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

1.1 金融数据模型

假设 A 市 C 银行为了方便对银行数据的管理和操作，引入了华为 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。

1.1.1 E-R 图

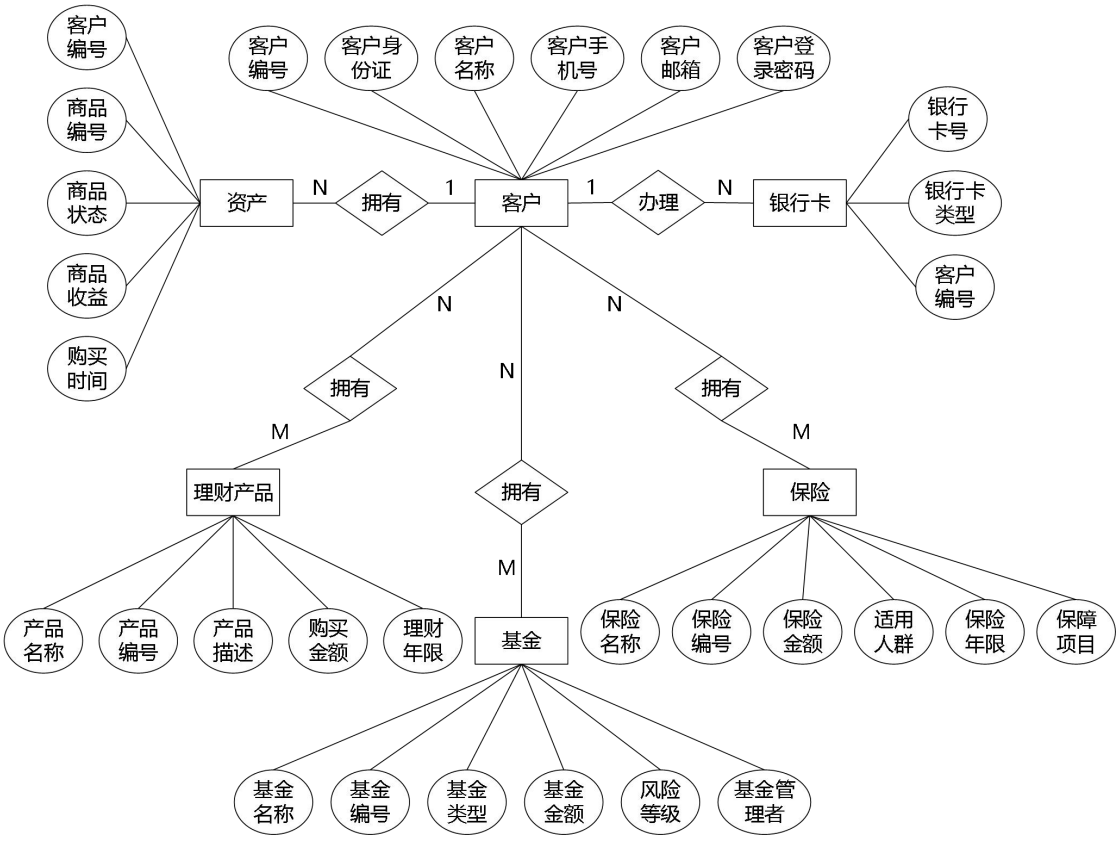


图 1-1 E-R 图

1.1.2 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

1.1.3 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：

步骤 1 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;
```

```
openGauss=# CREATE DATABASE finance ENCODING 'UTF8' template = template0;
CREATE DATABASE
openGauss=# \connect finance
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "finance" as user "omm".
finance=# CREATE SCHEMA finance;
CREATE SCHEMA
finance=# SET search_path TO finance;
SET
```

步骤 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。

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
);
```

```
finance=# DROP TABLE IF EXISTS client;
NOTICE:  table "client" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE client
finance-# (
finance(#         c_id INT PRIMARY KEY,
finance(#         c_name VARCHAR(100) NOT NULL,
finance(#         c_mail CHAR(30) UNIQUE,
finance(#         c_id_card CHAR(20) UNIQUE NOT NULL,
finance(#         c_phone CHAR(20) UNIQUE NOT NULL,
finance(#         c_password CHAR(20) NOT NULL
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "client_pkey" for table "client"
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "client_c_mail_key" for table "client"
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "client_c_id_card_key" for table "client"
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "client_c_phone_key" for table "client"
CREATE TABLE
finance=#
```

步骤 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。

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
);
```

```
finance=# DROP TABLE IF EXISTS bank_card;
NOTICE:  table "bank_card" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE bank_card
finance-# (
finance(#         b_number CHAR(30) PRIMARY KEY,
finance(#         b_type CHAR(20),
finance(#         b_c_id INT NOT NULL
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "bank_card_pkey" for table "bank_card"
CREATE TABLE
finance=#
```

步骤 4 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
);
```

```
finance=# DROP TABLE IF EXISTS finances_product;
NOTICE:  table "finances_product" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE finances_product
finance-# (
finance(#         p_name VARCHAR(100) NOT NULL,
finance(#         p_id INT PRIMARY KEY,
finance(#         p_description VARCHAR(4000),
finance(#         p_amount INT,
finance(#         p_year INT
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "finances_product_pkey" for table "finances_product"
CREATE TABLE
finance=#
```

步骤 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。

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)
);
```

```
finance=# DROP TABLE IF EXISTS insurance;
NOTICE:  table "insurance" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE insurance
finance-# (
finance(#         i_name VARCHAR(100) NOT NULL,
finance(#         i_id INT PRIMARY KEY,
finance(#         i_amount INT,
finance(#         i_person CHAR(20),
finance(#         i_year INT,
finance(#         i_project VARCHAR(200)
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "insurance_pkey" for table "insurance"
CREATE TABLE
finance=#
```

步骤 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。

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
);
```

```
finance=# DROP TABLE IF EXISTS fund;
NOTICE:  table "fund" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE fund
finance-# (
finance(#         f_name VARCHAR(100) NOT NULL,
finance(#         f_id INT PRIMARY KEY,
finance(#         f_type CHAR(20),
finance(#         f_amount INT,
finance(#         risk_level CHAR(20) NOT NULL,
finance(#         f_manager INT NOT NULL
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "fund_pkey" for table "fund"
CREATE TABLE
finance=#
```

步骤 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。

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
);
```

```
finance=# DROP TABLE IF EXISTS property;
NOTICE:  table "property" does not exist, skipping
DROP TABLE
finance=# CREATE TABLE property
finance-# (
finance(#         pro_c_id INT NOT NULL,
finance(#         pro_id INT PRIMARY KEY,
finance(#         pro_status CHAR(20),
finance(#         pro_quantity INT,
finance(#         pro_income INT,
finance(#         pro_purchase_time DATE
finance(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "property_pkey" for table "property"
CREATE TABLE
finance=#
```

1.1.4 插入表数据

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

步骤 1 对 client 表进行数据初始化。

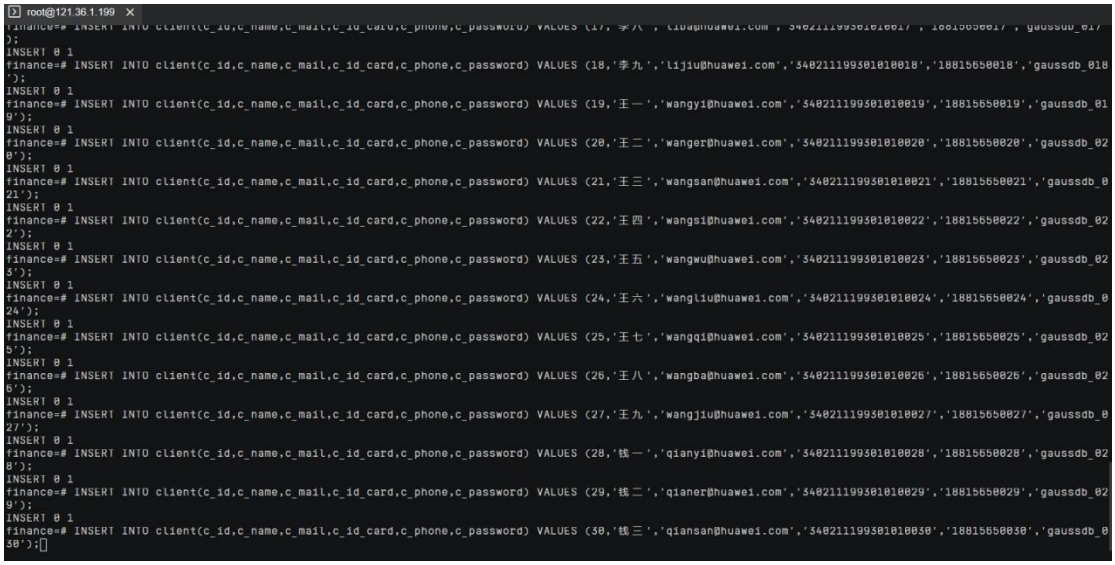
执行 insert 操作。

```
   INSERT INTO
client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES (1,'
张一
','zhangyi@huawei.com','340211199301010001','18815650001','gaussd
b_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','gaussd
b_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','gauss
db_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','gaussd
b_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','gaussd
b_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','gauss
db_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','gaussd
b_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','gaussd
b_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','gauss
db_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_0
10');
    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_0
11');
    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_0
13');
   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_0
14');
   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_0
16');
   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_0
17');
   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','gaussd
b_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','gaussd
b_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','gaussd
b_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','gaussd
b_030');
```

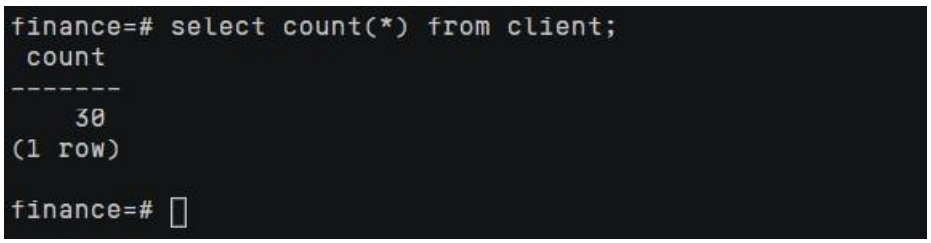


查询插入结果。

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

结果为：

```
count(*)
-------------
30
```

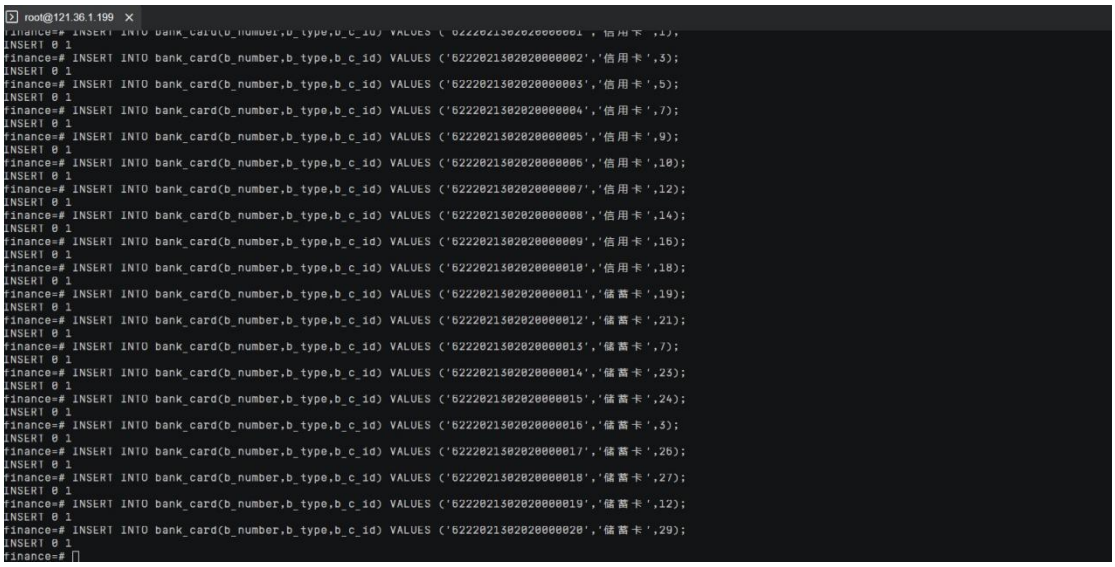


步骤 2 对 bank_card 表进行数据初始化。

执行 insert 操作。

```
   INSERT INTO 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
```

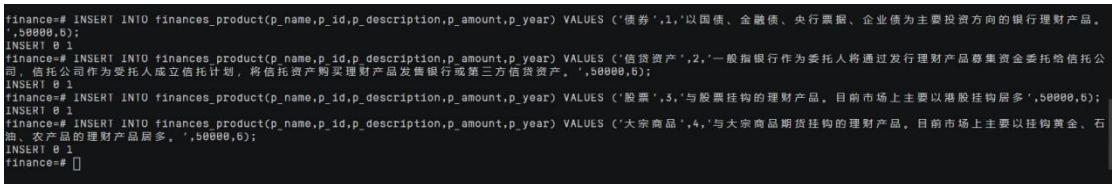
```
finance=# select count(*) from bank_card;
 count
-------
    20
(1 row)

finance=#
```

步骤 3 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
```

```
finance=# select count(*) from finances_product;
 count
-------
     4
(1 row)

finance=#
```

步骤 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');
```

```
finance=# 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 0 1
finance=# 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 0 1
finance=# 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 0 1
finance=# 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 0 1
finance=# 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');
INSERT 0 1
finance=#
```

查询插入结果。

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

结果为:

```
    count(*)
    -------------
    5
```

```
finance=# select count(*) from insurance;
 count
-------
     5
(1 row)
```

步骤 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);
```

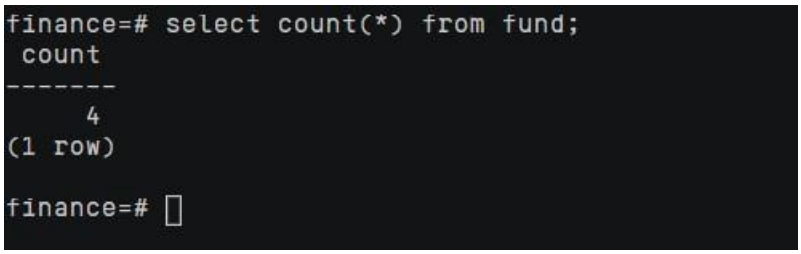
```
    INSERT INTO
fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('国
债',3,'货币型',10000,'低',3);
    INSERT INTO
fund(f_name,f_id,f_type,f_amount,risk_level,f_manager) VALUES ('沪
深 300 指数',4,'指数型',10000,'中',4);
```

查询插入结果。

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

结果为:

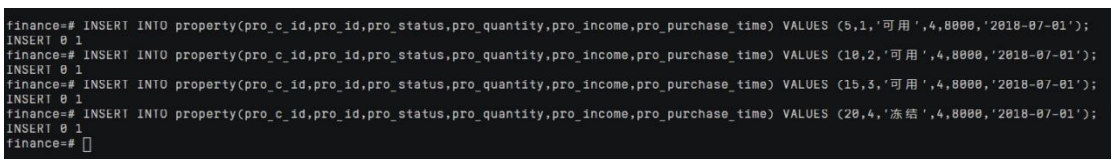
```
    count(*)
    -------------
    4
```



步骤 6 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_p
urchase_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_p
urchase_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_p
urchase_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_p
urchase_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
```

```
finance=# select count(*) from property;
 count
-------
     4
(1 row)

finance=#
```

1.1.5 手工插入一条数据

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

步骤 1 在金融数据库的客户信息表中添加一个客户的信息。（属性冲突的场景）

c_id_card 和 c_phone 非唯一。

```
    INSERT INTO
client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) VALUES
(31,'李丽
','lili@huawei.com','340211199301010005','18815650005','gaussdb_0
05');
```

错误信息如下：

```
    duplicate key value violates unique constraint
"client_c_id_card_key"
```

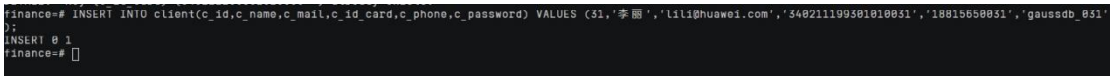
说明：由于在表的创建过程中，实验定义了 c_id_card 和 c_phone 为唯一且非空（ UNIQUE NOT NULL ），所以当表中存在时，插入数据失败。

```
finance=# 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'
);
ERROR:  duplicate key value violates unique constraint "client_c_id_card_key"
DETAIL:  Key (c_id_card)=(340211199301010005  ) already exists.
finance=#
```

步骤 2 在金融数据库的客户信息表中添加一个客户的信息。（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_0
31');
```

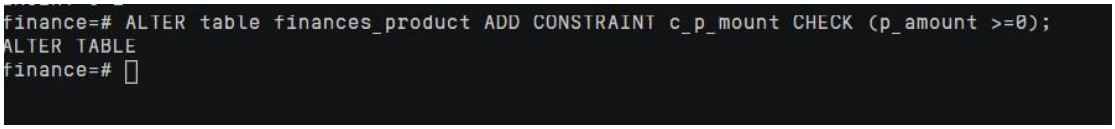


1.1.6 添加约束

步骤 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。

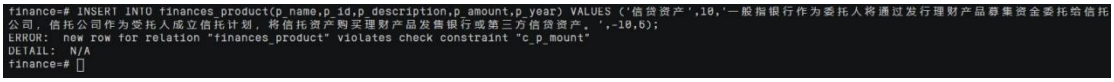
```
    ALTER table finances_product ADD CONSTRAINT c_p_mount CHECK
(p_amount >=0);
```



步骤 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"。



步骤 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);
```

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。

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

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

1.1.7 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。

步骤 1 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
```

```
finance=# SELECT b_number,b_type FROM bank_card;
            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
(20 rows)

finance=#
```

关系代数表达式： π (b_number, b_type)(bank_card)

步骤 2 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
```

```
finance=# select * from property where pro_status='可用';
 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
(3 rows)

finance=#
```

关系代数表达式： σ (pro_status = '可用')(property)

步骤 3 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
```

```
finance=# SELECT count(*) FROM client;
 count
-------
    31
(1 row)

finance=#
```

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
```

```
finance=# SELECT b_type,COUNT(*) FROM bank_card GROUP BY b_type;
     b_type      | count
-----------------+-------
 储蓄卡          |    10
 信用卡          |    10
(2 rows)

finance=#
```

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
```

```
finance=# SELECT AVG(i_amount) FROM insurance;
         avg
-----------------------
 2700.0000000000000000
(1 row)
```

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
```

```
finance=# select i_name,i_amount from insurance where i_amount in (select max(i_amount) from insurance)
finance-# union
finance-# select i_name,i_amount from insurance where i_amount in (select min(i_amount) from insurance);
    i_name    | i_amount
--------------+----------
 意外保险     |     5000
 财产损失保险 |     1500
(2 rows)

finance=#
```

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)))

步骤 4 连接查询。

（1） 半连接。

- 查询用户编号在银行卡表中出现的用户的编号，用户姓名和身份证。

```
    SELECT c_id,c_name,c_id_card FROM 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))

（2） 反连接。

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

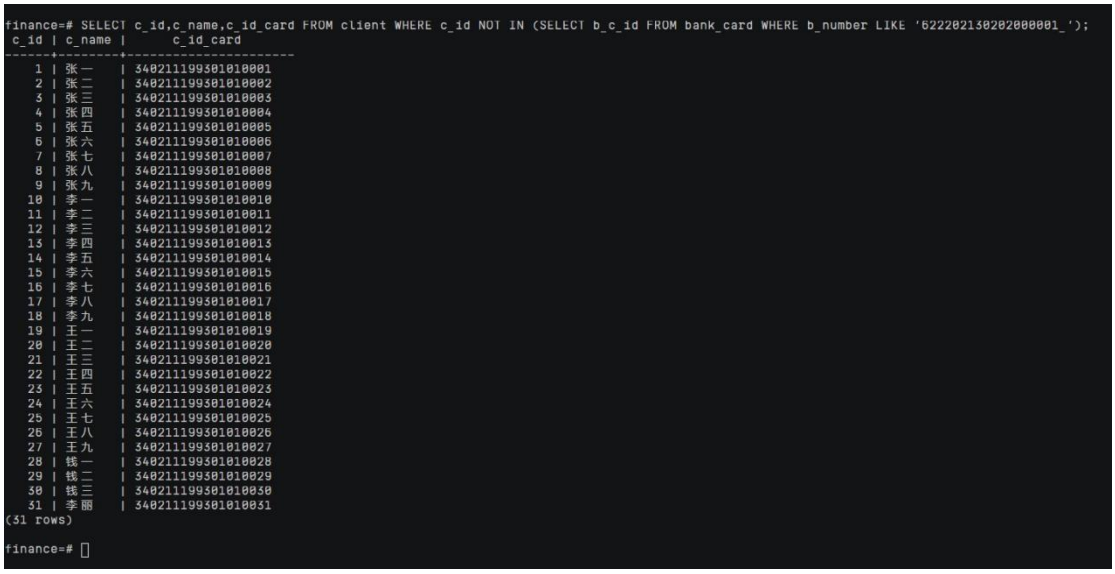
```
SELECT c_id,c_name,c_id_card FROM client WHERE c_id NOT IN (SELECT b_c_id FROM bank_card WHERE b_number LIKE '622202130202000001_');
```

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))

步骤 5 子查询

- 通过子查询，查询保险产品中保险金额大于平均值的保险名称和适用人群。

```
    SELECT i1.i_name,i1.i_amount,i1.i_person FROM 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
```

```
finance=# SELECT i1.i_name,i1.i_amount,i1.i_person FROM insurance i1 WHERE i_amount > (SELECT avg(i_amount) FROM insurance i2);
  i_name  | i_amount |      i_person
----------+----------+--------------------
 人寿保险 |     5000 | 老人
 意外保险 |     5000 | 所有人
(2 rows)

finance=#
```

关系代数表达式：π(i_name, i_amount, i_person)(σ (i_amount > AVG(i_amount))(insurance))

步骤 6 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
```

```
finance=# SELECT i_name,i_amount,i_person FROM insurance WHERE i_id>2 ORDER BY i_amount DESC;
    i_name    | i_amount |     i_person
--------------+----------+------------------
 意外保险     |     5000 | 所有人
 医疗保险     |     2000 | 所有人
 财产损失保险 |     1500 | 中年人
(3 rows)

finance=#
```

关系代数表达式：π(i_name, i_amount, i_person)(σ (i_id > 2)(insurance) ⋈_{i_amount DESC} insurance)

（2） 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
```

```
finance=# SELECT p_year,count(p_id) FROM finances_product GROUP BY p_year;
 p_year | count
--------+-------
      6 |     4
(1 row)

finance=#
```

关系代数表达式：γ(p_year, COUNT(p_id))(finances_product)

步骤 7 HAVING 和 WITH AS。

（1） HAVING 子句。

- 查询保险金额统计数量等于 2 的适用人群数。

```
SELECT i_person,count(i_amount) FROM 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。

```
finance=# SELECT i_person,count(i_amount) FROM insurance GROUP BY i_person HAVING count(i_amount)=2;
     i_person      | count
-------------------+-------
 老人              |     2
 所有人            |     2
(2 rows)

finance=#
```

关系代数表达式：γ(i_person, COUNT(i_amount))(σ (COUNT(i_amount) = 2)(insurance))

（2） WITH AS 子句。

- 使用 WITH AS 查询基金信息表。

```
WITH temp AS (SELECT f_name,ln(f_amount) FROM 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。

```
finance=# WITH temp AS (SELECT f_name,ln(f_amount) FROM fund ORDER BY f_manager DESC) SELECT * FROM temp;
   f_name    |        ln
-------------+------------------
 沪深300指数 | 9.21034037197618
 国债        | 9.21034037197618
 投资        | 9.21034037197618
 股票        | 9.21034037197618
(4 rows)

finance=#
```

关系代数表达式：temp ← π(f_name, LN(f_amount))(σ (f_manager DESC)(fund))
π(temp)

1.1.8 视图

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

步骤 1 创建视图。

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

```
CREATE VIEW 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
```

步骤 2 修改视图内容

修改视图，在原有查询的基础上，过滤出信用卡用户。

```
CREATE OR REPLACE VIEW 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
```

步骤 3 修改视图名称。

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

步骤 4 删除视图。

将 v_client 视图删除，删除视图不影响基表。

```
DROP VIEW v_client_new;
```

```
finance=# ALTER VIEW v_client RENAME TO v_client_new;
ALTER VIEW
finance=# DROP VIEW v_client_new;
DROP VIEW
finance=#
```

1.1.9 索引

步骤 1 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
```

步骤 2 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;
```

```
finance=# ALTER VIEW v_client RENAME TO v_client_new;
ALTER VIEW
finance=# DROP VIEW v_client_new;
DROP VIEW
finance=# CREATE INDEX idx_property ON property(pro_c_id DESC,pro_income,pro_purchase_time);
CREATE INDEX
finance=# DROP INDEX idx_property;
DROP INDEX
finance=# CREATE INDEX idx_property ON property(pro_c_id DESC,pro_income,pro_purchase_time);
CREATE INDEX
finance=# ALTER INDEX idx_property RENAME TO idx_property_temp;
ALTER INDEX
finance=# DROP INDEX idx_property_temp;
DROP INDEX
finance=#
```

1.1.10 数据的修改和删除

步骤 1 修改数据。

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

查看表数据。

```
SELECT * FROM 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
```

```
finance=# SELECT * FROM bank_card where b_c_id<10 ORDER BY b_c_id;
          b_number          |     b_type      | b_c_id
----------------------------+-----------------+--------
 6222021302020000001        | 信用卡          |      1
 6222021302020000016        | 储蓄卡          |      3
 6222021302020000002        | 信用卡          |      3
 6222021302020000003        | 信用卡          |      5
 6222021302020000004        | 信用卡          |      7
 6222021302020000013        | 储蓄卡          |      7
 6222021302020000005        | 信用卡          |      9
(7 rows)

finance=#
```

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
```

```
finance=# 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;
UPDATE 7
finance=# SELECT * FROM bank_card ORDER BY b_c_id;
          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
(20 rows)

finance=#
```

步骤 2 删除指定数据。

- 删除基金信息表中编号小于 3 的行。

  删除前查询结果。

```
SELECT * FROM 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
```

```
finance=# SELECT * FROM fund;
   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
(4 rows)

finance=#
```

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
```

```
finance=# DELETE FROM fund WHERE f_id<3;
DELETE 2
finance=# SELECT * FROM fund;
   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
(2 rows)

finance=#
```

1.1.11 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：

步骤 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';
```

步骤 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;
```

```
finance=# CREATE USER dbuser IDENTIFIED BY 'Gauss#3demo';
CREATE ROLE
finance=# GRANT SELECT,INSERT ON finance.bank_card TO dbuser;
GRANT
finance=# GRANT ALL ON SCHEMA finance to dbuser;
GRANT
finance=#
```

1.1.12 新用户连接数据库

步骤 1 在 gsql 登录数据库，使用新用户连接。

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;
```

步骤 2 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
```

```
finance=# select * from finance. bank_card where b_c_id<10;
          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
(7 rows)

finance=#
```

1.1.13 删除 Schema

步骤 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
```

步骤 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
```

```
finance=# gsql -d finance -p 26000
finance-# \dn
     List of schemas
      Name       | Owner
-----------------+--------
 blockchain      | omm
 cstore          | omm
 db4ai           | omm
 dbe_perf        | omm
 dbe_pldebugger  | omm
 dbe_pldeveloper | omm
 dbe_sql_util    | omm
 dbuser          | dbuser
 finance         | omm
 pkg_service     | omm
 public          | omm
 snapshot        | omm
 sqladvisor      | omm
(13 rows)

finance-#
```

步骤 3 设置默认查询为 finance。

```
set search_path to finance;
```

步骤 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}
```

```
finance=# \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}
(6 rows)

finance=#
```

步骤 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.
```

```
finance=# DROP SCHEMA finance;
ERROR:  cannot drop schema finance because other objects depend on it
DETAIL:  table client depends on schema finance
table bank_card depends on schema finance
table finances_product depends on schema finance
table insurance depends on schema finance
table fund depends on schema finance
table property depends on schema finance
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
finance=#
```

步骤 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
```

```
finance=# DROP SCHEMA finance CASCADE;
NOTICE:  drop cascades to 6 other objects
DETAIL:  drop cascades to table client
drop cascades to table bank_card
drop cascades to table finances_product
drop cascades to table insurance
drop cascades to table fund
drop cascades to table property
DROP SCHEMA
finance=#
```

步骤 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.
```

```
finance=# \dt
No relations found.
finance=#
```

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。