

华为认证 HCIA 系列教程

HCIA-openGauss

实验指导手册

版本：1.0



华为技术有限公司

版权所有 © 华为技术有限公司 2021。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系，包含ICT技术架构与应用认证、云服务与平台认证两类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIA-openGauss定位于openGauss数据库的工程师级别认证，适用于数据库开发工程师以及致力于向数据库领域发展的人员的技能提升。该认证提供了丰富的随堂实验和行业案例，旨在提升学员的实践能力，促进数据库人才的培养。

华为认证协助您打开行业之窗，开启改变之门，屹立在WLAN网络世界的潮头浪尖！



前言

简介

本书为 HCIA-openGauss 认证培训教程，适用于准备参加 HCIA-openGauss 考试的学员或者希望了解 openGauss 和 GaussDB(for openGauss)的基础知识、SQL 语法分类及使用场景等相关 GaussDB 数据库技术的读者。

内容描述

本实验指导书共包含 5 个部分，从 openGauss 的安装部署开始，逐一介绍了数据库及对象管理，openGauss 数据库中的 SQL 语法，GaussDB(for openGauss)云数据库，场景化综合实验。

- 实验一为 openGauss 数据库环境搭建及操作实验，通过华为云上购买 ECS 部署 openGauss 数据库，帮助读者熟悉掌握 openGauss 数据库部署和数据库连接的基本操作。
- 实验二为 openGauss 数据库及对象管理实验，通过对 openGauss 数据库中的表空间、数据库、对象、用户等进行管理操作，掌握 openGauss 数据库相关的对象管理能力。
- 实验三为 SQL 语法基础，通过 DDL\DML 等实验，掌握 SQL 语法的基础能力。
- 实验四为云数据库 GaussDB(for openGauss)实验，通过华为云购买使用云数据库 GaussDB(for openGauss)，并使用 DAS 服务连接数据库，完成数据库上的增删改查功能。
- 实验五为综合实验，以金融场景为例，完成 openGauss 数据库的相关操作，包括对象管理、连接管理、用户管理、SQL 语句的掌握、索引视图等，旨在通过场景实践，掌握 openGauss 数据库。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的计算机知识背景，同时熟悉华为云界面，了解基本 Linux 知识。

实验环境说明

组网说明

本实验环境面向准备 HCIA-openGauss 考试的数据库工程师。每套实验环境包括 GaussDB(for openGauss)服务一个，数据库管理服务一个，弹性公网 IP 一个，ECS 服务器一台。

设备介绍

为了满足 HCIA-openGauss 实验需要，建议每套实验环境采用以下配置：

设备名称、型号与版本的对应关系如下：

表1-1 对应关系

设备名称	设备型号	软件版本
GaussDB数据库	GaussDB(for openGauss)	1.4版本
数据库管理服务	DAS服务	-
弹性公网服务	EIP	-
ECS	4vCPUs 8GiB弹性云服务器	-
openGauss数据库	openGauss数据库	2.0.0

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	4
1 openGauss 数据库安装与操作	9
1.1 实验介绍	9
1.1.1 关于本实验	9
1.1.2 实验目的	9
1.2 数据库安装	9
1.2.1 购买弹性云服务器 ECS (openEuler ARM 操作系统)	9
1.2.2 修改操作系统配置	14
1.2.3 安装 openGauss 数据库	16
1.3 gsql 连接数据库	24
1.3.1 gsql 连接数据库	24
1.4 数据库工具使用	29
1.4.1 前提条件	29
1.4.2 基本操作步骤	29
1.4.3 工具使用操作步骤	32
2 数据库及对象管理	43
2.1 实验介绍	43
2.1.1 关于本实验	43
2.1.2 实验目的	43
2.2 实验内容	44
2.2.1 登录数据库	44
2.2.2 表空间的创建、查询、修改与删除	44
2.2.3 数据库的创建、查看、修改与删除	46
2.2.4 行存表、列存表、MOT 表的创建、查看、修改与删除	47
2.2.5 用户、角色及 schema 管理	50
2.2.6 数据导入导出	55
2.3 清理实验环境	64

2.4 实验小结	65
2.5 思考题	65
3 SQL 语法基础实验.....	66
3.1 实验介绍	66
3.1.1 关于本实验	66
3.1.2 实验目的.....	66
3.1.3 内容描述.....	66
3.2 数据准备	66
3.2.1 实验任务.....	66
3.3 数据查询	69
3.3.1 实验任务.....	70
3.4 数据更新	73
3.4.1 实验任务.....	73
3.5 数据定义	74
3.5.1 实验任务.....	74
3.6 常用函数和操作符	81
3.6.1 实验任务.....	81
3.7 实验小结	84
3.8 思考题	84
4 云数据库 GaussDB(for openGauss)	86
4.1 实验介绍	86
4.1.1 关于本实验	86
4.1.2 实验目的.....	86
4.2 购买 GaussDB (for openGauss)数据库实例	86
4.2.1 登录华为云官网	86
4.2.2 购买数据库实例	88
4.3 DAS 使用	90
4.3.1 DAS 连接数据库.....	90
4.3.2 创建数据库	91
4.3.3 创建 schema	92
4.3.4 创建表	93
4.3.5 插入数据.....	94
4.3.6 删除数据.....	94
4.3.7 修改数据.....	95
4.4 实验小结	96

4.5 思考题	96
5 综合实验	97
5.1 金融数据模型	97
5.1.1 E-R 图	97
5.1.2 关系模式	98
5.1.3 物理模型	99
5.1.4 连接数据库设置	101
5.1.5 创建数据库及 schema	103
5.1.6 创建、设置表空间	104
5.1.7 新用户的创建和授权	105
5.1.8 新用户连接数据库	106
5.1.9 创建数据表	106
5.1.10 插入表数据	109
5.1.11 手工插入一条数据	114
5.1.12 添加约束	114
5.1.13 查询数据	115
5.1.14 视图	121
5.1.15 索引	123
5.1.16 数据的修改和删除	124
5.1.17 删除 Schema 和删除 Database	126
5.2 实验小结	128
5.3 思考题	128
6 附录一：Linux 操作系统相关命令	129
6.1 vi/vim	129
6.2 cd	129
6.3 mv	130
6.4 curl	131
6.5 yum	132
6.6 wget	132
6.7 ln	133
6.8 mkdir	134
6.9 chmod 命令	134
6.10 chown	135
6.11 ls	136
6.12 cp	137
6.13 rm	137

6.14 cat.....	138
7 附录二：openGauss 数据库基本操作.....	139
7.1 查看数据库对象.....	139
7.2 其他操作.....	140

1 openGauss 数据库安装与操作

1.1 实验介绍

1.1.1 关于本实验

本实验主要描述 openGauss 数据库在 openEuler 弹性云服务器上的安装部署。

1.1.2 实验目的

- 了解 openGauss 数据库部署方式；
- 掌握 openGauss 数据库安装部署方法。

1.2 数据库安装

1.2.1 购买弹性云服务器 ECS（openEuler ARM 操作系统）

1.2.1.1 登录华为云

步骤 1 进入华为云官网。

华为云官网：<https://www.huaweicloud.com/>，进入华为云官网，点击登录。



步骤 2 输入账号名和密码，点击登录。

账号登录

账号名/邮箱

密码

手机号登录

记住登录名

登录

[免费注册](#)
[忘记密码](#)
[IAM用户登录](#)

使用其他账号登录

如果还没有注册，点击免费注册，按步骤进行注册后进行登录。

1.2.1.2 购买弹性云服务器 ECS

步骤 1 在华为云主页(<https://www.huaweicloud.com/>)点击产品，选择基础服务，再选择弹性云服务器 ECS。



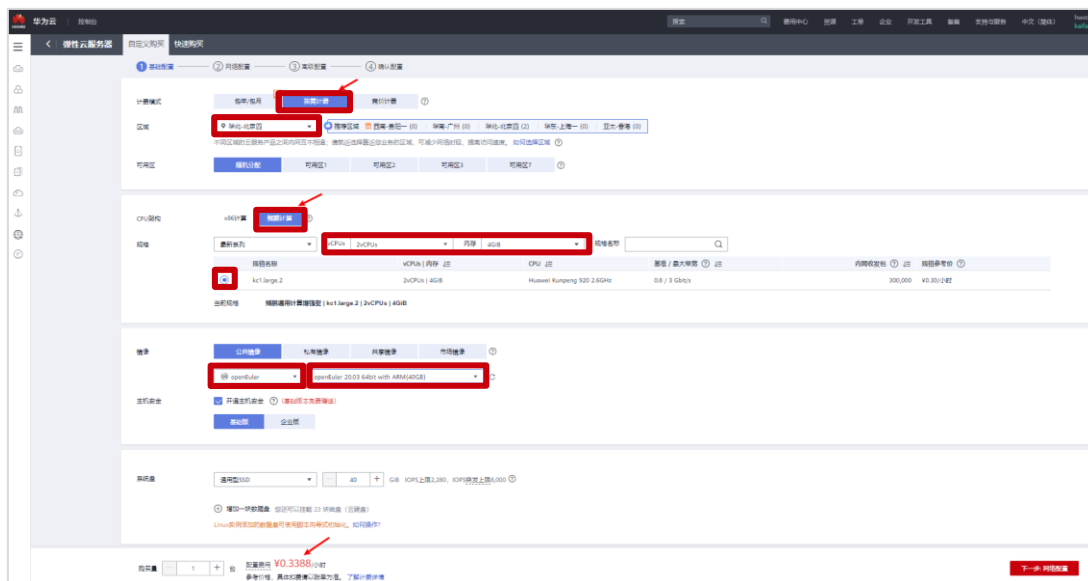
步骤 2 进入弹性云服务器 ECS 购买界面。



步骤 3 自定义购买进行基础配置。

表1-1 ECS 基础配置

配置选项	配置值
计费模式	按需计费（一定要选按需计费，注意配置费用）
区域	华北-北京四（推荐，其他区域可能会导致无法选择openEuler公共镜像）
CPU架构	鲲鹏计算
规格	最新系列 2vCPU5 4GB
镜像	公共镜像： openEuler openEuler 20.03 64bit with ARM(40GB)



其余默认即可，点击下一步网络配置。

步骤 4 自定义购买进行网络配置。

表1-2 ECS 网络配置

配置选项	配置值
网络	Vpc-default (192.168.0.0/16) (选现有默认网络即可)
弹性公网IP	现在购买
公网带宽	按流量计费
带宽大小	5

The screenshot shows the 'VPC Configuration' page in the Huawei Cloud console. The 'VPC' section shows a selected VPC with CIDR block 192.168.0.0/24. The 'Security Group' section shows a selected security group with a list of rules. The 'Network Interface' section shows a selected network interface. The 'Pricing' section at the bottom shows the total cost of the configuration.

安全组名称	优先级	策略	协议端口	类型	源地址	描述
1	允许	TCP	8022	IPv4	119.3.115.16/32	--
1	允许	TCP	80	IPv4	0.0.0.0/0	Used to access websites over HTTP
1	允许	TCP	443	IPv4	0.0.0.0/0	Used to access websites over HTTPS
1	允许	TCP	20-21	IPv4	0.0.0.0/0	Used to upload and download files
1	允许	ICMP	全部	IPv4	0.0.0.0/0	Used to test the ECS connectivity with the ping command.

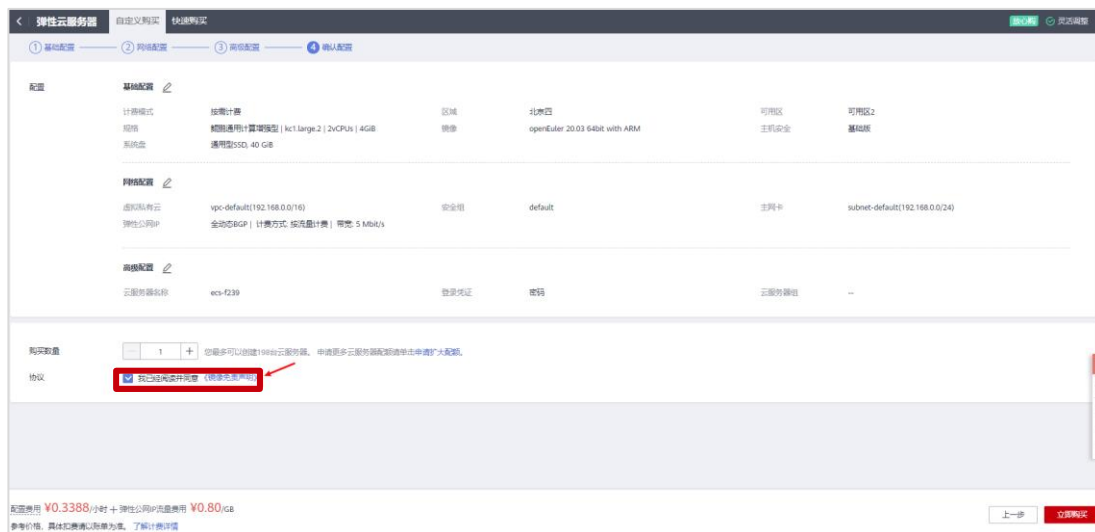
其余默认即可，点击下一步高级配置。

步骤 5 自定义购买进行高级配置。

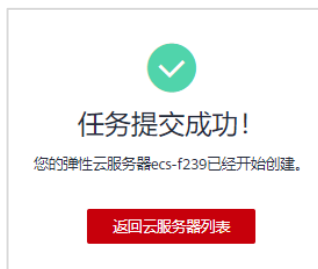
The screenshot shows the 'VPC Configuration' page in the Huawei Cloud console. The 'VPC' section shows a selected VPC with CIDR block 192.168.0.0/24. The 'Security Group' section shows a selected security group with a list of rules. The 'Network Interface' section shows a selected network interface. The 'Pricing' section at the bottom shows the total cost of the configuration.

记住用户名为 root，然后输入自定义密码和确认密码，其余默认即可，点击下一步确认设置。

步骤 6 确认配置购买成功。



确认设置信息，尤其是配置费用，然后勾选协议“我已经阅读并同意《华为镜像免责声明》”，点击立即购买。



查看云服务器列表

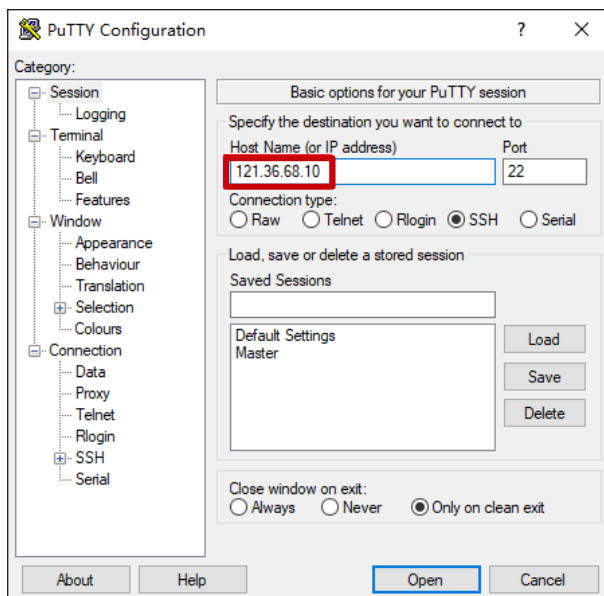


购买成功！

注意：本次购买鲲鹏服务器价格为公测价格，具体价格以华为云官网为准。

1.2.2 修改操作系统配置

为了操作方便，可以使用 SSH 工具（比如：Putty 等）从本地电脑通过配置弹性云服务器的弹性公网 IP 地址（如:121.36.68.10）来连接 ECS，并使用 ROOT 用户来登录。



1.2.2.1 设置字符集参数

将各数据库节点的字符集设置为相同的字符集，可以在/etc/profile 文件中添加"export LANG=XXX"（XXX 为 Unicode 编码）。

步骤 1 在/etc/profile 文件中添加"export LANG= en_US.UTF-8"。

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

步骤 2 输入如下命令，使修改生效。

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

1.2.2.2 修改 python 版本并安装 libaio 包

之前安装过程中 openGauss 用户互信，openEuler 服务器需要用到 Python-3.7.x 命令，但是默认 Python 版本为 Python-2.7.x，所以需要切换 Python 版本。

步骤 1 进入/usr/bin 文件，备份 python 文件。

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

备份 python 文件。

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

步骤 2 建立 python3 软连接。

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

步骤 3 验证 python 版本。

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

显示如下，说明切换成功：

```
Python 3.7.4
```

步骤 4 Python 版本切换成功，后续安装需要 libaio 包，下载进行安装。

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

1.2.3 安装 openGauss 数据库

1.2.3.1 下载数据库安装包

步骤 1 以 root 用户登录待安装 openGauss 的主机，并按规划创建存放安装包的目录。

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

注：不建议把安装包的存放目录规划到 openGauss 用户的家目录或其子目录下，可能导致权限问题。openGauss 用户须具有/opt/software/openGauss 目录的读写权限。

步骤 2 使用 wget 下载数据库安装包到安装包目录。

切换到安装目录：

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

使用 wget 下载安装包：

```
[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
```

注：https://openGauss.obs.cn-south-1.myhuaweicloud.com/2.0.0/arm/openGauss-2.0.0-openEuler-64bit-all.tar.gz 是数据库安装包下载网址，输入时不需要进行换行。

下载成功显示如下：

```
.....
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 创建 XML 配置文件

- 安装 openGauss 前需要创建 XML 文件。XML 文件包含部署 openGauss 的服务器信息、安装路径、IP 地址以及端口号等。用于告知 openGauss 如何部署。用户需根据不同场合配置对应的 XML 文件。
- 以单节点配置的方案为例，说明如何创建 XML 配置文件。

步骤 1 以 root 用户登录待安装 openGauss 的主机，切换到存放安装包的目录。

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

步骤 2 创建 XML 配置文件，用于数据库安装。

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

步骤 3 输入” i” 进入 INSERT 模式，添加文本如下，加粗字体内容为示例，可自行替换。其中“**ecs-f239**”是弹性云服务器的名称，“**192.168.0.193**”为弹性云服务器的 IP 地址（私有），其他 value 值可以不进行修改。

```
<?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/wisquery" />
    <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>
```

弹性云服务器名称及私有 IP 查看：

☐ 名称/ID	监控	可用区 ▼	状态 ▼	规格/镜像	IP地址
☐ ecs-f239 b88e1a26-46f1-4cde-8d89-c37463f0c...		可用区2	 运行中	2vCPUs 4GiB kc1.large.2 openEuler 20.03 64bit with ARM	121.36.68.10 (弹性公网...) 192.168.0.193 (私有)

步骤 4 点击“Esc”退出 INSERT 模式，然后输入“:wq”后回车退出编辑并保存文本。

表1-3 配置文件参数附加说明

实例类型	参数	说明
整理信息	clusterName	openGauss名称。

	nodeNames	openGauss中主机名称。
	backIp1s	主机在后端存储网络中的IP地址（内网IP）。所有openGauss主机使用后端存储网络通讯。
	gaussdbAppPath	openGauss程序安装目录。此目录应满足如下要求： - 磁盘空间>1GB - 与数据库所需其它路径相互独立，没有包含关系。
	gaussdbLogPath	openGauss运行日志和操作日志存储目录。此目录应满足如下要求： - 磁盘空间建议根据主机上的数据库节点数规划。数据库节点预留1GB空间的基础上，再适当预留冗余空间。 - 与openGauss所需其它路径相互独立，没有包含关系。 此路径可选。不指定的情况下，openGauss安装时会默认指定“\$GAUSSLOG/安装用户名”作为日志目录。
	tmpdbPath	数据库临时文件存放目录。 若不配置tmpdbPath，默认存放在/opt/huawei/wisquery/perfadm_db目录下。
	gaussdbToolPath	openGauss系统工具目录，主要用于存放互信工具等。此目录应满足如下要求： - 磁盘空间>100MB - 固定目录，与数据库所需其它目录相互独立，没有包含关系。 此目录可选。不指定的情况下，openGauss安装时会默认指定“/opt/huawei/wisquery”作为数据库系统工具目录。
	corePath	openGauss core文件的指定目录。

须知：

- “/opt/huawei/newsq1/tools” 存放互信等工具，避免权限问题，不要把实例数据目录放在此目录下。
- 安装目录和数据目录需为空或者不存在，否则可能导致安装失败。

- 在对数据库节点的实例进行具体配置时，需确保配置的目录之间不相互耦合。即各个配置目录不关联，删除其中任意一个目录，不会级联删除其它目录。如 gaussdbAppPath 为 "/opt/gaussdb/app"，gaussdbLogPath 为 "/opt/gaussdb/app/omm"。当 gaussdbAppPath 目录被删除时，会级联删除 gaussdbLogPath 目录，从而引起其它问题。
- 若需要自动创建安装用户时，各配置的目录需保证不与系统创建的默认用户目录耦合关联。
- 配置 openGauss 路径和实例路径时，路径中不能包含 "|", ";", "&", "\$", "<", ">", "`", "\\", "'", "\"", "{", "}", "(", ")", "[", "]", "~", "*", "?" 特殊字符。

1.2.3.3 初始化安装环境

为了保证 openGauss 的正确安装，请首先对主机环境进行配置。

1.2.3.3.1 准备安装用户及环境

创建完 openGauss 配置文件后，在执行安装前，为了后续能以最小权限进行安装及 openGauss 管理操作，保证系统安全性，需要运行安装前置脚本 gs_preinstall 准备好安装用户及环境。

1.2.3.3.2 前提条件

已完成安装准备的所有任务。

1.2.3.3.3 注意事项

- 用户需要检查上层目录权限，保证安装用户对安装包和配置文件目录读写执行的权限；
- xml 文件中主机的名称与 IP 映射配置正确；
- 只能使用 root 用户执行 gs_preinstall 命令。

1.2.3.3.4 操作步骤

步骤 1 修改 performance.sh 文件

使用 vi 打开文件 "/etc/profile.d/performance.sh"，具体如下：

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

输入 "i"，进入 INSERT 模式。用 # 注释 sysctl -w vm.min_free_kbytes=112640 &> /dev/null 这行。

```
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
```

点击 "Esc" 退出 INSERT 模式。输入 ":wq" 后回车，保存退出。

步骤 2 为确保 openssl 版本正确，执行预安装前加载安装包中 lib 库。

执行命令如下，其中 packagePath 为用户安装包放置的路径，本示例中为 /opt/software/openGauss。

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

输入 i，进入 INSERT 模式，在文件的底部添加如下代码，加载安装包中 lib 库。按下 “Esc” 退出 INSERT 模式，输入 “:wq” 后回车，保存后退出。

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

配置完成后，输入如下命令，使设置生效。

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

步骤 3 在安装包所在的目录下，解压安装包。

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

解压缩安装包：

先解压 openGauss-2.0.0-openEuler-64bit-all.tar.gz 包。

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

再解压 openGauss-2.0.0-openEuler-64bit-om.tar.gz 包。

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

解压后如下，用 ls 命令查看如下：

```
[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
```

安装包解压后，会在 /opt/software/openGauss 路径下自动生成 script 子目录，并且在 script 目录下生成 gs_preinstall 等各种 OM 工具脚本。

步骤 4 使用 gs_preinstall 准备好安装环境，切换到 gs_preinstall 命令所在目录。

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

script 中内容显示如下：

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

步骤 5 采用交互模式执行，并在执行过程中会创建 root 用户互信和 openGauss 用户互信：

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

这里的 omm 为操作系统用户（注：同时 omm 也是 openGauss 的数据库管理员账号，在下面的 1.4.4 环节中会创建），dbgrp 为运行 openGauss 的操作系统用户的群组名称，/opt/software/openGauss/clusterconfig.xml 为 openGauss 配置文件路径。在执行过程中，用户根据提示选择是否创建互信，并输入 root 用户或 openGauss 用户的密码。

对 root 创建 trust，输入 root 的密码，购买弹性服务云时自定义的密码。

```
Are you sure you want to create trust for root (yes/no)? yes
Please enter password for root.
Password:-说明：此处输入密码时，屏幕上不会有任何反馈，不用担心，这是 LINUX 操作系统对密码的保护。
Creating SSH trust for the root permission user.
```

创建操作系统 omm 用户，并对 omm 创建 trust，并设置密码，设置为 Admin@123(建议用户自定义设置密码)。

```
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:-说明：此处输入密码时，屏幕上不会有任何反馈，不用担心，这是 LINUX 操作系统对密码的保护。
Please enter password for cluster user again.
Password:-说明：此处输入密码时，屏幕上不会有任何反馈，不用担心，这是 LINUX 操作系统对密码的保护。
Successfully created [omm] user on all nodes.
```

成功后显示为：

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

1.2.3.4 执行安装

执行前置脚本准备好 openGauss 安装环境之后，按照启动安装过程部署 openGauss。

1.2.3.4.1 前提条件

- 已成功执行前置脚本 gs_preinstall；
- 服务器操作系统和网络均正常运行。

1.2.3.4.2 操作步骤

步骤 1 修改文件权限。

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

步骤 2 登录到 openGauss 的主机，并切换到 omm 用户。

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

注：

- omm 指的是前置脚本 gs_preinstall 中-U 参数指定的用户。
- 安装脚本 gs_install 必须以前置脚本中指定的 omm 执行，否则，脚本执行会报错。

步骤 3 使用 gs_install 安装 openGauss。

执行以下命令进行安装，具体如下：

```
[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 为 openGauss 配置文件的路径。在执行过程中，用户需根据提示输入数据库管理员 **omm 用户** 的密码，密码具有一定的复杂度，为保证用户正常使用该数据库，请记住输入的数据库密码。

按照设置密码要求，设置密码（**建议用户自定义设置密码**）：

```
encrypt cipher and rand files for database.
Please enter password for database: --说明：此处输入密码时，屏幕上不会有任何反馈
Please repeat for database: --说明：此处输入密码时，屏幕上不会有任何反馈
begin to create CA cert files
```

设置的密码要符合复杂度要求：

- 最少包含 8 个字符；
- 不能和用户名和当前密码（ALTER）相同，或和当前密码反序；
- 至少包含大写字母（A-Z），小写字母（a-z），数字，非字母数字字符（限定为 ~!@#\$\$%^&*()-_+=\|{}];<.>/?）四类字符中的三类字符。

如果安装成功，显示如下：

```
.....
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 安装生成的目录

安装后的目录及各目录下的文件说明请参见下表。

表1-4 安装后的目录及各目录下的文件说明

序号	项目目录说明	目录	子目录	说明
1	集群 openGauss 安装目录	/opt/gaussdb/app	etc	cgroup工具配置文件。
			include	存放数据库运行所需要的头文件。

			lib	存放数据库的库文件的目录。
			share	存放数据库运行所需要的公共文件，如配置文件模板。
2	集群 openGauss 数据目录	/gaussdb/data	data_dnxxx	DBnode 实例的数据目录，其中主实例的目录名为“data_dnxxx”，备实例的为 data_dnSxxx。xxx 代表 DBnode 编号。
3	集群 openGauss 日志目录	/var/log/gaussdb/用户名	bin	二进制程序的日志目录。
			gs_profile	数据库内核性能日志目录。
			om	OM 的日志目录。例如：部分 local 脚本产生的日志，增删数据库节点接口的日志，gs_om 接口的日志，前置接口的日志，节点替换接口的日志等。
			pg_audit	数据库审计日志目录。
			pg_log	数据库节点实例的运行日志目录。
4	集群 openGauss 系统工具目录	/opt/huawei/wi sequery	script	用于 openGauss 用户进行 openGauss 管理的脚本文件。
			lib	bin 目录下的二进制文件依赖的库文件。


```
# 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 为数据库主节点的端口号。

请在实际操作中记录数据库主节点实例的服务器 IP 地址，数据路径和端口号，并在之后操作中按照实际情况进行替换。

1.3.1.2 本地连接数据库

步骤 1 切换到 omm 用户，以操作系统用户 omm 登录数据库主节点。

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

步骤 2 启动数据库服务

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

显示如下，启动成功。

```
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.
```

步骤 3 连接数据库。

执行如下命令连接数据库。

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

其中 postgres 为需要连接的数据库名称，26000 为数据库主节点的端口号。请根据实际情况替换。

连接成功后，系统显示类似如下信息：

```
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=#
```

omm 用户是管理员用户，因此系统显示 “DBNAME=#”。若使用普通用户身份登录和连接数据库，系统显示 “DBNAME=>”。

“Non-SSL connection” 表示未使用 SSL 方式连接数据库。如果需要高安全性时，请用 SSL 进行安全的 TCP/IP 连接。

步骤 4 退出数据库。

```
postgres=# \q
```

1.3.1.3 gsql 获取帮助

1.3.1.3.1 前提条件

以下操作在 openGauss 的数据库主节点所在主机上执行（本地连接数据库），切换到 omm 用户。

```
su - omm
```

1.3.1.3.2 连接数据库时，可以使用如下命令获取帮助信息

```
gsql --help
```

显示如下帮助信息：

```
[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/wisquery/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 连接到数据库后，可以使用如下命令获取帮助信息

步骤 1 使用如下命令连接数据库。

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

步骤 2 输入 help 命令

```
postgres=# help
```

显示如下帮助信息：

```
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
```

步骤 3 查看版权信息。

```
postgres=# \copyright
```

显示如下版本信息。

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

步骤 4 查看 openGauss 支持的所有 SQL 语句。

```
postgres=# \h
```

显示如下信息：

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

步骤 5 查看 CREATE DATABASE 命令的参数可使用下面的命令。

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

显示如下帮助信息：

```
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 ]}{...} ;
```

步骤 6 查看 gsql 支持的元命令。

```
postgres=# \?
```

显示如下信息：

```
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
.....
```

步骤 7 退出数据库

```
postgres=# \q
```

1.4 数据库工具使用

本节描述使用数据库的基本操作。通过此节您可以完成创建数据库、创建表及向表中插入数据和查询表中数据等操作。

1.4.1 前提条件

- openGauss 正常运行。
- 由于本实验是对 openGauss 数据库的基本使用，需要掌握 openGauss 数据库的基本操作和 SQL 语法，openGauss 数据库支持 SQL2003 标准语法，数据库基本操作参见附录二。

1.4.2 基本操作步骤

步骤 1 在数据库主节点服务器上，切换至 omm 操作系统用户环境。

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

若不确定数据库主节点部署在哪台服务器，请确认连接信息。

步骤 2 启动数据库服务（可选操作，如未启动，请按此步骤启动）。

启动服务命令：

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

查看服务是否启动：

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

cluster_state   : Normal
redistributing  : No
-----
```

步骤 3 连接数据库。

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

当结果显示为如下信息，则表示连接成功。

```
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=#
```

其中，postgres 为 openGauss 安装完成后默认生成的数据库。初始可以连接到此数据库进行新数据库的创建。26000 为数据库主节点的端口号，需根据 openGauss 的实际情况做替换，请确认连接信息获取。

引申信息：

- 使用数据库前，需先使用客户端程序或工具连接到数据库，然后就可以通过客户端程序或工具执行 SQL 来使用数据库了。
- gsql 是 openGauss 数据库提供的命令行方式的数据库连接工具。

步骤 4 创建数据库用户

默认只有 openGauss 安装时创建的管理员用户可以访问初始数据库，您还可以创建其他数据库用户帐号。

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

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

如上创建了一个用户名为 joe，密码为 Bigdata@123 的用户。

步骤 5 创建数据库。

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

当结果显示为如下信息，则表示创建成功。

```
CREATE DATABASE
```

步骤 6 使用新用户连接到此数据库执行接下来的创建表等操作。当然，也可以选择继续在默认的 postgres 数据库下做后续的体验。

退出 postgres 数据库。

```
postgres=# \q
```

使用新用户连接到此数据库。

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

当结果显示为如下信息，则表示连接成功。

```
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=>
```

步骤 7 创建 SCHEMA。

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

当结果显示为如下信息，则表示创建 SCHEMA 成功。

```
CREATE SCHEMA
```

步骤 8 创建表。

创建一个名称为 mytable，只有一列的表。字段名为 firstcol，字段类型为 integer。

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

步骤 9 向表中插入数据：

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

当结果显示为如下信息，则表示插入数据成功。

```
INSERT 0 1
```

0：表示 OID，1：表示插入的条数。

查看表中数据：

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

1.4.3 工具使用操作步骤

1.4.3.1 gs_checkos

步骤 1 用 root 用户登录装有 openGauss 数据库服务的操作系统。在 root 用户下执行 gs_checkos 先对系统参数进行检查。

```
[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.
```

说明事项:

Normal 为正常项, Abnormal 为必须处理项, Warning 可以不处理。

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

表示: 总共检查 14 项, 其中 Abnormal 必须处理项为 0, Warning 告警项为 3。

步骤 2 调整系统参数值。

在参数配置文件 (/etc/sysctl.conf) 中将参数:

vm.min_free_kbytes(表示: 内核内存分配保留的内存量) 的值调整为 348844。

net.ipv4.tcp_retries1 的值由 3 调整为 5。

net.ipv4.tcp_syn_retries 的值由 6 调整为 5。

net.sctp.path_max_retrans 的值由 5 调整为 10。

net.sctp.max_init_retransmits 的值由 8 调整为 10。

具体设置如下:

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

输入“i”进入 INSERT 模式，进行修改。

```
[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
```

参数值修改好后，按“**ESC**”键退出编辑模式，然后输入“**:wq**”后回车进行保存。接着通过执行 `sysctl -p` 命令使刚才修改的参数生效，具体如下：

```
[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
```

步骤 3 通过执行 `gs_checkos -i A` 查看系统参数检查是否能通过。

```
[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.
```

从检查结果可以看出，系统参数检查已经通过。

1.4.3.2 gs_check

`gs_check` 能够帮助用户在 openGauss 运行过程中，全量的检查 openGauss 运行环境，操作系统环境，网络环境及数据库执行环境，也有助于在 openGauss 重大操作之前对各类环境进行全面检查，有效保证操作执行成功。

本实验主要是通过 `gs_check` 工具来检查 openGauss 数据库运行状态。先进行场景设置，然后根据检查结果进行数据库调整。

语法如下：

单项检查：

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

场景检查：

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

场景检查项。默认的场景有 `inspect`（例行巡检）、`upgrade`（升级前巡检）、`binary_upgrade`（就地升级前巡检）、`health`（健康检查巡检）、`install`(安装)等，用户可以根据需求自己编写场景。

显示帮助信息。

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

1.4.3.2.1 操作步骤

步骤 1 用 ROOT 用户登录装有 openGauss 数据库服务的操作系统然后用 su - omm 命令切换至 OMM 用户环境，登录后信息如下。

```
[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
```

步骤 2 确认 openGauss 数据库服务是否启动。

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

```
-----
cluster_name   : dbCluster
cluster_state  : Normal
redistributing : No
-----
```

cluster_state : Normal 表示已启动，可以正常使用。如果状态为非 Normal 表示不可用
为了实验场景设置，如果数据库服务已经启动，请执行步骤 3 先关闭服务。

步骤 3 关闭 openGauss 数据库服务。

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

```
Stopping cluster.
```

```
=====
Successfully stopped cluster.
```

```
=====
End stop cluster.
```

步骤 4 检查 openGauss 实例连接。

```
[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/wisquery/script/gspylib/inspection/output/CheckReport_2020072139449163171.tar.gz

```

说明：

CheckDBConnection.....NG 表示连接检查项无用；

The database can not be connected. 表示实例不能连接；

Failed. All check items run completed. Total:1 NG:1 表示共检查 1 项并且检查结果未通过。

步骤 5 启动 openGauss 数据库服务。

```

[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.

```

步骤 6 确认 openGauss 数据库服务已启动。

```

[omm@ecs-opengauss ~]$ gs_om -t status;
-----
cluster_name      : dbCluster
cluster_state     : Normal
redistributing    : No
-----

```

步骤 7 再次检查 openGauss 实例连接。

```

[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/wisquery/script/gspylib/inspection/output/CheckReport_2020072140672174672.tar.gz
```

说明：

CheckDBConnection.....OK 表示连接检查项正常；

Success. All check items run completed. Total:1 Success:1 表示共检查 1 项并且检查结果成功。

1.4.3.3 gs_checkperf

1.4.3.3.1 通过 gs_checkperf 工具来检查数据库性能

说明：

gs_checkperf 可以对以下级别进行检查：

- openGauss 级别（主机 CPU 占用率、Gauss CPU 占用率、I/O 使用情况等）
- 节点级别（CPU 使用情况、内存使用情况、I/O 使用情况）
- 会话/进程级别（CPU 使用情况、内存使用情况、I/O 使用情况）
- SSD 性能（写入、读取性能）

其中检查 SSD 性能要用 root 用户执行，检查 openGauss 性能要用 openGauss 安装用户执行本实验为检查 openGauss 性能。

步骤 1 用 ROOT 用户登录装有 openGauss 数据库服务的操作系统然后用 su - omm 命令切换至 OMM 用户环境，登录后信息如下。

```
[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
```

步骤 2 先启动数据库服务，再用 `gs_checkperf` 检查下，再使用 `gsq` 客户端以管理员用户身份连接 `postgres` 数据库，假设端口号为 26000。

先启动数据库服务。

```
[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.
```

用 `gs_checkperf` 检查下。

```
[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
```

确认 openGauss 数据库服务是否正常。

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

cluster_state : Unavailable
redistributing : No
-----
```

`cluster_state : Normal` 表示已启动，可以正常使用。如果状态为 `Unavailable` 表示不可用
为了实验继续进行，请先启动数据库服务。

启动数据库服务（如果数据库服务是正常的，此步骤可以不执行）。

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

```
Starting cluster.
=====
=====
Successfully started.
```

然后连接 postgres 数据库。

```
[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=#
```

步骤 3 对 PMK 模式下的表进行统计信息收集。

```
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=#
```

说明：

- gs_checkperf 工具的监控信息依赖于 pmk 模式下的表的数据，如果 pmk 模式下的表未执行 analyze 操作，则可能导致 gs_checkperf 工具执行失败。

步骤 4 执行简要性能检查。

用 \q 先退出 postgres 数据库，然后在操作系统用户 omm 环境下去执行 gs_checkperf 检查工具，具体如下：

```
postgres=#
postgres=# \q
[omm@ecs-opengauss ~]$ gs_checkperf
Cluster statistics information:
Host CPU busy time ratio          : 11.76      % -----主机 CPU 占用率
MPPDB CPU time % in busy time    : 1.40      % ----Gauss CPU 占用率
Shared Buffer Hit ratio           : 99.50     % ----共享内存命中率
In-memory sort ratio             : 0         ---内存中排序比率
Physical Reads                   : 398       ---物理读次数
Physical Writes                  : 98        ---物理写次数
DB size                          : 38        MB---DB 大小
Total Physical writes            : 98        ---总物理写次数
Active SQL count                 : 4         ---当前 SQL 执行数
Session count                   : 6         ---Session 数量
```

步骤 5 执行详细性能检查。

```
[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 数据库及对象管理

2.1 实验介绍

2.1.1 关于本实验

本实验主要描述表空间、数据库的创建、查询、修改、删除管理以及行存表、列存表、MOT表的创建、查询、修改和删除管理，同时介绍用户、角色及 schema 管理和数据导入导出的各种方式。

2.1.2 实验目的

- 掌握表空间的创建、查询、修改、删除。

- 掌握数据库的创建、查询、修改、删除。
- 掌握行存表、列存表、MOT 表的创建、查询、修改、删除。
- 掌握用户、角色及 schema 管理。
- 掌握不同的数据导入导出方式。

2.2 实验内容

2.2.1 登录数据库

步骤 1 使用 Shell 工具登录 ECS 弹性云服务器

步骤 2 从 root 系统用户切换到 omm 数据库用户

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

步骤 3 登录数据库

```
[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 表空间的创建、查询、修改与删除

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

1. 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种固态设备。一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上。
2. 管理员通过表空间可以设置占用的磁盘空间。用以在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间。
3. 表空间可以控制数据库数据占用的磁盘空间，当表空间所在磁盘的使用率达到 90%时，数据库将被设置为只读模式，当磁盘使用率降到 90%以下时，数据库将恢复到读写模式。
4. 表空间对应于一个文件系统目录，用户需要对该目录拥有读写权限。
5. 使用表空间配额管理会使性能有 30%左右的影响，MAXSIZE 指定每个数据库节点的配额大小，误差范围在 500MB 以内。请根据实际情况确认是否需要设置表空间的最大值。

步骤 1 创建表空间。

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

当结果显示为“CREATE TABLESPACE”时，表示创建成功。

其中，test_tbs 为新创建的表空间，/opt/opengauss/tablespace/test_tbs1 为用户拥有读写权限的空目录，也是数据文件存放的目录。

步骤 2 创建用户，并授予 test_tbs 表空间的访问权限。

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

其中'xxxxxxx'为用户密码，需自定义设置并符合密码复杂度要求。授予完成后，用户 jack 就具有了在 test_tbs 下创建数据库对象的权限。

步骤 3 在 test_tbs 表空间下创建表。

方法一：

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

方法二：

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

步骤 4 查询表空间。

方法一：检查 pg_tablespace 系统表。如下命令可查到系统和用户定义的全部表空间。

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

方法二：使用 gsql 程序的元命令查询表空间。

```
postgres=# \db

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

步骤 5 查询表空间使用率。

查询表空间的当前使用情况，其中得到的结果表示表空间的大小，单位为字节。

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

计算表空间使用率。

表空间使用率=PG_TABLESPACE_SIZE/表空间所在目录的磁盘大小。

步骤 6 修改表空间。

通过以下命令重命名表空间。

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

步骤 7 删除表空间。

通过以下命令删除表空间。用户必须是表空间的 owner 或者系统管理员才能删除表空间。

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

发现有报错，提示表空间不为空。此时需要先删除在表空间内创建的对象，再删除表空间。

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

2.2.3 数据库的创建、查看、修改与删除

用户必须拥有数据库创建的权限或者是数据库的系统管理员权限才能创建数据库。

步骤 1 创建两个新的表空间 db_tbs, 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
```

步骤 2 在 db_tbs 表空间内新建数据库 testdb。

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

步骤 3 查看数据库。

方式一：通过系统表 pg_database 查询数据库列表。

```
postgres=# SELECT datname FROM pg_database;
datname
```



```
-----
template1
testdb
template0
postgres
(4 rows)
```

方式二：使用\l 元命令查看数据库系统的数据库列表。

```
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)
```

步骤 4 修改数据库。

修改数据库设置默认的模式搜索路径。

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

修改数据库表空间。

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

重命名数据库。

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

步骤 5 删除数据库。

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

2.2.4 行存表、列存表、MOT 表的创建、查看、修改与删除

行存表的适用场景：

点查询（返回记录少，基于索引的简单查询）；

增、删、改操作较多的场景。

列存表的适用场景：

统计分析类查询（关联、分组操作较多的场景）；

即席查询（查询条件不确定，行存表扫描难以使用索引）。

步骤 1 创建行存表。默认就是行存表，即 WITH (ORIENTATION = ROW)可省略。

```
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);
```

步骤 2 创建列存表。

```
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);
```

步骤 3 查看表信息。

使用\d 元命令查看表列表。

```
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)
```

步骤 4 修改表的属性。

增加列。

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

增加列上的默认值。

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

删除列上的默认值。

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

修改字段的数据类型。

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

修改列的名称。

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

删除列。

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

步骤 5 创建 MOT 表。

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

如出现报错，ERROR: Cannot create MOT tables while incremental checkpoint is enabled。需要修改参数 enable_incremental_checkpoint，重启数据库后可创建 MOT 表。

```
[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
gsqll ((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
```

步骤 6 为 MOT 表创建索引。

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

步骤 7 删除表。

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

2.2.5 用户、角色及 schema 管理

2.2.5.1 用户管理

通过 CREATE USER 创建的用户，默认具有 LOGIN 权限。

通过 CREATE USER 创建用户的同时系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA。其他数据库中，则不自动创建同名的 SCHEMA；

用户可使用 CREATE SCHEMA 命令，会分别在其他数据库中，为该用户创建同名 SCHEMA。

步骤 1 连接数据库后，进入 SQL 命令界面。创建用户 jim，登录密码为 Bigdata@123。

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

密码规则如下：

密码默认不少于 8 个字符；

不能与用户名及用户名倒序相同；

至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为 ~!@#\$\$%^&*()-_+=|{[]};,<.>/?）四类字符中的三类字符；

创建用户时，应当使用双引号或单引号将用户密码括起来。

步骤 2 查看用户列表。

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

步骤 3 创建有“创建数据库”权限的用户，则需要加 CREATEDB 关键字。

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

步骤 4 将用户 jim 的登录密码由 Bigdata@123 修改为 Abcd@123。

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

步骤 5 为用户 jim 追加有创建角色的 CREATEROLE 权限。

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

步骤 6 锁定 jim 帐户。

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

步骤 7 解锁 jim 帐户。

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

步骤 8 删除用户。

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

2.2.5.2 角色管理

角色是拥有数据库对象和权限的实体；

在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者；

在数据库中添加一个新角色，角色无登录权限；

创建角色的用户必须具备 CREATE ROLE 的权限或者是系统管理员。

步骤 1 创建一个角色，名为 manager，密码为 Bigdata@123。

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

步骤 2 创建一个角色，从 2020 年 7 月 1 日开始生效，到 2020 年 12 月 1 日失效。

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

步骤 3 修改角色 manager 为系统管理员。

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

步骤 4 删除角色 manager。

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

步骤 5 查看角色。

```
postgres=# SELECT * FROM PG_ROLES;
 rolname | rolsuper | rolinherit | rolcreatorole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconndef | rol
password |   rolvalidbegin   |   rolvaliduntil   |   rolrespool   | rolparentid | roltabspace |
rolconfig | oid | roluseft | rolkind | nodegroup | rolte
mpspace | rolspillspace
-----+-----+-----+-----+-----+-----+-----+-----+-----+
rolname | rolsuper | rolinherit | rolcreatorole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconndef | 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 管理

Schema 又称作模式。

通过管理 Schema，允许多个用户使用同一数据库而不相互干扰；

每个数据库包含一个或多个 Schema；

在数据库创建用户时，系统会自动帮助用户创建一个同名 Schema。

步骤 1 创建模式 ds。

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

步骤 2 将当前模式 ds 更名为 ds_new。

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

步骤 3 创建用户 jack。

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

步骤 4 将 DS_NEW 的所有者修改为 jack。

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

步骤 5 查看 Schema 所有者。

```
postgres=# SELECT s.nspname,u.username 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)
```

步骤 6 删除用户 jack 和模式 ds_new。

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

2.2.5.4 将系统权限授权给用户或者角色

步骤 1 创建名为 joe 的用户，并将 sysadmin 权限授权给他。

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

2.2.5.5 将数据库对象授权给角色或用户

步骤 1 撤销 joe 用户的 sysadmin 权限，然后创建 tpceds 模式，并给 tpceds 模式下创建一张 reason 表。

```
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
```

步骤 2 将模式 tpcds 的使用权限和表 tpcds.reason 的所有权限授权给用户 joe。

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

授权成功后，joe 用户就拥有了 tpcds.reason 表的所有权限，包括增删改查等权限。

步骤 3 将 tpcds.reason 表中 r_reason_sk、r_reason_id、r_reason_desc 列的查询权限，r_reason_desc 的更新权限授权给 joe。

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

步骤 4 将数据库 postgres 的连接权限授权给用户 joe，并给予其在 postgres 中创建 schema 的权限，而且允许 joe 将此权限授权给其他用户。

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

步骤 5 创建角色 tpcds_manager，将模式 tpcds 的访问权限授权给角色 tpcds_manager，并授予该角色在 tpcds 下创建对象的权限，不允许该角色中的用户将权限授权给他人。

```
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 将用户或者角色的权限授权给其他用户或角色

步骤 1 创建角色 manager，将 joe 的权限授权给 manager，并允许该角色将权限授权给其他人

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

步骤 2 创建用户 senior_manager，将用户 manager 的权限授权给该用户


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

2.2.5.7 权限回收

步骤 1 撤销权限，并清理用户

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

2.2.6 数据导入导出

gs_dump 是 openGauss 用于导出数据库相关信息的工具，用户可以自定义导出一个数据库或其中的对象（模式、表、视图等）。支持导出的数据库可以是默认数据库 postgres，也可以是自定义数据库。

2.2.6.1 创建备份目录

使用 omm 用户，创建备份目录

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

2.2.6.2 创建表并插入数据

步骤 1 登录数据库。

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

步骤 2 创建 customer_t1 表。

```
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
```

步骤 3 向 customer_t1 表中插入数据。

```
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
```

步骤 4 查看表 customer_t1 数据。

```
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)
```

步骤 5 创建 joe 用户。

```
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
```

步骤 6 在 joe 用户下创建表 mytable。

```
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
```

步骤 7 向 mytable 表插入数据。

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

步骤 8 查看表 mytable 文件。

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

步骤 9 查看当前表信息。

```
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 copy 数据导出

步骤 1 切换到默认 omm 用户下。

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

步骤 2 使用 copy 命令进行导出。

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

步骤 3 查看导出文件。

```
[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 gs_dump 数据导出

步骤 1 使用 gs_dump 命令将 postgres 数据库导出，导出为纯文本格式。

gs_dump 部分重要指令解释：

-f: 将导出文件发送至指定目录文件夹。

-F: 选择导出文件格式。-F 参数值如下：

p: 纯文本格式

c: 自定义归档

d: 目录归档格式

t: tar 归档格式

-n: 只导出与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。

-t: 指定导出的表（或视图、序列、外表），可以使用多个-t 选项来选择多个表，也可以使用通配符指定多个表对象。

-T: 不转储的表（或视图、或序列、或外表）对象列表，可以使用多个-T 选项来选择多个表，也可以使用通配符指定多个表对象。

```
[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
```

步骤 2 查看导出文件。

```
[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
--
```

步骤 3 使用 gs_dump，导出 postgres 数据库，导出为 tar 格式。

```
[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
```

步骤 4 查看导出文件。

```
[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
```

步骤 5 使用 gs_dump 导出依赖于表的视图。

连接数据库，并创建依赖于表 customer_t1 的视图 vw_customer_t1，后退出。

```
[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
```

导出依赖于表 customer_t1 的视图 vw_customer_t1。

```
[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
```

查看导出文件。

```
[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 gs_dumpall 数据导出。

gs_dumpall 是 openGauss 用于导出所有数据库相关信息工具，它可以导出 openGauss 数据库的所有数据，包括默认数据库 postgres 的数据、自定义数据库的数据、以及 openGauss 所有数据库公共的全局对象。

gs_dumpall 在导出 openGauss 所有数据库时分为两部分：

- gs_dumpall 自身对所有数据库公共的全局对象进行导出，包括有关数据库用户和组，表空间以及属性（例如，适用于数据库整体的访问权限）信息。
- gs_dumpall 通过调用 gs_dump 来完成 openGauss 中各数据库的 SQL 脚本文件导出，该脚本文件包含将数据库恢复为其保存时的状态所需要的全部 SQL 语句。

以上两部分导出的结果为纯文本格式的 SQL 脚本文件，使用 gsql 运行该脚本文件可以恢复 openGauss 数据库。

步骤 1 使用 gs_dumpall 一次导出 openGauss 的所有数据库。

```
[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
```

步骤 2 查看导出文件。

```
[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 gsql 数据导入

对于.sql 文件，可使用 gsql 直接导入。

步骤 1 删除之前导出的表。

```
[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
```

步骤 2 使用 gsql 将导出的表恢复。（出现 ERROR: schema "joe" already exists 报错可忽略）

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

步骤 3 查看恢复后的表。

```
[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 copy 数据导入

对于使用 copy 导出的数据，同样可以使用 copy 导入。

步骤 1 删除使用 copy 命令导出过的表。

```
[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
```


步骤 2 使用 copy 命令进行导入。

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

步骤 3 查看导入的表数据。

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

postgres=# \q
```

2.2.6.8 gs_restore 数据导入

gs_restore 是 openGauss 提供的针对 gs_dump 导出数据的导入工具。通过此工具可由 gs_dump 生成的导出文件进行导入。gs_restore 工具由操作系统用户 omm 执行。

步骤 1 删除使用 gs_dump 导出的表和视图。

```
[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
```

步骤 2 使用 gs_restore 导入 gsdump_post.tar 文件内数据到 tpcc 数据库。

新建 tpcc 数据库。

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

导入 gsdump_post.tar 文件内数据到 tpcc 数据库。

```
[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
```

步骤 3 登录 tpcc 数据库查看恢复的数据。

```
[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 清理实验环境

删除创建的数据库及用户。

```
[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 实验小结

本实验介绍了如何登录 openGauss 数据库，并对 openGauss 数据库进行如表空间、数据库、表、用户角色等维护，并使用导入导出工具，将数据导入至 openGauss 数据库中，通过本节的实验，希望能够熟练掌握 openGauss 数据库及对象的管理。

2.5 思考题

1. 尝试使用 COPY 的方式导出查询结果数据集（即 where 条件语句）。
2. 假设有以下操作：
 - a. 创建表 customer_t1，并导入 4 条数据；
 - b. 创建依赖于表 customer_t1 的视图 vw_customer_t1；

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

此时查看视图 vw_customer_t1，显示 2 条数据。

- c. 导出依赖于表 customer_t1 的视图 vw_customer_t1 到文件 vw_customer.sql；
- d. 重命名表 customer_t1 为 customer_t1_bak；
- e. 新建表 customer_t1，不插入数据；
- f. 导入 vw_customer.sql；

请问此时查看视图 vw_customer，显示有几条数据？

参考答案：视图 vw_customer 中无数据。

3 SQL 语法基础实验

3.1 实验介绍

3.1.1 关于本实验

本实验共 4 个子实验，从数据准备开始，逐一介绍了 OpenGauss 数据库中的 SQL 语法，包括数据查询、数据更新、数据定义以及常用函数和操作符。

3.1.2 实验目的

学习 SQL 相关操作，如常用 DQL、DML、DDL、常用操作符和函数；掌握其基本语法和使用方法。

3.1.3 内容描述

- 子实验 1.2 为预置 SQL 脚本准备实验数据。
- 子实验 1.3 为数据查询实验，通过基本的 DQL 语言介绍，帮助读者掌握从一个或多个表或者视图中检索数据的操作。
- 子实验 1.4 为数据更新实验，通过对 DML 语言基本语法和使用方法的介绍，帮助读者掌握如何对数据库表中数据进行更新操作，包括数据插入、数据修改和数据删除。
- 子实验 1.5 为数据定义实验，介绍了 DDL 的类型、语法格式和使用场景，帮助读者熟练掌握如何用数据定义语言定义或修改数据库中的对象。
- 子实验 1.6 为操作符和函数，详细介绍了常用的操作符和函数以及使用场景，帮助读者掌握如何使用数据库内置的操作符和函数。

3.2 数据准备

通过预置 SQL 脚本创建金融数据模型示例，为后续章节的 SQL 语法学习准备数据。

往金融数据库里相关表插入数据，为后续实验做数据支撑。

3.2.1 实验任务

openGauss 提供一个金融示例库，供用户学习、验证数据库。数据库中包含 6 个表，该章节只使用其中 3 个表格，分别为：

- 客户（客户编号、客户名称、客户邮箱，客户身份证，客户手机号，客户登录密码）
- 银行卡（银行卡号，银行卡类型，所属客户编号）
- 保险（保险名称，保险编号，保险金额，适用人群，保险年限，保障项目）

上述属性对应的编号为：

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

对象之间的关系：

- 一个客户可以办理多张银行卡
- 一个客户可以购买多个保险，同一类保险可由多个客户购买

步骤 1 创建表

使用普通用户 omm 登录到 openGauss

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

启动数据库服务端，若数据库已启动，不需要执行此操作

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

以 postgres 账号连接数据库，5432 是连接端口，不同环境可能不同，请根据实际环境设置。

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

创建客户信息表。

```
--删除表 client
POSTGRES=# DROP TABLE IF EXISTS client;

Succeed.

--创建表 client
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.
```

创建银行卡信息表。

```
--删除表 bank_card
```

```
POSTGRES=# DROP TABLE IF EXISTS bank_card;

Succeed.

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

Succeed.

--给表 bank_card 添加外键约束
POSTGRES=# ALTER TABLE bank_card ADD CONSTRAINT fk_c_id FOREIGN KEY (b_c_id) REFERENCES
client(c_id) ON DELETE CASCADE;

Succeed.
```

创建保险信息表。

```
--删除表 insurance
POSTGRES=# DROP TABLE IF EXISTS insurance;

Succeed.

--创建表 insurance
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.
```

步骤 2 编辑 client.sql

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

client.sql 脚本为：

```
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');
```

步骤 3 编辑 bank_card.sql

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

bank_card.sql 脚本为：

```
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62220213020200000001','Credit Card', 1);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62220213020200000002','Credit Card', 3);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62220213020200000003','Credit Card',5);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62220213020200000004','Credit Card', 7);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('62220213020200000005','Credit Card', 9);
```

步骤 4 编辑 insurance.sql

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

insurance.sql 脚本为：

```
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');
```

步骤 5 在 SQL 界面执行脚本

```
--启动数据库后执行
[gaussdba@kwephicprd12118 bin]$ gsql -d postgres -p 5432

connected.

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

3.3 数据查询

通过 DQL 语言来演示如何从表中查询数据，包括简单查询、带条件查询、连接查询、排序和限制等。

- 了解数据查询 DQL 语法结构。
- 掌握不同场景下 DQL 语言的使用

3.3.1 实验任务

3.3.1.1 简单查询

日常查询中，最常用的是通过 FROM 子句实现的查询。

SELECT 后面使用*号查询 bank_card 表中的所有列。

```
SELECT * FROM bank_card;
b_number          | b_type          | b_c_id
-----+-----+-----
6222021302020000001 | Credit Card     | 1
6222021302020000002 | Credit Card     | 3
6222021302020000003 | Credit Card     | 5
```

查看 bank_card 表中的银行卡号和卡类型

```
SELECT b_number,b_type FROM bank_card;
b_number          | b_type
-----+-----
6222021302020000001 | Credit Card
6222021302020000002 | Credit Card
6222021302020000003 | Credit Card
```

查看客户为 1 的客户编号，客户名称，客户邮箱和银行卡号信息

```
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  | 6222021302020000001
```

别名使用

```
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 | 6222021302020000001
```

3.3.1.2 带条件查询

在 SELECT 语句中，可以通过设置条件以达到更精确的查询。

使用比较操作符 “>、<、>=、<=、!=、<>、=”来指定查询条件。

在银行卡表中查询卡类型是信用卡的银行卡信息。

```
SELECT * FROM bank_card WHERE b_type= 'Credit Card';
```

b_number	b_type	b_c_id
6222021302020000001	Credit Card	1
6222021302020000002	Credit Card	3
6222021302020000003	Credit Card	5
6222021302020000004	Credit Card	7
6222021302020000005	Credit Card	9

从 bank_card 中查询客户 ID 为 1 且银行卡类型为信用卡的银行卡信息

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

b_number	b_type	b_c_id
6222021302020000001	Credit Card	1

3.3.1.3 连接查询

实际应用中所需要的数据，经常会需要查询两个或两个以上的表。这种查询两个或两个以上数据表或视图的查询叫做连接查询。连接查询通常建立在存在相互关系的父子表之间。

- 内连接

内连接的关键字为 inner join，其中 inner 可以省略。使用内连接，连接执行顺序必然遵循语句中所写的表的顺序。

使用内连接联合查询表。查询客户 ID、银行卡卡号和银行卡类型。使用 client 和 bank_card 两个相关的列(c_id)做查询操作。

```
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_number	b_type
1	6222021302020000001	Credit Card
3	6222021302020000002	Credit Card

- 外连接

使用左外连接查询客户的编号，用户姓名，银行卡号和银行卡类型

```
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_number	b_type
1	Zhang Yi	6222021302020000001	Credit Card
3	Zhang San	62220213020200000002	Credit Card
2	Zhang Er	6222021302020000003	
4	Zhang Si	6222021302020000004	

使用右外连接查询表 client 和表 bank_card 中的数据

```
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_number	b_type
1	Zhang Yi	6222021302020000001	Credit Card

3	Zhang San	622220213020200000002	Credit Card
		622220213020200000003	Credit Card
		622220213020200000004	Credit Card
		622220213020200000005	Credit Card

通过全连接获取表 client 和表 bank_card 的完全连接数据

```
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_number	b_type
1	Zhang Yi	622220213020200000001	Credit Card
3	Zhang San	622220213020200000002	Credit Card
		622220213020200000003	Credit Card
		622220213020200000004	Credit Card
		622220213020200000005	Credit Card
2	Zhang Er		
4	Zhang Si		

3.3.1.4 数据排序

使用 ORDER BY 子句对查询语句返回的行根据指定的列进行排序。

按照保额降序查询保险编号大于 2 的保险名称，保额和适用人群。

```
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 数据限制

LIMIT 子句允许限制查询返回的行。可以指定偏移量，以及要返回的行数或行百分比。可以使用此子句实现 top-N 报表。要获得一致的结果，请指定 ORDER BY 子句以确保确定性排序顺序。

先不使用 LIMIT 子句进行查询，与使用 LIMIT 子句进行查询的结果进行对比。

查询下表 insurance 中的保险信息。通过增加 LIMIT 2 OFFSET 1 限定查询时跳过前 1 行后，查询总共 2 行数据。

```
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 数据更新

将介绍数据操作语言 DML (Data Manipulation Language)，演示如何通过 DML 语言来对数据库表中的数据进行更新操作。主要包括：数据插入、数据修改和数据删除。

- 了解数据操作语言 DML 语法结构。
- 掌握不同场景下 DML 语言的使用

3.4.1 实验任务

3.4.1.1 数据插入

在表中插入新的数据。

INSERT 语句有三种形式：

第一种是值插入，即构造一行记录并插入到表中。

第二种形式是查询插入，它通过 SELECT 子句返回的结果集构造一行或多行记录插入到表中。

第三种是先插入记录，如果报主键冲突错误则执行 UPDATE 操作，更新指定字段值。

向表 bank_card 中值插入数据。

```
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000030','Savings Card', 30);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000031','Savings Card', 31);
INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000032','Savings Card', 32);
```

通过子查询向表 bank_card1 表中插入 bank_card 表的所有数据

```
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;
```

主键冲突错误，执行 UPDATE 操作。

```
INSERT INTO bank_card VALUES ('6222021302020000001', 'Credit Card', 1) ON DUPLICATE KEY
UPDATE b_type = 'Savings Card';
```

数据修改

查询表 bank_card 记录。

```
SELECT * FROM bank_card;
b_number          | b_type          | b_c_id
-----+-----+-----
622202130202000001 | Savings Card    | 1
622202130202000002 | Credit Card     | 3
622202130202000003 | Credit Card     | 5
622202130202000004 | Credit Card     | 7
622202130202000005 | Credit Card     | 9
622202130202000030 | Savings Card    | 30
622202130202000031 | Savings Card    | 31
622202130202000032 | Savings Card    | 32
```

更新表 bank_card 中客户编码为 1 的 b_type 为信用卡。

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

3.4.1.2 数据删除

从表中删除行。

查询表 bank_card 记录。

```
SELECT * FROM bank_card;
```

删除表 bank_card 中同时匹配 b_type='Credit Card'和 b_c_id=1 的记录。

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

查询表 bank_card 记录

```
SELECT * FROM bank_card;
```

删除表 bank_card 的全部数据。

```
DELETE FROM bank_card;
```

查询表 bank_card 记录

```
SELECT * FROM bank_card;
```

3.5 数据定义

介绍数据定义语言 DDL (Data Definition Language) , 演示如何通过 DDL 语言来定义或修改数据库中的对象。如: 数据库, 模式, 表、索引、视图、数据库、序列、用户等。

- 了解数据定义语言 DDL 语法结构。
- 掌握不同场景下 DDL 语言的使用

3.5.1 实验任务

3.5.1.1 定义数据库

数据库是组织、存储和管理数据的仓库, 而数据库定义主要包括: 创建数据库、修改数据库属性, 以及删除数据库。数据库相关的 DDL 包括创建数据库、修改数据库属性、删除数据库。

3.5.1.1.1 创建数据库

创建数据库 jim 用户

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

创建数据库 music, 指定用模板 template0, 并指定所有者为 jim。

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

3.5.1.1.2 修改数据库属性

在创建数据库后如果因为业务场景变动, 使用 ALTER Database 命令可以更改数据库的定义。

修改数据库 music 的最大连接数。

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

修改数据名称。

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

修改数据库默认表空间。

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

修改数据库所有者。

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

3.5.1.1.3 删除数据库

当不再需要数据库时，可以使用 DROP DATABASE 命令删除此数据库

删除数据库

```
DROP DATABASE music01;
```

删除用户

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

3.5.1.2 定义模式

模式是一组数据库对象的集合，主要用于控制对数据库对象的访问。

3.5.1.2.1 创建模式

创建一个角色 role1

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

为用户 role1 创建一个同名 schema，子命令创建表 films 和 winners 的拥有者为 role1。

```
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 修改模式属性

在创建模式后如果因为业务场景变动，使用 ALTER SCHEMA 命令可以更改模式的定义。

修改模式名

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

修改模式的所有者

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

3.5.1.2.3 删除模式

当不再需要模式时，可以使用 DROP SCHEMA 命令删除此模式

删除模式

```
DROP SCHEMA role1_new;
```

```
# 删除用户
```

```
DROP USER jack;
```

3.5.1.3 定义表

表是数据库中的一种特殊数据结构，用于存储数据对象以及对象之间的关系。表相关的 DDL 包括创建表、修改表属性、删除表和删除表中所有数据。

3.5.1.3.1 创建表

普通表

```
# 创建表 bank_card。
```

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

临时表

```
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 修改表属性

在创建表后如果因为业务场景变动，使用 ALTER TABLE 命令可以更改表的定义。

```
# 将 bank_card 表重命名为 bank_card1。
```

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

```
# 在 bank_card1 表中增加列，数据类型为 Integer。
```

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

```
# 修改列的数据类型。
```

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

```
# 添加约束。
```

```
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);
```

```
# 删除列。
```

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

```
# 向表中插入数据若不满足约束会报错。
```

```
INSERT INTO bank_card1(b_number, b_type, b_c_id) VALUES ('6222021302020000001','Credit Card', 0);  
GS-01222, Check constraint violated
```

3.5.1.3.3 删除表中数据

有些情况下当表的数据不再使用时，需要删除表的所有行，即清空该表。

```
# 删除 bank_card1 表中所有的行。
```

```
DELETE FROM bank_card1;
```

3.5.1.3.4 删除表

当不再需要表数据及表定义时，可以使用 DROP TABLE 命令删除此表

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

3.5.1.4 定义分区表

openGauss 数据库支持的分区表为范围分区表、列表分区表、哈希分区表。

分区表和普通表相比具有以下优点：

改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。

增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。

方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

均衡 I/O：可以把不同的分区映射到不同的磁盘以平衡 I/O，改善整个系统性能。

分区表表相关的 DDL 包括创建分区表、修改分区表属性、删除分区表和删除分区表中所有数据。

3.5.1.4.1 创建分区表

创建表空间和分区表

```
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 修改分区表

在创建分区表后如果因为业务场景变动，使用 ALTER TABLE 命令可以更改表的定义。

修改分区表的表空间

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

重命名分区表

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

合并分区

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

删除分区

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

3.5.1.4.3 删除分区表

当不再需要分区表时，可以使用 DROP TABLE 命令删除此表

```
DROP TABLE bank_card;
```

删除表空间

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

3.5.1.5 定义索引

索引可以提高数据的访问速度，但同时也增加了插入、更新和删除表的处理时间。所以是否要为表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询条件或者被要求排序的字段来确定是否建立索引。索引相关的 DDL 包括创建索引、删除索引属性和删除索引。

3.5.1.5.1 索引建立原则

建立索引后，在查询的时候合理利用索引能够提高数据库性能。但是创建索引和维护索引需要时间消耗，索引文件占用物理空间，同时对表的数据进行 INSERT、UPDAT、DELETE 时候需要维护索引，会降低数据的维护效率。所以建议基于以下原则，合理使用索引。

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。
- where 子句的过滤条件字段上（尤其是范围条件）。
- 在经常出现在 order by、group by 和 distinct 后的字段。

3.5.1.5.2 创建索引

在使用 CREATE INDEX 命令创建索引。

在表 bank_card 的 b_number, b_type 上创建索引。

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

在分区表 education 上创建分区索引

```
--创建分区表 education。
postgres=# CREATE TABLE education(staff_id INTEGER NOT NULL, highest_degree CHAR(8),
graduate_school VARCHAR(64),graduate_date DATETIME, education_note VARCHAR(70))
PARTITION BY LIST(highest_degree)
(
PARTITION doctor VALUES ('Doctor'),
PARTITION master VALUES ('Master'),
PARTITION undergraduate VALUES ('Bachelor')
);
--创建分区索引，并制定索引分区的名称。
```



```
postgres=# CREATE INDEX idx_education ON education(staff_id ASC, highest_degree) LOCAL (PARTITION
doctor, PARTITION master, PARTITION undergraduate);
```

3.5.1.5.3 修改索引

- 重建索引

当对表执行大量增、删、改操作后，表的数据可能在磁盘的物理文件上存在碎片，影响访问速度。通过 ALTER INDEX 命令重建索引可以重组索引数据并释放不使用的空间，从而提高数据访问效率和空间使用效率。

数据库重启回滚期间不支持重建索引。

重建索引。

```
ALTER INDEX idx_bank_card REBUILD;
```

- 重命名索引

如果用户需要重新统一索引的命名风格，可以通过 RENAME 语法只修改索引的名称，而不改变索引的其他属性。

数据库重启回滚期间不支持重命名索引。

重命名索引

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

设置索引不可用

```
ALTER INDEX idx_bank_card UNUSABLE;
```

3.5.1.5.4 删除索引

使用 DROP INDEX 命令删除索引。删除索引有以下限制：

- 数据库重启回滚期间不支持删除索引。
- 开启“ENABLE_IDX_CONFS_NAME_DUPL”配置项后，不同表支持索引名重名，删除索引时必须指定表名。
- 不能删除已有的“UNIQUE KEY”键或“PRIMARY KEY”键约束相关的索引。
- 删除表则同时删除了该表的索引。

以删除 bank_card 表上的索引 idx_bank_card 为例。

```
DROP INDEX idx_bank_card;
```

3.5.1.6 定义视图

- 当用户对数据库中的一张或者多张表的某些字段的组合感兴趣，而又不想每次键入这些查询时，用户就可以定义一个视图，以便解决这个问题。视图相关的 DDL 包括创建视图和删除视图。

3.5.1.6.1 相关概念

- 视图与基本表不同，不是物理上实际存在的，是一个虚表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。视图每次被引用的时候都会运行一次。

3.5.1.6.2 创建视图

创建视图，可以使用 CREATE VIEW 命令。

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

CREATE VIEW 中的 OR REPLACE 可有可无，当存在 OR REPLACE 时，表示若以前存在该视图就进行替换。

查看视图中的数据，语法和查询表一样。

```
SELECT * FROM privilege_view;
```

查看视图结构。

```
\d privilege_view;
```

3.5.1.6.3 删除视图

删除视图，使用 DROP VIEW 命令。

删除视图。

```
DROP VIEW privilege_view;
```

3.5.1.7 定义序列

序列可以产生一组等间隔的数值，能自增，主要用于表的主键。序列的 DDL 包括创建序列、修改序列和删除序列。

3.5.1.7.1 创建序列

向当前数据库中增加一个新的序列生成器。当前用户为该生成器的所有者。

创建序列 seq_auto_extend，序列起点为 10，步长为 2，最大值为 200，序列到达最大值时可循环。

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

查询序列 seq_auto_extend 的下一个值

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

```
NEXTVAL
```

```
-----
```

```
10
```

3.5.1.7.2 修改序列

修改步长为 4，最大值为 400。

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

3.5.1.7.3 删除序列

删除序列 seq_auto_extend。

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

3.6 常用函数和操作符

介绍数据库常用函数和操作符，例如字符处理函数和操作符，数字操作函数和操作符，时间和日期处理函数和操作符等。

3.6.1 实验任务

3.6.1.1 字符处理函数

openGauss 提供的字符处理函数和操作符主要用于字符串与字符串、字符串与非字符串之间的连接，以及字符串的模式匹配操作。

instr(string1,string2,int1,int2)返回在 string1 中从 int1 位置开始匹配到第 int2 次 string2 的位置，第一个 int 表示开始匹配起始位置，第二个 int 表示匹配的次数。

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

overlay(string placing string FROM int [for int])替换子字符串。FROM int 表示从第一个 string 的第几个字符开始替换，for int 表示替换第一个 string 的字符数目。

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

position(substring in string)指定子字符串的位置。字符串区分大小写。

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

substring_inner(string [FROM int] [for int]) 截取子字符串，FROM int 表示从第几个字符开始截取，for int 表示截取几个字节。

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

replace(string text, FROM text, to text)把字符串 string 里出现地所有子字符串 FROM 的内容替换成子字符串 to 的内容。

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

substring(string [FROM int] [for int])截取子字符串，FROM int 表示从第几个字符开始截取，for int 表示截取几个字节。

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

3.6.1.2 数字操作函数和操作符

数字操作符

数字操作符	描述	数字操作符	描述
+	加	!!	阶乘（前缀操作符）
-	减		二进制OR
*	乘	&	二进制AND
/	除（除法操作符不会取整）	#	二进制XOR
%	模运算	~	二进制NOT
@	绝对值	^	幂（指数运算）
	平方根	<<	左移位
	立方根	>>	右移位

```
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), asin(exp): 返回表达式的绝对值，余弦值，正弦值，反余弦值和反正弦值。

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

bitand(exp1,exp2)计算两个数字与运算(&)的结果。

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

round(number[,decimals]) 将 number 类数值按照 decimals 指定的向小数点前后截断。

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

3.6.1.3 日期和时间处理函数和操作符

时间和日期操作符” +”

```
SELECT date '2021-5-28' + integer '7' AS RESULT;
      result
-----
2021-06-04 00:00:00

SELECT date '2021-05-28' + interval '1 hour' AS RESULT;
      result
-----
2021-05-28 01:00:00

SELECT date '2021-05-28' + time '03:00' AS RESULT;
      result
-----
2021-05-28 03:00:00

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

时间和日期操作符” -”

```
SELECT date '2021-05-01' - date '2021-04-28' AS RESULT;
      result
-----
3 days

SELECT date '2021-05-01' - integer '7' AS RESULT;
      result
-----
2021-04-24 00:00:00

SELECT date '2021-05-28' - interval '1 hour' AS RESULT;
      result
-----
2021-05-27 23:00:00

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

```
03:00:00
```

age(timestamp, timestamp)将两个参数相减，并以年、月、日作为返回值。若相减值为负，则函数返回亦为负，入参可以都带 timezone 或都不带 timezone。

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

current_time 当前时间

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

3.6.1.4 类型转换函数

to_char(int, text), to_clob(str), to_date(exp[,fmt]), to_number(n[,fmt])将指定入参转换为 char, clob, date, number 类型。

```
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)将 x 转换成 y 指定的类型。

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

hextoraw(string) 将一个十六进制构成的字符串转换为 RAW。

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

3.7 实验小结

本实验从数据准备开始，逐一介绍了 openGauss 数据库中的 SQL 语法，包括数据查询、数据更新、数据定义和数据控制。而且介绍了 openGauss 数据库中的常用函数和操作符的使用。

3.8 思考题

1. SQL 语句分成几类？简单描述各位 SQL 语句的作用。

参考答案：

DDL (Data Definition Language)数据定义语言，用于定义或修改数据库中的对象。如：表、索引、视图、数据库、序列、用户、角色、表空间、存储过程等。

DML (Data Manipulation Language)数据操纵语言，用于对数据库表中的数据进行操作，如插入，更新和删除。视图 vw_customer 中无数据。

DCL (Data Control Language)数据控制语言，用来设置或更改数据库事务、授权操作（用户或角色授权，权限回收，创建角色，删除角色等）、锁表（支持 SHARE 和 EXCLUSIVE 两种锁表模式）、停机等。

DQL (Data Query Language)数据查询语言，用来查询数据库内的数据，如查询数据、合并多个 select 语句的结果集。

2.什么是函数？设计函数的目的是什么？并列出具体的函数。

参考答案：函数作为数据库的一个对象，是在 SQL 基础上设计的独立的程序单元。

根据函数处理对象的数据类型分为：字符处理函数，二进制字符串函数，位串函数，数字操作函数，时间和日期处理函数，几何函数和类型转换函数等。

4 云数据库 GaussDB(for openGauss)

4.1 实验介绍

4.1.1 关于本实验

本实验主要描述 GaussDB (for openGauss)数据库的购买流程以及使用 DAS 客户端工具的使用和连接数据库的方法，并在 GaussDB (for openGauss)数据库中进行数据库创建、表创建、增删改查数据等简单操作。

4.1.2 实验目的

- 掌握 GaussDB (for openGauss)数据库的购买流程;
- 掌握 DAS 客户端工具使用方法;
- 掌握在 GaussDB (for openGauss)数据库中基础 SQL 语句的使用。

4.2 购买 GaussDB (for openGauss)数据库实例

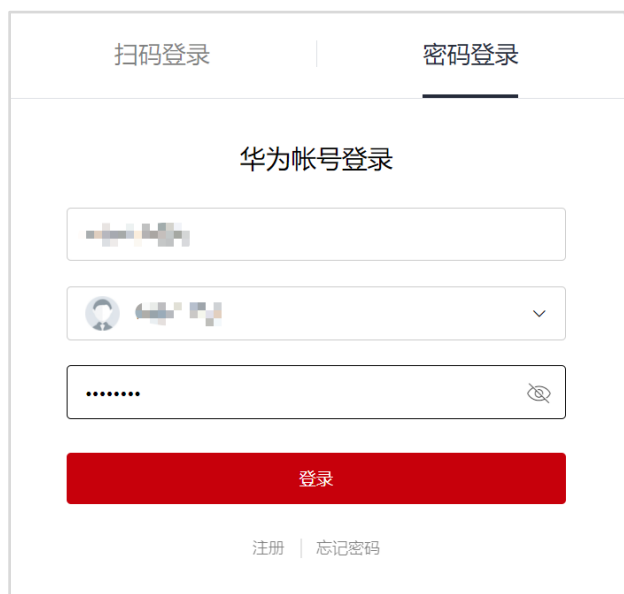
简介：GaussDB(for openGauss)是基于华为主导的 openGauss 生态推出的企业级分布式关系型数据库。该产品具备企业级复杂事务混合负载能力，同时支持分布式事务，同城跨 AZ 部署，数据 0 丢失，支持 1000+的扩展能力，PB 级海量存储。同时拥有云上高可用，高可靠，高安全，弹性伸缩，一键部署，快速备份恢复，监控告警等关键能力，能为企业提供功能全面，稳定可靠，扩展性强，性能优越的企业级数据库服务。

4.2.1 登录华为云官网

步骤 1 使用 PC 上的浏览器访问华为云官网：<https://www.huaweicloud.com/?locale=zh-cn>，单击页面右上角的“登录”，进入华为云账号登录页面。



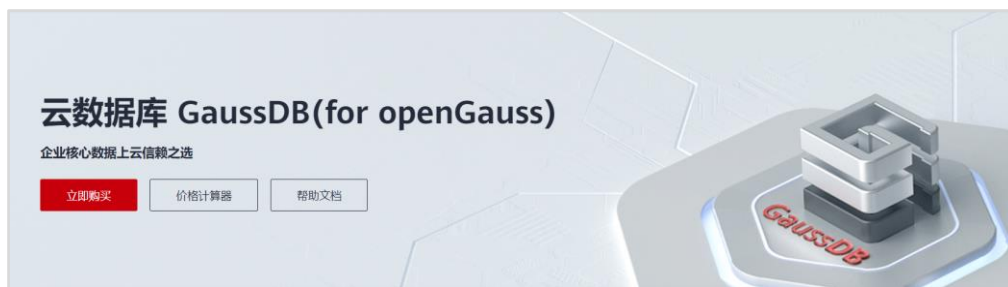
步骤 2 输入华为云账号的用户名和密码，单击下方的“登录”，登录华为云官网。



步骤 3 依次选择“产品”-“数据库”，点击“云数据库 GaussDB(for openGauss)”。



步骤 4 点击“立即购买”，进入购买页面。



4.2.2 购买数据库实例

步骤 1 “计费模式”选择“按需计费”“实例名称”可以根据自定义命名；“事务一致性”这里选择“强一致性”。由于本实验不考虑高可用部分，减少实验花费，因此“分片数量”与“协调节点数量”均选择“1”即可。如下图所示：



三 < | 购买数据库实例

计费模式 包年/包月 按需计费 ?

区域 华北-北京四 ?
不同区域的资源之间内网不互通。请选择靠近您客户的区域，可以降低网络时延、提高访问速度。

实例名称 gauss-test ?

数据库引擎 GaussDB(for openGauss)

数据库版本 2020企业版

实例类型 分布式版

部署形态 ? 独立部署

事务一致性 ? 强一致性 最终一致性

副本集数量 - 3 +

分片数量 - 1 +

协调节点数量 ? - 1 +

可用区 可用区二 可用区一 可用区七
只支持选择一个或者三个不同的可用区。

时区 UTC+08:00

步骤 2 “性能规格”中采用默认选项即可。如下图所示：

性能规格 ?

通用增强II型

规格名称

☒ 8 vCPUs | 64 GB 该规格不能用于生产环境

☐ 16 vCPUs | 128 GB

☐ 32 vCPUs | 256 GB

☒ 64 vCPUs | 512 GB (售罄)

当前选择实例 通用增强II型 | 8 vCPUs | 64 GB

存储类型

超高IO

您可以点此了解, 存储类型详情

存储空间 (GB)

160 GB

160 3,300 6,450 9,600 16,000

GaussDB给您提供相同大小的备份存储空间, 超出部分按照OBS计费规则收取费用。

磁盘加密

不加密

加密

推荐

?

步骤 3 剩余部分需要填入“管理员密码”并“确认密码”。其他选项默认即可，注意左下角显示“配置费用”约为 84 元。如下图所示：

① 虚拟私有云、子网、安全组与实例关系。

虚拟私有云 ?

default_vpc

default_subnet(192.168.0.0/24)

内网安全组 ?

default_securitygroup

查看内网安全组

数据库端口

默认端口8000

管理员用户名

root

管理员密码

.....

请妥善保管密码, 系统无法获取您设置的密码内容。

确认密码

.....

参数模板

Default-GaussDB(for openGauss)-Enterprise ...

查看参数模板

标签

如果您需要使用同一标签标识多种云资源, 即所有服务均可在标签输入框下拉选择同一标签, 建议在TMS中创建预定义标签。 查看预定义标签

您还可以添加 10 个标签。

配置费用

¥83.992/小时

?

步骤 4 点击“提交”，进入配置确认页面，确认后完成 GaussDB(for openGauss)实例购买。页面会跳转到控制台，通过运行状态可以看到数据库正在“创建中”。



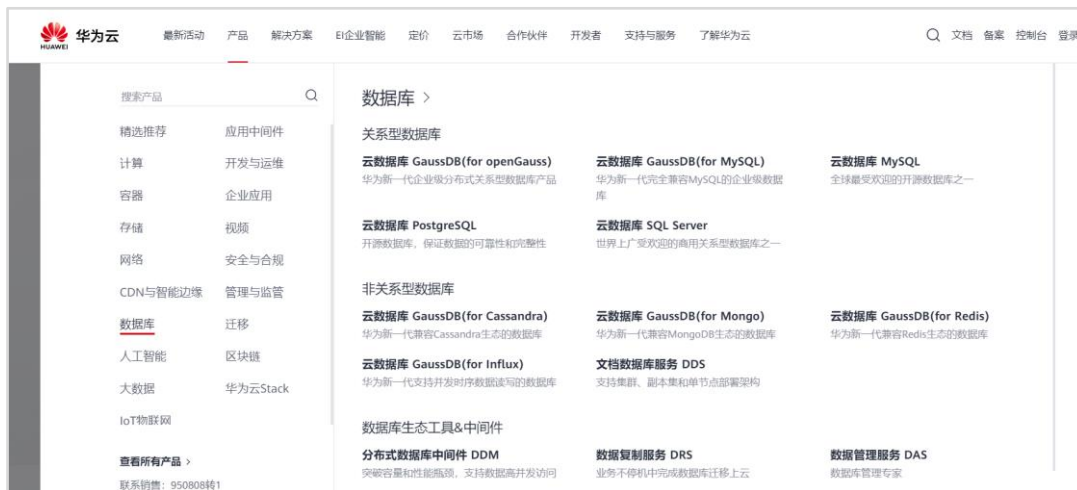
步骤 5 确认实例创建成功，页面实例状态显示“正常”，如图：



4.3 DAS 使用

4.3.1 DAS 连接数据库

步骤 1 登录华为云官网 <https://www.huaweicloud.com/>，点击“产品”-“数据库”，找到“数据库管理服务 DAS”并点击进入。



步骤 2 点击“数据库免费登录”



步骤 3 点击“开发工具”，找到 GaussDB(for openGauss)，点击登录。



输入创建数据库实例时的密码，勾选“记住密码”，点击“测试连接”。测试连接成功后点击“确定”。



4.3.2 创建数据库

步骤 1 点击“新建数据库”进行数据库创建。输入“数据库名称”，并选择“DBCOMPATIBILITY”数据库兼容性为“PostgreSQL”，点击“确定”。



步骤 2 点击“库管理”，对刚才新建的 db01 数据库进行操作。



4.3.3 创建 schema

步骤 1 点击“新建 Schema”创建新的模式，输入 schema 名，点击“确定”。

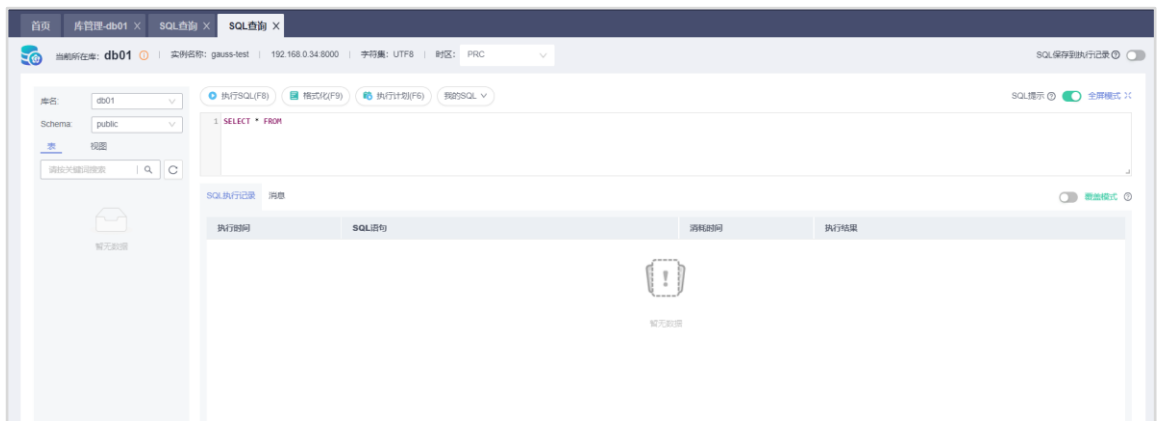


步骤 2 schema 创建完成，如下图所示：



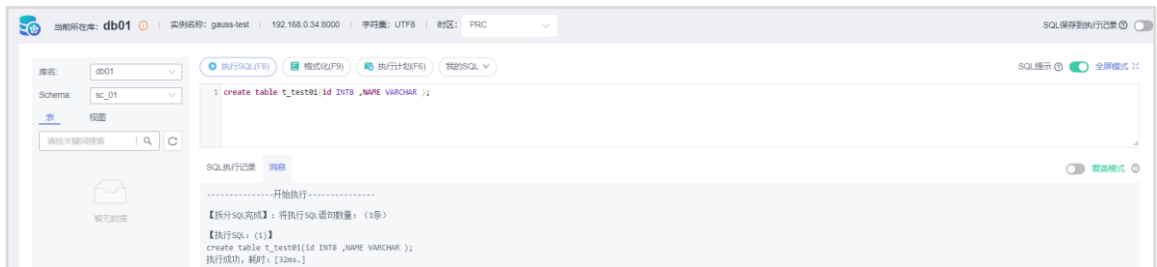
4.3.4 创建表

步骤 1 点击“SQL 窗口”按钮，跳转到“SQL 查询”页签，如下图所示：



步骤 2 在页面左侧选择刚才创建的“sc_01” schema，并键入创建表的 SQL 语句。点击“执行 SQL”，显示“执行成功”。

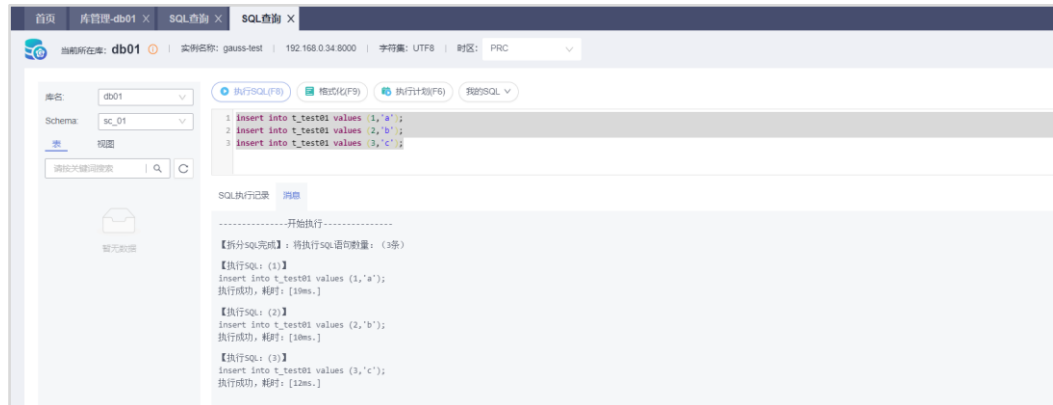
```
create table t_test01(id INT8 ,NAME VARCHAR );
```



4.3.5 插入数据

步骤 1 在当前页面，向创建的表中插入数据。点击“执行 SQL”。

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



步骤 2 查询已插入的数据。

```
select * from t_test01;
```



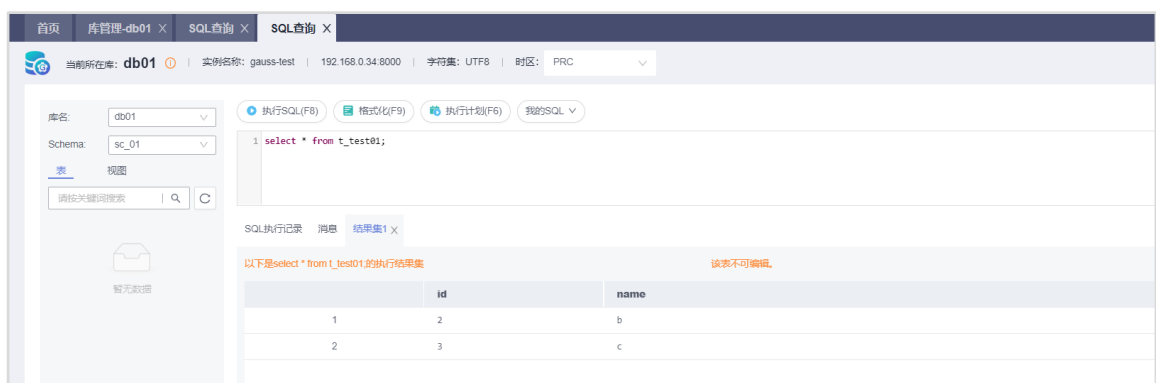
4.3.6 删除数据

步骤 1 键入 delete 语句，删除 id=1 的数据。

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




步骤 2 查看原表所有数据，确认 id=1 的记录已被删除。



4.3.7 修改数据

步骤 1 将 t_test01 表中，id=2 的记录，name 修改为 z。键入 update 语句进行修改。

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



步骤 2 查看修改结果。



4.4 实验小结

本实验通过实操进行了云数据库 GaussDB(for openGauss)的实例购买，并通过数据管理服务 DAS 进行数据库实例连接。通过在 DAS 上进行数据库、schema、表的创建，以及对表中的数据进行增删改查的简单操作，熟悉了基本 SQL 语句的使用方法，并对数据管理服务 DAS 工具的使用进行了解。

4.5 思考题

如何在数据库管理服务 DAS 上查看 SQL 语句的执行计划？尝试使用查看执行计划功能，模拟出在表上创建索引使得执行计划变优的场景。

5 综合实验

5.1 金融数据模型

假设 A 市 C 银行为了方便对银行数据的管理和操作，引入了 openGauss 数据库。针对 C 银行的业务，本实验主要将对象分为客户、银行卡、理财产品、保险和基金。因此，针对这些数据库对象，本实验假设 C 银行的金融数据库存在着以下关系：客户可以办理银行卡，同时客户可以购买不同的银行产品，如理财产品，基金和保险。那么，根据 C 银行的对象关系，本实验给出了相应的关系模式和 ER 图，并对其进行较为复杂的数据库操作。

5.1.1 E-R 图

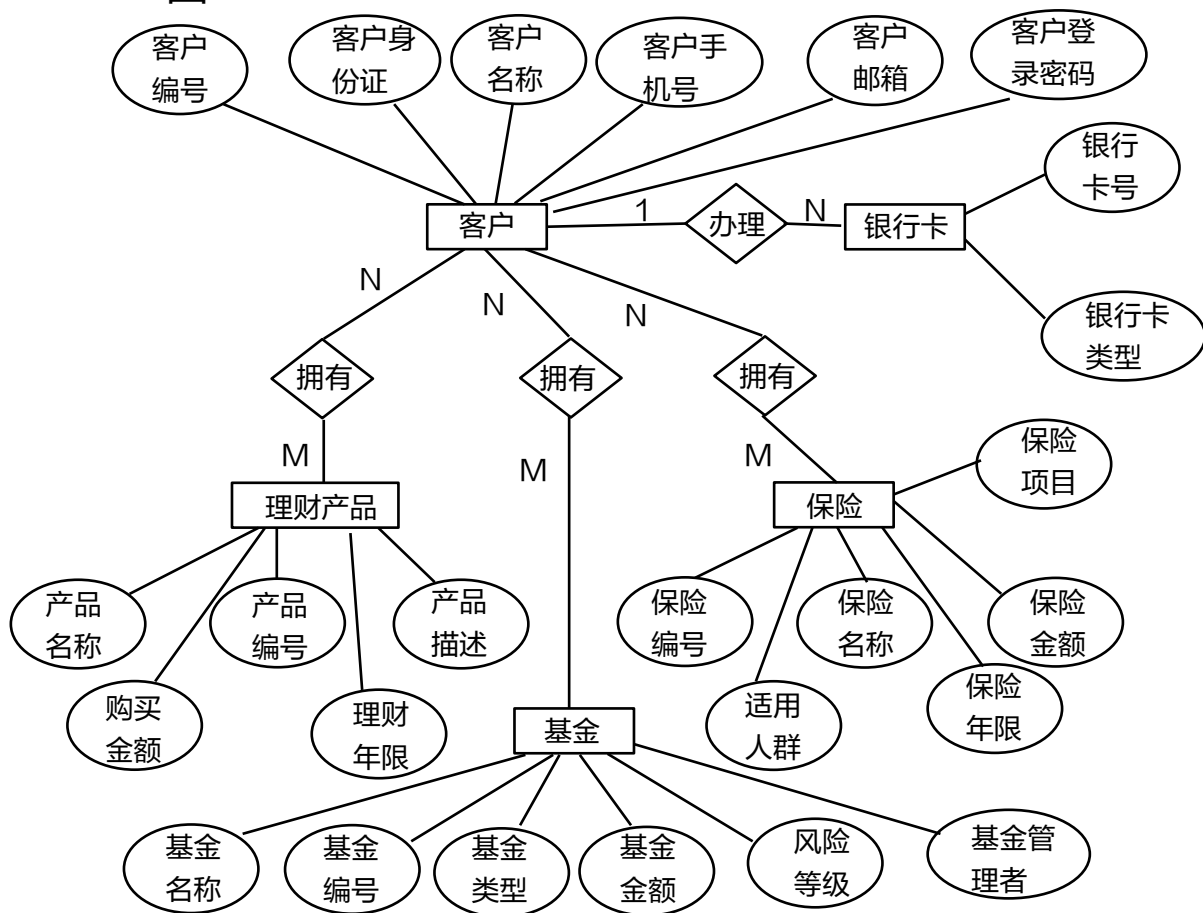


图5-1 E-R 图

5.1.2 关系模式

对于 C 银行中的 5 个对象，分别建立属于每个对象的属性集合，具体属性描述如下：

- 客户（客户编号、客户名称、客户邮箱，客户身份证，客户手机号，客户登录密码）
- 银行卡（银行卡号，银行卡类型）
- 理财产品（产品名称，产品编号，产品描述，购买金额，理财年限）
- 保险（保险名称，保险编号，保险金额，适用人群，保险年限，保障项目）
- 基金（基金名称，基金编号，基金类型，基金金额，风险等级，基金管理者）

对象之间的关系：

- 一个客户可以办理多张银行卡
- 一个客户可以购买多个理财产品，同一类理财产品可由多个客户购买
- 一个客户可以购买多个基金，同一类基金可由多个客户购买
- 一个客户可以购买多个保险，同一类保险可由多个客户购买

根据关系分析，设计关系模式如下：

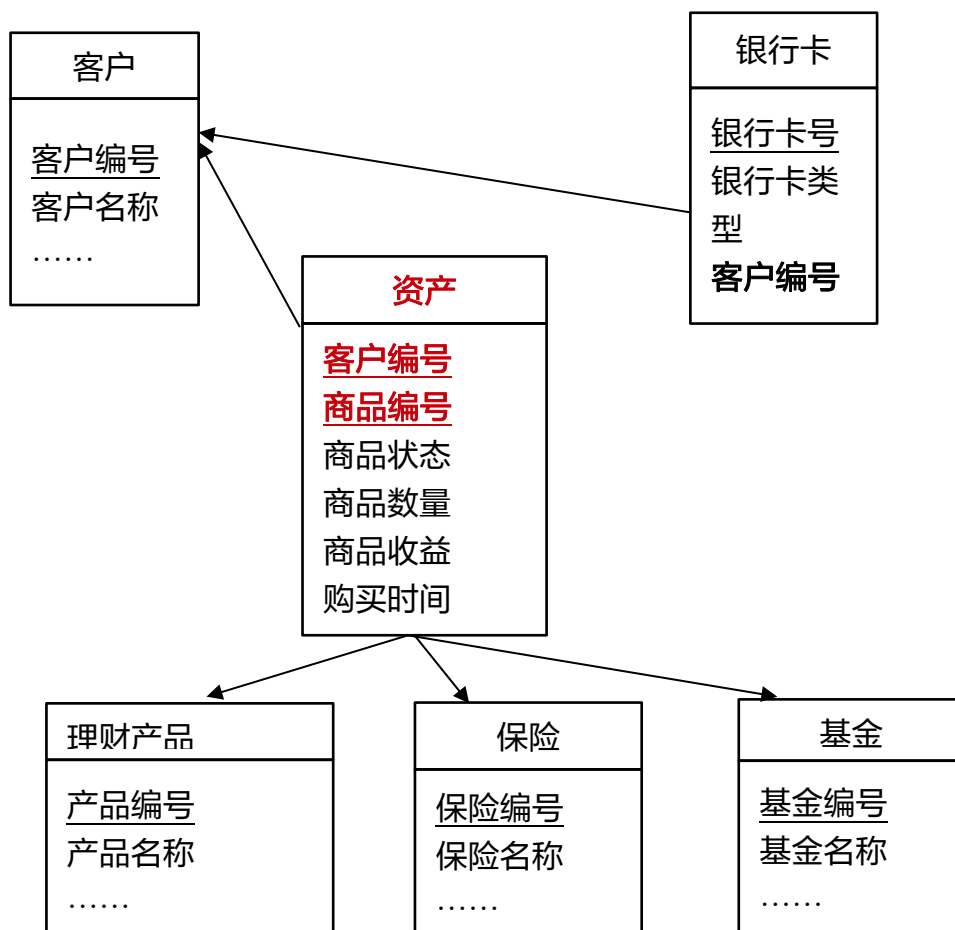


图5-2 金融数据关系模型设计图

说明：

- 由于一个客户可以办理多张银行卡，所以银行卡表引用客户表的客户编号作为外键。
- 由于一个客户可以购买多个理财产品，同一类理财产品可由多个客户购买。所以生成关系表——资产表。资产表引用客户表的商品编号作为外键，引用理财产品表的产品编号作为外键，并且添加商品状态、商品数量、商品收益和购买时间等属性。
- 客户和保险、客户和基金的关系同理，所以资产表同样作为生成的关系表，修改资产表的理财产品编号为商品编号，商品编号引用自理财产品表、保险和基金表的编号。

5.1.3 物理模型

对象及属性对应的编号为：

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

说明：

- 资产表（property）中由于商品编号(pro_pif_id)字段引用自理财产品表、保险和基金表的 id 字段，为了防止三商品的 id 字段相互冲突，添加商品类型(pro_type)字段区分三种商品。并且资产表添加资产编号(pro_id)字段作为主键。

表5-1 Client（客户）表

字段名称	字段类型	约束	说明
c_id	INTEGER	PRIMARY KEY	客户编码
c_name	VARCHAR(100)	NOT NULL	客户名称
c_mail	CHAR(30)	UNIQUE	客户邮箱
c_id_card	CHAR(20)	UNIQUE NOT NULL	客户身份证
c_phone	CHAR(20)	UNIQUE NOT NULL	客户手机号
c_password	CHAR(20)	NOT NULL	客户登录密码

表5-2 bank_card（银行卡）表

字段名称	字段类型	约束	说明
b_number	CHAR(30)	PRIMARY KEY	银行卡号
b_type	CHAR(20)		银行卡类型
b_c_id	INTEGER	NOT NULL FOREIGN KEY	所属客户编号 注：本字段引用自client表的c_id字段。

表5-3 finances_product（理财产品）信息表

字段名称	字段类型	约束	
p_name	VARCHAR(100)	NOT NULL	产品名称
p_id	INTEGER	PRIMARY KEY	产品编号
p_description	VARCHAR(4000)		产品描述
p_amount	INTEGER		购买金额
p_year	INTEGER		理财年限

表5-4 insurance（保险）表

字段名称	字段类型	约束	说明
i_name	VARCHAR(100)	NOT NULL	保险名称
i_id	INTEGER	PRIMARY KEY	保险编号
i_amount	INTEGER		保险金额
i_person	CHAR(20)		适用人群
i_year	INTEGER		保险年限
i_project	VARCHAR(200)		保障项目

表5-5 fund（基金）表

字段名称	字段类型	约束	说明
f_name	VARCHAR(100)	NOT NULL	基金名称

f_id	INTEGER	PRIMARY KEY	基金编号
f_type	CHAR(20)		基金类型
f_amount	INTEGER		基金金额
risk_level	CHAR(20)	NOT NULL	风险等级
f_manager	INTEGER	NOT NULL	基金管理者

表5-6 property（资本）表

字段名称	字段类型	约束	
pro_id	INTEGER	PRIMARY KEY	资产编号
pro_c_id	VARCHAR(100)	NOT NULL FOREIGN KEY	客户编号 说明：本字段引用自client表的c_id字段。
pro_pif_id	INTEGER	NOT NULL FOREIGN KEY	商品编号 说明：本字段引用自finances_product表、insurance表和fund表三个表的id字段。
pro_type	INTEGER	NOT NULL	商品类型 说明：1表示理财产品；2表示保险；3表示基金。
pro_status	CHAR(20)		商品状态
pro_quantity	INTEGER		商品数量
pro_income	INTEGER		商品收益
pro_purchase_time	DATE		购买时间

接下来进行 openGauss 数据模型表操作。

5.1.4 连接数据库设置

数据库创建完成后，需要配置数据库的监听、设置数据库的白名单后，才能使用客户端如可视化的 Data Studio、JDBC 或者 ODBC 等方式连接数据库。

步骤 1 修改数据库的 pg_hba.conf 文件。

在 GS_HOME 中查找 pg_hba.conf 文件，本实验中数据库 GS_HOME 设置的为 /gaussdb/data，实际操作中 GS_HOME 地址可以查看安装时的配置文件：<PARAM name="dataNode1" value="/gaussdb/data"/>。

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

找到对应位置，然后输入“i”切换到 INSERT 模式，将以下内容添加进 pg_hba.conf 文件，添加后按下“ESC”键，退出 INSERT 模式，输入“:wq”后回车保存。此处 host all all 0.0.0.0/0 sha256 的含义对应为“host DATABASE USER ADDRESS METHOD”，即 host 的方式；对数据库 DATABASE 和用户 USER 都是 all 的设置；对地址设置为 0.0.0.0/0 即全部的网段，采用的加密使用 sha256 的加密方式。

```
# 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
```

步骤 2 修改数据库监听地址。

在 GS_HOME 中，本实验中数据库 GS_HOME 设置的为 /gaussdb/data。

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

找到对应位置，然后输入“i”切换到 INSERT 模式，将 listen_addresses 的值修改成为*，修改后按下“ESC”键，退出 INSERT 模式，输入“:wq”后回车保存。设置为*表明，监听所有的 IP 请求。

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

修改参数信息，也可以使用 gs_guc 命令进行参数的修改，如果使用直接修改 postgresql.conf 文件，可直接执行下一步骤，步骤 3。

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

获得结果为：

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!

步骤 3 重启数据库，使参数生效

修改完成后重启数据库生效（-D 后面的数据库默认路径，需要根据实际情况进行修改，本实验中数据库 GS_HOME 设置的为/gaussdb/data）。

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

5.1.5 创建数据库及 schema

根据 C 银行的场景描述，统一创建数据库命名为：finance，并创建对应的模式(schema)：finance。具体的步骤如下：

步骤 1 创建金融数据库 finance。

切换到 omm 用户（如果当前是 omm 用户无需切换），以操作系统用户 omm 登录数据库主节点。

```
su - omm
```

步骤 2 使用 gsql 工具登录数据库。

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

步骤 3 创建金融数据库 finance。

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

步骤 4 连接 finance 数据库。

```
\connect finance
```

步骤 5 创建名为 finance 的 schema，并设置 finance 为当前的 schema。

```
CREATE SCHEMA finance;
```

步骤 6 使用元命令查看数据库

```
\l
```

获得的结果为：

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)					

步骤 7 将 finance 数据库默认的模式搜索路径修改为 public 和 finance。

```
ALTER DATABASE finance SET search_path TO finance,public;
```

步骤 8 查看 finance 数据库默认的模式搜索路径。

```
show search_path
```

获得结果为:

```
search_path
```

```
-----
```

```
finance, public
```

```
(1 row)
```

步骤 9 退出数据库:

```
\q
```

5.1.6 创建、设置表空间

表空间的设置,是为了更好从逻辑上管理数据库中的表。为 finance 数据库创建 finance_tbs 表空间,并将 finance_tbs 表空间设置为 finance 默认的表空间。

步骤 1 连接数据库。

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

步骤 2 创建表空间。

```
CREATE TABLESPACE finance_tbs LOCATION '/opt/opengauss/tablespace/finance_tbs1';
```

步骤 3 使用元命令查看表空间。

```
\db
```

获得结果为:

List of tablespaces

Name	Owner	Location
------	-------	----------

```
-----+-----+-----
```

finance_tbs	omm	/opt/opengauss/tablespace/finance_tbs1
-------------	-----	--

pg_default	omm	
------------	-----	--

pg_global	omm	
-----------	-----	--

```
(3 rows)
```

步骤 4 查询表空间使用率

查询表空间的当前使用情况,其中得到的结果表示表空间的大小,单位为字节。

```
SELECT PG_TABLESPACE_SIZE('finance_tbs');
```

获得结果为:

```
pg_tablespace_size
```

```
-----  
4096  
(1 row)
```

步骤 5 将创建的表空间设置为 finance 数据库默认的表空间。

```
ALTER DATABASE finance SET tablespace finance_tbs;
```

得到如下的报错：

```
ERROR: cannot change the tablespace of the currently open database
```

原因为连接在 finance 数据库上的会话没法修改数据库默认表空间，因此需要切换会话连接到默认的 postgres 数据库上，再进行修改。

```
\connect postgres
```

再次修改 finance 默认表空间

```
ALTER DATABASE finance SET tablespace finance_tbs;
```

返回结果为，表示修改成功：

```
ALTER DATABASE
```

步骤 6 退出数据库：

```
\q
```

5.1.7 新用户的创建和授权

在本章中，假设 C 银行的某个应用需要访问数据库，因此针对该业务创建 bank_app 用户，并对该用户授予一定的权限，具体操作如下：

步骤 1 连接数据库。

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

步骤 2 连接数据库后，进入 SQL 命令界面。创建用户 bank_app，密码为 Gauss#3demo。

```
CREATE USER bank_app IDENTIFIED BY 'Gauss#3demo';
```

步骤 3 切换到 finance 数据库。

```
\connect finance;
```

步骤 4 将 SCHEMA 的权限也授予 bank_app 用户。

```
GRANT ALL ON SCHEMA finance to bank_app;
```

步骤 5 退出数据库：

```
\q
```

5.1.8 新用户连接数据库

步骤 1 在 gsql 登录数据库，使用新用户连接。

本实验通过 gsql 方式登录数据库，同学们也可以使用其他客户端如 Data Studio 连接数据库，本实验暂不涉及，感兴趣的同学可以自行学习 Data Studio 的使用手册 (<https://opengauss.obs.cn-south-1.myhuaweicloud.com/2.0.0/Data%20Studio%20%E7%94%A8%E6%88%B7%E6%89%8B%E5%86%8C.pdf>)。

使用操作系统 omm 用户在新的窗口登陆并执行以下命令，并输入对应的密码(如：Gauss#3demo)。

```
gsql -d finance -U bank_app -p 26000 -r
```

步骤 2 访问 finance 数据库的表 bank_card。

```
\du;
```

结果如下：

```
List of roles
Role name | Attributes | Member of
-----+-----+-----
bank_app |           | {}
```

步骤 3 将默认搜索路径设为 finance。

(1.1.5 中设置过的，可以不用重新设置)

```
SET search_path TO finance;
```

步骤 4 查看 search_path

```
show search_path;
```

获得结果为：

```
search_path
```

```
-----
```

```
finance
```

```
(1 row)
```

5.1.9 创建数据表

根据 C 银行的场景描述，本实验分别针对客户(client)，银行卡(bank_card)，理财产品(finances_product)，保险(insurance)，基金(fund)和资产(property)创建相应的表。具体的实验步骤如下所示：

步骤 1 客户信息表的创建。

在上一个实验中，未退出连接，可以执行创建表的 SQL，如果退出连接，可以重新执行 1.1.7 的步骤。

在 SQL 编辑框中输入如下语句，创建客户信息表 client。

删除表 client。

```
DROP TABLE IF EXISTS finance.client;
```

创建表 client。

```
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
);
```

添加注释。

```
COMMENT ON TABLE finance.client IS '客户表';
```

步骤 2 银行卡信息表的创建。

在 SQL 编辑框中输入如下语句，创建银行卡信息表 bank_card。

删除表 bank_card。

```
DROP TABLE IF EXISTS finance.bank_card;
```

创建表 bank_card 并添加注释。

```
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 '银行卡表';
```

步骤 3 理财产品信息表的创建。

创建理财产品信息表 finances_product。

删除表 finances_product。

```
DROP TABLE IF EXISTS finance.finances_product;
```

创建表 finances_product。

```
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 '理财产品表';
```

步骤 4 保险信息表的创建。

在 SQL 编辑框中输入如下语句，创建保险信息表 insurance。

删除表 insurance。

```
DROP TABLE IF EXISTS finance.insurance;
```

创建表 insurance。

```
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 '保险信息表';
```

步骤 5 基金信息表的创建。

在 SQL 编辑框中输入如下语句，创建保险信息表 fund。

删除表 fund。

```
DROP TABLE IF EXISTS finance.fund;
```

创建表 fund。

```
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 '基金信息表';
```

步骤 6 资产信息表的创建。

在 SQL 编辑框中输入如下语句，创建资产信息表 property。

删除表 property。

```
DROP TABLE IF EXISTS finance.property;
```

创建表 property。

```
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 '资产信息表';
```

步骤 7 查看创建的表

```
\d+;
```

获得结果为：

List of relations						
Schema	Name	Type	Owner	Size	Storage	
Description						
-----+-----+-----+-----+-----+-----+-----						

finance	bank_card	table	bank_app	8192 bytes	{orientation=row,compression=no} 银行卡	
finance	client	table	bank_app	8192 bytes	{orientation=row,compression=no} 客户表	
finance	finances_product	table	bank_app	16 kB	{orientation=row,compression=no} 理财产品表	
finance	fund	table	bank_app	8192 bytes	{orientation=row,compression=no} 基金信息表	
finance	insurance	table	bank_app	8192 bytes	{orientation=row,compression=no} 保险信息表	
finance	property	table	bank_app	8192 bytes	{orientation=row,compression=no} 资产信息表	
(6 rows)						

5.1.10 插入表数据

为了实现对表数据的相关操作，本实验需要以执行 SQL 语句的方式对金融数据库的相关表插入部分数据。

步骤 1 对 client 表进行数据初始化。

执行 insert 操作。

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (1,'左晓婷',
'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,'虞成刚',
'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,'朱长刚',
'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,'任高峰',
'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,'安艳芳',
'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,'滕长丽',
'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,'傅小芳',
'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,'卞兰娟',
'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,'邵小婷',
'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,'章晓峰',
'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,'康晓鹏',
'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,'冯长强',
'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,'应晓鹏',
'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,'赵晓鹏',
'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,'苏小刚',
'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,'尹高峰',
'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,'孟小轩',
'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,'唐高伟',
'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,'洪高鹏',
'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,'宋成峰',
'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,'邱长强',
'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,'卫高鹏',
'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,'潘成轩',
'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,'季小梅','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,'李春婷','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,'倪兰芳','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,'姜小芳','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,'李兰芳','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,'娄晓婷','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,'傅晓婷','fuxiaoting@huawei.com','340211199301010030','18815650030','gaussdb_030');
```

查询插入结果。

```
select count(*) from finance.client;
```

结果为:

```
count(*)
```

```
-----
```

```
30
```

步骤 2 对 bank_card 表进行数据初始化。

执行 insert 操作。

```
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000001','信用卡',1);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000002','信用卡',3);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000003','信用卡',5);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000004','信用卡',7);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000005','信用卡',9);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000006','信用卡',10);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000007','信用卡',12);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000008','信用卡',14);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000009','信用卡',16);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000010','信用卡',18);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000011','储蓄卡',19);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000012','储蓄卡',21);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000013','储蓄卡',7);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000014','储蓄卡',23);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000015','储蓄卡',24);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000016','储蓄卡',3);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000017','储蓄卡',26);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000018','储蓄卡',27);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000019','储蓄卡',12);
INSERT INTO finance.bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000020','储蓄卡',29);
```

查询插入结果。

```
select count(*) from finance.bank_card;
```

结果为：

```
count(*)
```

```
-----
```

```
20
```

步骤 3 对 finances_product 表进行数据初始化。

执行 insert 操作。

```
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('债券',1,'以国债、金融债、央行票据、企业债为主要投资方向的银行理财产品。',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('信贷资产',2,'一般指银行作为委托人将通过发行理财产品募集资金委托给信托公司，信托公司作为受托人成立信托计划，将信托资产购买理财产品发售银行或第三方信贷资产。',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('股票',3,'与股票挂钩的理财产品。目前市场上主要以港股挂钩居多',50000,6);
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('大宗商品',4,'与大宗商品期货挂钩的理财产品。目前市场上主要以挂钩黄金、石油、农产品的理财产品居多。',50000,6);
```

查询插入结果。

```
select count(*) from finance.finances_product;
```

结果为：

```
count(*)
```

```
-----
```

```
4
```

步骤 4 对 insurance 表进行数据初始化。

执行 insert 操作。

```
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('健康保险',1,2000,'老人',30,'平安保险');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('人寿保险',2,3000,'老人',30,'平安保险');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('意外保险',3,5000,'所有人',30,'平安保险');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('医疗保险',4,2000,'所有人',30,'平安保险');
INSERT INTO finance.insurance(i_name,i_id,i_amount,i_person,i_year,i_project) VALUES ('财产损失保险',5,1500,'中年人',30,'平安保险');
```

查询插入结果。

```
select count(*) from finance.insurance;
```

结果为：

```
count(*)
```

```
-----
```

```
5
```

步骤 5 对 fund 表进行数据初始化。

执行 insert 操作。

```
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('股票',1,'股票型',10000,'高',1);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('投资',2,'债券型',10000,'中',2);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('国债',3,'货币型',10000,'低',3);
INSERT INTO finance.fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('沪深 300 指数',4,'指数型',10000,'中',4);
```

查询插入结果。

```
select count(*) from finance.fund;
```

结果为:

```
count(*)
```

```
-----
```

```
4
```

步骤 6 对 property 表进行数据初始化。

执行 insert 操作。

```
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,'可用',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,'可用',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,'可用',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,'冻结',4,8000,'2018-07-01');
```

查询插入结果。

```
select count(*) from finance.property;
```

结果为:

```
count(*)
```

```
-----
```

```
4
```

5.1.11 手工插入一条数据

当 C 银行有新的信息需要加入数据库时，系统需要在对应的数据表中手动插入一条新的数据。因此，针对主键属性定义的场景，介绍如何手动插入一条数据。

步骤 1 在金融数据库的客户信息表中添加一个客户的信息。（属性冲突的场景）

c_id_card 和 c_phone 非唯一。

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (31,'李丽','lili@huawei.com','340211199301010005','18815650005','gaussdb_005');
```

错误信息如下：

ERROR: duplicate key value violates unique constraint "client_c_id_card_key"

DETAIL: Key (c_id_card)=(340211199301010005) already exists.

说明：由于在表的创建过程中，实验定义了 c_id_card 和 c_phone 为唯一且非空（UNIQUE NOT NULL），所以当表中存在相同记录时，插入数据失败。

步骤 2 在金融数据库的客户信息表中添加一个客户的信息，满足表上的约束条件。（插入成功的场景）。

插入成功的示例。

```
INSERT INTO finance.client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (31,'李丽','lili@huawei.com','340211199301010031','18815650031','gaussdb_031');
```

步骤 3 验证插入结果

```
select * from finance.client where c_id=31;
```

得到结果为：

c_id	c_name	c_mail	c_id_card	c_phone	c_password
31	李丽	lili@huawei.com	340211199301010031	18815650031	gaussdb_031

5.1.12 添加约束

步骤 1 对表添加外键约束，在银行信息表和资产信息表中，都存在每个银行卡必须有一个持卡人者、每份资产必须都有一个资产拥有者这样的对应关系。因此针对这种对应关系，创建外键约束。

给表 bank_card 添加外键约束

```
ALTER TABLE finance.bank_card ADD CONSTRAINT fk_c_id FOREIGN KEY (b_c_id) REFERENCES finance.client(c_id) ON DELETE CASCADE;
```

给表 property 添加外键约束

```
ALTER TABLE finance.property ADD CONSTRAINT fk_pro_c_id FOREIGN KEY (pro_c_id) REFERENCES
finance.client(c_id) ON DELETE CASCADE;
```

备注：

- 银行卡信息表中的 b_c_id 与客户信息表中的 c_id 一致，且每个银行卡都必须有一个持卡人。
- 在进行表删除时，需要先删除 bank_card 表，再删除 client 表，因为两个表存在约束。
- 资产信息表中的 pro_c_id 与客户信息表中的 c_id 一致，且每一份资产都必须有一个资产拥有者。
- 在进行表删除时，需要先删除 property 表，再删除 client 表，因为两个表存在约束。
- 在实际应用系统中，外键约束几乎不会创建在数据库表上，一切外键概念都在应用层解决。

步骤 2 在理财产品表、保险信息表和基金信息表中，都存在金额这个属性，在现实生活中，金额不会存在负数。因此针对表中金额的属性，增加大于 0 的约束条件。

为 finances_product 表的 p_amount 列添加大于等于 0 的约束。

```
ALTER table finance.finances_product ADD CONSTRAINT c_p_mount CHECK (p_amount >=0);
```

步骤 3 尝试手工插入一条金额小于 0 的记录。

```
INSERT INTO finance.finances_product(p_name,p_id,p_description,p_amount,p_year) VALUES ('信贷资产',10,'一般指银行作为委托人将通过发行理财产品募集资金委托给信托公司，信托公司作为受托人成立信托计划，将信托资产购买理财产品发售银行或第三方信贷资产。',-10,6);
```

执行失败，失败原因：

ERROR: new row for relation "finances_product" violates check constraint "c_p_mount"

DETAIL: Failing row contains (信贷资产, 10, 一般指银行作为委托人将通过发行理财产品募集资金..., -10, 6).

步骤 4 向 fund 表添加约束。

为 fund 表的 f_amount 列添加大于等于 0 的约束。

```
ALTER table finance.fund ADD CONSTRAINT c_f_mount CHECK (f_amount >=0);
```

步骤 5 向 insurance 表添加约束。

为 insurance 表的 i_amount 列添加大于等于 0 的约束。

```
ALTER table finance.insurance ADD CONSTRAINT c_i_mount CHECK (i_amount >=0);
```

5.1.13 查询数据

在本章的金融数据库实验中，主要目的是为了让读者学习到更深一层的查询操作，让学习者能够更深入的去了解 openGauss 数据库的复杂操作。

步骤 1 单表查询。

- 查询理财产品中有哪些理财产品。

```
SELECT * from finance.finances_product;
```

结果如下:

p_name	p_id	p_description	p_amount	p_year
债券	1	以国债、金融债、央行票据、企业债为主要投资方向的银行理财产品。	50000	6
信贷资产	2	一般指银行作为委托人将通过发行理财产品募集资金委托给信托公司，信托公司作为受托人成立信托计划，将信托资产购买理财产品发售银行或第三方信贷资产。	50000	6
股票	3	与股票挂钩的理财产品。目前市场上主要以港股挂钩居多	50000	6
大宗商品	4	与大宗商品期货挂钩的理财产品。目前市场上主要以挂钩黄金、石油、农产品的理财产品居多。	50000	6

(4 rows)

步骤 2 条件查询。

- 查询资产信息中‘可用’的资产数据。

```
SELECT * from finance.property where pro_status='可用';
```

结果如下:

pro_id	pro_c_id	pro_pif_id	pro_type	pro_status	pro_quantity	pro_income	pro_purchase_time
1	5	1	1	可用	4		8000 2018-07-01 00:00:00
2	10	2	2	可用	4		8000 2018-07-01 00:00:00
3	15	3	3	可用	4		8000 2018-07-01 00:00:00

(3 rows)

步骤 3 聚合查询。

- 查询用户表中有多少个用户。

```
SELECT count(*) FROM finance.client;
```

结果如下：

count

31

- 查询银行卡信息表中，储蓄卡和信用卡的个数。

```
SELECT b_type,COUNT(*) FROM finance.bank_card GROUP BY b_type;
```

结果如下：

b_type	count
储蓄卡	10
信用卡	10

- 查询保险信息表中，保险金额的平均值。

```
SELECT AVG(i_amount) FROM finance.insurance;
```

结果如下：

avg
2700.0000000000000000

- 查询保险信息表中保险金额的最大值和最小值所对应的险种和金额。

```
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);
```

结果如下：

i_name	i_amount
财产损失保险	1500
意外保险	5000

步骤 4 连接查询。

(1) 半连接。

- 查询用户编号在银行卡表中出现的用户的编号，用户姓名和身份证。

```
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);
```

结果如下：

c_id	c_name	c_id_card
-----+-----+-----		

```

1 | 左晓婷 | 340211199301010001
3 | 朱长刚 | 340211199301010003
5 | 安艳芳 | 340211199301010005
7 | 傅小芳 | 340211199301010007
9 | 邵小婷 | 340211199301010009
10 | 章晓峰 | 340211199301010010
12 | 冯长强 | 340211199301010012
14 | 赵晓鹏 | 340211199301010014
16 | 尹高峰 | 340211199301010016
18 | 唐高伟 | 340211199301010018
19 | 洪高鹏 | 340211199301010019
21 | 邱长强 | 340211199301010021
23 | 潘成轩 | 340211199301010023
24 | 季小梅 | 340211199301010024
26 | 倪兰芳 | 340211199301010026
27 | 姜小芳 | 340211199301010027
29 | 娄晓婷 | 340211199301010029

```

(17 rows)

备注：半连接是一种特殊的连接类型，在 SQL 中没有指定的关键字，通过在 WHERE 后面使用 IN 或 EXISTS 子查询实现。当 IN/EXISTS 右侧的多行满足子查询的条件时，主查询也只返回一行与 EXISTS 子查询匹配的行，而不是复制左侧的行。

(2) 反连接。

- 查询银行卡号不是'622202130202000001*'（*表示未知）的用户的编号，姓名和身份证。

```
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 '622202130202000001_*');
```

结果如下：

```

c_id | c_name |      c_id_card
-----+-----+-----
1 | 左晓婷 | 340211199301010001
2 | 虞成刚 | 340211199301010002
3 | 朱长刚 | 340211199301010003
4 | 任高峰 | 340211199301010004
5 | 安艳芳 | 340211199301010005
6 | 滕长丽 | 340211199301010006

```



```

7 | 傅小芳 | 340211199301010007
8 | 卞兰娟 | 340211199301010008
9 | 邵小婷 | 340211199301010009
10 | 章晓峰 | 340211199301010010
11 | 康晓鹏 | 340211199301010011
12 | 冯长强 | 340211199301010012
13 | 应晓鹏 | 340211199301010013
14 | 赵晓鹏 | 340211199301010014
15 | 苏小刚 | 340211199301010015
16 | 尹高峰 | 340211199301010016
17 | 孟小轩 | 340211199301010017
18 | 唐高伟 | 340211199301010018
19 | 洪高鹏 | 340211199301010019
20 | 宋成峰 | 340211199301010020
21 | 邱长强 | 340211199301010021
22 | 卫高鹏 | 340211199301010022
23 | 潘成轩 | 340211199301010023
24 | 季小梅 | 340211199301010024
25 | 李春婷 | 340211199301010025
26 | 倪兰芳 | 340211199301010026
27 | 姜小芳 | 340211199301010027
28 | 李兰芳 | 340211199301010028
29 | 娄晓婷 | 340211199301010029
30 | 傅晓婷 | 340211199301010030
31 | 李丽 | 340211199301010031

```

(31 rows)

备注：反连接是一种特殊的连接类型，在 SQL 中没有指定的关键字，通过在 WHERE 后面使用 NOT IN 或 NOT EXISTS 子查询实现。返回所有不满足条件的行。这个关系的概念跟半连接相反。

步骤 5 子查询。

- 通过子查询，查询保险产品中保险金额大于平均值的保险名称和适用人群。

```
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);
```

结果如下：

```

i_name | i_amount | i_person
-----+-----+-----
  人寿保险 |      3000 |  老人
  意外保险 |      5000 | 所有人
(2 rows)

```

步骤 6 ORDER BY 和 GROUP BY。

(1) ORDER BY 子句。

- 按照保额降序查询保险编号大于 2 的保险名称，保额和适用人群。

```
SELECT i_name,i_amount,i_person FROM finance.insurance WHERE i_id>2 ORDER BY i_amount DESC;
```

结果如下：

```

i_name | i_amount | i_person
-----+-----+-----
  意外保险 |      5000 | 所有人
  医疗保险 |      2000 | 所有人
  财产损失保险 |      1500 | 中年人
(3 rows)

```

(2) GROUP BY 子句。

- 查询各理财产品信息总数，按照 p_year 分组。

```
SELECT p_year,count(p_id) FROM finance.finances_product GROUP BY p_year;
```

结果如下：

```

p_year | count
-----+-----
      6 |      4

```

步骤 7 HAVING 和 WITH AS。

(1) HAVING 子句。

- 查询保险金额统计数量等于 2 的适用人群数。

```
SELECT i_person,count(i_amount) FROM finance.insurance GROUP BY i_person HAVING count(i_amount)=2;
```

结果如下：

```

i_person | count
-----+-----
  所有人 |      2
  老人   |      2

```

备注：HAVING 子句依附于 GROUP BY 子句而存在。

(2) WITH AS 子句。

- 使用 WITH AS 查询基金信息表。

```
WITH temp AS (SELECT f_name,ln(f_amount) FROM finance.fund ORDER BY f_manager DESC) SELECT *
FROM temp;
```

结果如下：

f_name	ln
沪深 300 指数	9.21034037197618
国债	9.21034037197618
投资	9.21034037197618
股票	9.21034037197618

备注：该语句为定义一个 SQL 片段，该 SQL 片段会被整个 SQL 语句用到。

可以使 SQL 语句的可读性更高。存储 SQL 片段的表与基本表不同，是一个虚表。数据库不存放对应的定义和数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从存储 SQL 片段的表中查询出的数据也随之改变。

5.1.14 视图

视图是一个**虚拟表**，是 sql 的查询结果，其内容由查询定义。对于来自多张关联表的复杂查询，就不得不使用十分复杂的 SQL 语句进行查询，造成极差的体验感。使用视图之后，可以极大的简化操作，使用视图不需要关心相应表的结构、关联条件等。

步骤 1 创建视图。

针对“查询用户编号在银行卡表中出现的用户的编号，用户姓名和身份证”的查询，创建视图。

```
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);
```

使用视图进行查询。

```
SELECT * FROM finance.v_client;
```

结果如下：

c_id	c_name	c_id_card
1	左晓婷	340211199301010001
3	朱长刚	340211199301010003
5	安艳芳	340211199301010005
7	傅小芳	340211199301010007
9	邵小婷	340211199301010009
10	章晓峰	340211199301010010

```

12 | 冯长强 | 340211199301010012
14 | 赵晓鹏 | 340211199301010014
16 | 尹高峰 | 340211199301010016
18 | 唐高伟 | 340211199301010018
19 | 洪高鹏 | 340211199301010019
21 | 邱长强 | 340211199301010021
23 | 潘成轩 | 340211199301010023
24 | 季小梅 | 340211199301010024
26 | 倪兰芳 | 340211199301010026
27 | 姜小芳 | 340211199301010027
29 | 姜晓婷 | 340211199301010029

```

(17 rows)

步骤 2 修改视图内容

修改视图，在原有查询的基础上，过滤出信用卡用户。

```

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='信用卡');

```

使用视图进行查询。

```

select * from finance.v_client;

```

结果如下：

```

c_id | c_name |      c_id_card
-----+-----+-----
1 | 左晓婷 | 340211199301010001
3 | 朱长刚 | 340211199301010003
5 | 安艳芳 | 340211199301010005
7 | 傅小芳 | 340211199301010007
9 | 邵小婷 | 340211199301010009
10 | 章晓峰 | 340211199301010010
12 | 冯长强 | 340211199301010012
14 | 赵晓鹏 | 340211199301010014
16 | 尹高峰 | 340211199301010016
18 | 唐高伟 | 340211199301010018

```

(10 rows)

步骤 3 修改视图名称。

```
ALTER VIEW finance.v_client RENAME TO v_client_new;
```

步骤 4 删除视图。

将 v_client 视图删除，删除视图不影响基表。

```
DROP VIEW finance.v_client_new;
```

5.1.15 索引

步骤 1 创建索引。

- 在普通表 property 上创建索引。

```
CREATE INDEX finance.idx_property ON finance.property(pro_c_id DESC,pro_income,pro_purchase_time);
```

结果如下：

```
CREATE INDEX
```

步骤 2 使用元命令，查看创建的索引

```
\di idx_property
```

获得结果为：

List of relations

Schema	Name	Type	Owner	Table	Storage
finance	idx_property	index	bank_app	property	

(1 row)

步骤 3 重命名索引。

- 在普通表 property 上重建及重命名索引。

重建索引。

```
DROP INDEX finance.idx_property;
```

```
CREATE INDEX finance.idx_property ON finance.property(pro_c_id DESC,pro_income,pro_purchase_time);
```

重命名索引。

```
ALTER INDEX finance.idx_property RENAME TO idx_property_temp;
```

步骤 4 使用元命令，查看创建的索引

```
\di idx_property_temp
```

获得结果为：

List of relations

Schema	Name	Type	Owner	Table	Storage
finance	idx_property_temp	index	bank_app	property	

(1 row)

步骤 5 删除索引。

- 删除索引 idx_property_temp。

```
DROP INDEX finance.idx_property_temp;
```

5.1.16 数据的修改和删除

步骤 1 修改数据。

- 修改/更新银行卡信息表中 b_c_id 小于 10 和客户信息表中 c_id 相同的记录的 b_type 字段。

查看表数据。

```
SELECT * FROM finance.bank_card where b_c_id<10 ORDER BY b_c_id;
```

结果如下：

b_number	b_type	b_c_id
6222021302020000001	信用卡	1
6222021302020000016	储蓄卡	3
6222021302020000002	信用卡	3
6222021302020000003	信用卡	5
6222021302020000004	信用卡	7
6222021302020000013	储蓄卡	7
6222021302020000005	信用卡	9

(7 rows)

开始更新数据：

```
UPDATE finance.bank_card SET bank_card.b_type='借记卡' from finance.client where bank_card.b_c_id = client.c_id and bank_card.b_c_id<10;
```

重新查询数据情况。

```
SELECT * FROM finance.bank_card ORDER BY b_c_id;
```

结果如下：

b_number	b_type	b_c_id
6222021302020000001	借记卡	1
6222021302020000002	借记卡	3
6222021302020000016	借记卡	3
6222021302020000003	借记卡	5

6222021302020000013	借记卡	7
6222021302020000004	借记卡	7
6222021302020000005	借记卡	9
6222021302020000006	信用卡	10
6222021302020000007	信用卡	12
6222021302020000019	储蓄卡	12
6222021302020000008	信用卡	14
6222021302020000009	信用卡	16
6222021302020000010	信用卡	18
6222021302020000011	储蓄卡	19
6222021302020000012	储蓄卡	21
6222021302020000014	储蓄卡	23
6222021302020000015	储蓄卡	24
6222021302020000017	储蓄卡	26
6222021302020000018	储蓄卡	27
6222021302020000020	储蓄卡	29

步骤 2 删除指定数据。

- 删除基金信息表中编号小于 3 的行。

删除前查询结果。

```
SELECT * FROM finance.fund;
```

结果如下：

f_name	f_id	f_type	f_amount	risk_level	f_manager
股票	1	股票型	10000	高	1
投资	2	债券型	10000	中	2
国债	3	货币型	10000	低	3
沪深 300 指数	4	指数型	10000	中	4

开始删除数据：

```
DELETE FROM finance.fund WHERE f_id<3;
```

查询删除结果。

```
SELECT * FROM finance.fund;
```

结果如下：

f_name	f_id	f_type	f_amount	risk_level	f_manager
沪深 300 指数	4	指数型	10000	中	4

国债		3	货币型		10000	低		3
沪深 300 指数		4	指数型		10000	中		4

步骤 3 退出数据库

```
\q
```

5.1.17 删除 Schema 和删除 Database

此类操作一定确认系统需要下线，并且不做保留时，方可执行删除操作，切勿在生产环境上进行实验。

步骤 1 使用管理员用户登陆 finance 数据库。

通过 gsql 来登录 finance 数据库，新建 session。

```
gsql -d finance -p 26000 -r
```

步骤 2 使用“\dn”查看数据库下的 schema。

```
\dn
```

查询结果为：

```
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)
```

步骤 3 使用“\dt”命令可以看到在 finance 中的对象。

```
\dt
```

查询结果为：

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)
```

步骤 4 使用 DROP SCHEMA 命令删除 finance 会有报错，因为 finance 下存在对象。

```
DROP SCHEMA finance;
```

报错如下：

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.

步骤 5 使用 DROP SCHEMA.....CASCADE 删除，会将 finance 连同下的对象一起删除。

```
DROP SCHEMA finance CASCADE;
```

结果如下：

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

步骤 6 使用“\dt”命令可以看到在 finance 和 public 中的对象，对象已删除。

```
\dt
```

结果为：

No relations found.

步骤 7 删除数据库。

```
DROP DATABASE finance;
```

结果为：

```
DROP DATABASE
```

步骤 8 使用元命令查看数据库。

```
\l
```

获得结果为，可以看到 finance 数据库已被删除：

```

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)

```

步骤 9 退出数据库：

```
\q
```

5.2 实验小结

本实验通过 ER 模型加深对数据库的理解和数据库设计知识的掌握，通过 SQL 语句的练习，使学员熟练掌握 SQL 语法。

5.3 思考题

问题 1、comment 的作用是什么？是否一定需要？

答案：comment 用于解释说明，可以对表、字段等信息进行解释说明；comment 不是一定要设置，但是建议对表和字段加以说明，方便运维管理。

问题 2：在查询中 where 条件子句和 having 条件子句有何不同？

where 不能使用聚合函数、having 中可以使用聚合函数

问题 3：查看索引时，发现自动创建了很多的索引，是什么原因？

在创建表时，指定了主键，因此会自动创建对应的主键索引。

6

附录一：Linux 操作系统相关命令

Linux 中的命令格式为：command [options] [arguments] 中括号表示可选的，即有些命令不需要选项也不需要参数，但有的命令在运行时需要多个选项或参数。

- options(选项)：选项是调整命令执行行为的开关，选项的不同决定了命令的显示结果不同。
- arguments(参数)：参数是指命令的作用对象。

6.1 vi/vim

文本编辑器，若文件存在则是编辑，若不存在则是创建并编辑文本。

命令语法：

```
vim [参数]
```

参数说明：可编辑的文件名。

命令示例：

- 编辑名为 clusterconfig 的 xml 文本：

```
vim clusterconfig.xml
```

注：

vim 编辑器有以下三种模式：

- 正常模式：其它模式下按 Esc 或 Ctrl+[进入，左下角显示文件名或为空。
- 插入模式：正常模式下按 i 键进入，左下角显示--INSERT--。
- 可视模式：正常模式下按 v 键进入，左下角显示--VISUAL--。

退出命令（正常模式下）：

- :wq 保存并退出。
- :q! 强制退出并忽略所有更改。
- :e! 放弃所有修改，并打开原有文件。

6.2 cd

显示当前目录的名称，或切换当前的目录（打开指定目录）。

命令语法:

```
cd [参数]
```

参数说明:

- 无参数: 切换用户当前目录。
- . : 表示当前目录;
- .. : 表示上一级目录;
- ~ : 表示 home 目录;
- / : 表示根目录。

命令示例:

- 切换到 usr 目录下的 bin 目录中:

```
cd /usr/bin
```

- 切换到用户 home 目录:

```
cd
```

- 切换到当前目录(cd 后面接一个.):

```
cd .
```

- 切换到当前目录上一级目录(cd 后面接两个.):

```
cd ..
```

- 切换到用户 home 目录:

```
cd ~
```

- 切换到根目录下:

```
cd /
```

注: 切换目录需要理解绝对路径和相对路径这两个概念。

- 绝对路径: 在 Linux 中, 绝对路径是从/(即根目录)开始的, 例如 /opt/software、/etc/profile, 如果目录以 / 就是绝对目录。
- 相对路径: 是以 . 或 .. 开始的目录。 . 表示用户当前操作所在的位置, 而 .. 表示上级目录。例如 ./gs_om 表示当前目录下的文件或者目录。

6.3 mv

文件或目录改名(move (rename) files)或将文件或目录移入其它位置,经常用来备份文件或者目录。

命令语法:

```
mv [选项] 参数 1 参数 2
```

常用选项:

- -b: 若需覆盖文件, 则覆盖前先行备份。

参数说明：

- 参数 1：源文件或目录。
- 参数 2：目标文件或目录。

命令示例：

- 将文件 python 重命名为 python.bak：

```
mv python python.bak
```

- 将/physical/backup 目录下的所有文件和目录移到/data/dbn1 目录下：

```
mv /physical/backup/* /data/dbn1
```

6.4 curl

在 Linux 中 curl 是一个利用 URL 规则在命令行下工作的文件传输工具。支持文件的上传和下载，是综合传输工具。

命令语法：

```
curl [选项] [URL]
```

常用选项：

- -A/--user-agent <string>：设置用户代理发送给服务器；
- -C/--continue-at <offset>：断点续转；
- -D/--dump-header <file>：把 header 信息写入到该文件中；
- -e/--referer：来源网址；
- -o/--output：把输出写到该文件中；
- -O/--remote-name：把输出写到该文件中，保留远程文件的文件名；
- -s/--silent：静默模式。不输出任何东西；
- -T/--upload-file <file>：上传文件；
- -u/--user <user[:password]>：设置服务器的用户和密码；
- -x/--proxy <host[:port]>：在给定的端口上使用 HTTP 代理；
- -#/--progress-bar：进度条显示当前的传送状态。

参数说明：

- URL：指定的文件传输 URL 地址。

命令示例：

- 将 url(<https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo>)的内容保存到/etc/yum.repos.d/CentOS-Base.repo 文件中。

```
curl -o /etc/yum.repos.d/CentOS-Base.repo https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo
```

- 如果在传输过程中掉线，可以使用-C 的方式进行续传。

```
curl -C -O https://mirrors.huaweicloud.com/repository/conf/CentOS-7-anon.repo
```

6.5 yum

Shell 前端软件包管理器。基于 RPM 包管理，能够从指定的服务器自动下载 RPM 包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无须繁琐地一次次下载和安装。

命令语法：

```
yum [options] [command] [package ...]
```

常用选项：

- -h: 查看帮助；
- -y: 当安装过程提示选择全部为 "yes"；
- -q: 不显示安装的过程。

参数说明：

- command: 要进行的操作。
- package: 安装的包名。

命令示例：

- 列出所有可更新的软件清单命令：

```
yum check-update
```

- 更新所有软件命令：

```
yum update
```

- 列出所有可安装的软件清单命令：

```
yum list
```

- 安装指定的软件：

```
yum install -y libaio-devel flex bison ncurses-devel glibc-devel patch lsb_release wget python3
```

6.6 wget

wget 是 Linux 下下载文件的最常用命令。wget 支持 HTTP,HTTPS 和 FTP 协议,支持自动下载,即可以在用户退出系统后在后台执行,直到下载结束。

命令语法：

```
wget [选项] [URL]
```

常用选项：

- -c: 接着下载没下载完的文件；
- -b: 启动后转入后台执行；
- -P: 指定下载目录；
- -O: 变更下载文件名；

- --ftp-user --ftp-password：使用 FTP 用户认证下载。

参数说明:

- 指定的文件下载 URL 地址。

命令示例:

- 下载 openGauss 数据库安装文件到当前文件夹:

```
wget https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.1.0/x86/openGauss-1.1.0-CentOS-64bit.tar.gz
```

- 使用 wget 断点续传:

```
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

为某一个文件在另外一个位置建立一个同步的链接（软硬链接，不带选项为硬链接）。

当需要在不同的目录，用到相同的文件时，就不需要在每一个需要要的目录下都放一个必须相同的文件，我们只要在某个固定的目录，放上该文件，然后在 其它的目录下用 ln 命令链接（link）它就可以，不必重复的占用磁盘空间。

命令语法:

```
ln [选项] 参数 1 参数 2
```

常用选项:

- -b --删除，覆盖以前建立的链接；
- -d --允许超级用户制作目录的硬链接；
- -s --软链接(符号链接)。

参数说明:

- 参数 1：源文件或目录。
- 参数 2：被链接的文件或目录。

命令示例:

- 为 python3 文件创建软链接/usr/bin/python，如果 python3 丢失，/usr/bin/python 将失效:

```
ln -s python3 /usr/bin/python
```

- 为 python3 创建硬链接/usr/bin/python，python3 与/usr/bin/python 的各项属性相同:

```
ln python3 /usr/bin/python
```

6.8 mkdir

创建指定的名称的目录，要求创建目录的用户在当前目录中具有写权限，并且指定的目录名不能是当前目录中已有的目录。

命令语法：

```
mkdir [选项] [参数]
```

常用选项：

- -p --可以是一个路径名称。此时若路径中的某些目录尚不存在,加上此选项后,系统将自动建立好那些尚不存在的目录,即一次可以建立多个目录（递归）；
- -v --每次创建新目录都显示信息；
- -m --设定权限<模式>（类似 chmod），而不是 rwxrwxrwx 减 umask。

参数说明：

- 需要创建的目录。

命令示例：

- 创建一个空目录：

```
mkdir test
```

- 递归创建多个目录：

```
mkdir -p /opt/software/openGauss
```

- 创建权限为 777 的目录(目录的权限为 rwxrwxrwx)：

```
mkdir -m 777 test
```

6.9 chmod 命令

更改文件权限。

命令语法：

```
chmod [选项] <mode> <file...>
```

常用选项：

- -R, --以递归的方式对目前目录下的所有文件与子目录进行相同的权限变更。

参数说明：

- mode：权限设定字串，详细格式如下：

```
[ugoa...][[+|=][rwxX]...][,...],
```

其中，[ugoa...]：u 表示该档案的拥有者，g 表示与该档案的拥有者属于同一个群体(group)者，o 表示其他以外的人，a 表示所有（包含上面三者）；[+|=]：+ 表示增加权限，- 表示取消权限，= 表示唯一设定权限；[rwxX]：r 表示可读取，w 表示可写入，x 表示可执行，X 表示只有当该档案是个子目录或者该档案已经被设定过为可执行。

- file：文件列表（单个或者多个文件、文件夹）。

命令示例：

- 设置所有用户可读取文件 cluterconfig.xml：

```
chmod ugo+r cluterconfig.xml  
或  
chmod a+r cluterconfig.xml
```

- 设置当前目录下的所有档案与子目录皆设为任何人可读写：

```
chmod -R a+rw *
```

数字权限使用格式：

- 这种使用方式中，规定数字 4、2 和 1 表示读、写、执行权限，即 $r=4, w=2, x=1$ 。
- 例： $rw x = 7 (4+2+1)$ ； $rw = 6 (4+2)$ ； $r-x = 5 (4+0+1)$ ； $r-- = 4 (4+0+0)$ ； $--x = 1 (0+0+1)$ ；

每个文件都可以针对三个粒度，设置不同的 $rw x$ (读写执行)权限。即我们可以用三个 8 进制数字分别表示 拥有者、群组、其它组(u、g、o)的权限详情，并用 $chmod$ 直接加三个 8 进制数字的方式直接改变文件权限。语法格式为：

```
chmod <abc> file...
```

其中，a,b,c 各为一个数字，分别代表 User、Group、及 Other 的权限，相当于简化版的 $chmod u=权限, g=权限, o=权限 file...$ ，而此处的权限将用 8 进制的数字来表示 User、Group、及 Other 的读、写、执行权限。

命令示例：

- 赋予 cluterconfig.xml 文件可读可写可执行权限（所有权限）：

```
chmod 777 cluterconfig.xml
```

- 赋予/opt/software/openGauss 目录下所有文件及其子目录 用户所有权限组可读可执行权限，其他用户可读可执行权限：

```
chmod R 755 /opt/software/openGauss
```

6.10 chown

利用 $chown$ 将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户 ID；组可以是组名或者组 ID；文件是以空格分开的要改变权限的文件列表，支持通配符。只有系统管理者(root)才有这样的权限。使用权限：root。

命令语法：

```
chown [选项] user[:group] file...
```

常用选项：

- -c：显示更改的部分的信息；
- -f：忽略错误信息；

- -R：处理指定目录以及其子目录下的所有文件。

参数说明

- user：新的文件拥有者的使用者 ID。
- group：新的文件拥有者的使用者组(group)。
- file：文件。

命令示例：

- 将文件 file1.txt 的拥有者设为 omm，群体的使用者 dbgrp:

```
chown omm:dbgrp /opt/software/openGauss/clusterconfig.xml
```

- 将目前目录下的所有文件与子目录的拥有者皆设为 omm，群体的使用者 dbgrp:

```
chown -R omm:dbgrp *
```

6.11 ls

列出文件和目录的内容。

命令语法：

```
ls [选项] [参数]
```

常用选项：

- -l --以长格式显示，列出文件的详细信息，如创建者，创建时间，文件的读写权限列表等等；
- -a --列出文件下所有的文件，包括以"."和".."开头的隐藏文件（Linux 下文件隐藏文件是以 .开头的，如果存在 .. 代表存在着父目录）；
- -d --列出目录本身而非目录内的文件，通常要与-l 一起使用；
- -R --同时列出所有子目录层，与-l 相似，只是不显示出文件的所有者，相当于编程中的“递归”实现；
- -t --按照时间进行文件的排序，Time（时间）；
- -s --在每个文件的后面打印出文件的大小，size（大小）；
- -S --以文件的大小进行排序。

参数说明：

- 目录或文件。

命令示例：

- 以长格式列出当前目录中的文件及目录：

```
ls -l
```

6.12 cp

复制文件或者目录。

命令语法：

```
cp [选项] 参数 1 参数 2
```

常用选项：

- -f --如果目标文件无法打开则将其移除并重试(当 -n 选项存在时则不需再选此项)；
- -n --不要覆盖已存在的文件(使前面的 -i 选项失效)；
- -l --覆盖前询问(使前面的 -n 选项失效)；
- -p --保持指定的属性(默认：模式,所有权,时间戳)，如果可能保持附加属性：环境、链接、xattr 等；
- -R,-r --复制目录及目录内的所有项目。

参数说明：

- 参数 1：源文件。
- 参数 2：目标文件。

命令示例：

- 将 home 目录中的 abc 文件复制到 opt 目录下：

```
cp /home/abc /opt
```

注：目标文件存在时，会询问是否覆盖。这是因为 cp 是 cp -i 的别名。目标文件存在时，即使加了 -f 标志，也还会询问是否覆盖。

6.13 rm

删除一个目录中的一个或多个文件或目录，它也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，只是删除了链接，原有文件均保持不变。

rm 是一个危险的命令，使用的时候要特别当心，否则整个系统就会毁在这个命令（比如在/（根目录）下执行 rm * rf）。所以，我们在执行 rm 之前最好先确认一下在哪个目录，到底要删除什么东西，操作时保持高度清醒的头脑。

命令语法：

```
rm [选项] 文件
```

常用选项：

- -f --忽略不存在的文件，从不给出提示；
- -r --指示 rm 将参数中列出的全部目录和子目录均递归地删除。

参数说明：

- 需要删除的文件或目录。

命令示例：

- 删除文件：

```
rm qwe
```

注：输入 rm qwe 命令后，系统会询问是否删除，输入 y 后就会删除文件，不想删除文件则输入 n。

- 强制删除某个文件：

```
rm-rf clusterconfig.log
```

6.14 cat

连接文件并在标准输出上输出。这个命令常用来显示文件内容，或者将几个文件连接起来显示，或者从标准输入读取内容并显示，它常与重定向符号配合使用。

命令语法：

```
cat [选项] [参数]
```

常用选项：

- -E --在每行结束显示\$；
- -n --由 1 开始对给所有输出行编号；
- -b 或 --number-nonblank：和 -n 相似，只不过对于空白行不编号；
- -v --使用 ^ 和 M- 符号，除了 LFD 和 TAB 之外。

参数说明：

- 可操作的文件名。

命令示例：

- 显示 testfile 文件的内容：

```
cat testfile
```

- 把 testfile1 和 testfile2 的文档内容加上行号（空白行不加）之后将内容追加到 testfile3 文档里：

```
cat -b testfile1 testfile2 >> testfile3
```

- 向/etc/profile 中追加内容（输入 EOF 表示结束追加）：

```
cat >>/etc/profile<<EOF
>export LD_LIBRARY_PATH=$packagePath/script/gspylib/clib:$LD_LIBRARY_PATH
>EOF
```

注：

- EOF 是 end of file 的缩写，表示"文字流"（stream）的结尾。"文字流"可以是文件（file），也可以是标准输入（stdin）。在 Linux 系统之中，EOF 是当系统读取到文件结尾，所返回的一个信号值（也就是-1）。

7 附录二：openGauss 数据库基本操作

7.1 查看数据库对象

- 查看帮助信息：

```
postgres=# \?
```

- 切换数据库：

```
postgres=# \c dbname
```

- 列举数据库：

使用 \l 元命令查看数据库系统的数据库列表。

```
postgres=# \l
```

使用如下命令通过系统表 pg_database 查询数据库列表。

```
postgres=# SELECT datname FROM pg_database;
```

- 列举表：

```
postgres=# \dt
```

- 列举所有表、视图和索引：

```
postgres=# \d+
```

使用 gsql 的 \d+ 命令查询表的属性。

```
postgres=# \d+ tablename
```

- 查看表结构：

```
postgres=# \d tablename
```

- 列举 schema：

```
postgres=# \dn
```

- 查看索引：

```
postgres=# \di
```

- 查询表空间：

使用 gsql 程序的元命令查询表空间。

```
postgres=# \db
```

检查 pg_tablespace 系统表。如下命令可查到系统和用户定义的全部表空间。

```
postgres=# SELECT spcname FROM pg_tablespace;
```

- 查看数据库用户列表：

```
postgres=# SELECT * FROM pg_user;
```

- 要查看用户属性：

```
postgres=# SELECT * FROM pg_authid;
```

- 查看所有角色：

```
postgres=# SELECT * FROM PG_ROLES;
```

7.2 其他操作

- 查看 openGauss 支持的所有 SQL 语句。

```
postgres=# \h
```

- 切换数据库：

```
postgres=# \c dbname
```

- 切换用户：

```
postgres=# \c - username
```

- 退出数据库：

```
postgres=# \q
```