数据库开发查询实验

2112060-信息安全-孙蕗

**实验要求：**

请完成场景化综合实验并提交实验报告，实验报告应包括但不限于如下内容：

1. 实验环境说明，并说明你选择此实验环境进行实验的原因。
2. 1.1.3-1.1.13中完成主要步骤后的执行结果截图（每小节截图不少于2张）。
3. 对于1.1.7中的每个查询需求，请分别提供对应的SQL查询语句和能够满足查询需求的关系代数表达式。
4. 如果你的初始SQL执行结果和要求的执行结果不符，其原因是什么？请就和要求结果不符的SQL执行内容分别进行说明。
5. 实验总时长分析及遇到的问题、以及实验中学习到的知识点分析。

1.0 前置环境：

在openGauss进行

## 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、基金和资产。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，如资产，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，并对其进行多种数据库操作。

### E-R图



E-R图

### 关系模式

对于C银行中的6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，所属客户编号）
* 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）
* 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）

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)
* 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\_c\_id，pro\_id，pro\_status，pro\_quantity，pro\_income，pro\_purchase\_time)

对象之间的关系：

* 一个客户可以办理多张银行卡
* 一个客户可有多笔资产
* 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

openGauss数据模型表操作

### 创建数据表 （请自行完成数据库表的创建，并参考手册内容，文字已设为白色，请完成此步骤后更改文字颜色进行对照）

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。

使用gsql工具登陆数据库。

gsql -d postgres -p 26000

创建数据库financeL。

CREATE DATABASE finance ENCODING 'UTF8' template = template0;

连接finance数据库。

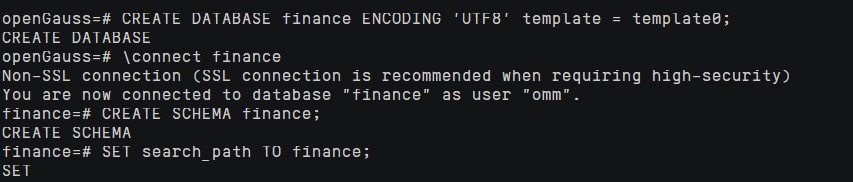
\connect finance

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,

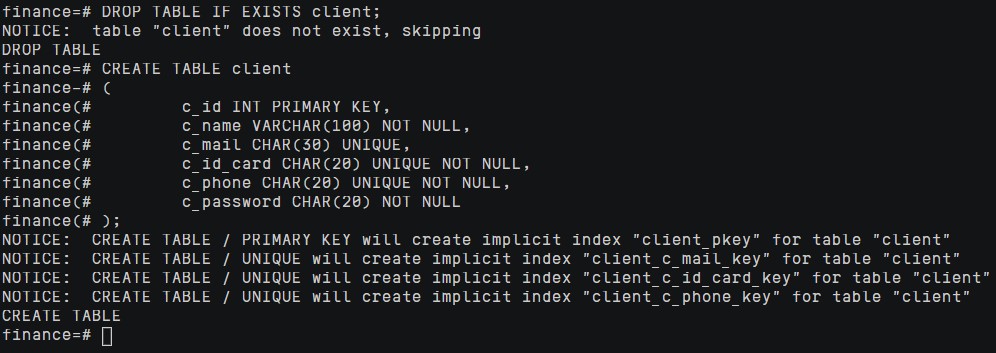
c\_mail CHAR(30) UNIQUE,

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

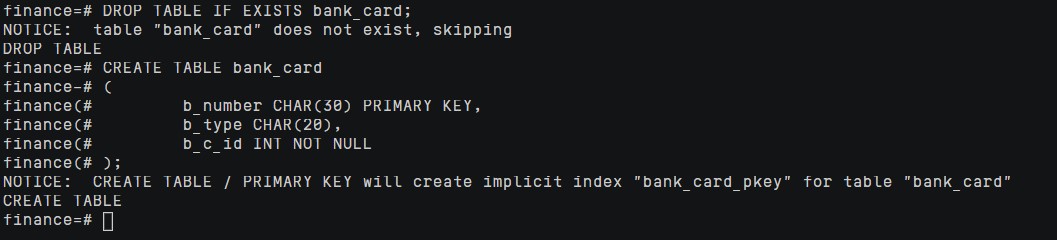
(

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,

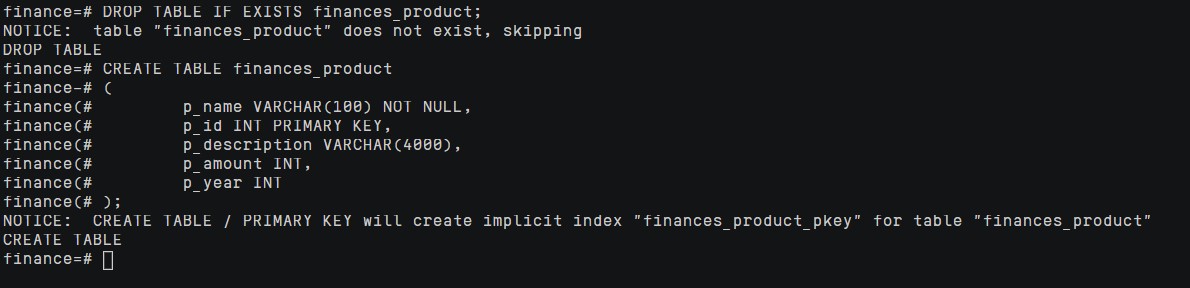
p\_id INT PRIMARY KEY,

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,

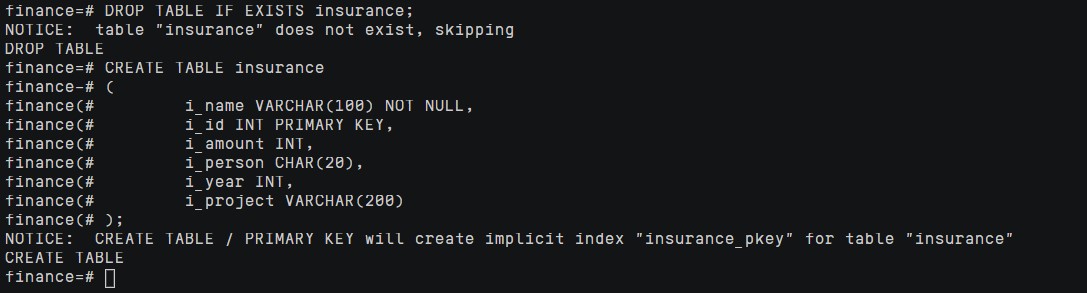
i\_amount INT,

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,

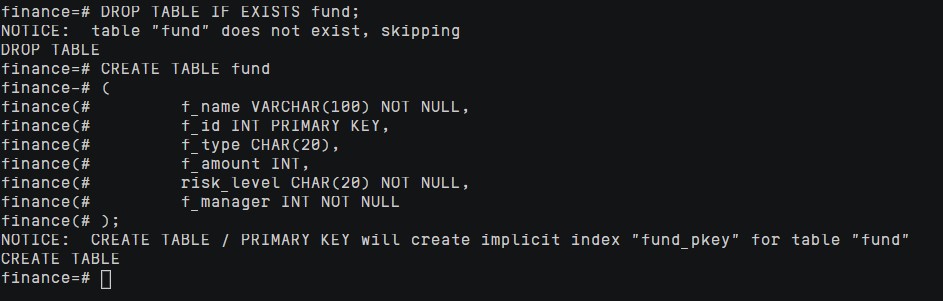
f\_type CHAR(20),

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\_c\_id INT NOT NULL,

pro\_id INT PRIMARY KEY,

pro\_status CHAR(20),

pro\_quantity INT,

pro\_income INT,

pro\_purchase\_time DATE

);

### 

### 7. 资产信息表的创建

### 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');

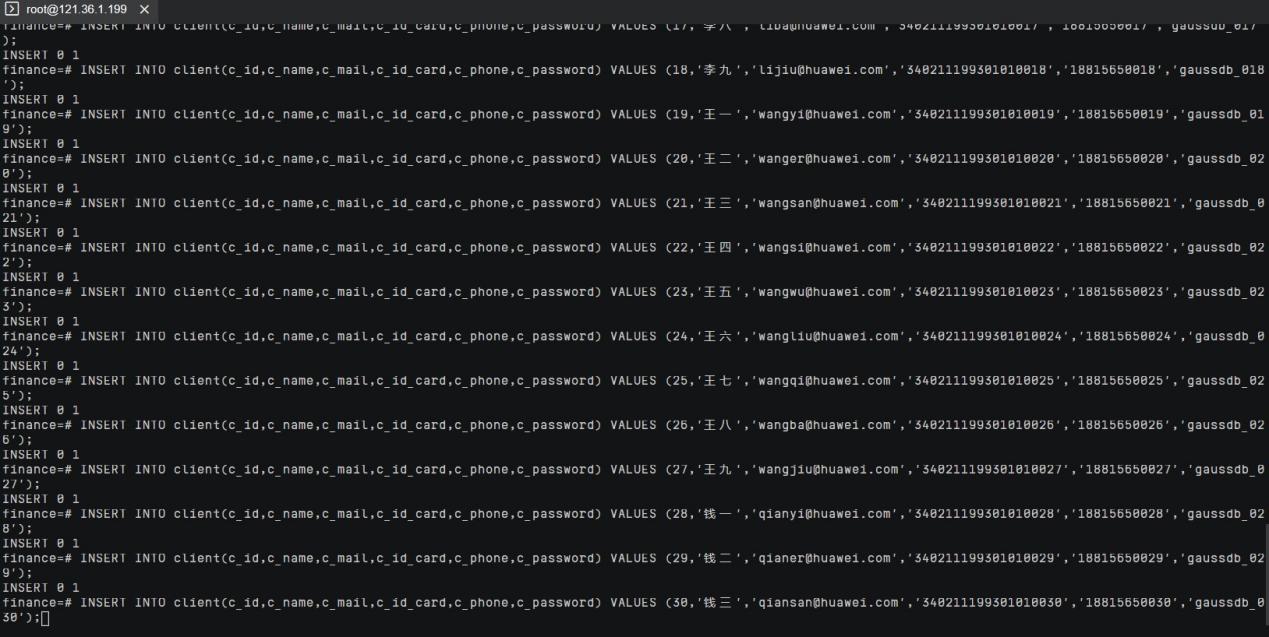
INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (26,'王八','wangba@huawei.com','340211199301010026','18815650026','gaussdb\_026');

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。

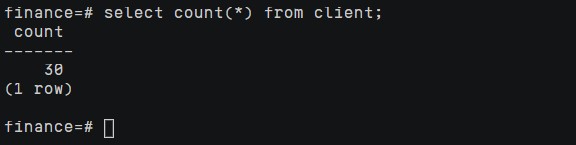
select count(\*) from client;

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);

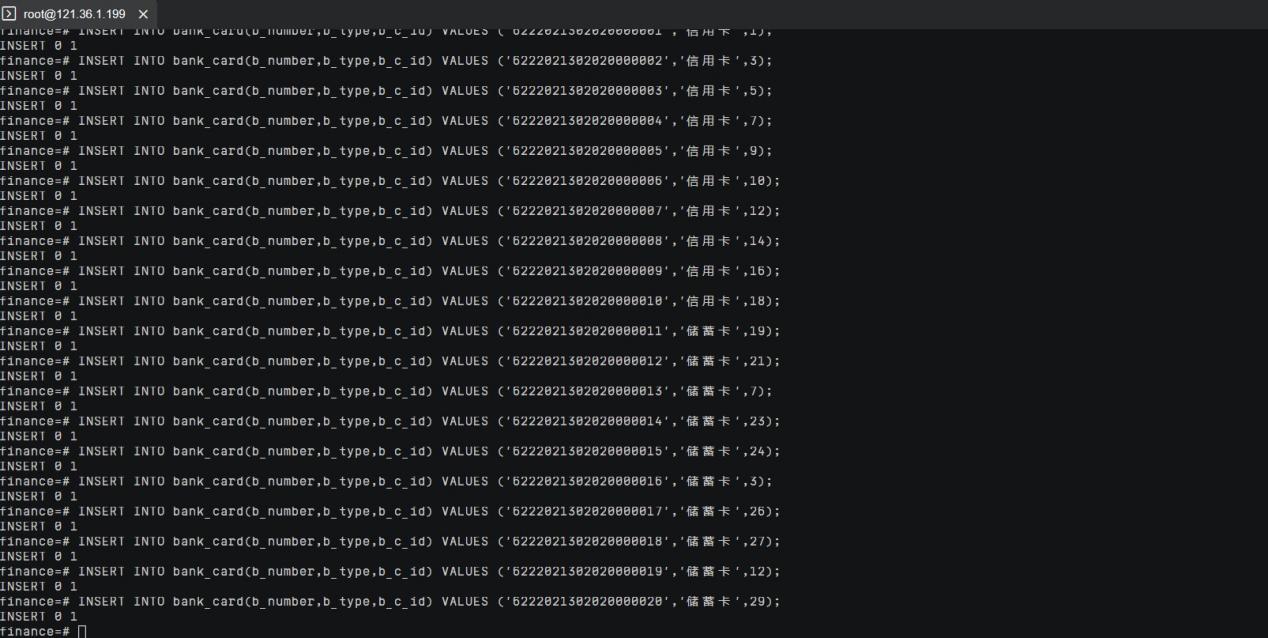
INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000016','储蓄卡',3);

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。

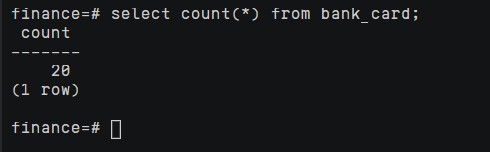
select count(\*) from bank\_card;

结果为：

count(\*)

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

20



对finances\_product表进行数据初始化。

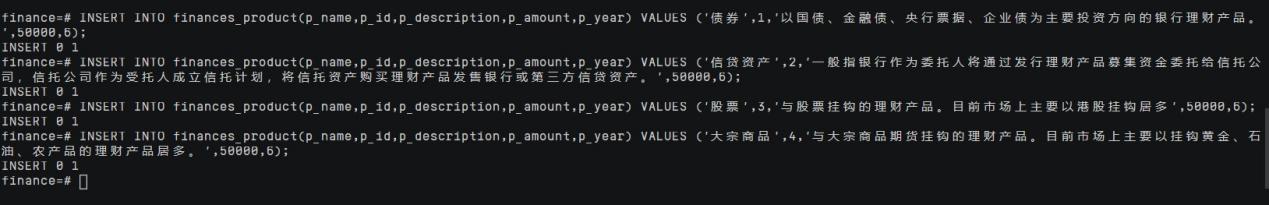
执行insert操作。

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。

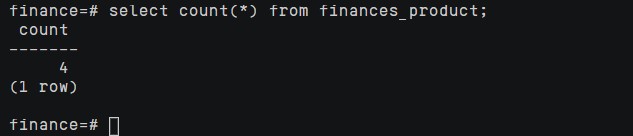
select count(\*) from finances\_product;

结果为：

count(\*)

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

4



对insurance表进行数据初始化。

执行insert操作。

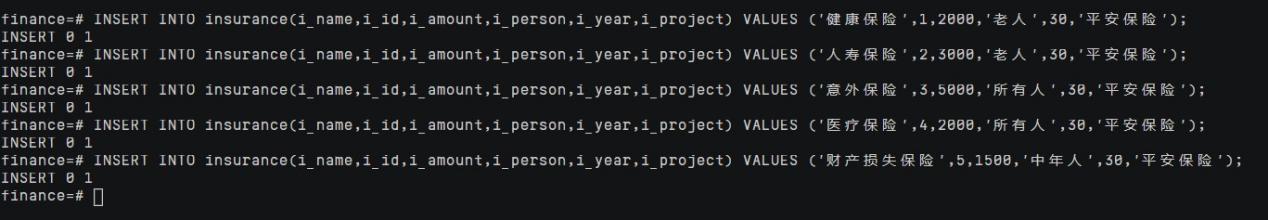
INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('健康保险',1,2000,'老人',30,'平安保险');

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。

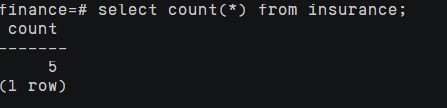
select count(\*) from insurance;

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。

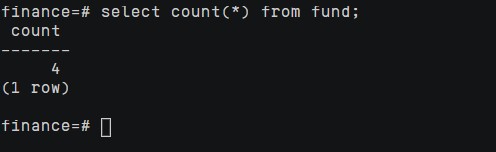
select count(\*) from fund;

结果为：

count(\*)

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

4



对property表进行数据初始化。

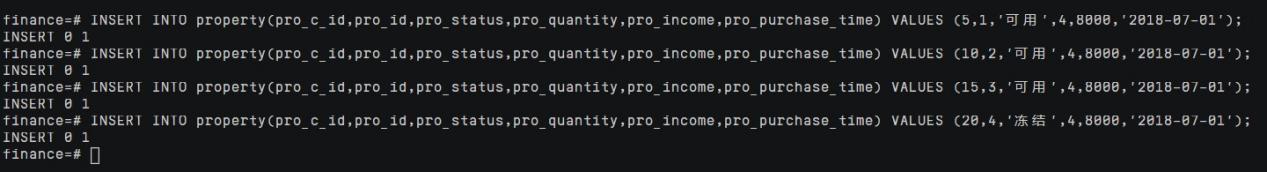
执行insert操作。

INSERT INTO property(pro\_c\_id,pro\_id,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (5,1,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_c\_id,pro\_id,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (10,2,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_c\_id,pro\_id,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (15,3,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_c\_id,pro\_id,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (20,4,'冻结',4,8000,'2018-07-01');



查询插入结果。

select count(\*) from property;

结果为：

count(\*)

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

4

### 

### 19. 查询插入结果

### 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');

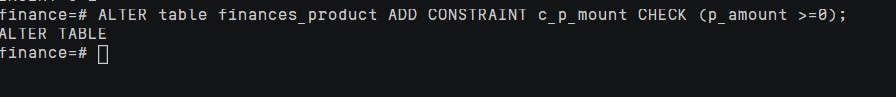
### 21. 在金融数据库的客户信息表中添加一个客户的信息。（插入成功的场景）。

### 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。

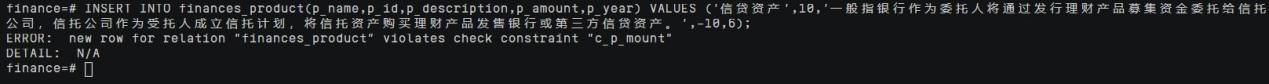
ALTER table finances\_product ADD CONSTRAINT c\_p\_mount CHECK (p\_amount >=0);



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);

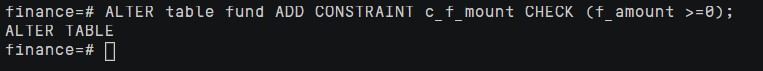
执行失败，失败原因：new row for relation "finances\_product" violates check constraint "c\_p\_mount"。



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);

### 25. 步骤 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

### 查询数据 （请根据要求自行完成相应SQL语句的查询，再和手册进行对照）

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

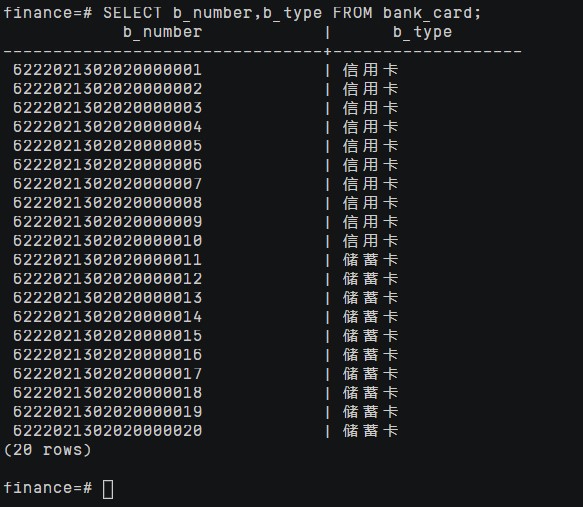
6222021302020000016 | 储蓄卡

6222021302020000017 | 储蓄卡

6222021302020000018 | 储蓄卡

6222021302020000019 | 储蓄卡

6222021302020000020 | 储蓄卡



关系代数表达式：π(b\_number, b\_type)(bank\_card)

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：

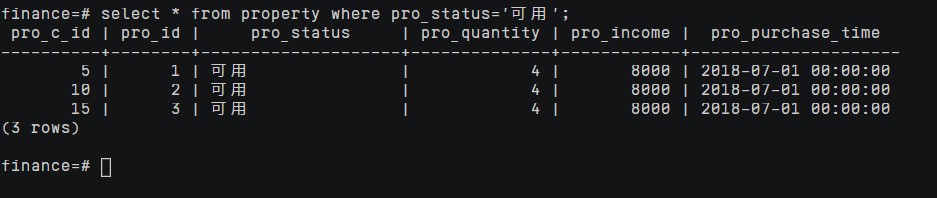
pro\_c\_id | pro\_id | pro\_status | pro\_quantity | pro\_income | pro\_purchase\_time

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

5 | 1 | 可用 | 4 | 8000 | 2018-07-01 00:00:00

10 | 2 | 可用 | 4 | 8000 | 2018-07-01 00:00:00

15 | 3 | 可用 | 4 | 8000 | 2018-07-01 00:00:00



关系代数表达式：σ(pro\_status = '可用')(property)

聚合查询。

* 查询用户表中有多少个用户。

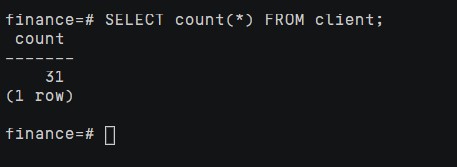
SELECT count(\*) FROM client;

结果如下：

count

-------

31



关系代数表达式：γ()(client)

* 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;

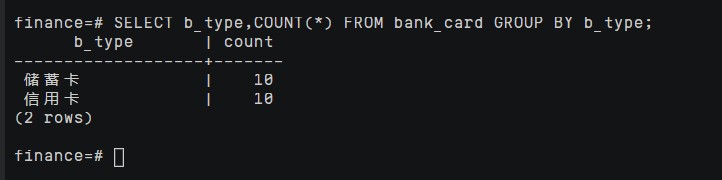
结果如下：

b\_type | count

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

储蓄卡 | 10

信用卡 | 10



关系代数表达式：γ(b\_type, COUNT)(bank\_card)

* 查询保险信息表中，保险金额的平均值。

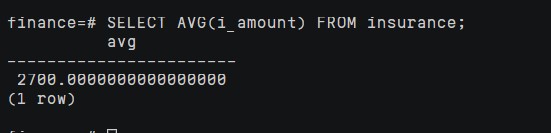
SELECT AVG(i\_amount) FROM insurance;

结果如下：

avg

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

2700.0000000000000000



关系代数表达式：γ(AVG(i\_amount))(insurance)

* 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);

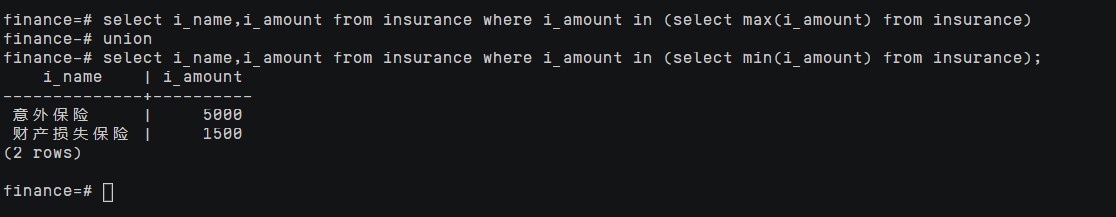
结果如下：

i\_name | i\_amount

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

财产损失保险 | 1500

意外保险 | 5000



关系代数表达式：(π(i\_name, i\_amount)(σ(i\_amount = MAX(i\_amount))(insurance))) ∪ (π(i\_name, i\_amount)(σ(i\_amount = MIN(i\_amount))(insurance)))

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

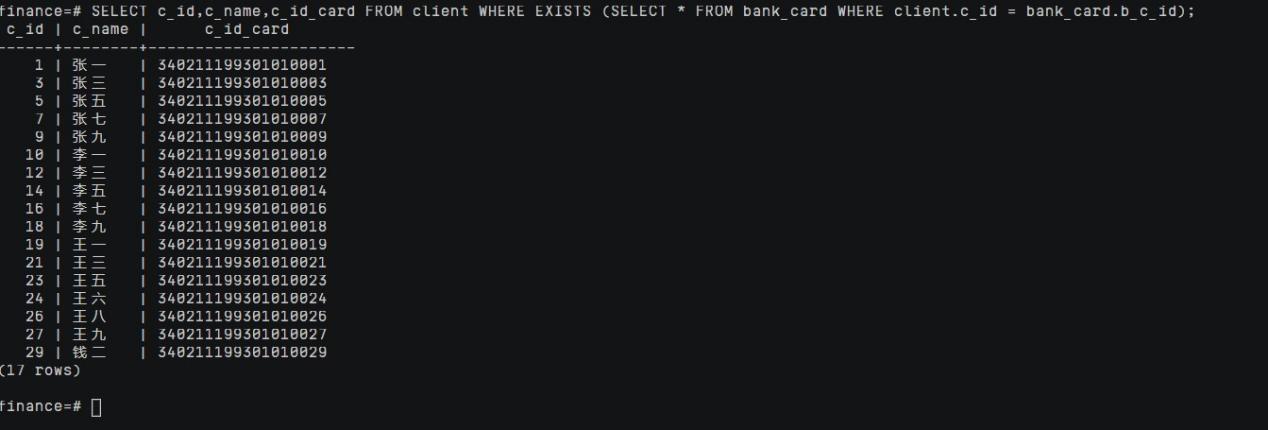
24 | 王六 | 340211199301010024

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。



关系代数表达式：π(c\_id, c\_name, c\_id\_card)(σ(∃b\_c\_id (client.c\_id = bank\_card.b\_c\_id))(client ⨝ bank\_card))

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。



关系代数表达式：π(c\_id, c\_name, c\_id\_card)(σ(c\_id ∉ (π(b\_c\_id)(σ(b\_number LIKE '622202130202000001\_')(bank\_card))))(client))

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);

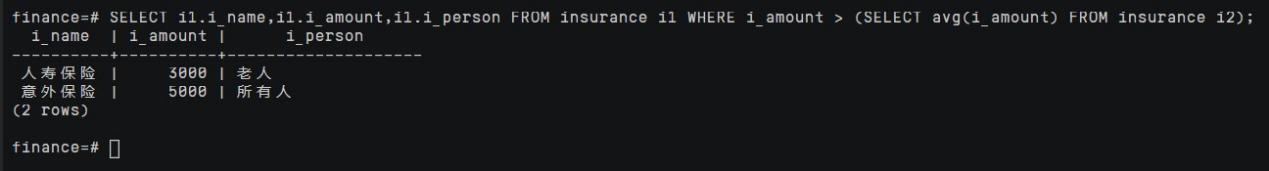
结果如下：

i\_name | i\_amount | i\_person

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

人寿保险 | 3000 | 老人

意外保险 | 5000 | 所有人



关系代数表达式：π(i\_name, i\_amount, i\_person)(σ(i\_amount > AVG(i\_amount))(insurance))

ORDER BY和GROUP BY。

1. ORDER BY子句。

* 按照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：

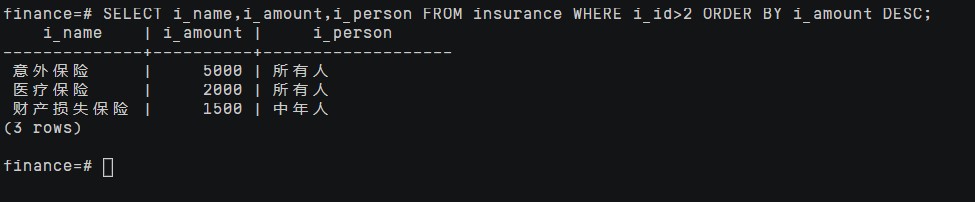
i\_name | i\_amount | i\_person

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

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



关系代数表达式：π(i\_name, i\_amount, i\_person)(σ(i\_id > 2)(insurance) ⨝\_{i\_amount DESC} insurance)

1. GROUP BY子句。

* 查询各保险信息总数，按照p\_year分组。

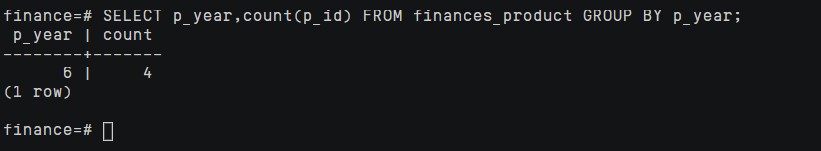
SELECT p\_year,count(p\_id) FROM finances\_product GROUP BY p\_year;

结果如下：

p\_year | count

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

6 | 4



关系代数表达式：γ(p\_year, COUNT(p\_id))(finances\_product)

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：

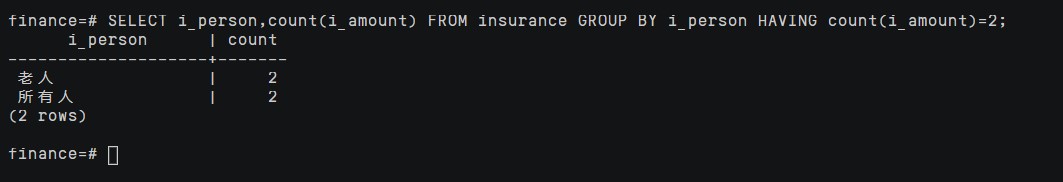
i\_person | count

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

所有人 | 2

老人 | 2

备注：HAVING子句依附于GROUP BY子句而存在。



关系代数表达式：γ(i\_person, COUNT(i\_amount))(σ(COUNT(i\_amount) = 2)(insurance))

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

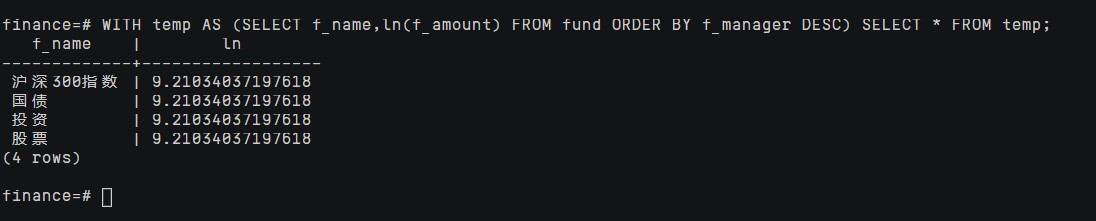
国债 | 9.21034037197618

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。



关系代数表达式：temp ← π(f\_name, LN(f\_amount))(σ(f\_manager DESC)(fund))

π(temp)

### 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

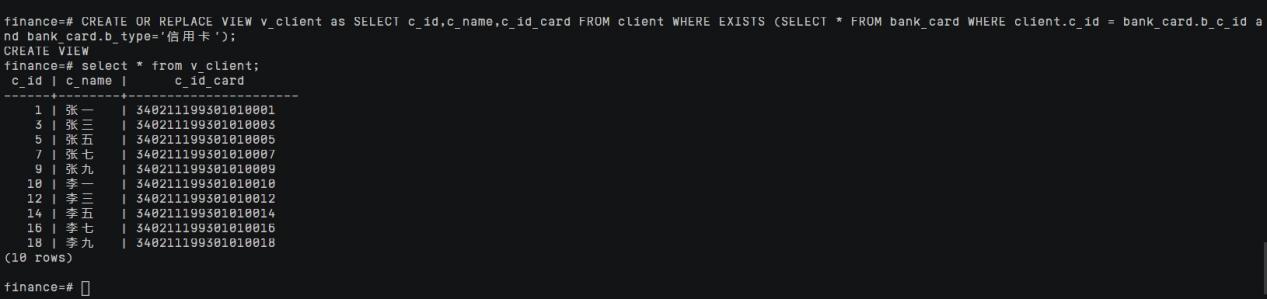
14 | 李五 | 340211199301010014

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;

### 

### 41. 修改视图，删除视图

### 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;

### 

### 42. 创建，重建，重命名，删除索引

### 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

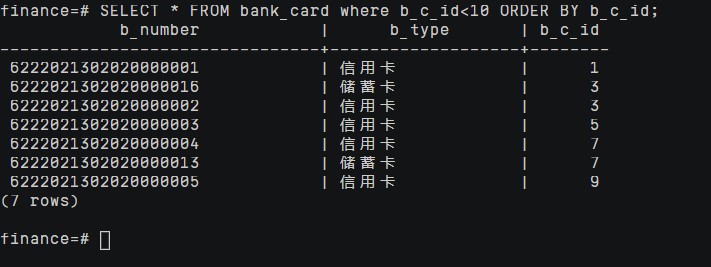
6222021302020000002 | 信用卡 | 3

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

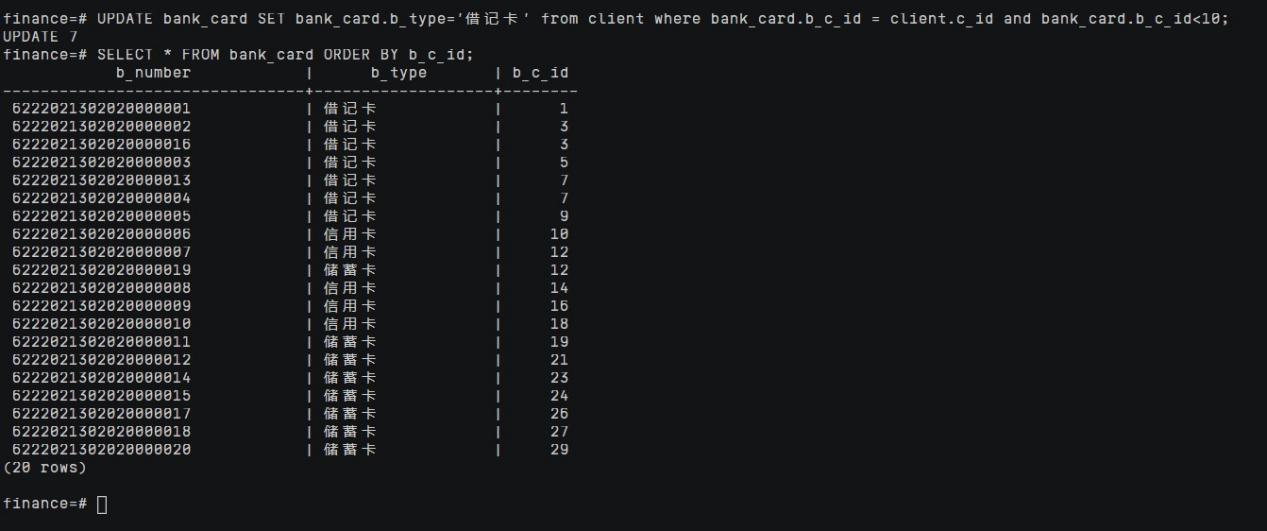
6222021302020000014 | 储蓄卡 | 23

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

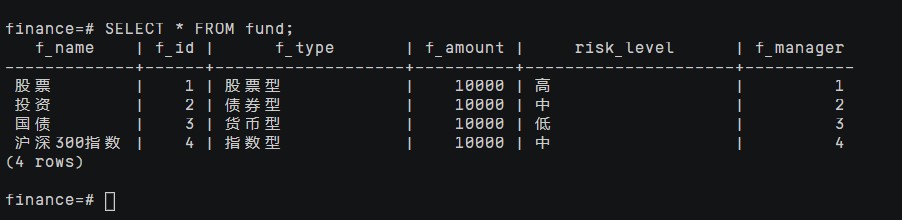
-------------+------+-------------------+----------+---------------------+-----------

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

### 

### 46. 删除数据，查询结果

### 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;

### 

### 47. 连接数据库后，进入SQL命令界面。创建用户dbuser，密码为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。

### 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;

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

### 

### 48. 访问finance数据库的表bank_card

### 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}

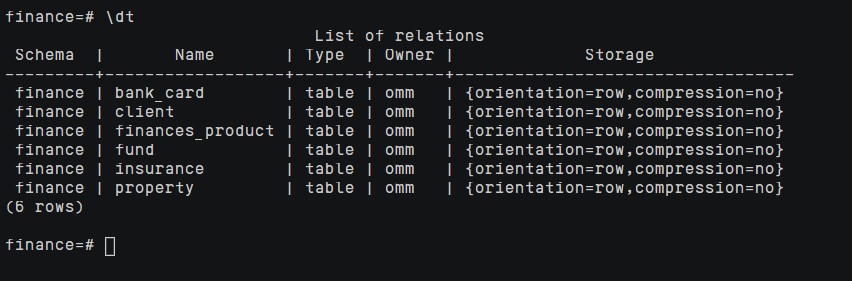
finance | client | table | omm | {orientation=row,compression=no}

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

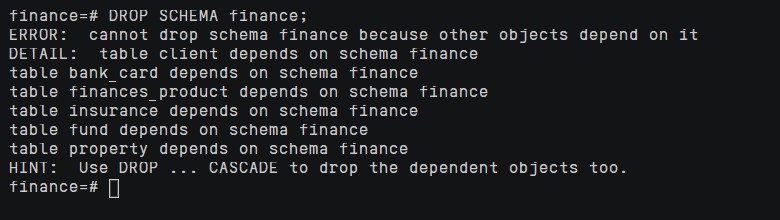
table finance.insurance depends on schema finance

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

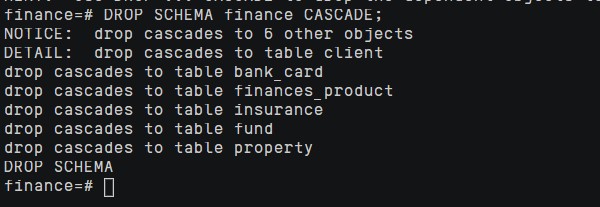
drop cascades to table insurance

drop cascades to table fund

drop cascades to table property

drop cascades to table finances\_product

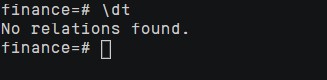
DROP SCHEMA



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.



实验中学习到的知识点分析：

数据库管理系统的基本操作：学习了如何连接数据库、登录用户、切换数据库，以及使用SQL命令界面进行数据库操作。

数据库的创建和删除：学习了使用CREATE DATABASE语句创建数据库，并使用DROP DATABASE语句删除数据库。

表的创建和修改：学习了使用CREATE TABLE语句创建表，定义表的结构和列的属性，以及使用ALTER TABLE语句修改表的结构。

数据的插入、查询、更新和删除：学习了使用INSERT INTO语句插入数据到表中，使用SELECT语句查询表中的数据，使用UPDATE语句更新表中的数据，以及使用DELETE语句删除表中的数据。

视图的创建、修改和删除：学习了使用CREATE VIEW语句创建视图，并可以在视图上执行查询操作，使用ALTER VIEW语句修改视图的名称，以及使用DROP VIEW语句删除视图。

索引的创建和删除：学习了使用CREATE INDEX语句创建索引，以加快对表的查询效率，使用DROP INDEX语句删除索引。

用户的创建和授权：学习了使用CREATE USER语句创建新用户，并使用GRANT语句授予用户访问数据库和表的权限。

数据库连接和访问：学习了使用不同用户身份和权限连接数据库，并在不同用户下访问和操作数据库对象。

Schema的管理：学习了使用DROP SCHEMA语句删除数据库中的Schema，包括其中的对象。

关系代数表达式：学习了关系代数的基本操作类型，如选择、投影、连接、并、差、交和除等，并了解了如何使用关系代数表达式描述数据操作过程。