openGauss 场景化综合实验(金融场景)实验报告

姓名：王娇妹 学号：2012679

实验步骤：

- 前置环境设置
- 创建数据表
- 插入表数据
- 手工插入一条数据
- 添加约束
- 查询数据
- 视图
- 索引
- 数据的修改和删除
- 新用户的创建和授权
- 新用户连接数据库
- 删除 Schema

实验报告

一、实验环境说明

此次实验我使用 openGauss 实验环境。选择 openGauss 的原因：

1、数据库系统这门课的前几次实验大多都是采用 openGauss 进行，我对 openGauss 更

加熟悉，而且实验手册是以 openGauss 为例进行说明的。

2、在实验前浏览实验指导手册时，手册里提到“基于 GaussDB 的操作可能没有权限进行 1.1.11 的实验步骤”，我想尝试一下 1.1.11~1.1.13 的实验，所以选择 openGauss。

二、1.1.3-1.1.13 执行结果截图

1.1.3 创建数据表

步骤 1 创建金融数据库 finance。

使用 gsql 工具登陆数据库。

```
[omm@ecs-ee4a ~]$ gs_om -t start;
Starting cluster.
=========================================
[SUCCESS] ecs-ee4a
2022-05-10 21:35:59.232 627a6a3f.1 [unknown] 281463176101904 [unknown] 0 dn_6001 01000  0 [BACKEND] WARNING:  could not create any HA TCP/IP sockets
2022-05-10 21:35:59.235 627a6a3f.1 [unknown] 281463176101904 [unknown] 0 dn_6001 01000  0 [BACKEND] WARNING:  Failed to initialize the memory protect for g
stance.attr.attr_storage.cstore_buffers (16 Mbytes) or shared memory (2363 Mbytes) is larger.
=========================================
Successfully started.
[omm@ecs-ee4a ~]$ gsql -d postgres -p 26000
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.
```

创建数据库 financeL 并连接，创建名为 finance 的 schema，并设置 finance 为当前的 schema，将默认搜索路径设为 finance。

```
postgres=# CREATE DATABASE finance ENCODING 'UTF8' template = template0;
CREATE DATABASE
postgres=# \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。

创建客户信息表 client。

```
finance=# DROP TABLE IF EXISTS client;
NOTICE:  table "client" does not exist, skipping
DROP TABLE
finance=# 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-# finance(# finance(# finance(# finance(# finance(# finance(# 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
```

步骤 3 银行卡信息表的创建。

创建银行卡信息表 bank_card。

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

步骤 4 理财产品信息表的创建。

创建理财产品信息表 finances_product。

```
finance=# DROP TABLE IF EXISTS finances_product;
NOTICE:  table "finances_product" does not exist, skipping
DROP TABLE
finance=# 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-# finance(# finance(# finance(# finance(# finance(# finance(#
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "finances_product_pkey" for table "finances_product"
CREATE TABLE
```

步骤 5 保险信息表的创建。

创建保险信息表 insurance。

```
finance=# DROP TABLE IF EXISTS insurance;
NOTICE:  table "insurance" does not exist, skipping
DROP TABLE
finance=# 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-# finance(# finance(# finance(# finance(# finance(# finance(#
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "insurance_pkey" for table "insurance"
CREATE TABLE
```

步骤 6 基金信息表的创建。

创建保险信息表 fund。

```
finance=# DROP TABLE IF EXISTS fund;
NOTICE:  table "fund" does not exist, skipping
DROP TABLE
finance=# 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-# finance(# finance(# finance(# finance(# finance(# finance(# finance(#
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "fund_pkey" for table "fund"
CREATE TABLE
```

步骤 7　资产信息表的创建。

创建资产信息表 property。

```
finance=# DROP TABLE IF EXISTS property;
NOTICE:  table "property" does not exist, skipping
DROP TABLE
finance=# 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-# finance(# finance(# finance(# finance(# finance(# finance(# finance(#
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "property_pkey" for table "property"
CREATE TABLE
```

1.1.4 插入表数据

步骤 1　对 client 表进行数据初始化。

执行 insert 操作，查询插入结果。

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

步骤 2　对 bank_card 表进行数据初始化。

执行 insert 操作，查询插入结果。

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

步骤 3　对 finances_product 表进行数据初始化。

执行 insert 操作，查询插入结果。

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

步骤 4　对 insurance 表进行数据初始化。

执行 insert 操作，查询插入结果。

```
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，查询插入结果。

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

步骤 6　对 property 表进行数据初始化。

执行 insert 操作，查询插入结果。

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

1.1.5 手工插入一条数据

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)

```
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.
```

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)。

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

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。

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

步骤 2　尝试手工插入一条金额小于 0 的记录。

```
finance=# 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◆◆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).
finance=#
```

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。

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

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。

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

1.1.7 查询数据

步骤 1　单表查询。

查询银行卡信息表。

```
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)
```

步骤 2　条件查询。

查询资产信息中‘可用’的资产数据。

```
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)
```

步骤 3　聚合查询。

查询用户表中有多少个用户。

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

查询银行卡信息表中，储蓄卡和信用卡的个数。

```
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)
```

查询保险信息表中，保险金额的平均值。

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

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。

```
finance=# 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);finance-# finance-#
    i_name    | i_amount
--------------+----------
 意外保险     |     5000
 财产损失保险 |     1500
(2 rows)
```

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。

```
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)
```

（2）反连接。

查询银行卡号不是‘6222021302020000001* ‘的用户的编号，姓名和身份证。

```
finance=# 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 '6222021302020000001_');
 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
(31 rows)
```

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。

```
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 |     3000 | 老人
 意外保险 |     5000 | 所有人
(2 rows)
```

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。

```
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)
```

（2）GROUP BY 子句。

查询各保险信息总数，按照 p_year 分组。

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

步骤 7　HAVING 和 WITH AS。

（1）HAVING 子句。

```
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)
```

（2）WITH AS 子句。

使用 WITH AS 查询基金信息表。

```
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)
```

1.1.8 视图

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。使用视图进行查询。

```
finance=# 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);
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)
```

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。使用视图进行查询。

```
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)
```

步骤 3　修改视图名称。

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

步骤 4　删除视图。

将 v_client 视图删除，删除视图不影响基表。

```
finance=# DROP VIEW v_client_new;
DROP VIEW
```

1.1.9 索引

步骤 1　创建索引。

在普通表 property 上创建索引。

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

步骤 2　重命名索引。

在普通表 property 上重建及重命名索引。

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

步骤 3　删除索引。

删除索引 idx_property_temp。

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

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。

```
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)
```

更新数据，重新查询数据情况。

```
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)
```

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。

```
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)
```

开始删除数据

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

查询删除结果。

```
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)
```

1.1.11 新用户的创建和授权

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。

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

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。

```
postgres=# \connect finance
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "finance" as user "omm".
finance=# GRANT SELECT,INSERT ON finance.bank_card TO dbuser;
GRANT
finance=# GRANT ALL ON SCHEMA finance to dbuser;
GRANT
```

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。

```
[omm@ecs-ee4a ~]$ 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.
```

步骤 2 访问 finance 数据库的表 bank_card。

```
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)
```

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。

```
[omm@ecs-ee4a ~]$ gsql -d finance -p 26000
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.
```

步骤 2　使用“\dn”查看数据库下的 schema。

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

步骤 3　设置默认查询为 finance。

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

步骤 4　使用“\dt”命令可以看到在 finance 中的对象。

```
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)
```

步骤 5　使用 DROP SCHEMA 命令删除 finance 会有报错，因为 finance 下存在对象。

```
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.
```

步骤 6　使用 DROP SCHEMA.....CASCADE 删除，会将 finance 连同下的对象一起删除。

```
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
```

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。

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

三、对于能够用关系代数表达的 SQL 执行需求，请写出对应小节及步骤的关系代数查询语句（标明小节和步骤以表示对应关系）。

1.1.7 查询数据

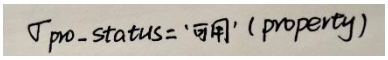
步骤 1　单表查询：查询银行卡信息表。

```
SELECT b_number,b_type FROM bank_card;
```

关系代数查询语句：$\Pi_{b\_number,b\_type}(bank\_card)$

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';
```

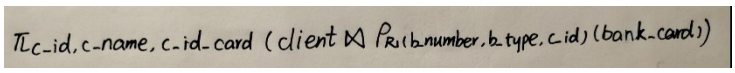
关系代数查询语句： 

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);
```

关系代数查询语句：



四、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。

在此次实验中，我有两处 SQL 执行结果和要求的执行结果不符。

1、在执行 1.1.11 的步骤 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”操作时，报错 schema“finance”不存在。如下图所示：

```
postgres=# GRANT SELECT,INSERT ON finance.bank_card TO dbuser;
ERROR:  schema "finance" does not exist
postgres=# GRANT ALL ON SCHEMA finance to dbuser;
ERROR:  schema "finance" does not exist
```

经过查询资料，我发现是因为在 1.1.11 中，我重新登陆了数据库，但是没有连接 finance 数据库，所以会提示 schema“finance” does not exist。

输入“\connect finance”命令后，操作可以正常进行。

```
postgres=# \connect finance
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "finance" as user "omm".
finance=# GRANT SELECT,INSERT ON finance.bank_card TO dbuser;
GRANT
finance=# GRANT ALL ON SCHEMA finance to dbuser;
GRANT
```

2、在执行 1.1.13 的步骤 4“使用‘\dt’命令可以看到在 finance 中的对象”操作时，报错 No relation found。如下图：

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

我发现这是漏掉了步骤 3“设置默认查询为 finance”操作导致的。因为没有设置默认搜索路径，所以平台无法判断\dt 命令查看谁的对象，产生错误。

输入“set search_path to finance;”命令后，用“\dt”可以查询到 finance 中的对象。

```
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)
```

五、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。

本次实验总用时 58 分钟。

通过这次实验，我发现自己对于 SQL 语句并不是很熟悉，一些 SQL 查询语句我需要思考一段时间才能写出来；甚至有些语句存在错误，要参考手册内容才能完成实验。在这次实验中，我复习了数据表的创建、插入数据、修改和删除数据、添加约束、查询数据等操作的基本 SQL 语句。

在 openGuass 中创建数据表不仅仅用一句“CREATE TABLE client（）；”，还要在前面添加“DROP TABLE IF EXISTS client;”，防止万一已经存在 client 数据表，影响到接下来的操作。插入数据使用“INSERT INTO ”语句，查询数据使用“SELECT FROM WHERE；”语句，修改数据使用“UPDATE … SET …”语句。添加约束使用“ALTER table 表名 ADD CONSTRAINT 列 CHECK (约束条件);”语句。视图与索引的基本操作类似，创建视图/索引使用“CREATE VIEW/ INDEX”语句，修改视图/索引名称使用“ALTER VIEW/ INDEX 旧名 RENAME TO 新名”语句，删除视图/ INDEX 使用“DROP VIEW/ INDEX 名称;”语句。