《数据库》

openGauss金融场景化实验



华为技术有限公司

目录

[前 言 3](#_Toc57396630)

[简介 3](#_Toc57396631)

[内容描述 3](#_Toc57396632)

[前置条件 3](#_Toc57396633)

[实验环境说明 3](#_Toc57396634)

[实验概览 4](#_Toc57396635)

[1 金融场景化实验 5](#_Toc57396636)

[1.1 金融数据模型 5](#_Toc57396637)

[1.1.1 E-R图 5](#_Toc57396638)

[1.1.2 关系模式 6](#_Toc57396639)

[1.1.3 物理模型 7](#_Toc57396640)

[1.1.4 创建数据表 10](#_Toc57396641)

[1.1.5 插入表数据 12](#_Toc57396642)

[1.1.6 手工插入一条数据 16](#_Toc57396643)

[1.1.7 添加约束 17](#_Toc57396644)

[1.1.8 查询数据 17](#_Toc57396645)

[1.1.9 视图 22](#_Toc57396646)

[1.1.10 索引 24](#_Toc57396647)

[1.1.11 数据的修改和删除 24](#_Toc57396648)

[1.1.12 新用户的创建和授权 26](#_Toc57396649)

[1.1.13 新用户连接数据库 26](#_Toc57396650)

[1.1.14 删除Schema 27](#_Toc57396651)

[1.2 使用JDBC连接数据库 29](#_Toc57396652)

[1.2.1 准备连接环境 29](#_Toc57396653)

[1.2.2 确定26000端口是否放开 30](#_Toc57396654)

[1.2.3 下载并安装JDK 33](#_Toc57396655)

[1.2.4 配置JDK环境变量 35](#_Toc57396656)

[1.2.5 连接openGauss并执行java代码 37](#_Toc57396657)

[1.3 实验小结 39](#_Toc57396658)

[2 附录一：openGauss数据库基本操作 40](#_Toc57396659)

[2.1 查看数据库对象 40](#_Toc57396660)

[2.2 其他操作 41](#_Toc57396661)

# 前 言

## 简介

本实验指导书以金融数据库模型为例，主要目的是为了让读者由浅渐深的熟悉openGauss数据库。

## 内容描述

本实验指导书以金融行业为场景，设计数据库模型，并使用华为openGauss构建金融场景下的数据库。通过对数据库中的对象（表、约束、视图、索引等）创建，掌握基础SQL语法，并通过对表中数据的增删改查，模拟金融场景下的业务实现。

本实验中的金融数据模型，主要是为了实现实验操作而构造的，若与现实场景中模型相似，纯属巧合。

## 前置条件

openGauss数据库场景化综合应用实验需要掌握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**附录一**。

## 实验环境说明

* 组网说明

本实验环境为华为云ECS服务器和openGauss数据库。

* 设备介绍

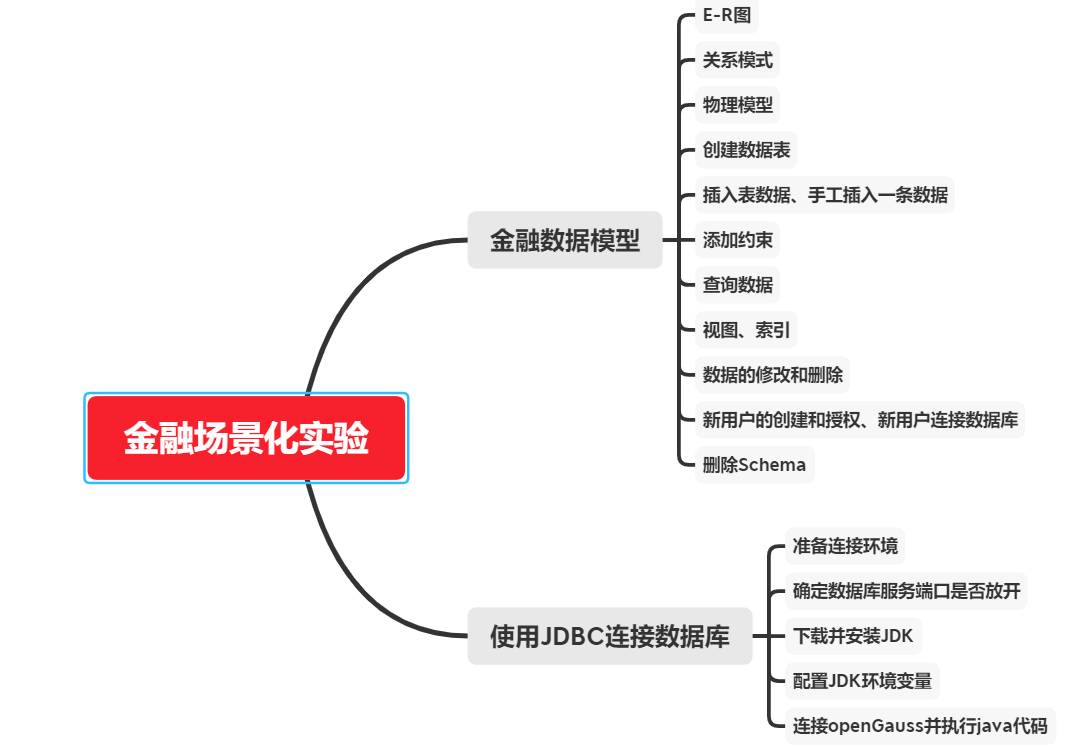
为了满足本实验需要，建议实验环境采用以下配置：

设备名称、型号与版本的对应关系如下：

设备明细表

|  |  |  |
| --- | --- | --- |
| 设备名称 | 设备型号 | 软件版本 |
| 数据库 | openGauss | openGauss 1.0.0 |
| 操作系统 | openEuler | openEuler 20.3LTS |

## 实验概览



本实验概览图

# 金融场景化实验

## 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，引入了华为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、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。

### E-R图

客户手机号

客户名称

客户身份证

客户编号

客户邮箱

客户登录密码

银行卡号

N

1

银行卡

办理

客户

N

N

N

银行卡类型

拥有

拥有

拥有

保险项目

M

M

M

保险

理财产品

保险金额

保险名称

保险编号

产品编号

产品名称

产品描述

保险年限

基金

适用人群

理财年限

购买金额

基金管理者

风险等级

基金金额

基金类型

基金编号

基金名称

E-R图

### 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：

客户

客户编号

客户名称

……

银行卡

银行卡号

银行卡类型

**客户编号**

**资产**

**客户编号**

**商品编号**

商品状态

商品数量

商品收益

购买时间

基金

基金编号

基金名称

……

保险

保险编号

保险名称

……

理财产品

产品编号

产品名称

……

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。

### 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。

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 |

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。 |

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 |

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 |

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  说明：本字段引用自银行员工表的id字段，由于简化系统没有在本实验展示员工表。 |

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。

### 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：

创建金融数据库finance。

切换到omm用户，以操作系统用户omm登录数据库主节点。

**su - omm**

使用gsql工具登陆数据库。

**gsql -d postgres -p 26000 -r**

创建数据库finance。

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

连接finance数据库。

**\connect finance**

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

**CREATE SCHEMA finance;**

将默认搜索路径设为finance。

**SET search\_path TO finance;**

客户信息表的创建。

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 client;**

创建表client。

**CREATE TABLE 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**

**);**

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 bank\_card;**

创建表bank\_card。

**CREATE TABLE bank\_card**

**(**

**b\_number CHAR(30) PRIMARY KEY,**

**b\_type CHAR(20),**

**b\_c\_id INT NOT NULL**

**);**

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 finances\_product;**

创建表finances\_product。

**CREATE TABLE finances\_product**

**(**

**p\_name VARCHAR(100) NOT NULL,**

**p\_id INT PRIMARY KEY,**

**p\_description VARCHAR(4000),**

**p\_amount INT,**

**p\_year INT**

**);**

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 insurance;**

创建表insurance。

**CREATE TABLE 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)**

**);**

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 fund;**

创建表fund。

**CREATE TABLE 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**

**);**

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 property;**

创建表property。

**CREATE TABLE 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**

**);**

### 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。

对client表进行数据初始化。

执行insert操作。

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (1,'张一','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,'张二','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,'张三','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,'张四','zhangsi@huawei.com','340211199301010004','18815650004','gaussdb\_004');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (5,'张五','zhangwu@huawei.com','340211199301010005','18815650005','gaussdb\_005');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (6,'张六','zhangliu@huawei.com','340211199301010006','18815650006','gaussdb\_006');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (7,'张七','zhangqi@huawei.com','340211199301010007','18815650007','gaussdb\_007');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (8,'张八','zhangba@huawei.com','340211199301010008','18815650008','gaussdb\_008');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (9,'张九','zhangjiu@huawei.com','340211199301010009','18815650009','gaussdb\_009');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (10,'李一','liyi@huawei.com','340211199301010010','18815650010','gaussdb\_010');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (11,'李二','lier@huawei.com','340211199301010011','18815650011','gaussdb\_011');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (12,'李三','lisan@huawei.com','340211199301010012','18815650012','gaussdb\_012');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (13,'李四','lisi@huawei.com','340211199301010013','18815650013','gaussdb\_013');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (14,'李五','liwu@huawei.com','340211199301010014','18815650014','gaussdb\_014');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (15,'李六','liliu@huawei.com','340211199301010015','18815650015','gaussdb\_015');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (16,'李七','liqi@huawei.com','340211199301010016','18815650016','gaussdb\_016');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (17,'李八','liba@huawei.com','340211199301010017','18815650017','gaussdb\_017');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (18,'李九','lijiu@huawei.com','340211199301010018','18815650018','gaussdb\_018');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (19,'王一','wangyi@huawei.com','340211199301010019','18815650019','gaussdb\_019');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (20,'王二','wanger@huawei.com','340211199301010020','18815650020','gaussdb\_020');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (21,'王三','wangsan@huawei.com','340211199301010021','18815650021','gaussdb\_021');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (22,'王四','wangsi@huawei.com','340211199301010022','18815650022','gaussdb\_022');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (23,'王五','wangwu@huawei.com','340211199301010023','18815650023','gaussdb\_023');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (24,'王六','wangliu@huawei.com','340211199301010024','18815650024','gaussdb\_024');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (25,'王七','wangqi@huawei.com','340211199301010025','18815650025','gaussdb\_025');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (26,'王八','wangba@huawei.com','340211199301010026','18815650026','gaussdb\_026');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (27,'王九','wangjiu@huawei.com','340211199301010027','18815650027','gaussdb\_027');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (28,'钱一','qianyi@huawei.com','340211199301010028','18815650028','gaussdb\_028');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (29,'钱二','qianer@huawei.com','340211199301010029','18815650029','gaussdb\_029');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (30,'钱三','qiansan@huawei.com','340211199301010030','18815650030','gaussdb\_030');

查询插入结果。

select count(\*) from client;

结果为：

count(\*)

-------------

30

对bank\_card表进行数据初始化。

执行insert操作。

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000001','信用卡',1);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000002','信用卡',3);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000003','信用卡',5);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000004','信用卡',7);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000005','信用卡',9);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000006','信用卡',10);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000007','信用卡',12);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000008','信用卡',14);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000009','信用卡',16);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000010','信用卡',18);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000011','储蓄卡',19);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000012','储蓄卡',21);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000013','储蓄卡',7);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000014','储蓄卡',23);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000015','储蓄卡',24);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000016','储蓄卡',3);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000017','储蓄卡',26);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000018','储蓄卡',27);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000019','储蓄卡',12);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000020','储蓄卡',29);

查询插入结果。

**select count(\*) from bank\_card;**

结果为：

count(\*)

-------------

20

对finances\_product表进行数据初始化。

执行insert操作。

INSERT INTO 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 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 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 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 finances\_product;

结果为：

count(\*)

-------------

4

对insurance表进行数据初始化。

执行insert操作。

INSERT INTO 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 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 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 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 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 insurance;

结果为：

count(\*)

-------------

5

对fund表进行数据初始化。

执行insert操作。

INSERT INTO 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 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 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 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 fund;

结果为：

count(\*)

-------------

4

对property表进行数据初始化。

执行insert操作。

INSERT INTO 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 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 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 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 property;

结果为：

count(\*)

-------------

4

### 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。

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 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：

duplicate key value violates unique constraint "client\_c\_id\_card\_key"

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），所以当表中存在时，插入数据失败。

在金融数据库的客户信息表中添加一个客户的信息。（插入成功的场景）。

插入成功的示例。

**INSERT INTO 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');**

### 添加约束

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 finances\_product ADD CONSTRAINT c\_p\_mount CHECK (p\_amount >=0);**

尝试手工插入一条金额小于0的记录。

**INSERT INTO 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：new row for relation "finances\_product" violates check constraint "c\_p\_mount"。

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 fund ADD CONSTRAINT c\_f\_mount CHECK (f\_amount >=0);**

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 insurance ADD CONSTRAINT c\_i\_mount CHECK (i\_amount >=0);**

### 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。

单表查询。

* 查询银行卡信息表。

**SELECT b\_number,b\_type FROM bank\_card;**

结果如下：

b\_number | b\_type

--------------------------------+-------------------

6222021302020000001 | 信用卡

6222021302020000002 | 信用卡

6222021302020000003 | 信用卡

6222021302020000004 | 信用卡

6222021302020000005 | 信用卡

6222021302020000006 | 信用卡

6222021302020000007 | 信用卡

6222021302020000008 | 信用卡

6222021302020000009 | 信用卡

6222021302020000010 | 信用卡

6222021302020000011 | 储蓄卡

6222021302020000012 | 储蓄卡

6222021302020000013 | 储蓄卡

6222021302020000014 | 储蓄卡

6222021302020000015 | 储蓄卡

6222021302020000016 | 储蓄卡

6222021302020000017 | 储蓄卡

6222021302020000018 | 储蓄卡

6222021302020000019 | 储蓄卡

6222021302020000020 | 储蓄卡

条件查询。

* 查询资产信息中‘可用’的资产数据。

**select \* from 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)

聚合查询。

查询用户表中有多少个用户。

**SELECT count(\*) FROM 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 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 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 insurance where i\_amount in (select max(i\_amount) from insurance)**

**union**

**select i\_name,i\_amount from insurance where i\_amount in (select min(i\_amount) from insurance);**

结果如下：

i\_name | i\_amount

--------------+----------

财产损失保险 | 1500

意外保险 | 5000

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 client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id);**

结果如下：

c\_id | c\_name | c\_id\_card

------+--------+----------------------

1 | 张一 | 340211199301010001

3 | 张三 | 340211199301010003

5 | 张五 | 340211199301010005

7 | 张七 | 340211199301010007

9 | 张九 | 340211199301010009

10 | 李一 | 340211199301010010

12 | 李三 | 340211199301010012

14 | 李五 | 340211199301010014

16 | 李七 | 340211199301010016

18 | 李九 | 340211199301010018

19 | 王一 | 340211199301010019

21 | 王三 | 340211199301010021

23 | 王五 | 340211199301010023

24 | 王六 | 340211199301010024

26 | 王八 | 340211199301010026

27 | 王九 | 340211199301010027

29 | 钱二 | 340211199301010029

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。

1. 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 client WHERE c\_id NOT IN (SELECT b\_c\_id FROM bank\_card WHERE b\_number LIKE '622202130202000001\_');**

结果如下：

c\_id | c\_name | c\_id\_card

------+--------+----------------------

1 | 张一 | 340211199301010001

2 | 张二 | 340211199301010002

3 | 张三 | 340211199301010003

4 | 张四 | 340211199301010004

5 | 张五 | 340211199301010005

6 | 张六 | 340211199301010006

7 | 张七 | 340211199301010007

8 | 张八 | 340211199301010008

9 | 张九 | 340211199301010009

10 | 李一 | 340211199301010010

11 | 李二 | 340211199301010011

12 | 李三 | 340211199301010012

13 | 李四 | 340211199301010013

14 | 李五 | 340211199301010014

15 | 李六 | 340211199301010015

16 | 李七 | 340211199301010016

17 | 李八 | 340211199301010017

18 | 李九 | 340211199301010018

19 | 王一 | 340211199301010019

20 | 王二 | 340211199301010020

21 | 王三 | 340211199301010021

22 | 王四 | 340211199301010022

23 | 王五 | 340211199301010023

24 | 王六 | 340211199301010024

25 | 王七 | 340211199301010025

26 | 王八 | 340211199301010026

27 | 王九 | 340211199301010027

28 | 钱一 | 340211199301010028

29 | 钱二 | 340211199301010029

30 | 钱三 | 340211199301010030

31 | 李丽 | 340211199301010031

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。

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 insurance i1 WHERE i\_amount > (SELECT avg(i\_amount) FROM 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

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 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

1. 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 finances\_product GROUP BY p\_year;**

结果如下：

p\_year | count

--------+-------

6 | 4

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 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。

1. 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 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。

### 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。

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 v\_client as SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id);**

使用视图进行查询。

**SELECT \* FROM v\_client;**

结果如下：

c\_id | c\_name | c\_id\_card

------+--------+----------------------

1 | 张一 | 340211199301010001

3 | 张三 | 340211199301010003

5 | 张五 | 340211199301010005

7 | 张七 | 340211199301010007

9 | 张九 | 340211199301010009

10 | 李一 | 340211199301010010

12 | 李三 | 340211199301010012

14 | 李五 | 340211199301010014

16 | 李七 | 340211199301010016

18 | 李九 | 340211199301010018

19 | 王一 | 340211199301010019

21 | 王三 | 340211199301010021

23 | 王五 | 340211199301010023

24 | 王六 | 340211199301010024

26 | 王八 | 340211199301010026

27 | 王九 | 340211199301010027

29 | 钱二 | 340211199301010029

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 v\_client as SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM 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 v\_client;**

结果如下：

c\_id | c\_name | c\_id\_card

------+--------+----------------------

7 | 张七 | 340211199301010007

3 | 张三 | 340211199301010003

5 | 张五 | 340211199301010005

9 | 张九 | 340211199301010009

12 | 李三 | 340211199301010012

14 | 李五 | 340211199301010014

18 | 李九 | 340211199301010018

10 | 李一 | 340211199301010010

16 | 李七 | 340211199301010016

1 | 张一 | 340211199301010001

修改视图名称。

**ALTER VIEW v\_client RENAME TO v\_client\_new;**

删除视图。

将v\_client视图删除，删除视图不影响基表。

**DROP VIEW v\_client\_new;**

### 索引

创建索引。

* 在普通表property上创建索引。

**CREATE INDEX idx\_property ON property(pro\_c\_id DESC,pro\_income,pro\_purchase\_time);**

结果如下：

CREATE INDEX

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 idx\_property;**

**CREATE INDEX idx\_property ON property(pro\_c\_id DESC,pro\_income,pro\_purchase\_time);**

重命名索引。

**ALTER INDEX idx\_property RENAME TO idx\_property\_temp;**

删除索引。

* 删除索引idx\_property\_temp。

**DROP INDEX idx\_property\_temp;**

### 数据的修改和删除

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 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

开始更新数据：

**UPDATE bank\_card SET bank\_card.b\_type='借记卡' from 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 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

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 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 fund WHERE f\_id<3;**

查询删除结果。

**SELECT \* FROM fund;**

结果如下：

f\_name | f\_id | f\_type | f\_amount | risk\_level | f\_manager

-------------+------+-------------------+----------+---------------------+-----------

国债 | 3 | 货币型 | 10000 | 低 | 3

沪深300指数 | 4 | 指数型 | 10000 | 中 | 4

### 新用户的创建和授权

在本章中，假设C银行的某新员工想要在自己的用户下去访问sys用户下的金融数据库，则该员工需要向sys申请添加相关权限，具体操作如下：

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

**CREATE USER dbuser IDENTIFIED BY 'Gauss#3demo';**

给用户dbuser授予finance数据库下bank\_card表的查询和插入权限，并将SCHEMA的权限也授予dbuser用户。

**GRANT SELECT,INSERT ON finance.bank\_card TO dbuser;**

**GRANT ALL ON SCHEMA finance to dbuser;**

退出数据库：

postgres=#**\q**

### 新用户连接数据库

在gsql登录数据库，使用新用户连接。

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。

**gsql -d finance -U dbuser -p 26000 -r**

访问finance数据库的表bank\_card。

**select \* from finance. bank\_card where b\_c\_id<10;**

结果如下：

b\_number | b\_type | b\_c\_id

--------------------------------+-------------------+--------

6222021302020000001 | 借记卡 | 1

6222021302020000002 | 借记卡 | 3

6222021302020000003 | 借记卡 | 5

6222021302020000004 | 借记卡 | 7

6222021302020000005 | 借记卡 | 9

6222021302020000013 | 借记卡 | 7

6222021302020000016 | 借记卡 | 3

退出数据库：

postgres=#**\q**

### 删除Schema

使用管理员用户登陆finance数据库。

使用操作系统omm用户使用gsql，新建session。

**gsql -d finance -p 26000**

使用“\dn”查看数据库下的schema。

**\dn**

List of schemas

Name | Owner

-----------+--------

cstore | omm

dbms\_perf | omm

dbuser | dbuser

finance | omm

public | omm

snapshot | omm

设置默认查询为finance。

**set search\_path to finance;**

使用“\dt”命令可以看到在finance中的对象。

**\dt**

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}

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.

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

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

**\dt**

No relations found.

退出数据库：

postgres=#**\q**

## 使用JDBC连接数据库

### 准备连接环境

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/db1，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/db1**"/>。

**cd /gaussdb/data/db1**

**vi pg\_hba.conf**

将以下内容添加进pg\_hba.conf文件。

**host all all 0.0.0.0/0 sha256**

使用omm用户登陆，使用gs\_ctl将策略生效。

**su - omm**

**gs\_ctl reload -D /gaussdb/data/db1/**

返回结果为：

[2020-07-23 15:39:55.398][71828][][gs\_ctl]: gs\_ctl reload ,datadir is -D "/gaussdb/data/db1"

server signaled

使用omm用户登陆数据库给dbuser用户授权，并退出数据库。

**gsql -d postgres -p 26000 -r**

postgres=# **alter role dbuser createrole createdb;**

ALTER ROLE

postgres=# **\q**

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/db1。

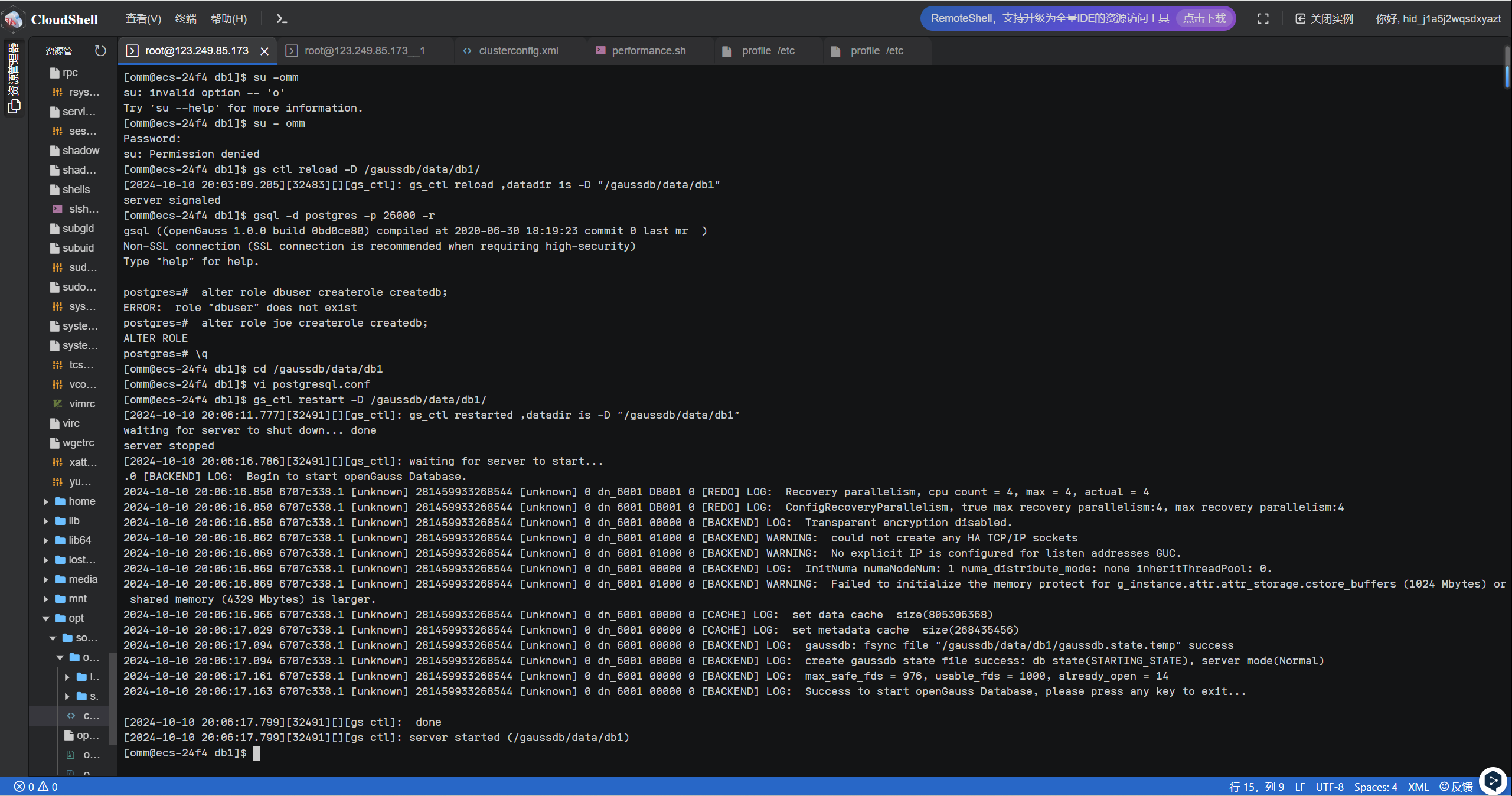
**cd /gaussdb/data/db1**

**vi postgresql.conf**

将listen\_addresses的值修改成为\*。

**listen\_addresses = '\*'**

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）。



**gs\_ctl restart -D /gaussdb/data/db1/**

下载Java连接openGauss的驱动包，并将其导入对应的使用工具。

通过以下链接，下载驱动包

<https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.0.0/arm/openGauss-1.0.0-JDBC.tar.gz>

假设文件存放在d:\Download目录下，并进行解压，解压后文件为“postgresql.jar”。

创建测试数据库demo。

使用gsql工具登陆数据库。登录用户为dbuser，密码为Gauss#3demo（或者为用户自定义的密码）。

**gsql -d postgres -p 26000 -U joe -r**

创建数据库demo。

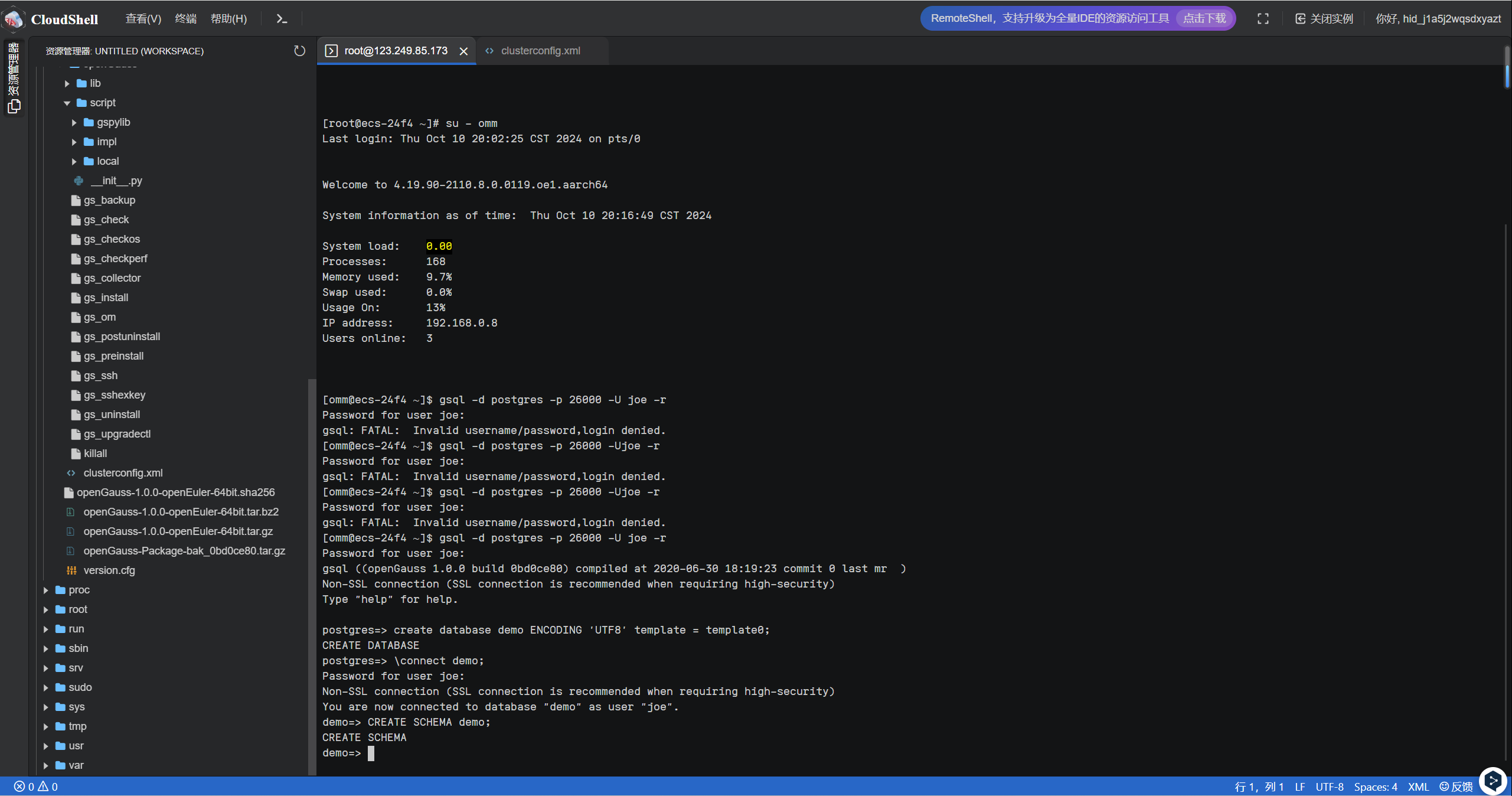
create database demo ENCODING 'UTF8' template = template0;

\connect demo;

创建名为demo的schema，并设置demo为当前的schema。

**CREATE SCHEMA demo;**

将默认搜索路径设为demo。



**SET search\_path TO demo;**

创建测试表websites。

**CREATE TABLE websites (**

**id int NOT NULL,**

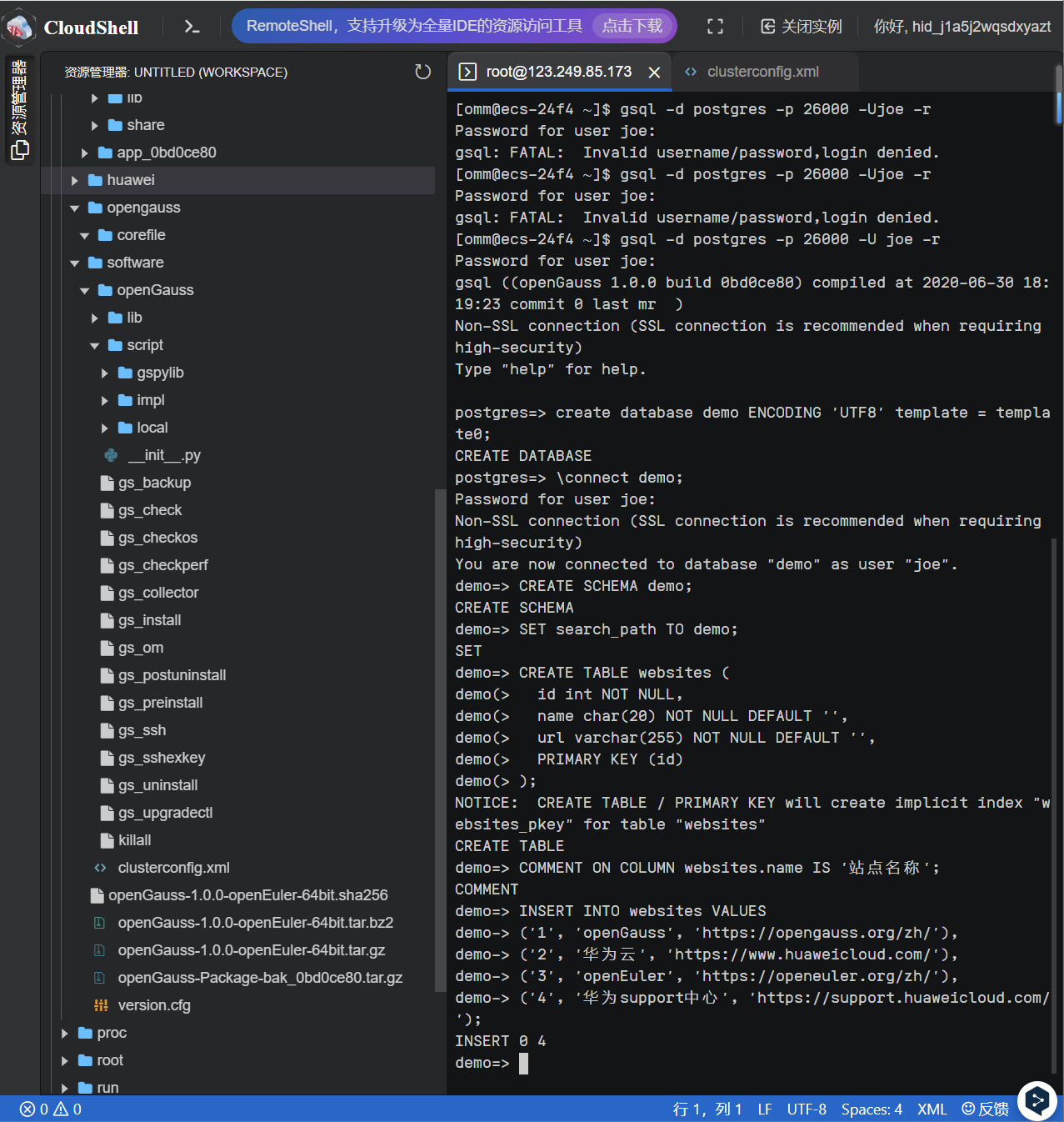
**name char(20) NOT NULL DEFAULT '',**

**url varchar(255) NOT NULL DEFAULT '',**

**PRIMARY KEY (id)**

**);**

**COMMENT ON COLUMN websites.name IS '站点名称';**

插入数据。

**INSERT INTO websites VALUES**

**('1', 'openGauss', 'https://opengauss.org/zh/'),**

**('2', '华为云', 'https://www.huaweicloud.com/'),**

**('3', 'openEuler', 'https://openeuler.org/zh/'),**

**('4', '华为support中心', 'https://support.huaweicloud.com/');**

退出数据库

**\q**

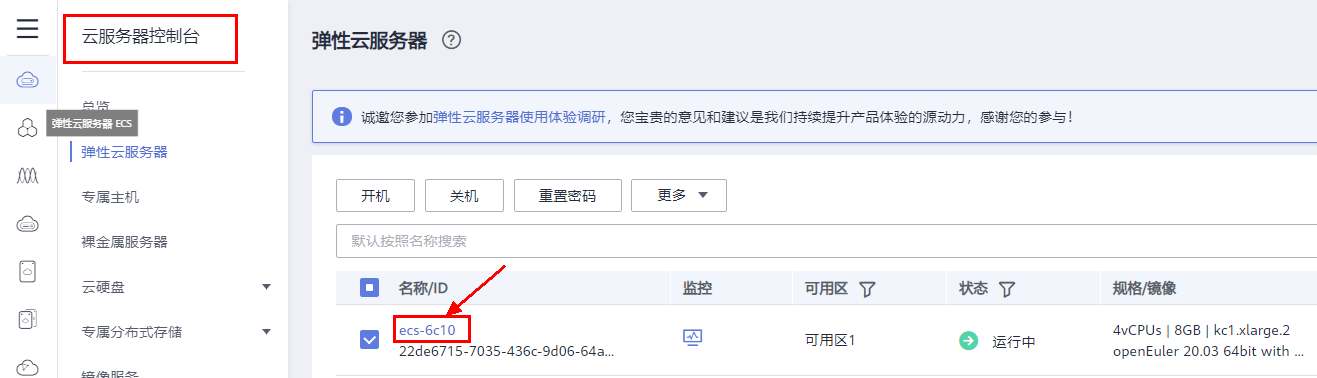
### 确定26000端口是否放开

打开华为云首页，登录后进入“控制台”，点击“弹性云服务器ECS”进入ECS列表。





在云服务器控制台找到安装数据库主机的ECS，点击查看基本信息，找到安全组。





点击进入安全组，选择“入方向规则”并“添加规则”，进行26000端口设置。



2.

1.



2.

1.

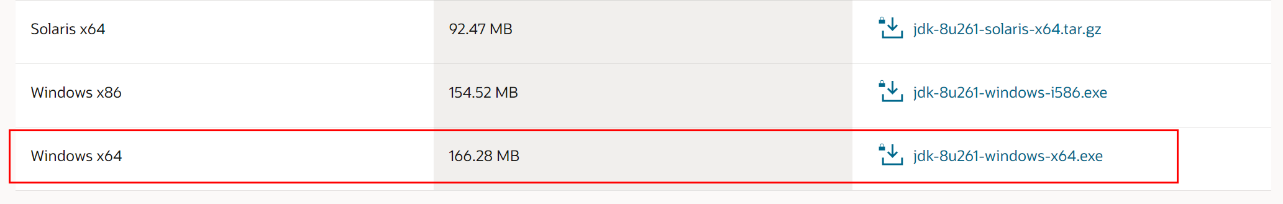
确定后，可以看到入网规则多了“TCP:26000”，如下图：



### 下载并安装JDK

下载JDK。

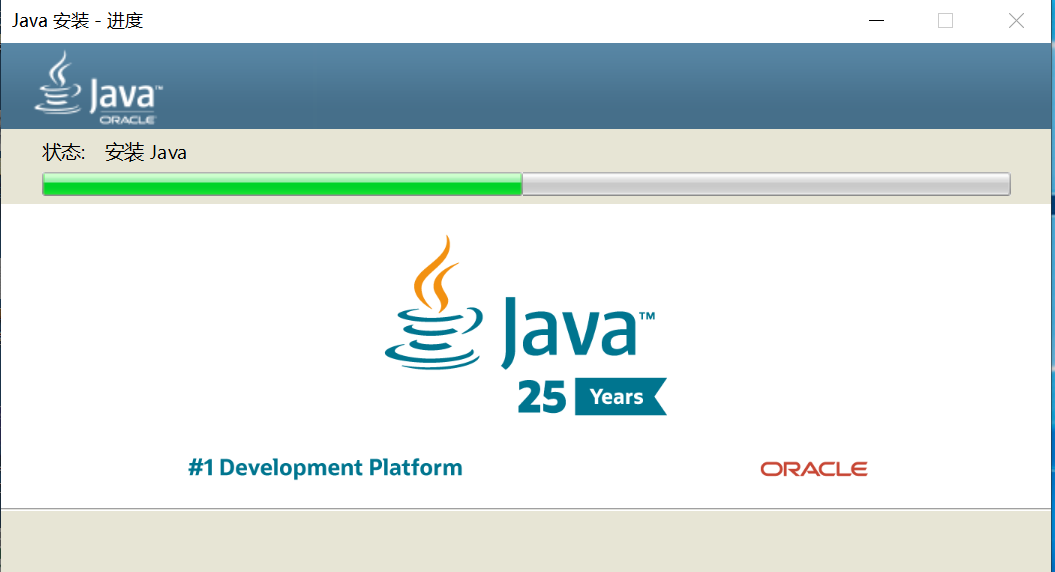
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



C:\Users\swx941157\AppData\Roaming\eSpace_Desktop\UserData\swx941157\imagefiles\956D0976-2FE5-4327-BE30-8724B20B7D78.png

点击jdk-8u261-windows-x64.exe进行安装。

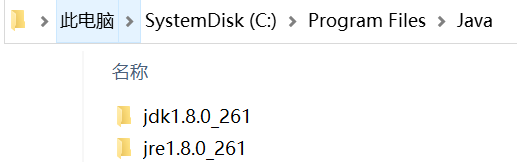
默认设置即可，出现安装进度。



如下显示表示安装成功：

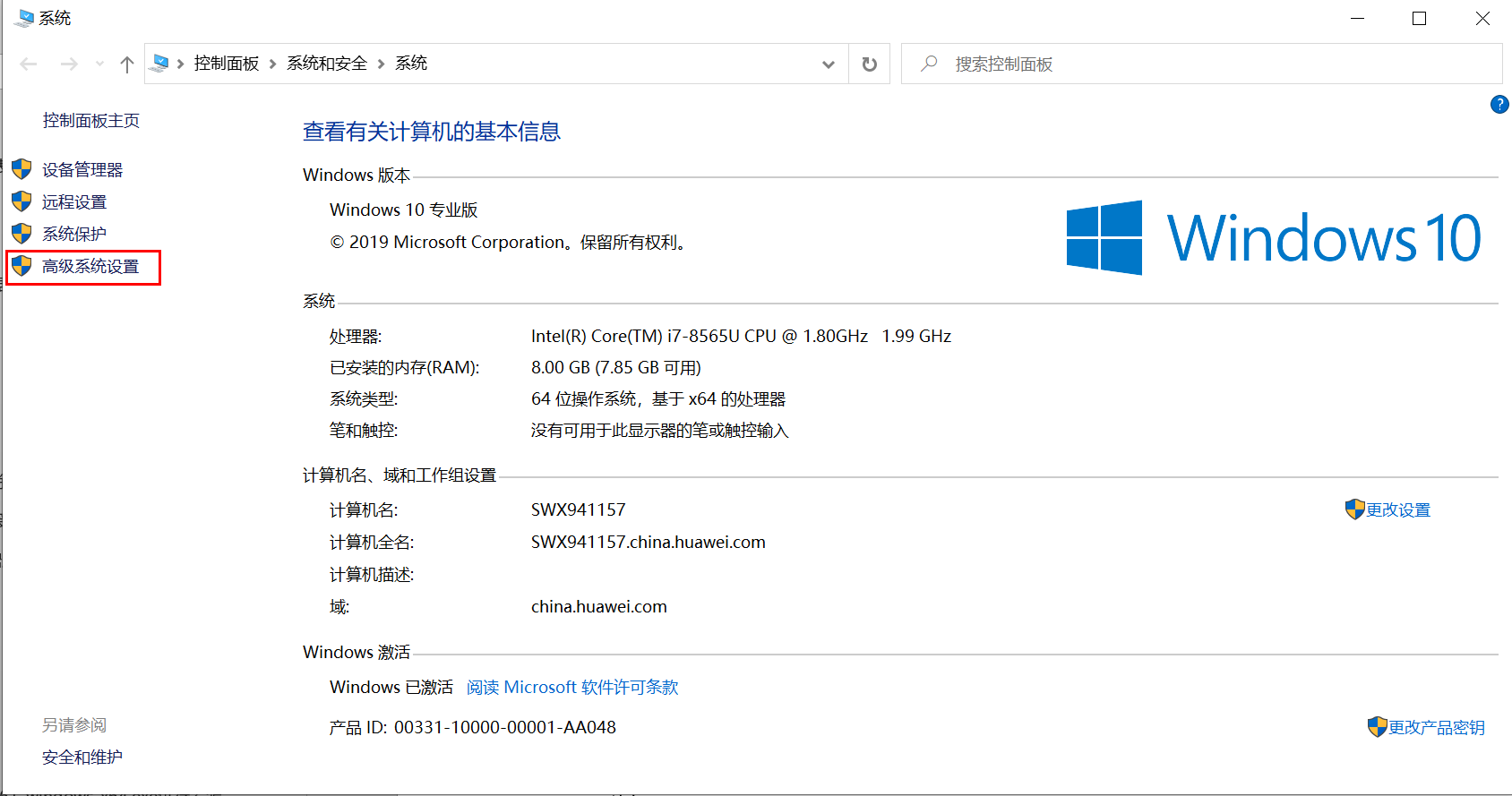


查看安装目录。



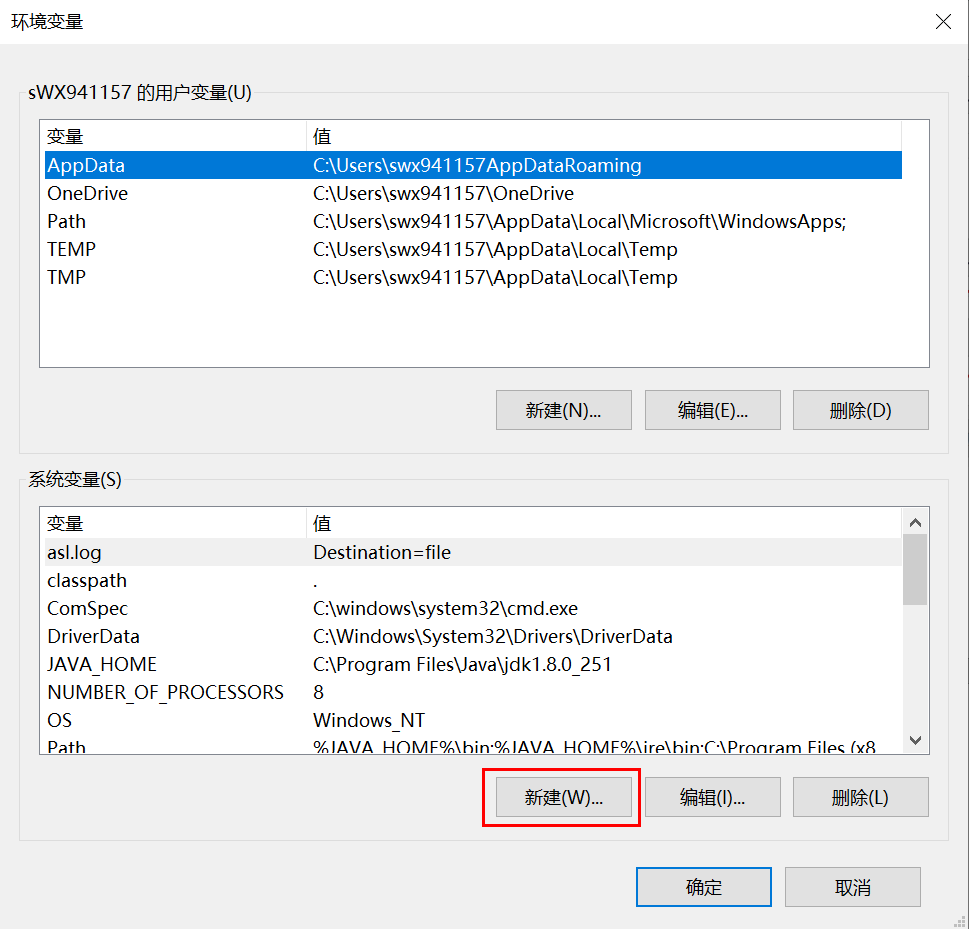
### 配置JDK环境变量

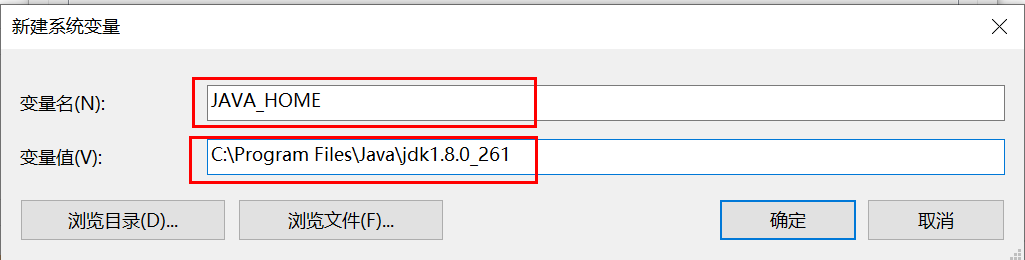
右击“此电脑”选择“属性”，点击“高级系统设置”。





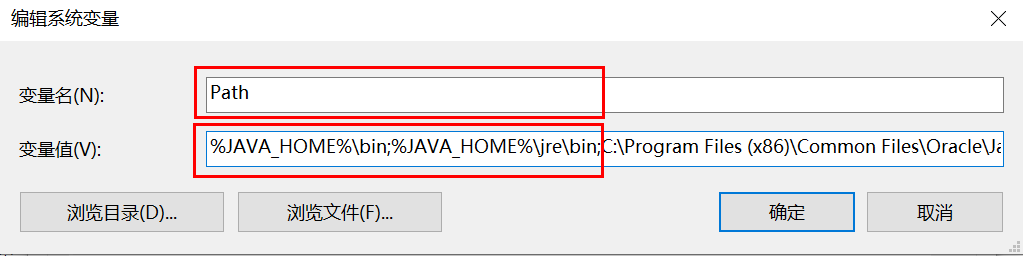
点击“环境变量”，新建系统变量“JAVA\_HOME”，输入JDK安装目录。





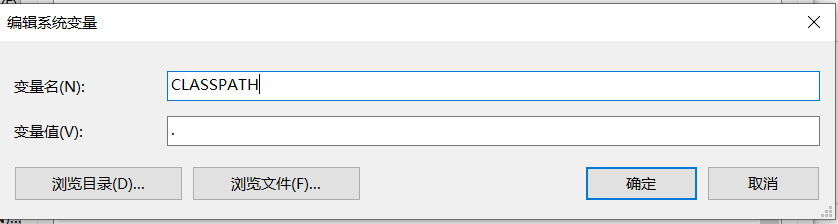
“C:\Program Files\Java\jdk1.8.0\_261”为JDK安装目录。

编辑系统变量“path”。



在变量值最后输入 %JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;**（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）。**

新建系统变量“CLASSPATH”变量，输入“.” 即可。



系统变量配置完毕，查询检验是否配置成功，运行cmd 输入java -version（java和 -version之间有空格）。

C:\Users\xxxxx>**java -version**

如下所示，显示版本信息，则说明安装和配置成功。

java version "1.8.0\_261"

Java(TM) SE Runtime Environment (build 1.8.0\_261-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)

### 连接openGauss并执行java代码

使用Java程序连接数据库并进行查询（注：请用户根据实际情况替换红字内容，修改jdbc:postgresql://**公网IP**:26000/demo中公网IP信息，USER = "**dbuser**"连接数据库的用户及密码PASS = "**Gauss#3demo**"）。

在d:\Download\中创建openGaussDemo.java文件，文件内容如如下，注意红字部分要按照实际情况进行替换：

import java.sql.\*;

public class openGaussDemo {

static final String JDBC\_DRIVER = "org.postgresql.Driver";

static final String DB\_URL = "jdbc:postgresql:// 123.249.85.173**:**26000/demo";

// 数据库的用户名与密码，需要根据自己的设置

static final String USER = "**joe**";

static final String PASS = "**Bigdata@123**";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try{

// 注册 JDBC 驱动

Class.forName(JDBC\_DRIVER);

// 打开链接

System.out.println("连接数据库...");

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

// 执行查询

System.out.println(" 实例化Statement对象...");

stmt = conn.createStatement();

String sql;

sql = "SELECT id, name, url FROM demo.websites";

ResultSet rs = stmt.executeQuery(sql);

// 展开结果集数据库

while(rs.next()){

// 通过字段检索

int id = rs.getInt("id");

String name = rs.getString("name");

String url = rs.getString("url");

// 输出数据

System.out.print("ID: " + id);

System.out.print(", 站点名称: " + name);

System.out.print(", 站点 URL: " + url);

System.out.print("\n");

}

// 完成后关闭

rs.close();

stmt.close();

conn.close();

}catch(SQLException se){

// 处理 JDBC 错误

se.printStackTrace();

}catch(Exception e){

// 处理 Class.forName 错误

e.printStackTrace();

}finally{

// 关闭资源

try{

if(stmt!=null) stmt.close();

}catch(SQLException se2){

}// 什么都不做

try{

if(conn!=null) conn.close();

}catch(SQLException se){

se.printStackTrace();

}

}

System.out.println("Goodbye!");

}

}

在安装Java的本机，打开cmd对Java程序编译后执行。

在cmd中，进入D:\Download\目录，先对Java程序进行编译（进入Java程序的目录）。

**C:\Users\username >D:**

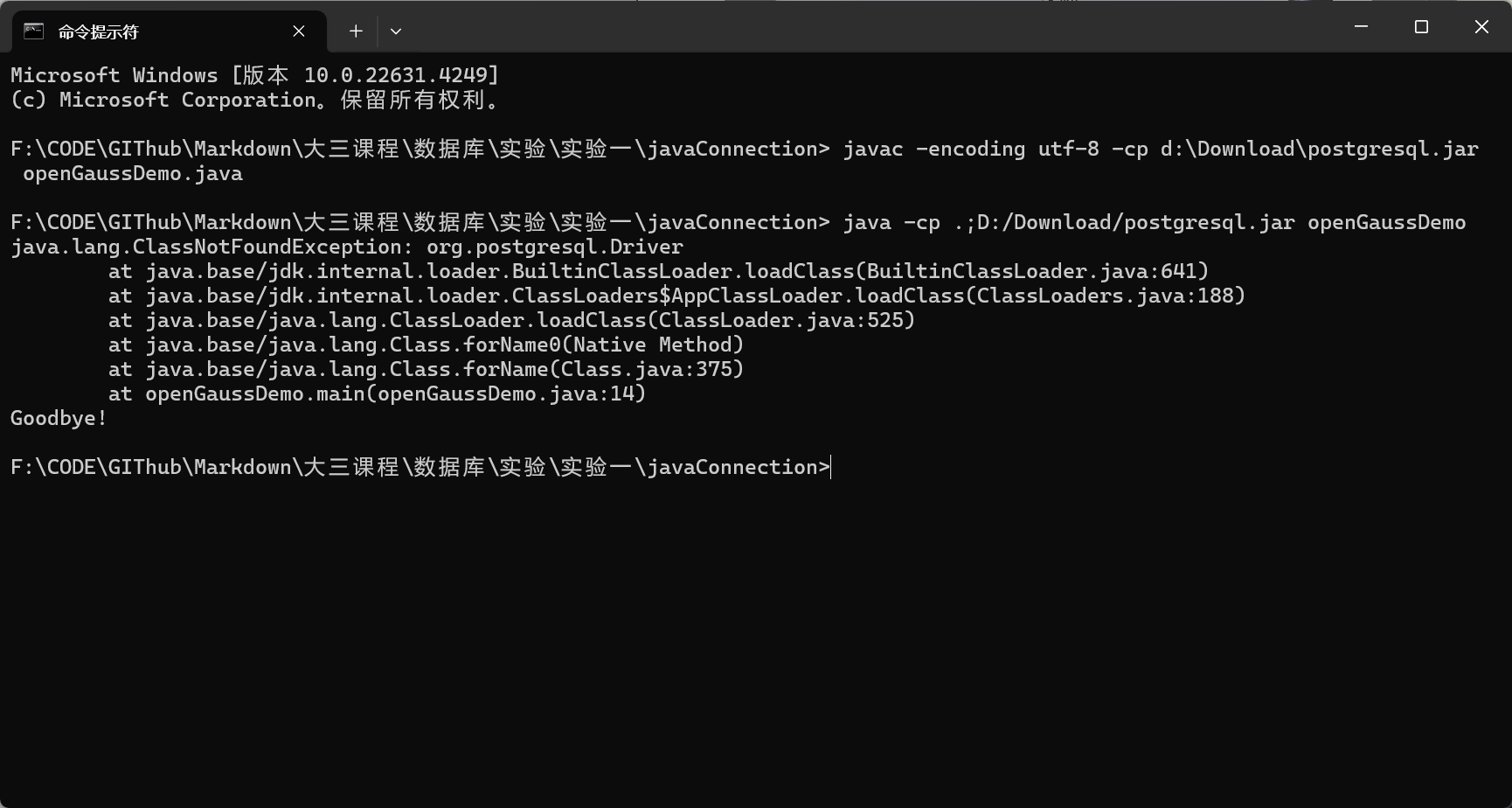
**D:\>cd download**

**D:\Download> javac -encoding utf-8 -cp d:\Download\postgresql.jar openGaussDemo.java**

再执行以下命令。

**D:\Download> java -cp .;D:/Download/postgresql.jar openGaussDemo**

执行结果。

执行结果如下：

连接数据库...

实例化Statement对象...

ID: 1, 站点名称: openGauss, 站点 URL: https://opengauss.org/zh/

ID: 2, 站点名称: 华为云, 站点 URL: https://www.huaweicloud.com/

ID: 3, 站点名称: openEuler, 站点 URL: https://openeuler.org/zh/

ID: 4, 站点名称: 华为support中心, 站点 URL: https://support.huaweicloud.com/

Goodbye!

## 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。

# 附录一：openGauss数据库基本操作

## 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;

## 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