCII | C       | C     | =c/omm               +
             |       |           |         |       | omm=CTc/omm
 template1   | omm   | SQL_ASCII | C       | C     | =c/omm               +
             |       |           |         |       | omm=CTc/omm
(3 rows)
```

Step 9 Exit the database.

```
\q
```

5.2 Summary

This experiment uses the ER model to deepen the understanding of the database and database design, and allows you to practice using SQL statements to master the SQL syntax.

5.3 Quiz

Question 1: What are the functions of a comment? Is it necessary?

Answer: A comment is used to describe a table or a column. It is not mandatory to set a comment, but you are advised to set a comment for a table or a column to facilitate O&M management.

Question 2: What is the difference between the **WHERE** clause and the **HAVING** clause in a query?

Aggregate functions cannot be used in the **WHERE** clause but can be used in the **HAVING** clause.

Question 3: Why are many indexes automatically created when an index is viewed?

When a table is created, the primary key is specified. Therefore, the corresponding primary key index is automatically created.

6 Appendix 1: Commands Related to the Linux OS

In Linux, the command format is **command [options] [arguments]**. The brackets indicate that some commands do not require options or arguments, while some commands require multiple options or arguments.

- **options** specifies whether to enable the function of adjusting command execution. The command output varies according to the value of **options**.
- **arguments** specifies the object on which a command is executed.

6.1 Vi/Vim

Vi and Vim are text editors. If a file exists, the text editors are used for editing. If no file exists, the text editors are used for creating and editing text.

Syntax:

```
vim [arguments]
```

Argument description: Specifies the name of a file that can be edited.

Command example:

- Command for editing the **clusterconfig** XML file:

```
vim clusterconfig.xml
```

Note:

The vim editor has the following modes:

- Normal mode: To switch to the normal mode from the other modes, press **Esc** or **Ctrl**+**[**. The file name is displayed in the lower left corner or is empty.
- Insert mode: In normal mode, press **i** to enter the insert mode. **--INSERT--** is displayed in the lower left corner.
- Visible mode: In normal mode, press **v** to enter the visible mode. **--VISUAL--** is displayed in the lower left corner.

Run the following commands to exit the normal mode:

- Run the **:wq!** command to save the settings and exit.
- Run the **:q!** command to forcibly exit and ignore all modifications.
- Run the **:e!** command to cancel all modifications and open the original file.

6.2 cd

cd is used to display the name of the current directory or to switch from the current directory to a specified directory.

Syntax:

```
cd [arguments]
```

Argument description:

- If no argument is specified, the current directory is switched.
- . The value **.** indicates the current directory.
- The value **..** indicates the upper-level directory.
- The value **~** indicates the home directory.
- The value **/** indicates the root directory.

Command example:

- Command for switching to the **bin** directory in the **usr** directory:

```
cd /usr/bin
```

- Command for switching to the home directory of the user:

```
cd
```

- Command (followed by a dot) for switching to the current directory:

```
cd .
```

- Command (followed by two dots) for switching to the upper-level directory of the current directory:

```
cd ..
```

- Command for switching to the home directory of the user:

```
cd ~
```

- Command for switching to the root directory:

```
cd /
```

Note: To switch the directory, you need to understand the absolute path and relative path.

- Absolute path: In Linux, an absolute path starts with a slash (/) (root directory), for example, **/opt/software** or **/etc/profile**. If a directory starts with a slash (/), it is an absolute directory.
- Relative path: directory that starts with a dot (.) or two dots (..). A dot (.) indicates the location of the current operation, and two dots (..) indicates the upper-level directory. For example, **./gs_om** indicates a file or directory in the current directory.

6.3 mv

mv is used to rename a file or directory or move a file or directory to another location to implement file or directory backup.

Syntax:

```
mv [options] argument 1 argument 2
```

Common options:

- **-b**: Backs up a file before overwriting it.

Argument description:

- *Argument 1*: Specifies the source file or directory.
- *Argument 2*: Specifies the target file or directory.

Command example:

- Command for renaming the **python** file to **python.bak**:

```
mv python python.bak
```

- Command for moving all the files and directories in the **/physical/backup** directory to the **/data/dbn1** directory:

```
mv    /physical/backup/*    /data/dbn1
```

6.4 cURL

In Linux, cURL is a file transfer tool that uses Uniform Resource Locator (URL) rules to work on the CLI. It supports file upload and download.

Syntax:

```
curl [options] [URL]
```

Common options:

- **-A/--user-agent <string>**: Sets the user agent to be sent to the server.
- **-C/--continue-at <offset>**: Resumes transmission when a disconnection occurs.
- **-D/--dump-header <file>**: Writes the header information to the file.
- **-e/--referer**: Specifies the source website.
- **-o/--output**: Writes the output to the file.
- **-O/--remote-name**: Writes the output to the file and retains the name of the remote file.
- **-s/--silent**: Specifies the silent mode. No information is output.
- **-T/--upload-file <file>**: Uploads the file.
- **-u/--user <user[:password]>**: Sets the username and password of the server.
- **-x/--proxy <host[:port]>**: Specifies that the HTTP proxy is used on the specified port.
- **-#/--progress-bar**: Displays the current transmission status.

Argument description:

- **URL**: Specifies the file transfer URL.

Command example:

- Command for saving the content of https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo) to the **/etc/yum.repos.d/CentOS-Base.repo** file:

```
curl -o /etc/yum.repos.d/CentOS-Base.repo https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo
```

- Command for resuming transmission through the -C mode when a disconnection occurs during the transmission:

```
curl -C -O https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo
```

6.5 Yum

Yellowdog Update, Modified (Yum) is a software package manager with an interactive shell. Based on the RedHat package manager (RPM), Yum can automatically download RPM packages from a specified server and install them, automatically process dependencies, and install all RPM packages at a time.

Syntax:

```
yum [options] [command] [package ...]
```

Common options:

- **-h**: Views the help information.
- **-y**: Selects **yes** for all items when a prompt is received during the installation.
- **-q**: Does not display the installation process.

Argument description:

- **command**: Specifies the operation to be performed.
- **package**: Specifies the name of the package to be installed.

Command example:

- Command for listing all software that can be updated:

```
yum check-update
```

- Command for updating all software:

```
yum update
```

- Command for listing all software that can be installed:

```
yum list
```

- Command for installing the specified software:

```
yum install -y libaio-devel flex bison ncurses-devel glibc.devel patch lsb_release wget python3
```

6.6 wget

wget is the most commonly used command for downloading files in Linux. **wget** supports the HTTP, HTTPS, and FTP protocols as well as automatic download. That is, **wget** can be executed in the background after a user logs out of the system until the download is complete.

Syntax:

```
wget [options] [URL]
```

Common options:

- **-c**: Downloads the file that is not completely downloaded.
- **-b**: Switches to background execution after startup.
- **-P**: Specifies the downloaded directory.
- **-O**: Changes the name of the downloaded file.
- **--ftp-user --ftp-password**: Uses FTP authentication for download.

Argument description:

- Specifies the URL for downloading a specified file.

Command example:

- Command for downloading the installation file of openGauss to the current folder:

```
wget https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.1.0/x86/openGauss-1.1.0-CentOS-64bit.tar.gz
```

- Command for resuming transmission by using the **wget** command:

```
wget –c https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.1.0/x86/openGauss-1.1.0-CentOS-64bit.tar.gz
```

6.7 ln

ln is used to create a link for synchronization (soft or hard link) for a file in another location. If no option is specified, the link is a hard link.

When the same file is required in different directories, you do not need to place the same file in each required directory. Instead, you only need to place the file in a fixed directory and then run the **ln** command to create a link to the file in other directories to save disk space.

Syntax:

```
ln [options] argument 1 argument 2
```

Common options:

- **-b**: Deletes and overwrites the existing link.
- **-d**: Allows the superuser to create hard links to directories.
- **-s**: Specifies a soft link (symbolic link).

Argument description:

- *Argument 1*: Specifies the source file or directory.
- *Argument 2*: Specifies the file or directory to which a link is created.

Command example:

- Command for creating the soft link **/usr/bin/python** for the **python3** file: (If the **python3** file is lost, **/usr/bin/python** becomes invalid.)

```
ln -s python3 /usr/bin/python
```

- Command for creating the hard link **/usr/bin/python** for the **python3** file: (The attributes of **python3** are the same as those of **/usr/bin/python**.)

```
ln python3 /usr/bin/python
```

6.8 mkdir

mkdir is used to create a specified directory. The user who creates the directory must have the write permission on the current directory, and the specified directory cannot be an existing directory in the current directory.

Syntax:

```
mkdir [options] [arguments]
```

Common options:

- **-p**: Specifies a path. If some directories in the path do not exist, with this option specified, the system automatically creates the directories that do not exist. That is, multiple directories can be created at a time recursively.
- **-v**: Displays information each time when a directory is created.
- **-m**: Sets permissions (similar to **chmod**).

Argument description:

- Specifies the directory to be created.

Command example:

- Command for creating a directory:

```
mkdir test
```

- Command for creating multiple directories recursively:

```
mkdir -p /opt/software/openGauss
```

- Command for creating a directory on which the permission is **777** (the permission is **rwxrwxrwx**):

```
mkdir –m 777 test
```

6.9 chmod

chmod is used to change the permission on a file.

Syntax:

```
chmod [options] <mode> <file...>
```

Common options:

- **-R**: Recursively changes the same permission on all files and subdirectories in a specified directory.

Argument description:

- **mode**: Sets strings for permissions. The detailed format is as follows:

```
[ugoa...][[+-=][rwxX]...][,...],
```

In **[ugoa...]**, **u** indicates the owner of a file, **g** indicates those who belong to the same group as the owner of the file, **o** indicates those who do not belong to the same group as the owner of the file, and **a** indicates all (including the preceding three kinds of people). In **[+-=]**, **+** indicates that permissions are added, **-** indicates that permissions are canceled, and **=** indicates that a unique permission is set. In **[rwxX]**, **r** indicates that the file can be read, **w** indicates that the file can be written, **x** indicates that the file can be executed, and **X** indicates that the file can be executed only when the file is a subdirectory or the file has been set to be executable.

- **file**: Specifies the file list (containing one or more files or folders).

Command example:

- Command for setting the **cluterconfig.xml** file to be readable for all users:

```
chmod ugo+r cluterconfig.xml
Or
chmod a+r    cluterconfig.xml
```

- Command for setting all the files and subdirectories in the current directory to be readable and writable to anyone:

```
chmod -R a+rw *
```

The format of digital permissions is as follows:

- In this format, digits 4, 2, and 1 indicate the read, write, and execute permissions, that is, r = 4, w = 2, and x = 1.
- Example: rwx = 7 (4 + 2 + 1), rw = 6 (4 + 2), r-x = 5 (4 + 0 + 1), r-- = 4 (4 + 0 + 0), --x = 1 (0 + 0 + 1)

You can set different read, write, and execute (specified by **r**, **w**, and **x**) permissions on each file based on the three granularities. That is, you can use three octal digits to indicate the permission details of the owner, group, and other groups (specified by **u**, **g**, and **o**), and run the **chmod** command to add the three octal digits to change the file permission. The syntax is as follows:

```
chmod <abc> file...
```

In the preceding information, **a**, **b**, and **c** indicate the permissions of the owner, group, and other groups respectively, which is similar to the simplified **chmod u = *permission*, g = *permission*, o = *permission file*...**. The permissions here are represented by octal numbers, indicating the read, write, and execute permissions of the owner, group, and other groups.

Command example:

- Command for granting the read, write, and execute permissions (all permissions) on the **cluterconfig.xml** file:

```
chmod 777 cluterconfig.xml
```

- Command for granting the read and execute permissions on all files and subdirectories in the **/opt/software/openGauss** directory to both the user group and other users:

```
chmod    R 755 /opt/software/openGauss
```

6.10 chown

chown is used to change the owner of a specified file to a particular user or group. The user can be a username or user ID, and the group can be a group name or group ID. The file lists the files whose permissions need to be changed. The file names are separated by spaces. Wildcards (*) are supported. Only the system administrator (the **root** user) has the permission to run this command. Permission: **root**

Syntax:

```
chown [options] user[:group] file...
```

Common options:

- **-c**: Displays the modified information.
- **-f**: Ignores error information.
- **-R**: Processes all files in a specified directory and its subdirectories.

Argument description:

- **user**: ID of the new file owner
- **group**: group to which the new file owner belongs
- **file**: file

Command example:

- Command for setting the owner of the **file1.txt** file to **omm** and the owner group to **dbgrp**:

```
chown omm:dbgrp /opt/software/openGauss/clusterconfig.xml
```

- Command for setting the owner of all files and subdirectories in the current directory to **omm** and the owner user to **dbgrp**:

```
chown -R omm:dbgrp *
```

6.11 ls

ls is used to list the content of files and directories.

Syntax:

```
ls [options] [arguments]
```

Common options:

- **-l**: Displays the detailed information about files, including the creators, creation time, as well as read and write permissions in long format.
- **-a**: Lists all files in a file, including hidden files whose names start with a dot (.) or two dots (..). (The names of hidden files in Linux start with a dot. A file whose name starts with two dots is a parent directory.)
- **-d**: Lists only the directory instead of files in the directory. It is usually used together with **-l**.
- **-R**: Lists all subdirectories at the same time. It works similar to **-l**. The only difference is that it does not display the file owners, which is similar to recursion in programming.
- **-t**: Sorts files by time.
- **-s**: Displays the file size after each file.
- **-S**: Sorts files by file size.

Argument description:

- Specifies a directory or file.

Command example:

- Command for listing the files and directories in the current directory in long format:

```
ls -l
```

6.12 cp

cp is used to copy a file or directory.

Syntax:

```
cp [options] argument 1 argument 2
```

Common options:

- **-f**: Removes the target file and tries again if it cannot be opened. (If the **-n** option exists, you do not need to select this option.)
- **-n**: Does not overwrite the existing file (to disable the **-i** option).
- **-I**: Performs a query before overwriting (to disable the **-n** option).
- **-p**: Retains the specified attributes (default attributes: mode, ownership, and timestamp) and additional attributes such as environment, link, and xattr, if possible.
- **-R/-r**: Copies a directory and all items in the directory.

Argument description:

- *Argument 1*: Specifies the source file.
- *Argument 2*: Specifies the target file.

Command example:

- Command for copying the **abc** file from the home directory to the **opt** directory:

```
cp /home/abc /opt
```

Note: If the target file exists, the system asks you whether to overwrite it. This is because **cp** is the alias of **cp -i**. If the target file exists, the system asks you whether to overwrite the file even if the **-f** flag is added.

6.13 rm

rm is used to delete one or more files or directories from a directory. It can also delete a directory and all its files and subdirectories. For a link file, only the link is deleted but the original file remains unchanged.

Exercise caution when using the **rm** command because it is a dangerous command, and the entire system may be ruined by this command. (For example, **rm * -rf** is performed under the root directory **/**.) Therefore, before running the **rm** command, you are advised to determine the directory from which the items are to be deleted and keep a clear mind during the operation.

Syntax:

```
rm [options] file
```

Common options:

- **-f**: Ignores the files that do not exist and does not display any message.
- **-r**: Instructs **rm** to recursively delete all directories and subdirectories listed in the argument.

Argument description:

- Specifies the file or directory to be deleted.

Command example:

- Command for deleting a file:

```
rm qwe
```

Note: After you run the **rm qwe** command, the system asks you whether to delete the file. If you enter **y**, the file is deleted. If you do not want to delete the file, enter **n**.

- Command for forcibly deleting a file:

```
rm-rf clusterconfig.log
```

6.14 cat

cat is used to connect to a file and export the file through the standard export. This command is often used to display the content of a file, connect several files and display them, or read content from the standard input and display it. It is often used together with a redirection symbol.

Syntax:

```
cat [options] [arguments]
```

Common options:

- **-E**: Displays **$** at the end of each line.
- **-n**: Numbers all lines starting from 1.

- **-b/--number-nonblank**: Functions similarly as **-n**. The difference lies in that this option does not number empty lines.
- **-v**: Uses ^ and M- symbols, except LFD and TAB.

Argument description:

- Specifies the name of the file to be operated.

Command example:

- Command for displaying the content of the **testfile** file:

```
cat textfile
```

- Command for adding the content of the **textfile1** and **textfile2** files to the **textfile3** file after adding line numbers to the **textfile1** and **textfile2** files.

```
cat -b textfile1 textfile2 >> textfile3
```

- Command for adding content to the **/etc/profile** file (you can enter **EOF** to end the addition):

```
cat >>/etc/profile<<EOF
>export LD_LIBRARY_PATH=$packagePath/script/gspylib/clib:$LD_LIBRARY_PATH
>EOF
```

Note:

- EOF is short for end of file, indicating the end of the text stream. The text stream can be a file · or standard input. In Linux, EOF indicates that a signal value (– **1**) is returned when the system reads the end of a file.

7 Appendix 2: Basic Operations on openGauss

7.1 Viewing Database Objects

- View help information.

```
postgres=# \?
```

- Switch the database.

```
postgres=# \c dbname
```

- List the databases.

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

```
postgres=# \l
```

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

```
postgres=# SELECT datname FROM pg_database;
```

- List tables.

```
postgres=# \dt
```

- List all tables, views, and indexes.

```
postgres=# \d+
```

Run the **\d+** command of the **gsql** tool to query table attributes.

```
postgres=# \d+ tablename
```

- View the table structure.

```
postgres=# \d tablename
```

- List schemas.

```
postgres=# \dn
```

- View indexes.

```
postgres=# \di
```

- Query tablespaces.

Run the meta-command of the **gsql** program to query the tablespaces.

```
postgres=# \db
```

Check the **pg_tablespace** system catalog. Run the following command to view all the tablespaces defined by the system and users:

```
postgres=# SELECT spcname FROM pg_tablespace;
```

- Query the database user list.

```
postgres=# SELECT * FROM pg_user;
```

- Query user attributes.

```
postgres=# SELECT * FROM pg_authid;
```

- Query all roles.

```
postgres=# SELECT * FROM PG_ROLES;
```

7.2 Other Operations

- Query all SQL statements supported by openGauss.

```
postgres=#\h
```

- Switch the database.

```
postgres=# \c dbname
```

- Switch the user.

```
postgres=# \c – username
```

- Exit the database.

```
postgres=# \q
```