基于 openGauss 的场景化综合实验

姓名：汤清云　　学号：2013536

实验步骤：

- 独立完成关于数据库的实验操作
- SQL 查询语句自己编写：

实验报告

实验环境：openGauss 避免在给各个表添加命名时混淆。

实验步骤：

1. 前期准备工作

1.登录 omm 用户：

```
su – omm
```

2.启动数据库服务：

```
gs_om -t start;
```

```
[omm@ecs-6002 ~]$ gs_om -t start;
Starting cluster.
=========================================
[SUCCESS] ecs-6002
2022-05-07 16:32:06.506 62762e86.1 [unknown] 281463157751824 [unknown] 0 dn_600
1 01000  0 [BACKEND] WARNING:  could not create any HA TCP/IP sockets
2022-05-07 16:32:06.510 62762e86.1 [unknown] 281463157751824 [unknown] 0 dn_600
1 01000  0 [BACKEND] WARNING:  Failed to initialize the memory protect for g_in
stance.attr.attr_storage.cstore_buffers (16 Mbytes) or shared memory (2363 Mbyt
es) is larger.
```

3.连接 postgres 数据库：

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

4.创建 finance 数据库，使用 UTF8 编码方式：

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

```
postgres=# CREATE DATABASE finance ENCODING 'UTF8' template = template0;
ERROR:  database "finance" already exists
```

5.连接 finance 数据库

```
\connect finance
```

```
postgres=# \connect finance
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "finance" as user "omm".
```

6.创建模式 finance 并连接：

```
CREATE SCHEMA finance;
SET search_path TO finance;
```

```
finance=# CREATE SCHEMA finance;
CREATE SCHEMA
finance=# SET search_path TO finance;
SET
```

2. 创建各个表

此处创建了 7 个表，语句依次如下：(由于我自己写的创建语句与答案只有大小写以及字符长度存在差异，故此处直接使用手册中所给的语句)

```
DROP TABLE IF EXISTS 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
);
DROP TABLE IF EXISTS bank_card;
CREATE TABLE bank_card
(
        b_number CHAR(30) PRIMARY KEY,
        b_type CHAR(20),
        b_c_id INT NOT NULL
);
DROP TABLE IF EXISTS 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
);
DROP TABLE IF EXISTS 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)
);
DROP TABLE IF EXISTS 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
);
DROP TABLE IF EXISTS 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
```

3. 对各个表插入所需数据，实现表的初始化。（语句略）

关系代数如：表名：=表名∪（属性）

例：client:=client∪(1,'张一

','zhangyi@huawei.com','340211199301010001','18815650001','gaussdb_0

01')

```
finance=# select count(*) from client;
 count
-------
    30
(1 row)
```

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

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

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

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

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

4. 手工插入

1. 当插入数据属性冲突时（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:=finance∪(31,’李丽’,’lili@huawei.com’,’32021199301010005)

```
finance=# INSERT INTO client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) V
ALUES (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.
```

2. 插入数据满足要求时手工插入自然成功：

```
finance=# INSERT INTO client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) V
ALUES (31,'李 丽 ','lili@huawei.com','340211199301010031','18815650031','gaussdb_
031');
INSERT 0 1
```

5. 添加约束

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_amount CHECK (p_amount>=0);
```

添加后尝试插入一条 p_amount<0 的记录则报错：

```
finance=# ALTER table finances_product ADD CONSTRAINT c_p_mount CHECK (p_amount
 >=0);
ALTER TABLE
finance=# INSERT INTO finances_product(p_name,p_id,p_description,p_amount,p_yea
r) 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◆◆购买理财产品发售银行或第三
方信贷资产。',-10,6);
ERROR:  new row for relation "finances_product" violates check constraint "c_p_
mount"
DETAIL:  Failing row contains (信贷资产, 10, 一般指银行作为委托人将通过发行理财
产品募集..., -10, 6).
```

同理，为 fund 表的 f_amount 列添加大于等于 0 的约束；为 insurance 表的 i_amount 列添加大于等于 0 的约束。

```
ALTER TABLE fund ADD CONSTRAINT c_f_amount CHECK (f_amount>=0);
ALTER TABLE insurance ADD CONSTRAINT c_i_amount CHECK (i_amount>=0);
```

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

6. 查询数据（以下为自己所编写 SQL 语句）

1. 查询银行卡信息表。

```
Select * from bank_card;
```

$$R_2 := \pi_{(b\_number, b\_type, b\_c\_id)}(bank\_card)$$

```
finance=# Select * from bank_card;
          b_number          |      b_type      | b_c_id
----------------------------+------------------+--------
 6222021302020000001        | 信用卡           |      1
 6222021302020000002        | 信用卡           |      3
 6222021302020000003        | 信用卡           |      5
 6222021302020000004        | 信用卡           |      7
 6222021302020000005        | 信用卡           |      9
 6222021302020000006        | 信用卡           |     10
 6222021302020000007        | 信用卡           |     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
 6222021302020000013        | 储蓄卡           |      7
 6222021302020000014        | 储蓄卡           |     23
 6222021302020000015        | 储蓄卡           |     24
 6222021302020000016        | 储蓄卡           |      3
 6222021302020000017        | 储蓄卡           |     26
 6222021302020000018        | 储蓄卡           |     27
 6222021302020000019        | 储蓄卡           |     12
 6222021302020000020        | 储蓄卡           |     29
(20 rows)
```

2. 查询资产信息中‘可用’的资产数据。

```
Select * from property where pro_status='可用';
```

$$\pi_{pro\_c\_id, pro\_id, pro\_status, pro\_quantity, pro\_income, pro\_purchase\_time}(\sigma_{pro\_status='可用'}(property))$$

```
finance=# select * from property where pro_status='可用';
 pro_c_id | pro_id |      pro_status      | pro_quantity | pro_income |   pro_purc
hase_time
----------+--------+----------------------+--------------+------------+----------
-----------
        5 |      1 | 可用                 |            4 |       8000 | 2018-07-0
1 00:00:00
       10 |      2 | 可用                 |            4 |       8000 | 2018-07-0
1 00:00:00
       15 |      3 | 可用                 |            4 |       8000 | 2018-07-0
1 00:00:00
(3 rows)
```

3. 查询用户表中有多少个用户：

```
Select count(*) from Client;
```

$$COUNT(c\_id)$$

```
finance=# Select count(*) from Client;
 count
-------
    31
(1 row)
```

4. 查询银行卡信息表中储蓄卡和信用卡的个数

```
Select count(*),b_type from bank_card group by b_type;
```

$$COUNT(\gamma_{b\_type}(bank\_card))$$

```
finance=# Select count(*),b_type from bank_card group by b_type;
 count |       b_type
-------+--------------------
    10 | 储蓄卡
    10 | 信用卡
(2 rows)
```

5. 查询保险信息表中保险金额的平均值：

```
Select AVG(i_amount) from insurance;
```

$$AVG(i\_amount)$$

```
finance=# Select AVG(i_amount) from insurance;
          avg
------------------------
 2700.0000000000000000
(1 row)
```

6. 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>=ALL( select i_amount from insurance);
Select i_name,i_amount from insurance where i_amount<=ALL( select i_amount from insurance);
```

$$\pi_{i\_name,i\_amount}\left(\sigma_{i\_amount=MAX(i\_amount)}(insurance)\right)$$

$$\pi_{i\_name,i\_amount}(\sigma_{i\_amount=MIN(i\_amount)}(insurance))$$

```
finance=# Select i_name,i_amount from insurance where i_amount>=ALL( select i_amount from insurance);
  i_name   | i_amount
-----------+----------
 意外保险  |      5000
(1 row)

finance=# Select i_name,i_amount from insurance where i_amount<=ALL( select i_amount from insurance);
     i_name     | i_amount
----------------+----------
 财产损失保险  |      1500
(1 row)
```

7. 查询用户编号在银行卡表中出现的用户编号，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);
```

$$\pi_{c\_id,c\_name,c\_id\_card}\,(client \bowtie bank\_card)$$

```
finance=# 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);
 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
(17 rows)
```

8. 查询银行卡号不是‘6222021302020000001*’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 LIKE
'622202130202000001_';
```

此条查询语句无法使用关系代数查询。

```
finance=# Select c_id,c_name,c_id_card from client where c_id NOT LIKE '6222021
302020000001_';
 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
```

```
    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
(31 rows)
```

9. 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 i_name,i_person from insurance where i_amount > (select AVG(i_amount)
from insurance );
```

$$\pi_{i\_name,i\_person}(\sigma_{i\_amount>AVG(i\_amount)}(insurance))$$

```
finance=# Select i_name,i_person from insurance where i_amount > (select AVG(i_
amount) from insurance );
   i_name   |       i_person
------------+----------------------
 人寿保险   | 老人
 意外保险   | 所有人
(2 rows)
```

10. 按照降序查找保险编号大于 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;
```

无法使用关系代数实现“降序”操作，仅查询操作实现如下：

$$\pi_{i\_name,i\_amount,i\_person}(\sigma_{i\_id>2}(insurance))$$

```
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)
```

11. 查询各保险信息总数，按 pyear 分组：

```
Select p_year,count(p_id) from finances_product group by p_year;
```

$$R1 = \rho_{p\_year,count}(\pi_{p\_year,COUNT(p\_id)}(\gamma_{p\_year}(finance\_product)))$$

```
finance=# Select p_year,count(p_id) from finances_product group by p_year;
 p_year | count
--------+-------
      6 |     4
(1 row)
```

12. 查询保险金额统计数量=2 的适用人群数：

```
Select i_person,count(i_amount) from insurance group by i_person having count(i_amount)=2;
```

$$\pi_{i\_person,COUNT(i\_person)}(\sigma_{COUNT(i\_amount)=2}(insurance))$$

```
finance=# Select i_person,count(i_amount) from insurance group by i_person havi
ng count(i_amount)=2;
      i_person      | count
--------------------+-------
 老人               |     2
 所有人             |     2
(2 rows)
```

13. 使用 with as 查询基金信息表：

```
WITH temp AS (SELECT * FROM fund)
SELECT * FROM temp;
```

$temp \coloneqq fund$

```
finance=# with temp as (select * from fund) select * from temp;
   f_name     | f_id |       f_type        | f_amount |      risk_level       | f_ma
nager
--------------+------+---------------------+----------+-----------------------+-----
------
 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)
```

7. 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);
```

```
finance=# CREATE VIEW v_client as SELECT c_id,c_name,c_id_card FROM client WHER
E EXISTS (SELECT * FROM bank_card WHERE client.c_id = bank_card.b_c_id);
CREATE VIEW
```

```
finance=# SELECT * FROM v_client;
 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
(17 rows)
```

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');
```

```
finance=# 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');
CREATE VIEW
```

```
finance=# select * from v_client;
 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
(10 rows)
```

修改视图名称：

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

删除视图：

```
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
```

8. 创建索引：

普通表 property 上创建索引。

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

```
finance=# CREATE INDEX idx_property ON property(pro_c_id DESC,pro_income,pro_pu
rchase_time);
CREATE INDEX
```

重命名索引：

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

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

```
finance=# DROP INDEX idx_property;
DROP INDEX
finance=# CREATE INDEX idx_property ON property(pro_c_id DESC,pro_income,pro_pu
rchase_time);
CREATE INDEX
finance=# ALTER INDEX idx_property RENAME TO idx_property_temp;
ALTER INDEX
```

删除索引：

```
DROP INDEX idx_property_temp;
```

```
finance=# DROP INDEX idx_property_temp;
DROP INDEX
```

9. 数据的修改和删除

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。

```
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;
```

$bank_{card}$:

$= bank\_card - (\sigma_{bank\_card.b\_(c\_id)=client.c\_id \cap bank\_card.b\_(c\_id)<10}(bank\_card))$

$\cup (\pi_{b\_number}(\sigma_{bank\_card.b\_(c\_id)=client.c\_id \cap bank\_card.b\_(c\_id)<10}(bank\_card), '借记卡'), \pi_{b\_c\_id}(\sigma_{bank\_card.b\_(c\_id)=client.c\_id \cap bank\_card.b\_(c\_id)<10}(bank\_card)))$

修改前：

```
finance=# SELECT * FROM bank_card 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
 6222021302020000013        | 储蓄卡               |      7
 6222021302020000004        | 信用卡               |      7
 6222021302020000005        | 信用卡               |      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)
```

修改后：

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

```
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
```

删除基金信息表中编号小于 3 的行：

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

$$fund := fund - (\sigma_{f\_id<3}(fund))$$

删除前：

```
SELECT * FROM fund;
```

```
finance=# SELECT * FROM fund;
   f_name    | f_id |      f_type       | f_amount |     risk_level      | f_ma
nager
-------------+------+-------------------+----------+---------------------+-----
------
 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)
```

删除后：

```
SELECT * FROM fund;
```

```
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_ma
nager
-------------+------+-------------------+----------+---------------------+-----
------
 国债        |    3 | 货币型            |    10000 | 低                  |
    3
 沪深300指数 |    4 | 指数型            |    10000 | 中                  |
    4
(2 rows)
```

10.选做部分：

1. 新用户创建授权：

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';
```

```
finance=# CREATE USER dbuser IDENTIFIED BY 'Gauss#3demo';
CREATE ROLE
```

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=# GRANT SELECT,INSERT ON finance.bank_card TO dbuser;
GRANT
finance=# GRANT ALL ON SCHEMA finance to dbuser;
GRANT
```

2. 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;
```

```
[omm@ecs-6002 ~]$ gsql -d finance -U dbuser -p 26000;
Password for user dbuser:
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0
 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
```

访问 finance 数据库的表 bank_card。

```
select * from finance. bank_card where b_c_id<10;
```

```
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)
```

3. 删除 schema 操作：

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
```

```
finance=# \dn
   List of schemas
    Name     | Owner
-------------+--------
 cstore      | omm
 dbe_perf    | omm
 dbuser      | dbuser
 finance     | omm
 pkg_service | omm
 public      | omm
 snapshot    | omm
(7 rows)
```

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
```

```
finance=# set search_path to finance;
SET
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)
```

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

```
DROP SCHEMA finance;
```

```
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=#
```

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

```
DROP SCHEMA finance CASCADE;
```

```
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=#
```

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

```
\dt
```

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

11.相关说明：

1. 实验环境说明：

   选择实验环境为 openGauss，避免之后因语句插入时不同导致信息插入失败，后续对应实验手册出现不匹配情况。

2. 执行结果截图附在每条语句后。

3. 能够使用关系代数表达的 SQL 查询语句均已标注在撰写的 SQL 查询语句后。

4. 初始 SQL 查询语句与要求查询结果不同的包括以下语句：

   老师所给语句为：

```
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);
```

   我所用语句为：

```
Select i_name,i_amount from insurance where i_amount>=ALL( select i_amount from insurance);
Select i_name,i_amount from insurance where i_amount<=ALL( select i_amount from insurance);
```

使用老师给的SQL语句的话，能够将最大最小值一并输出，因为是使用了union（并集），而使用我的语句必须将结果拆分，分开输出最大值与最小值。

结果如下：

```
i_name     | i_amount
---------------+----------
 财产损失保险 |     1500
 意外保险     |     5000
```

5. 实验总时长为：前后多次进行实验并截图，共约 4h

6. 遇到的问题有：不了解 with as 语句，后上网查询学习到了 with as 的用法：相当于子查询，给某个表建立别名；对华为云的 openGauss 的操作并不理解导致实验进度缓慢，截图不精确导致需要多次进行操作截图。

7. 实验中学习到的知识点：关系代数中τ只能够输出升序排序语句，而无法实现降序输出的操作；学习到了 with as 语句。