



**数据库实验报告**

实验题目openGauss场景化综合应用实验

学 院 计算机学院

专 业 计算机科学与技术3班

年 级 2020

姓 名 张俊鸿

指导教师 谢锐

（2022年 11 月）

### 实验二 **openGauss场景化综合应用实验**

**实验项目名称：openGauss场景化综合应用实验**

**实验项目性质：验证型**

**所属课程名称：《数据库应用》**

**实验计划学时：3**

**一、实验目的**

1、掌握openGauss数据库的基本操作和SQL语法；

2、运用openGauss数据库的实现场景化应用；

**二、实验内容和要求**

1、在openGauss中创建数据库、模式、用户、表，并熟悉openGauss的操作命令与方式；

2、利用openGauss数据库解决金融场景的实际应用；；

**三、实验主要仪器设备和材料**

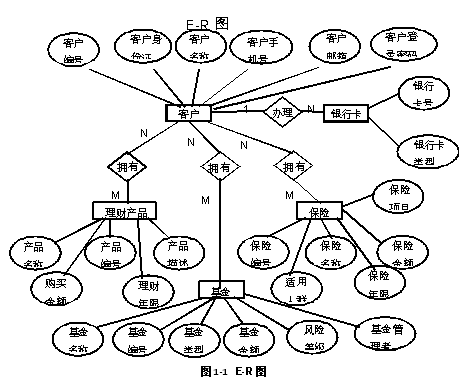
1．计算机及操作系统：PC机，Windows系统/Centos系统；

2．数据库管理系统：openGauss；

**四、实验背景**

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。

**1 实验E-R图**



**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）

**3.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

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。

**4.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 | 说明 |
| 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 | 说明 |
| 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 |

**五、实验方法、步骤及结果测试**

**1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：

**（1）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;

**（2）客户信息表的创建。**

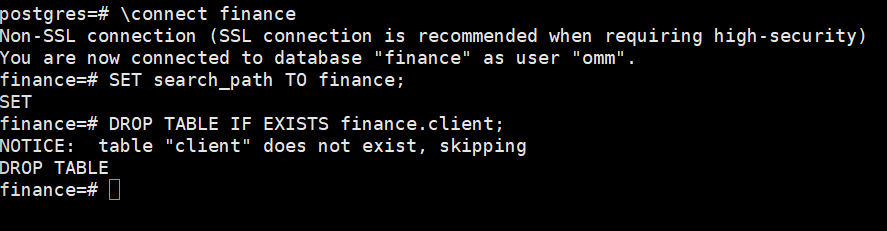
在SQL编辑框中输入如下语句，创建客户信息表client。

如果存在client则先删除表client。

SQL语句：

DROP TABLE IF EXISTS finance.client;

SQL语句及执行结果截图显示



创建表client。

SQL语句：

CREATE TABLE client (c\_id INT PRIMARY KEY,

c\_name VARCHAR(100) NOT NULL ,

c\_mail CHAR(30) UNIQUE,

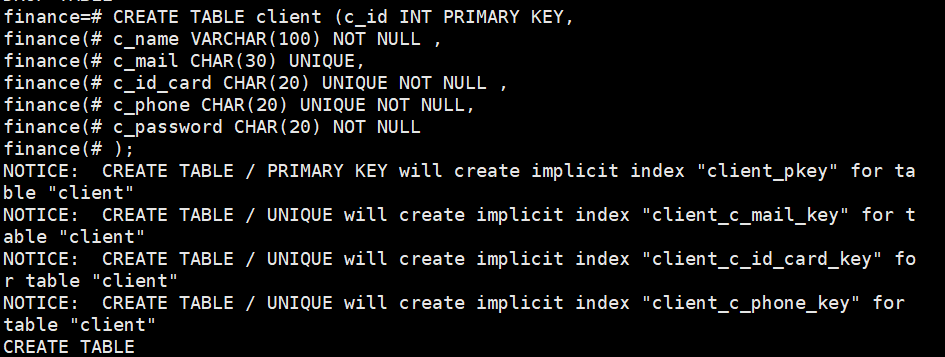
c\_id\_card CHAR(20) UNIQUE NOT NULL ,

c\_phone CHAR(20) UNIQUE NOT NULL,

c\_password CHAR(20) NOT NULL

);

SQL语句及执行结果截图显示



**（3）银行卡信息表的创建。**

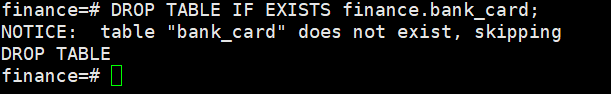
在SQL编辑框中输入如下语句，创建银行卡信息表bank\_card。

如果存在则删除表bank\_card。

SQL语句：

DROP TABLE IF EXISTS finance.bank\_card;

SQL语句及执行结果截图显示



创建表bank\_card。

SQL语句：

CREATE TABLE bank\_card(

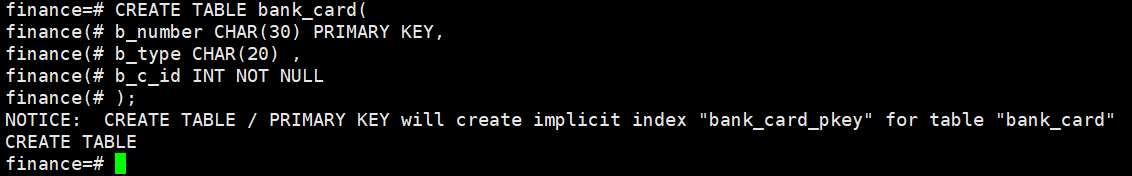
b\_number CHAR(30) PRIMARY KEY,

b\_type CHAR(20) ,

b\_c\_id INT NOT NULL

);

SQL语句及执行结果截图显示



**（4）理财产品信息表的创建。**

创建理财产品信息表finances\_product。

如果存在则删除表finances\_product。

SQL语句：

DROP TABLE IF EXISTS finance.finances\_product;

SQL语句及执行结果截图显示



创建表finances\_product。

SQL语句：

CREATE TABLE finances\_product(

p\_name VARCHAR(100) NOT NULL ,

p\_id INT PRIMARY KEY,

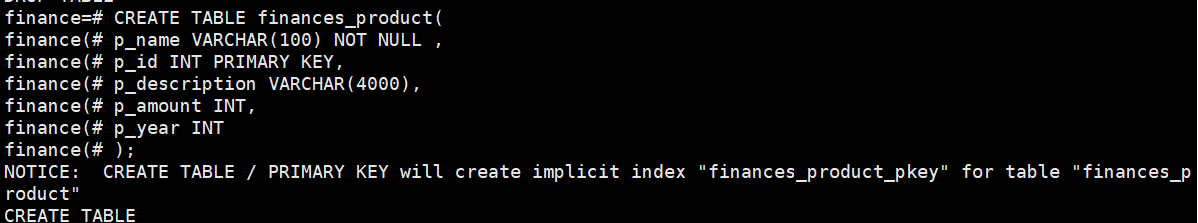
p\_description VARCHAR(4000),

p\_amount INT,

p\_year INT

);

SQL语句及执行结果截图显示



**（5）保险信息表的创建。**

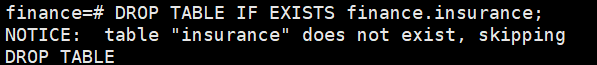
在SQL编辑框中输入如下语句，创建保险信息表insurance。

如果存在则删除表insurance。

SQL语句：

DROP TABLE IF EXISTS finance.insurance;

SQL语句及执行结果截图显示



创建表insurance。

SQL语句：

CREATE TABLE insurance(

i\_name VARCHAR(100) NOT NULL,

i\_id INTEGER PRIMARY KEY,

i\_amount INTEGER,

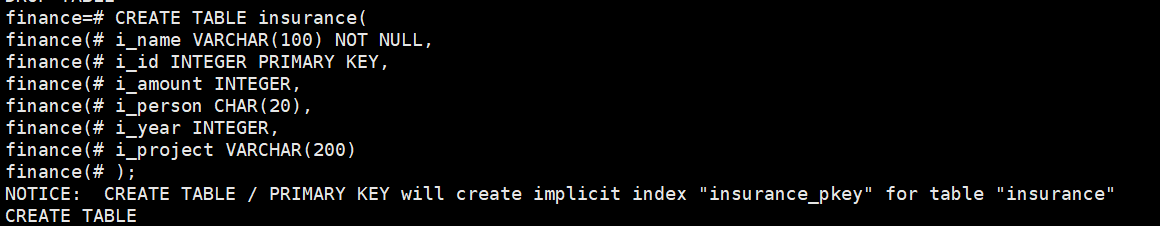
i\_person CHAR(20),

i\_year INTEGER,

i\_project VARCHAR(200)

);

SQL语句及执行结果截图显示



**（6）基金信息表的创建。**

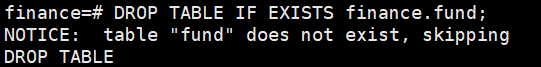
在SQL编辑框中输入如下语句，创建保险信息表fund。

如果存在则删除表fund。

SQL语句：

DROP TABLE IF EXISTS finance.fund;

SQL语句及执行结果截图显示



创建表fund。

SQL语句：

CREATE TABLE fund(

f\_name VARCHAR(100) NOT NULL,

f\_id INTEGER PRIMARY KEY,

f\_type CHAR(20),

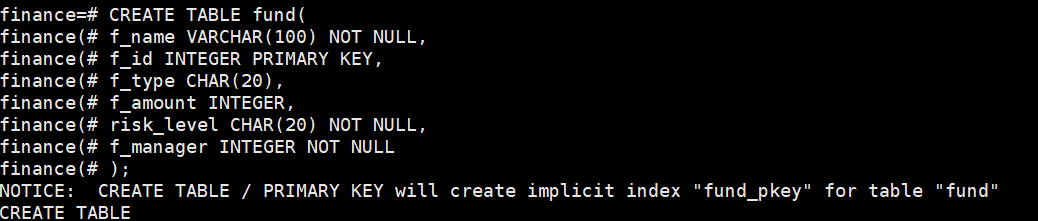
f\_amount INTEGER,

risk\_level CHAR(20) NOT NULL,

f\_manager INTEGER NOT NULL

);

SQL语句及执行结果截图显示



**（7）资产信息表的创建。**

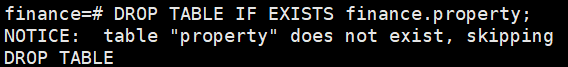
在SQL编辑框中输入如下语句，创建资产信息表property。

如果存在则删除表property。

SQL语句：

DROP TABLE IF EXISTS finance.property;

SQL语句及执行结果截图显示



创建表property。

SQL语句：

CREATE TABLE property(

pro\_id INTEGER PRIMARY KEY,

pro\_c\_id INTEGER NOT NULL,

pro\_pif\_id INTEGER NOT NULL,

pro\_type INTEGER NOT NULL,

pro\_status CHAR(20),

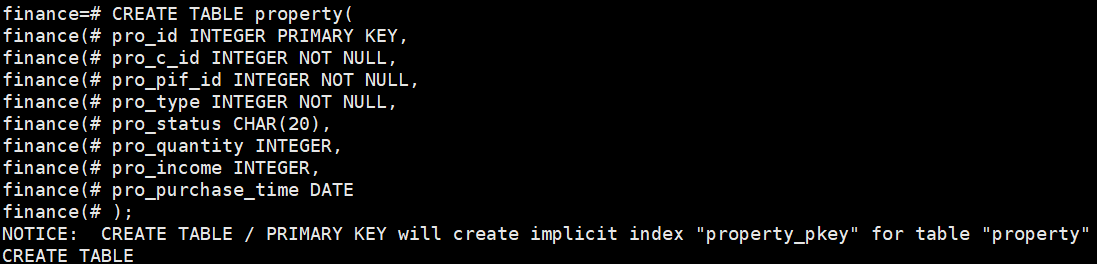
pro\_quantity INTEGER,

pro\_income INTEGER,

pro\_purchase\_time DATE

);

SQL语句及执行结果截图显示



**2.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。

**（1）对client表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password)

VALUES

(1,'张俊鸿','1017328759@qq.com','440514200210133051','15014318971','zjhzjh001'),

(2,'张俊白','1017328758@qq.com','440514200210133052','15014318972','zjhzjh002'),

(3,'张俊红','1017328757@qq.com','440514200210133003','15014318973','zjhzjh003'),

(4,'张俊黑','1017328756@qq.com','440514200210133004','15014318974','zjhzjh004'),

(5,'张俊绿','1017328755@qq.com','440514200210133005','15014318975','zjhzjh005'),

(6,'张俊紫','1017328754@qq.com','440514200210133006','15014318976','zjhzjh006'),

(7,'张俊一','1017328753@qq.com','440514200210133007','15014318977','zjhzjh007'),

(8,'张俊二','1017328752@qq.com','440514200210133008','15014318978','zjhzjh008'),

(9,'张俊三','1017328751@qq.com','440514200210133009','15014318979','zjhzjh009'),

(10,'张俊四','1017328750@qq.com','440514200210133010','15875415010','zjybj110'),

(11,'张俊五','2017328759@qq.com','440514200210133011','15875415011','zjybj111'),

(12,'张俊六','3017328759@qq.com','440514200210133012','15875415012','zjybj112'),

(13,'张俊七','4017328759@qq.com','440514200210133013','15875415013','zjybj113');

SQL语句及执行结果截图显示

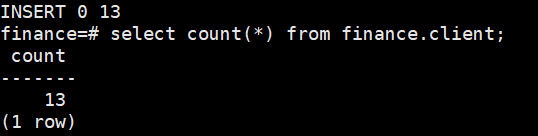


查询插入结果。

SQL语句：

select count(\*) from finance.client;

SQL语句及执行结果截图显示



**（2）对bank\_card表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id)

VALUES

('3120005041','信用卡',1),

('3120005042','信用卡',2),

('3120005043','储蓄卡',3),

('3120005044','信用卡',4),

('3120005045','信用卡',5),

('3120005046','储蓄卡',6),

('3120005047','储蓄卡',7),

('3120005048','信用卡',8),

('3120005049','储蓄卡',9),

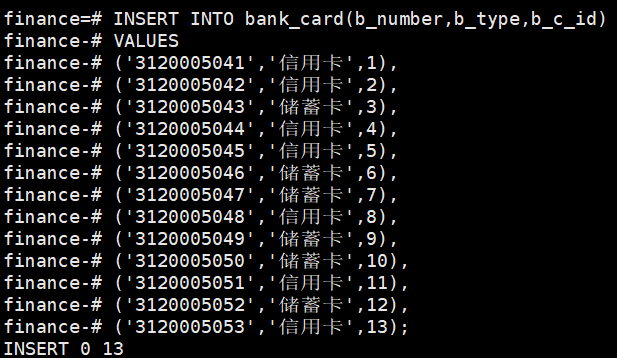
('3120005050','储蓄卡',10),

('3120005051','信用卡',11),

('3120005052','储蓄卡',12),

('3120005053','信用卡',13);

SQL语句及执行结果截图显示

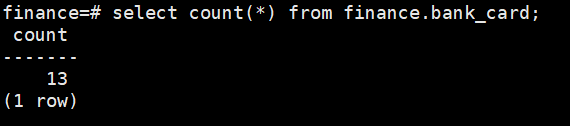


查询插入结果。

SQL语句：

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

SQL语句及执行结果截图显示



**（3）对finances\_product表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO finances\_product(p\_name,P\_id,p\_description,p\_amount,p\_year)

VALUES

('股票',1,'是一种有价证券',8000,6),

('债券',2,'是发行者为筹集资金而发行、在约定时间支付一定比例的利息，并在到期时偿还本金的一种有价证券。',12000,6),

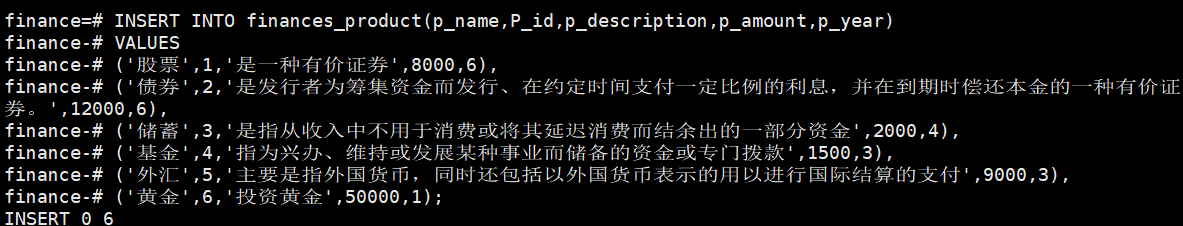
('储蓄',3,'是指从收入中不用于消费或将其延迟消费而结余出的一部分资金',2000,4),

('基金',4,'指为兴办、维持或发展某种事业而储备的资金或专门拨款',1500,3),

('外汇',5,'主要是指外国货币，同时还包括以外国货币表示的用以进行国际结算的支付',9000,3),

('黄金',6,'投资黄金',50000,1);

SQL语句及执行结果截图显示

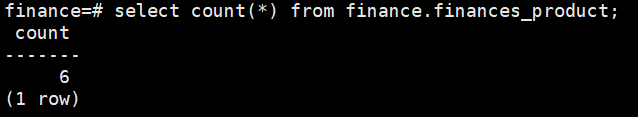


查询插入结果。

SQL语句：

select count(\*) from finance.finances\_product;

SQL语句及执行结果截图显示



**（4）对insurance表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO insurance(i\_name, i\_id, i\_amount, i\_person, i\_year, i\_project)

VALUES ('养老险',1,5000,'老人',15,'养老费用'),

('重病险',2,10000,'所有人',3,'医疗费用'),

('失业险',3,5000,'从业者',12,'保障失业后的基本开支'),

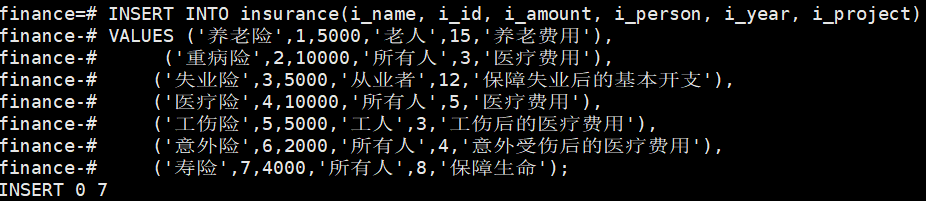
('医疗险',4,10000,'所有人',5,'医疗费用'),

('工伤险',5,5000,'工人',3,'工伤后的医疗费用'),

('意外险',6,2000,'所有人',4,'意外受伤后的医疗费用'),

('寿险',7,4000,'所有人',8,'保障生命');

SQL语句及执行结果截图显示

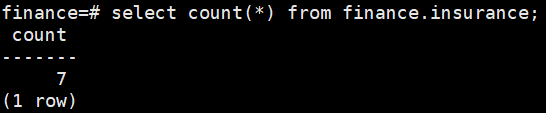


查询插入结果。

SQL语句：

select count(\*) from finance.insurance;

SQL语句及执行结果截图显示



**（5）对fund表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO fund(f\_name, f\_id, f\_type, f\_amount, risk\_level, f\_manager)

VALUES ('易方达基金', 1, '股票型', 10000, '中', '11'),

('广发基金', 2, '股票型', 10000, '中', '22'),

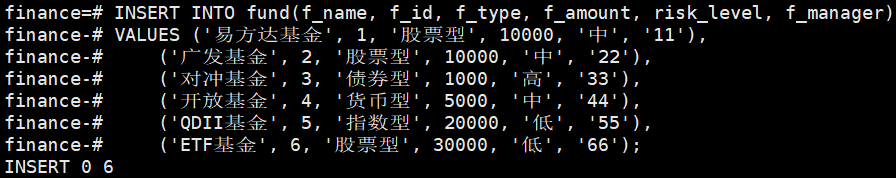
('对冲基金', 3, '债券型', 1000, '高', '33'),

('开放基金', 4, '货币型', 5000, '中', '44'),

('QDII基金', 5, '指数型', 20000, '低', '55'),

('ETF基金', 6, '股票型', 30000, '低', '66');

SQL语句及执行结果截图显示

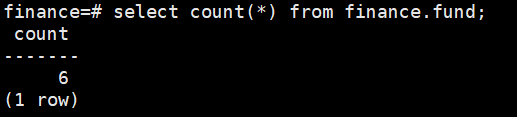


查询插入结果。

SQL语句：

select count(\*) from finance.fund;

SQL语句及执行结果截图显示



**（6）对property表进行数据初始化。**

执行insert操作。

SQL语句：

INSERT INTO property(pro\_id,pro\_c\_id,pro\_pif\_id,pro\_type,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time)

VALUES

(1,1,1,1,'可用',5,2000,'2022-11-07'),

(2,2,3,1,'可用',5,500,'2022-11-08'),

(3,3,5,1,'可用',5,1000,'2022-11-09'),

(4,4,2,2,'可用',5,4000,'2022-11-10'),

(5,5,1,2,'不可用',4,2000,'2022-11-11'),

(6,6,3,2,'可用',4,800,'2022-11-12'),

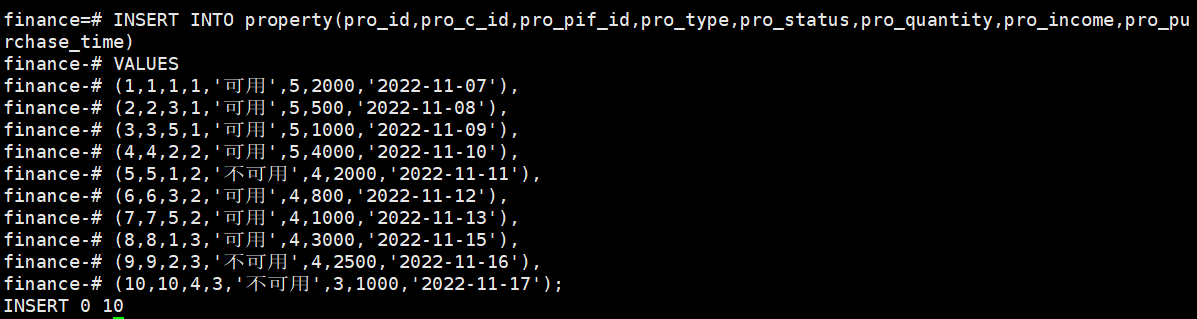
(7,7,5,2,'可用',4,1000,'2022-11-13'),

(8,8,1,3,'可用',4,3000,'2022-11-15'),

(9,9,2,3,'不可用',4,2500,'2022-11-16'),

(10,10,4,3,'不可用',3,1000,'2022-11-17');

SQL语句及执行结果截图显示

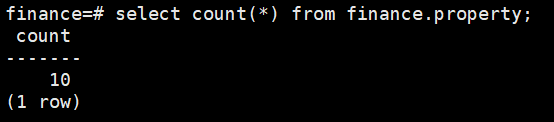


查询插入结果。

SQL语句：

select count(\*) from finance.property;

SQL语句及执行结果截图显示



**3.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。

**（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 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。

**（2）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');

**4.添加约束**

**（1）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);

**（2）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"。

**（3）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);

**（4）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);

**5.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。

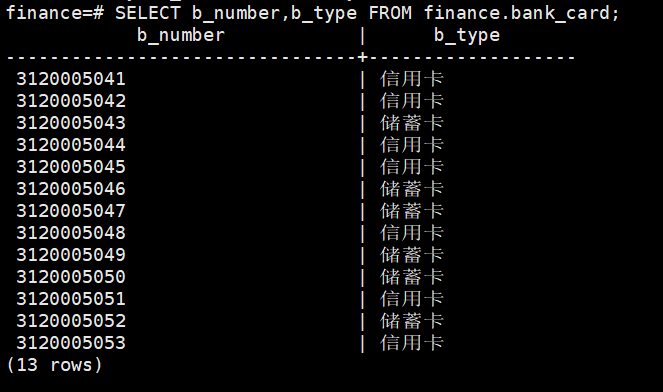
**（1）单表查询。**

* 查询银行卡信息表。

SQL语句：

SELECT b\_number,b\_type FROM finance.bank\_card;

SQL语句及执行结果截图显示



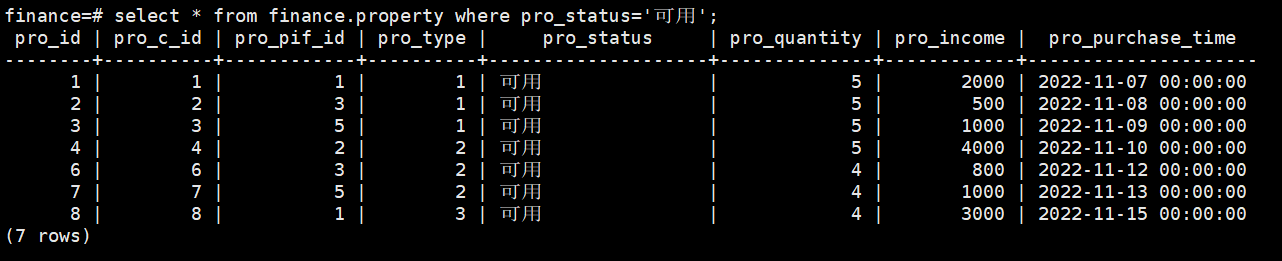
**（2）条件查询。**

* 查询资产信息中‘可用’的资产数据。

SQL语句：

select \* from finance.property where pro\_status='可用';

SQL语句及执行结果截图显示



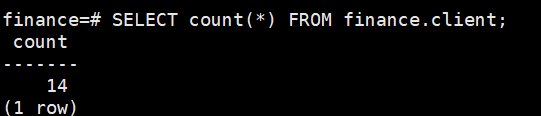
**（3）聚合查询。**

查询用户表中有多少个用户。

SQL语句：

SELECT count(\*) FROM finance.client;

SQL语句及执行结果截图显示

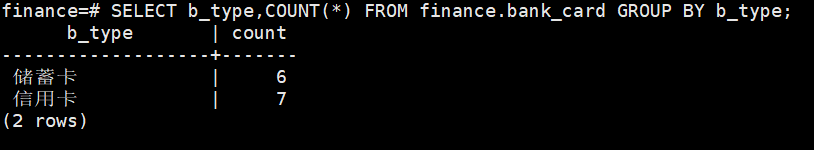


查询银行卡信息表中，储蓄卡和信用卡的个数。

SQL语句：

SELECT b\_type,COUNT(\*) FROM finance.bank\_card GROUP BY b\_type;

SQL语句及执行结果截图显示

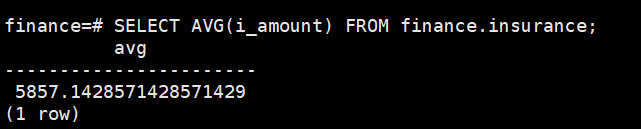


查询保险信息表中，保险金额的平均值。

SQL语句：

SELECT AVG(i\_amount) FROM finance.insurance;

SQL语句及执行结果截图显示



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。

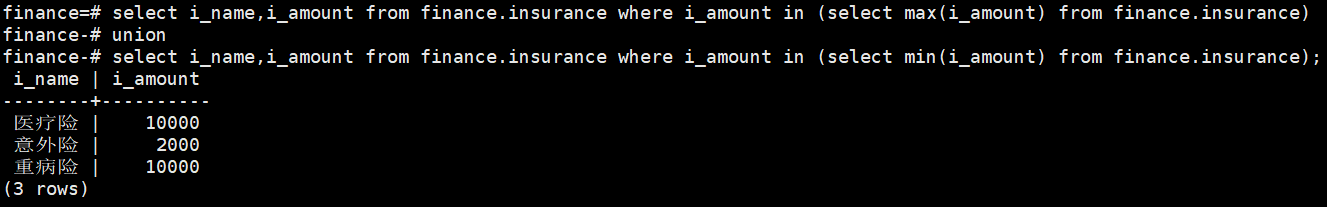
SQL语句：

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);

SQL语句及执行结果截图显示



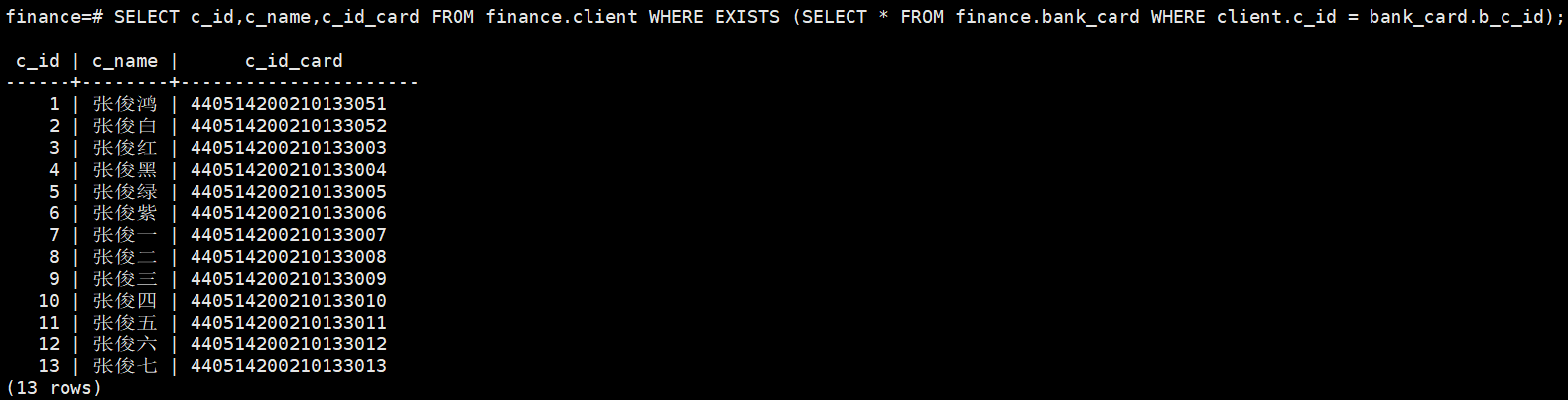
**（4）连接查询（半连接）**

* 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。

SQL语句：

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);

SQL语句及执行结果截图显示



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。

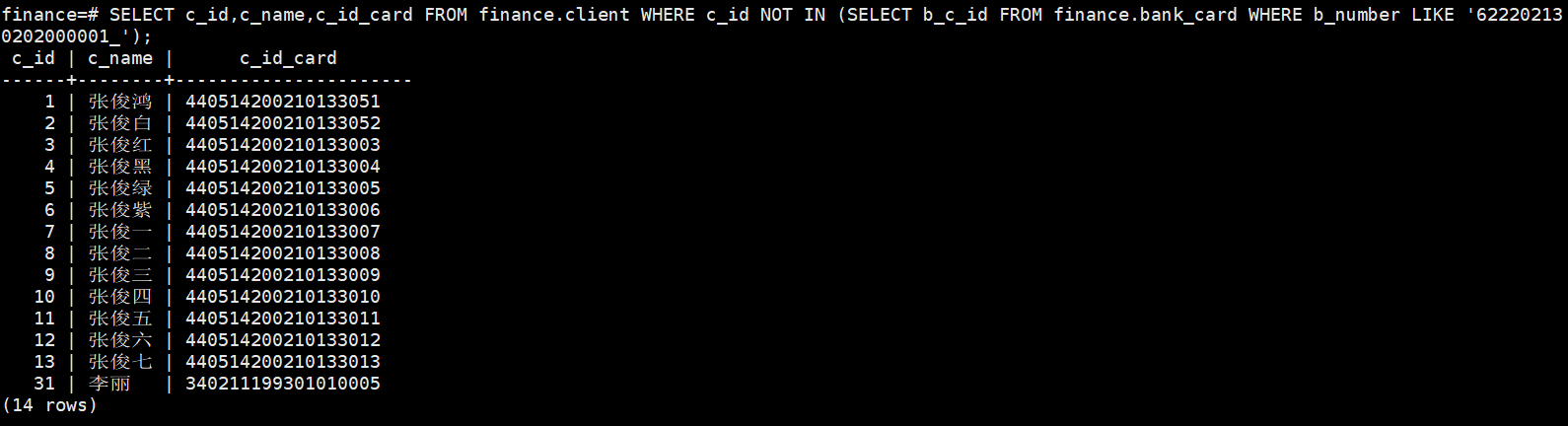
**（5）连接查询（反连接）。**

* 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。

SQL语句：

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\_');

SQL语句及执行结果截图显示



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。

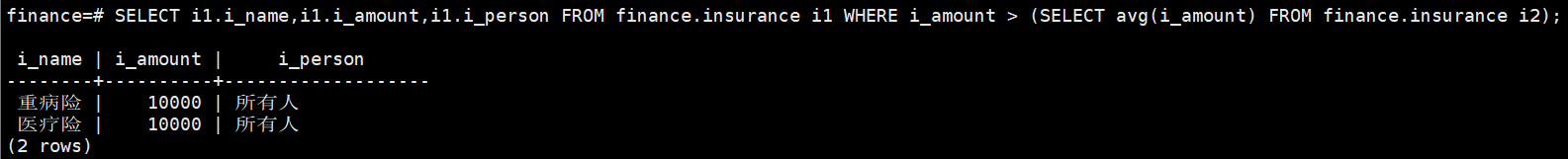
**（6）子查询。**

* 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。

SQL语句：

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);

SQL语句及执行结果截图显示



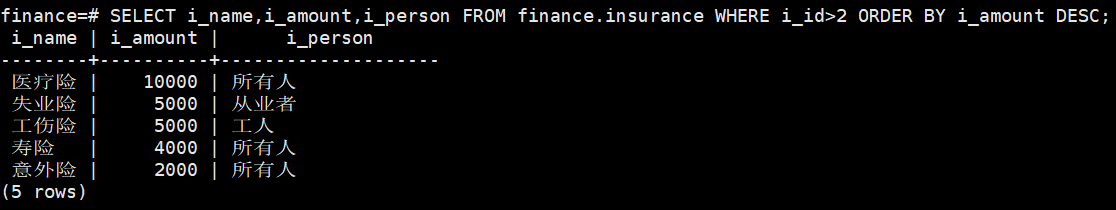
**（7）ORDER BY子句。**

* 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。

SQL语句：

SELECT i\_name,i\_amount,i\_person FROM finance.insurance WHERE i\_id>2 ORDER BY i\_amount DESC;

SQL语句及执行结果截图显示



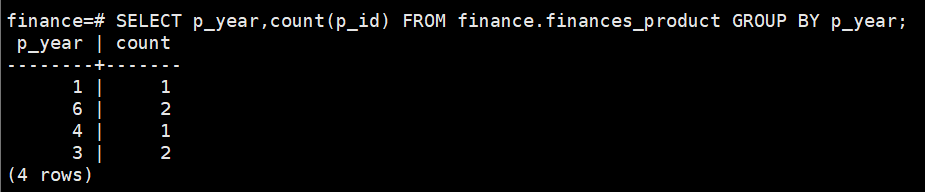
**（8）GROUP BY子句。**

* 查询各理财产品信息总数，按照p\_year分组。

SQL语句：

SELECT p\_year,count(p\_id) FROM finance.finances\_product GROUP BY p\_year;

SQL语句及执行结果截图显示



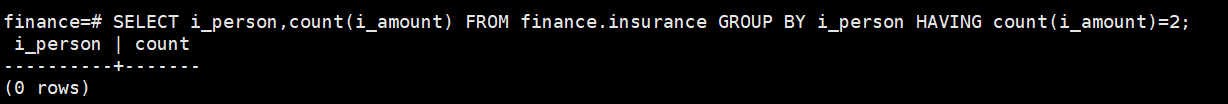
**（9）HAVING子句。**

* 查询保险金额统计数量等于2的适用人群数。

SQL语句：

SELECT i\_person,count(i\_amount) FROM finance.insurance GROUP BY i\_person HAVING count(i\_amount)=2;

SQL语句及执行结果截图显示



备注：HAVING子句依附于GROUP BY子句而存在。

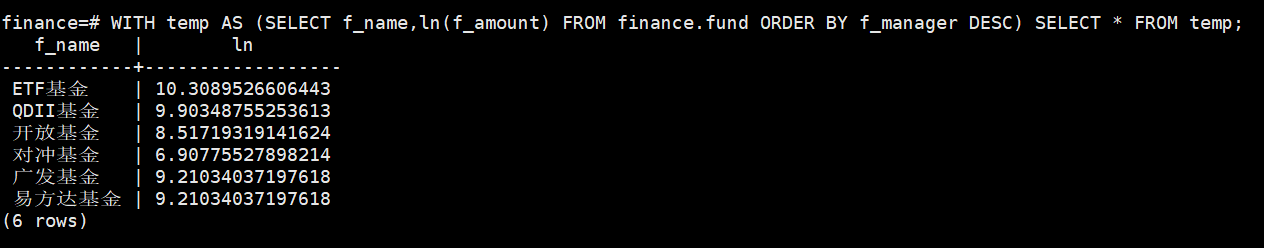
**（10）WITH AS子句。**

* 使用WITH AS查询基金信息表。

SQL语句：

WITH temp AS (SELECT f\_name,ln(f\_amount) FROM finance.fund ORDER BY f\_manager DESC) SELECT \* FROM temp;

SQL语句及执行结果截图显示



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。

**6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。

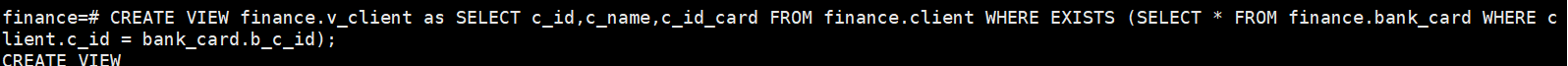
**（1）创建视图。**

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。

SQL语句：

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);

SQL语句及执行结果截图显示

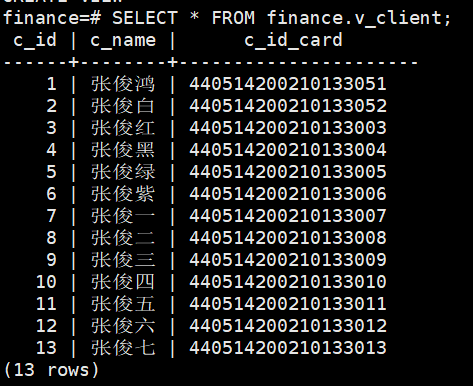


使用视图进行查询。

SQL语句：

SELECT \* FROM finance.v\_client;

SQL语句及执行结果截图显示



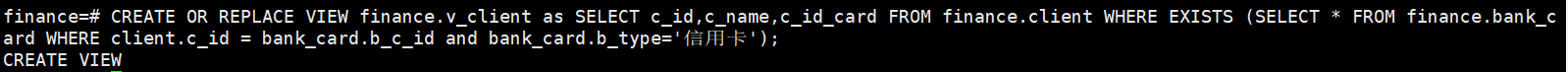
**（2）修改视图内容**

修改视图，在原有查询的基础上，过滤出信用卡用户。

SQL语句：

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');

SQL语句及执行结果截图显示

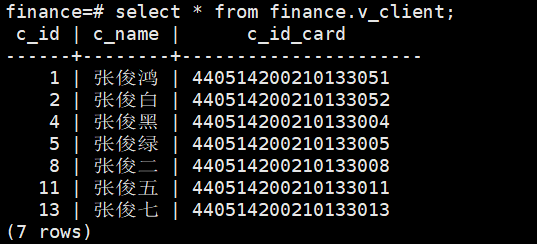


使用视图进行查询。

SQL语句：

select \* from finance.v\_client;

SQL语句及执行结果截图显示



**（3）修改视图名称。**

SQL语句：

ALTER VIEW finance.v\_client RENAME TO v\_client\_new;

SQL语句及执行结果截图显示



**（4）删除视图。**

将v\_client视图删除，删除视图不影响基表。

SQL语句：

DROP VIEW finance.v\_client\_new;

SQL语句及执行结果截图显示



**7索引**

**（1）创建索引。**

* 在普通表property上创建索引。

SQL语句：

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

SQL语句及执行结果截图显示



**（2）重命名索引。**

* 在普通表property上重建及重命名索引。

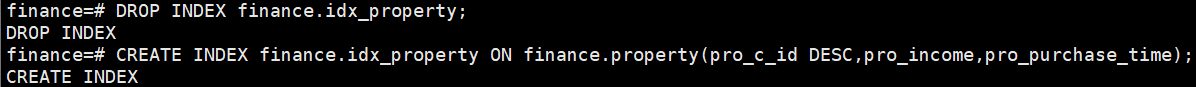
重建索引。

SQL语句：

DROP INDEX finance.idx\_property;

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

SQL语句及执行结果截图显示



重命名索引。

SQL语句：

ALTER INDEX finance.idx\_property RENAME TO idx\_property\_temp;

SQL语句及执行结果截图显示



**删除索引。**

* 删除索引idx\_property\_temp。

SQL语句：

DROP INDEX finance.idx\_property\_temp;

SQL语句及执行结果截图显示



**8数据的修改和删除**

**（1）修改数据。**

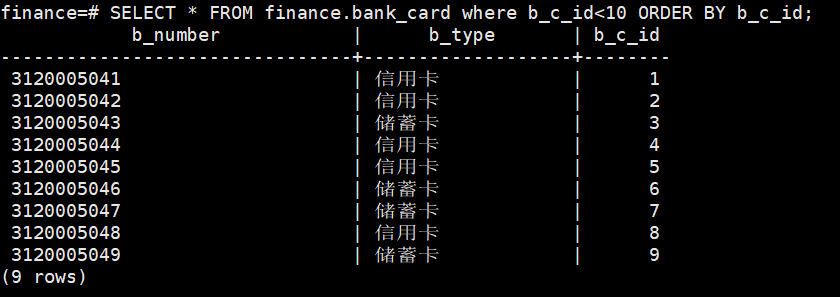
* 修改/更新银行卡信息表中b\_c\_id小于10和客户信息表中c\_id相同的记录的b\_type字段。

查看表数据。

SQL语句：

SELECT \* FROM finance.bank\_card where b\_c\_id<10 ORDER BY b\_c\_id;

SQL语句及执行结果截图显示

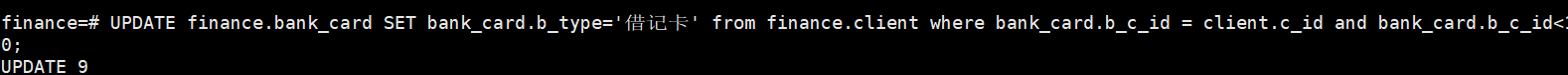


开始更新数据：

SQL语句：

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;

SQL语句及执行结果截图显示

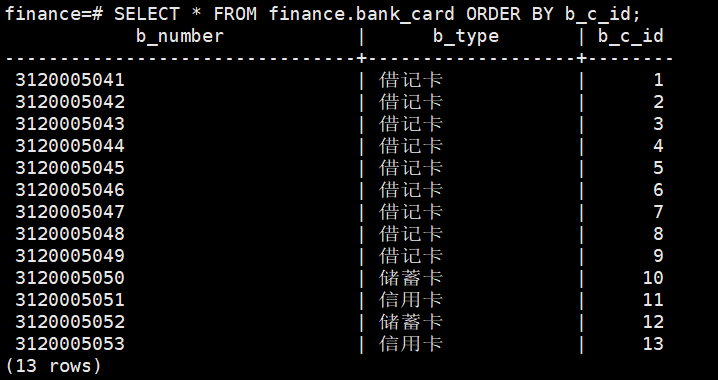


重新查询数据情况。

SQL语句：

SELECT \* FROM finance.bank\_card ORDER BY b\_c\_id;

SQL语句及执行结果截图显示



**（2）删除指定数据。**

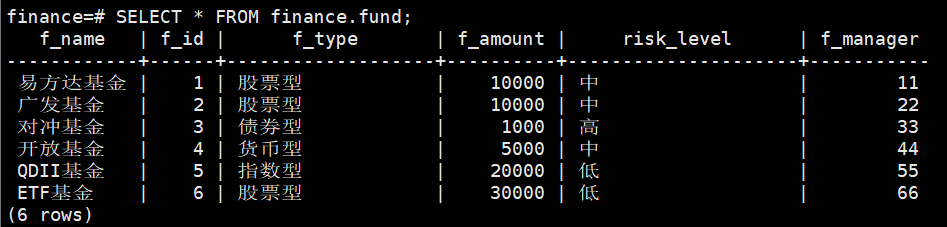
删除基金信息表中编号小于3的行。

删除前查询结果。

SQL语句：

SELECT \* FROM finance.fund;

SQL语句及执行结果截图显示



开始删除数据：

SQL语句：

DELETE FROM finance.fund WHERE f\_id<3;

SQL语句及执行结果截图显示

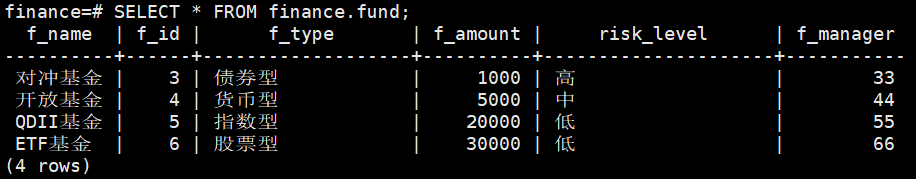


查询删除结果。

SQL语句：

SELECT \* FROM finance.fund;

SQL语句及执行结果截图显示



**9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：

**（1）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';

**（2）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;

**（3）退出数据库：**

postgres=#**\q**

**10新用户连接数据库**

**（1）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。

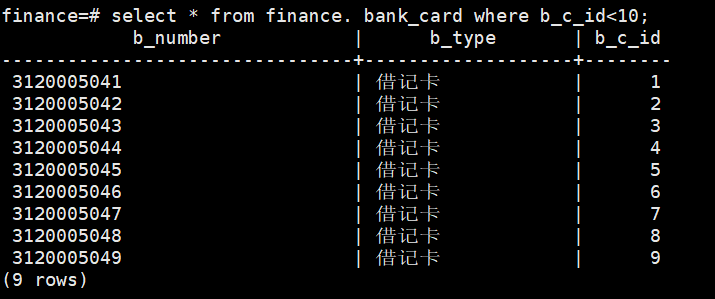
gsql -d finance -U dbuser -p 26000 -r

**（2）访问finance数据库的表bank\_card。**

SQL语句：

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

SQL语句及执行结果截图显示



**退出数据库：**

postgres=#**\q**

**11删除Schema**

**（1）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

**（2）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

**（3）设置默认查询为finance。**

set search\_path to finance;

**（4）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}

**（5）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.

**（6）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

**（7）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.

**（8）退出数据库：**

postgres=#**\q**

**六、实验总结：**

**七、问题与思考：**

1. openGauss数据库的特点；
   1. 高性能：提供Sql-bypass智能快速引擎技术
   2. 高可用：支持主备同步、数据页CRC校验
   3. 高安全：慢SQL诊断，提供在线自学习的SQL时间预测
   4. 全开放：允许对代码自由修改、使用
2. 如何应用openGauss数据库解决实际问题；

以课程设计平行志愿模拟录取为例，可以使用openGauss数据库系统设计相应的表和模式，录入学生的考生号，分数等信息，以及结合学校的招生计划表，实现志愿顺序录取以及调剂录取等功能。