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：

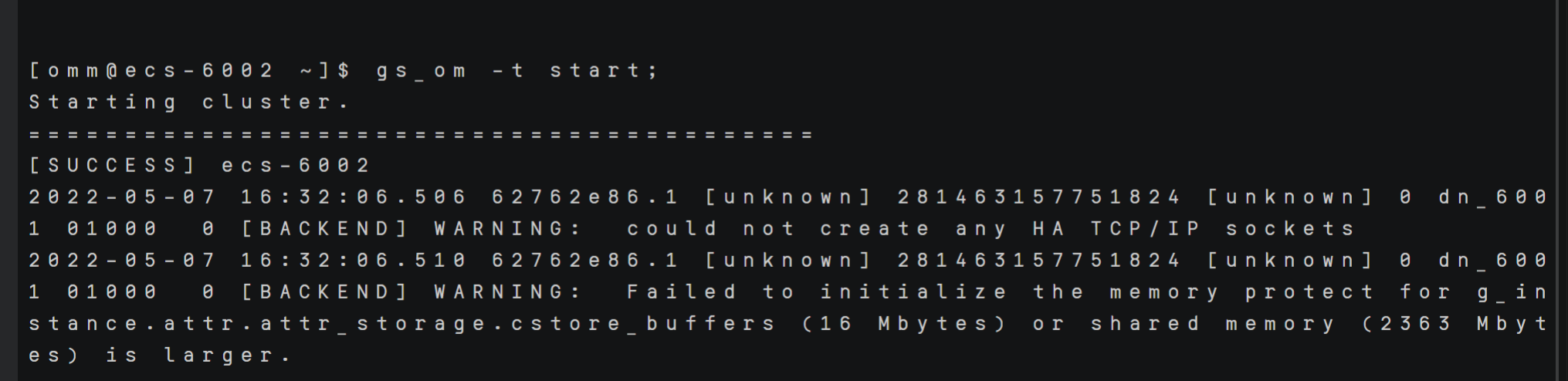
1. 前期准备工作

1.登录omm用户：

**su – omm**

2.启动数据库服务：

**gs\_om -t start;**

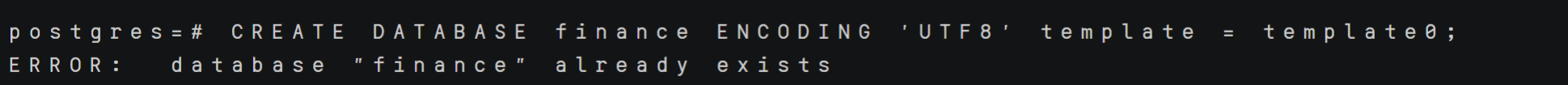


3.连接postgres数据库：

**gsql -d postgres -p 26000**

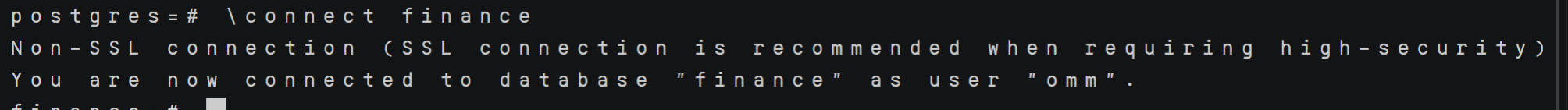
4.创建finance数据库，使用UTF8编码方式：

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



5.连接finance数据库

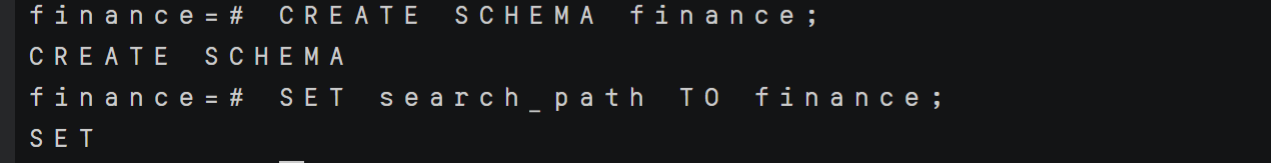
**\connect finance**



6.创建模式finance并连接：

**CREATE SCHEMA finance;**

**SET search\_path TO finance;**



1. 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,**

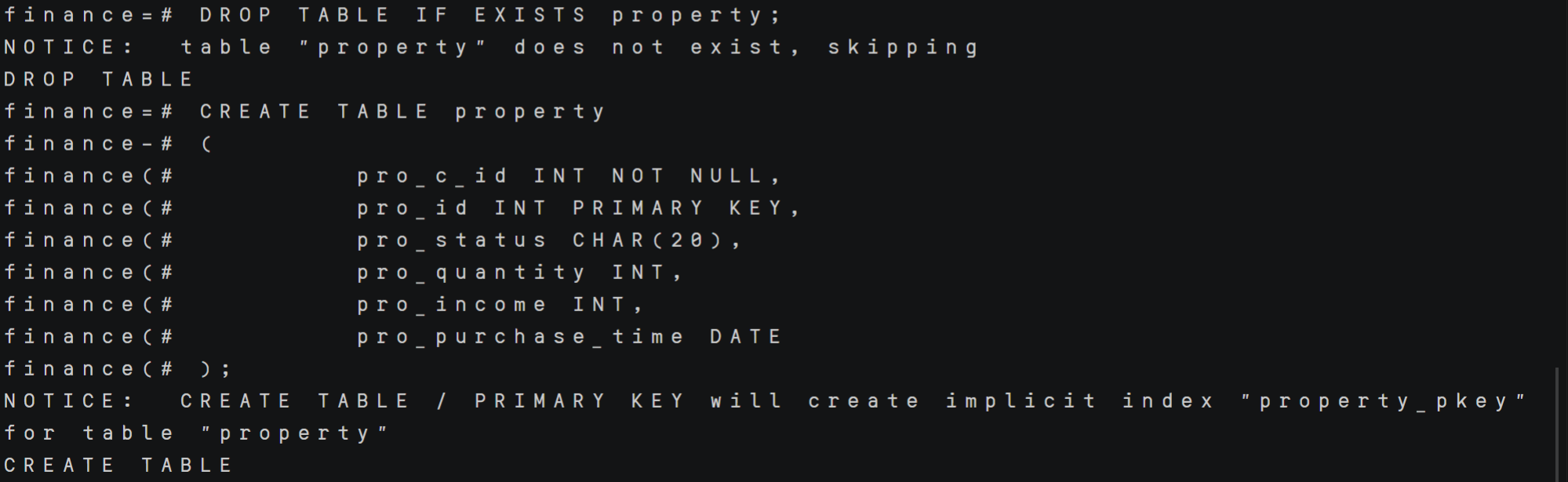
**pro\_status CHAR(20),**

**pro\_quantity INT,**

**pro\_income INT,**

**pro\_purchase\_time DATE**

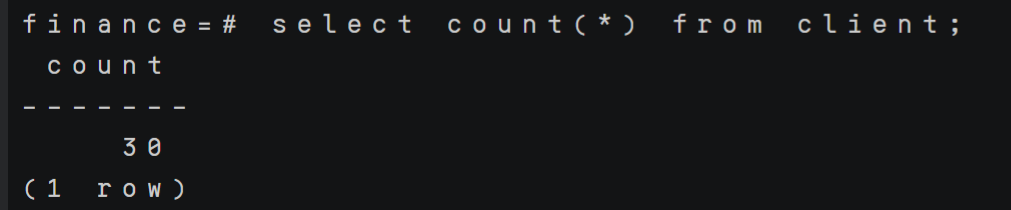
**);**

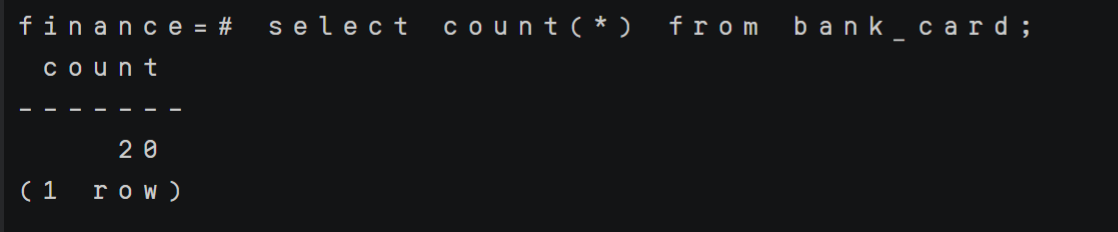


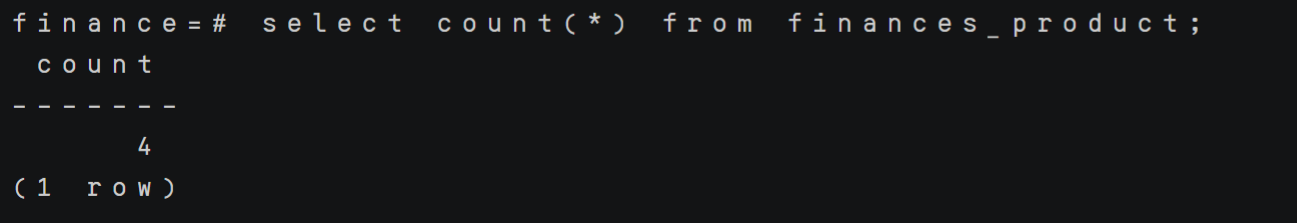
1. 对各个表插入所需数据，实现表的初始化。（语句略）

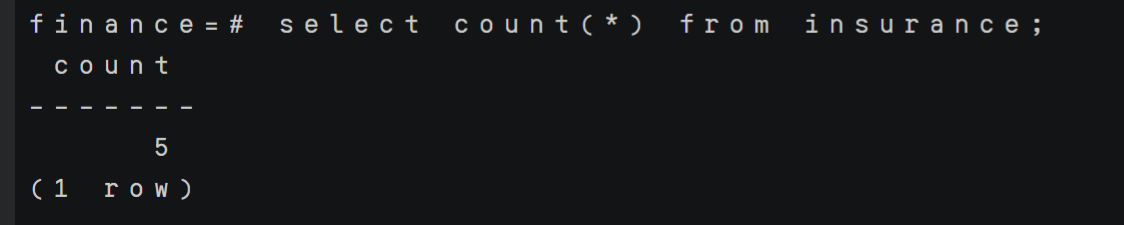
关系代数如：表名：=表名∪（属性）

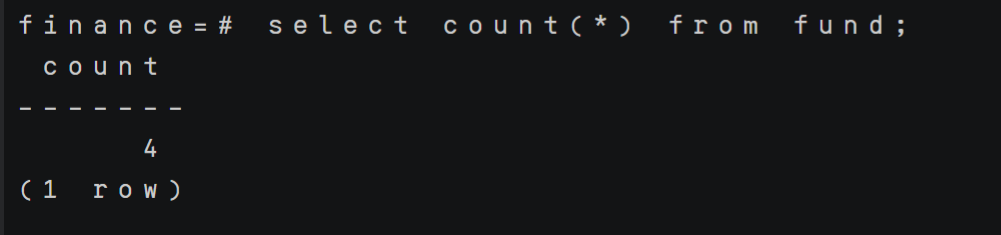
例：client:=client∪(1,'张一','zhangyi@huawei.com','340211199301010001','18815650001','gaussdb\_001')

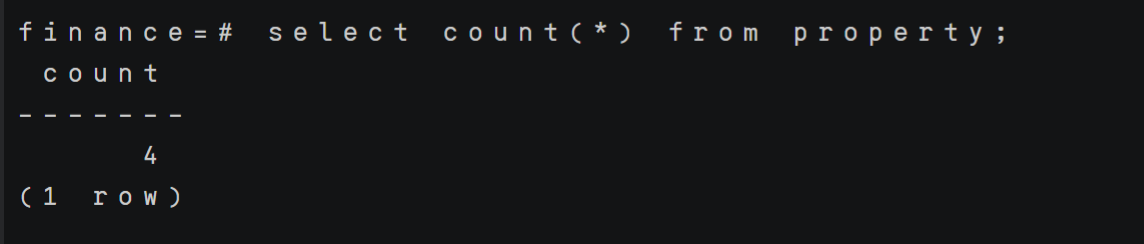






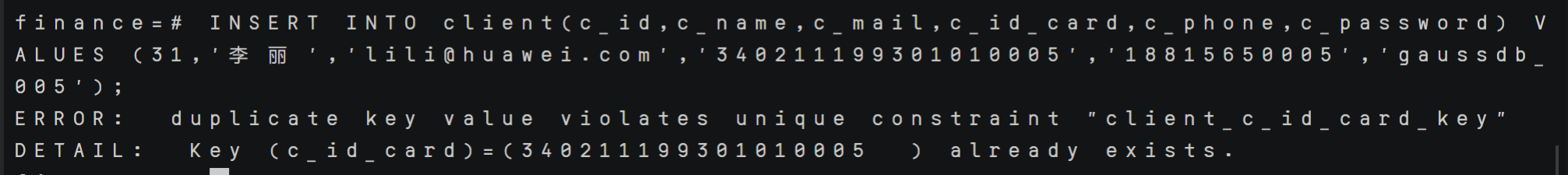




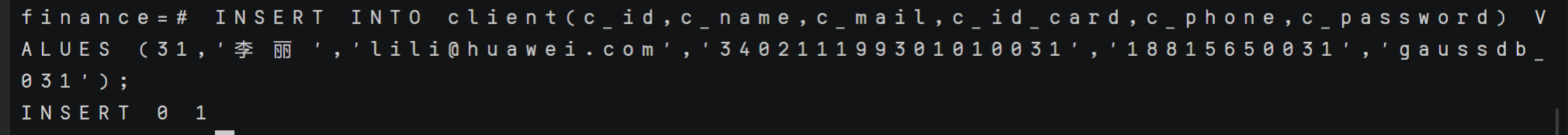


1. 手工插入
2. 当插入数据属性冲突时（由于在表的创建过程中，实验定义了c\_id\_card和c\_phone为唯一且非空（UNIQUE NOT NULL），所以当表中存在时，插入数据失败）：

Finance:=finance∪(31,’李丽’,’lili@huawei.com’,’32021199301010005)



1. 插入数据满足要求时手工插入自然成功：

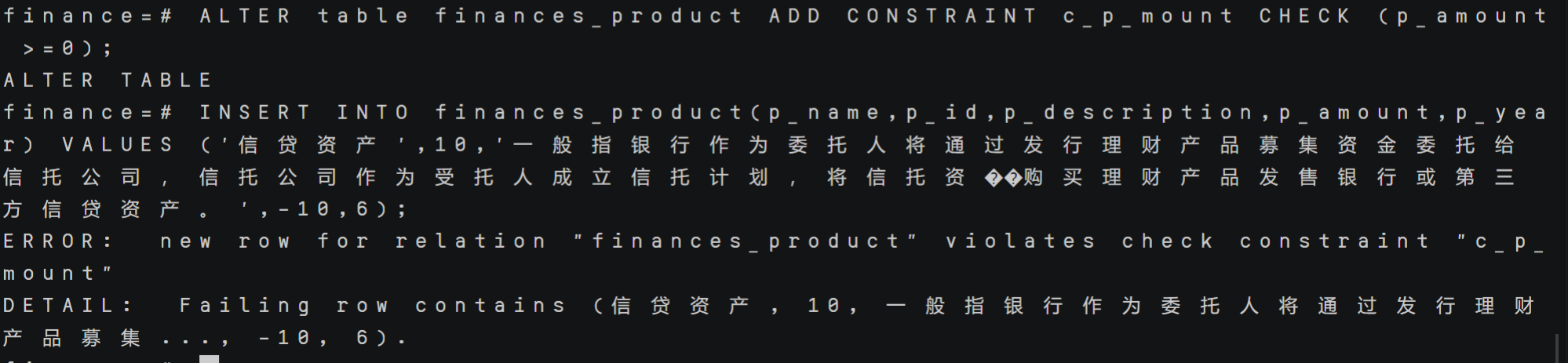


1. 添加约束

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);**

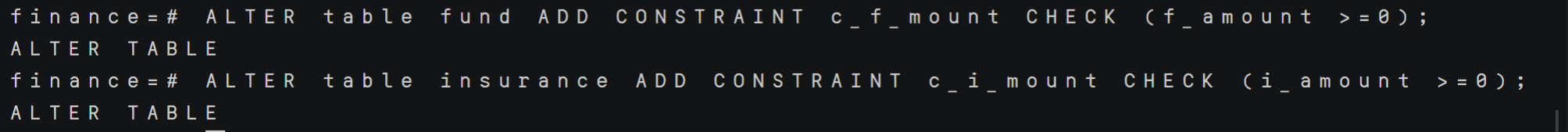
添加后尝试插入一条p\_amount<0的记录则报错：



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。

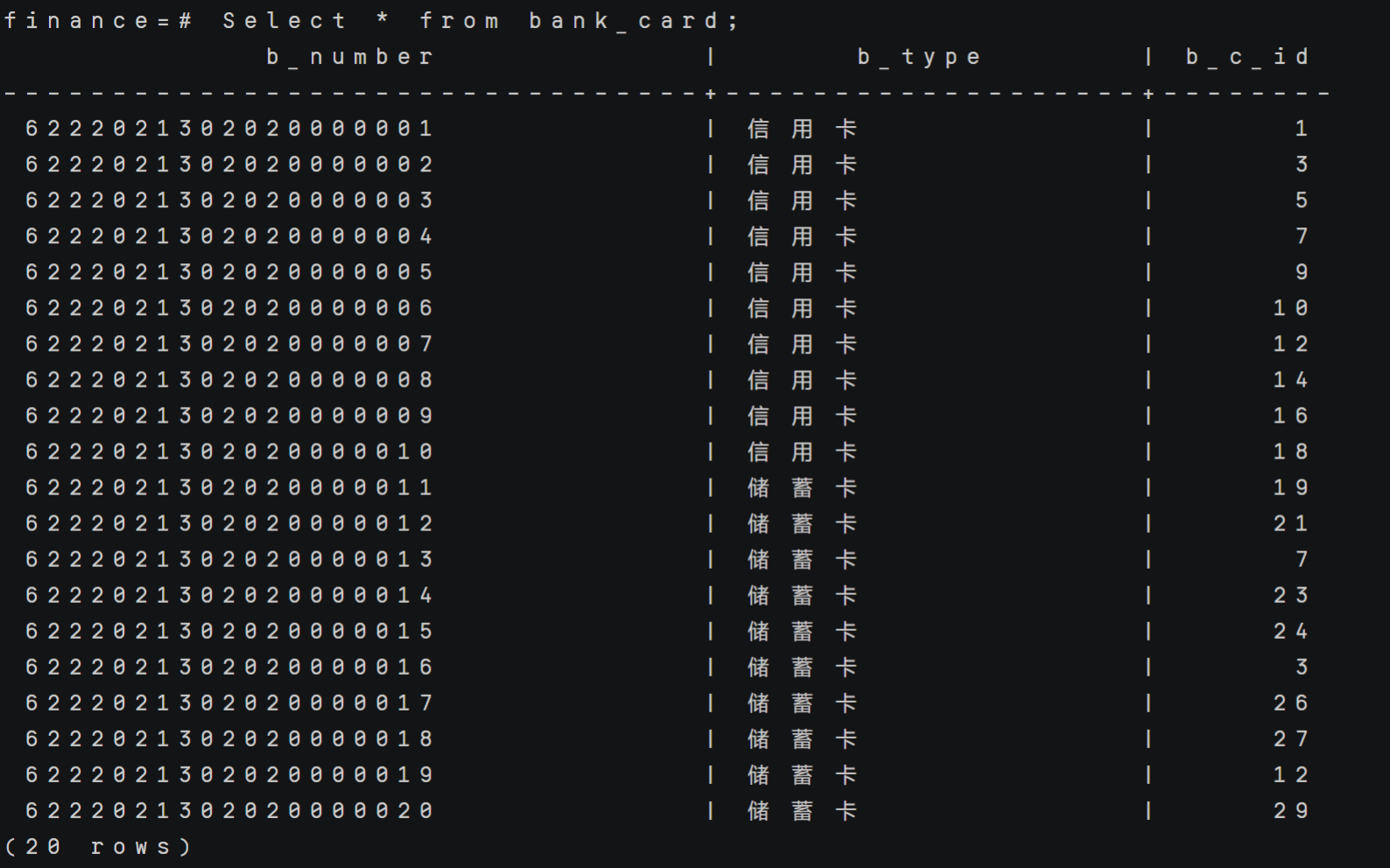
**ALTER TABLE fund ADD CONSTRAINT c\_f\_amount CHECK (f\_amount>=0);**

**ALTER TABLE insurance ADD CONSTRAINT c\_i\_amount CHECK (i\_amount>=0);**



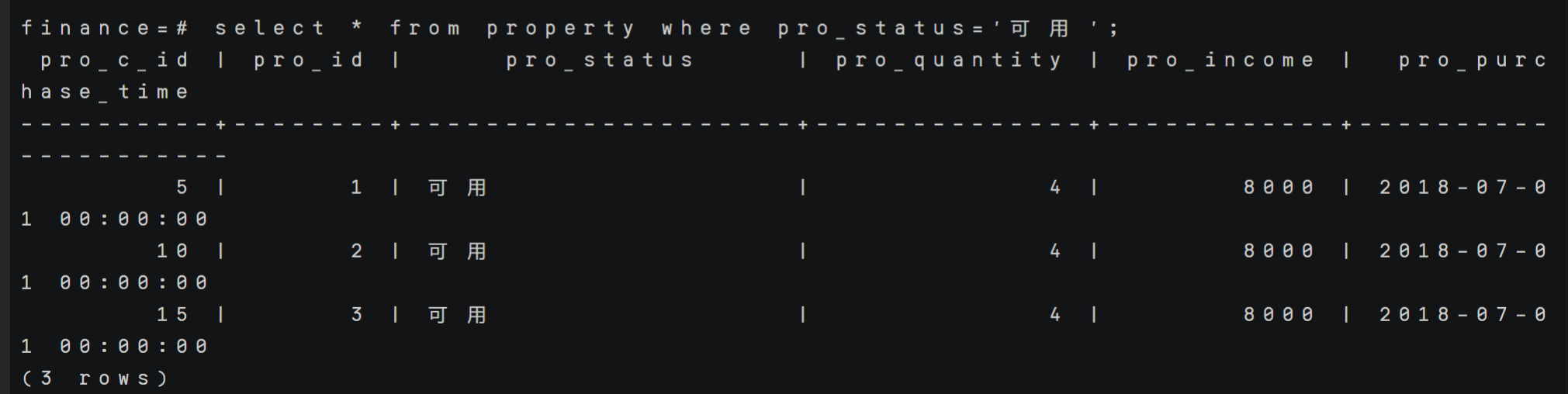
1. 查询数据（以下为自己所编写SQL语句）
2. 查询银行卡信息表。

**Select \* from bank\_card;**



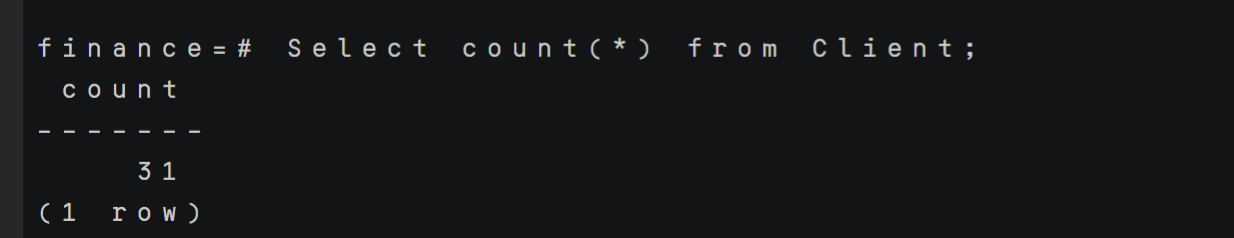
1. 查询资产信息中‘可用’的资产数据。

**Select \* from property where pro\_status=’可用’;**



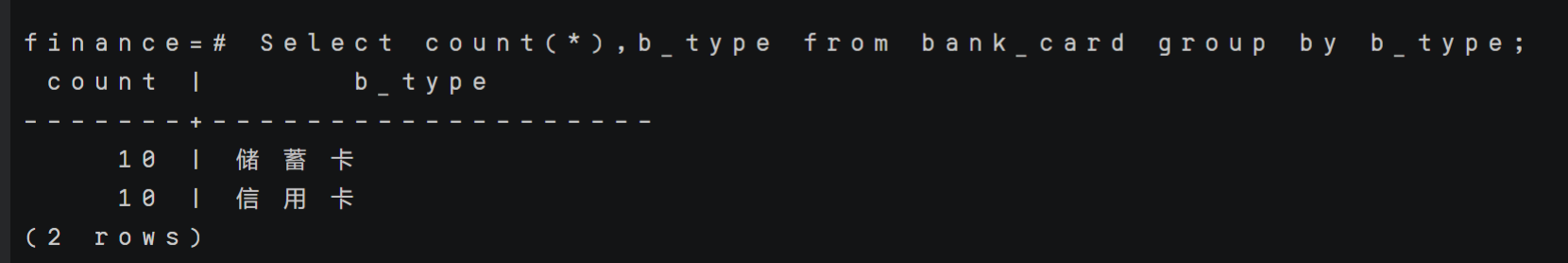
1. 查询用户表中有多少个用户：

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



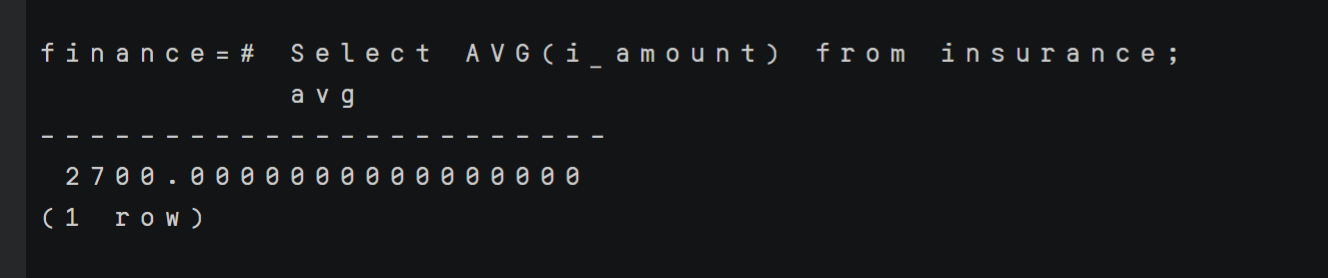
1. 查询银行卡信息表中储蓄卡和信用卡的个数

**Select count(\*),b\_type from bank\_card group by b\_type;**



1. 查询保险信息表中保险金额的平均值：

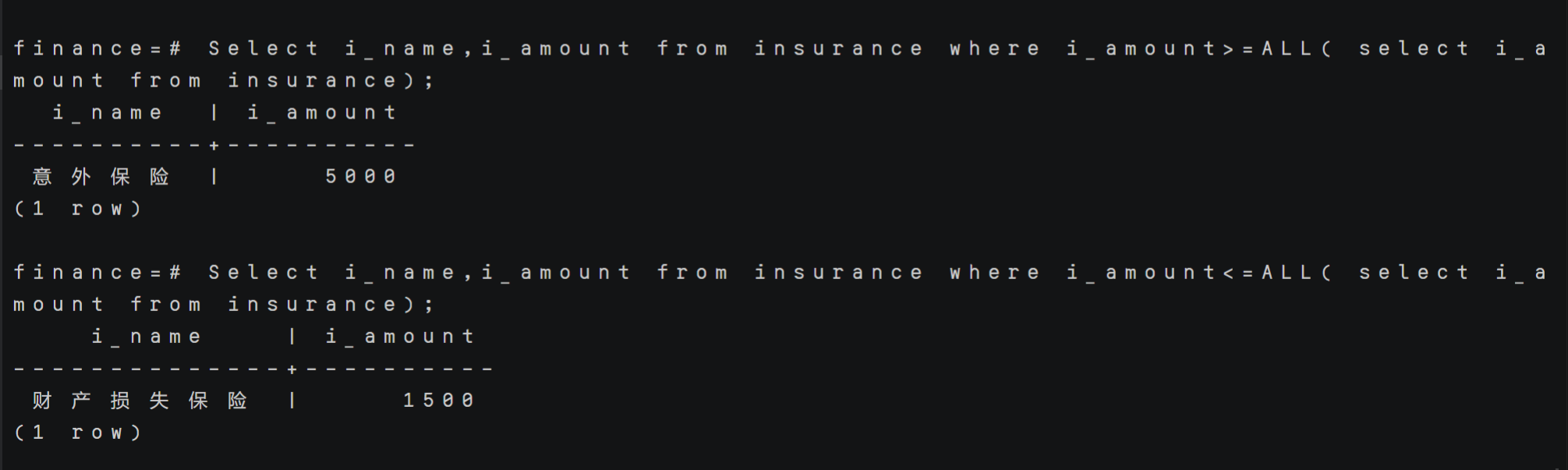
**Select AVG(i\_amount) from insurance;**



1. 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);**



1. 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：

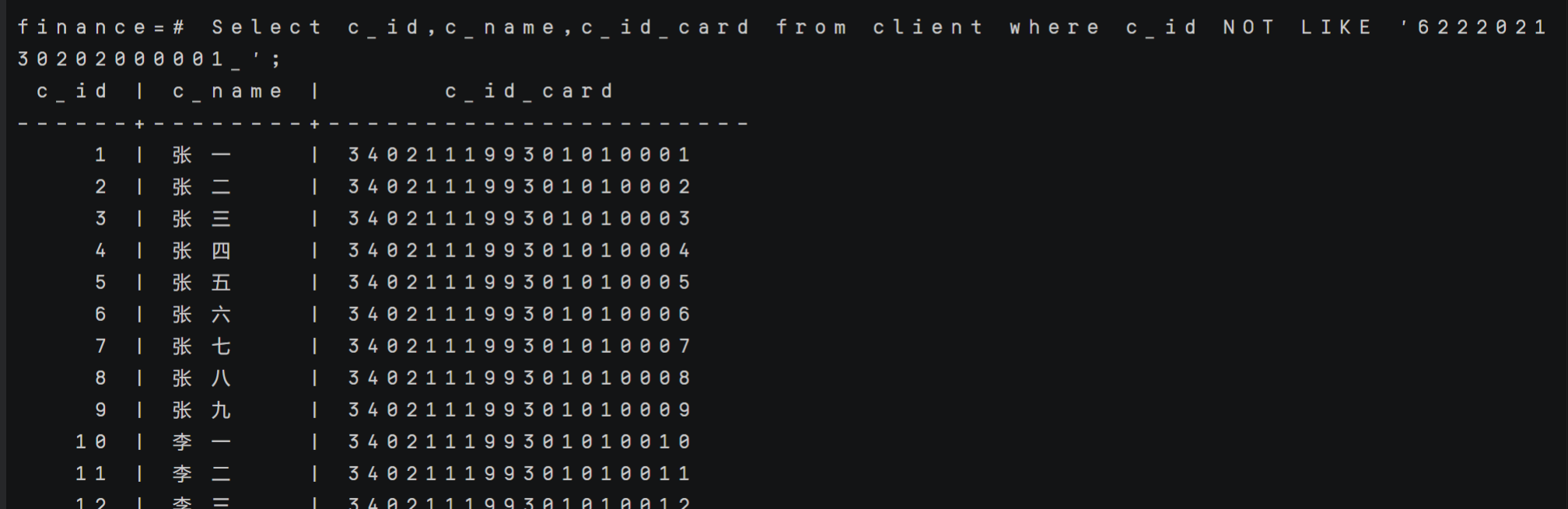
**Select c\_id,c\_name,c\_id\_card from client where exists( select \* from bank\_card where client.c\_id=bank\_card.b\_c\_id);**

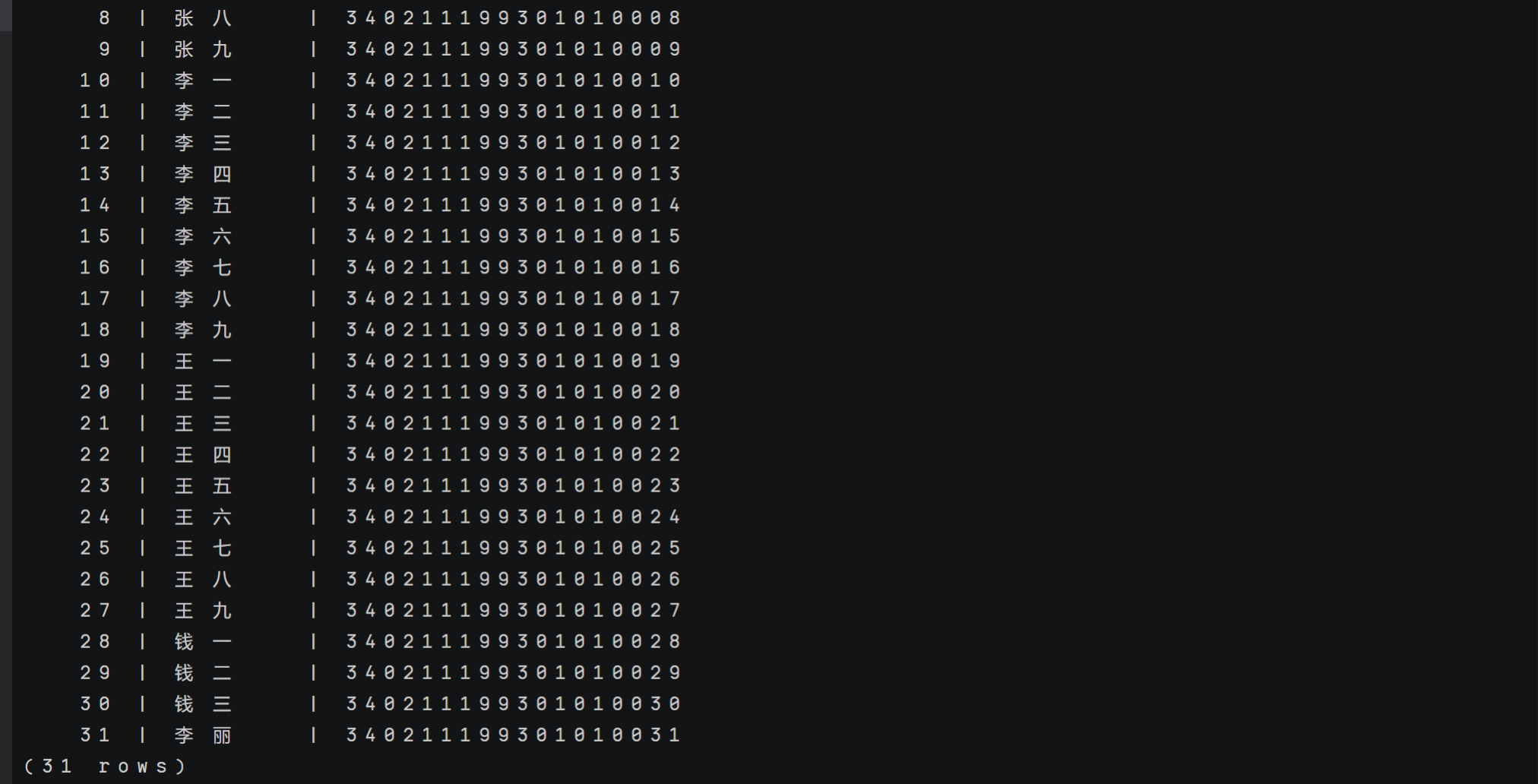


1. 查询银行卡号不是‘622202130202000001\*’的用户的编号，姓名和身份证：

**Select c\_id,c\_name,c\_id\_card from client where c\_id NOT LIKE '622202130202000001\_';**

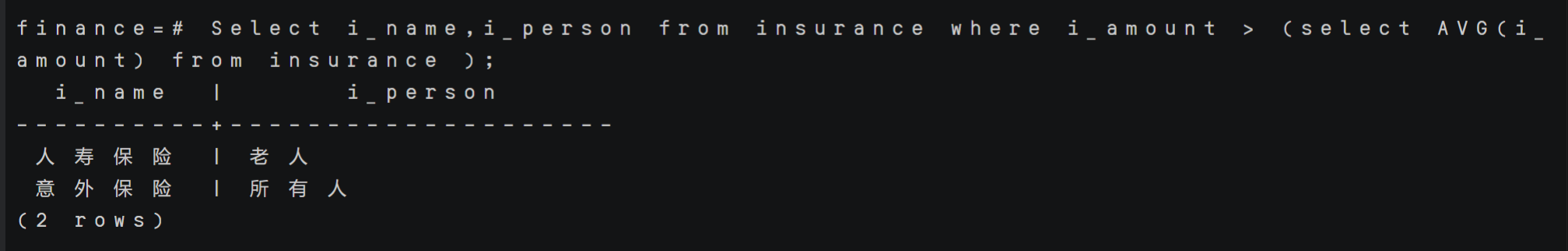
此条查询语句无法使用关系代数查询。





1. 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：

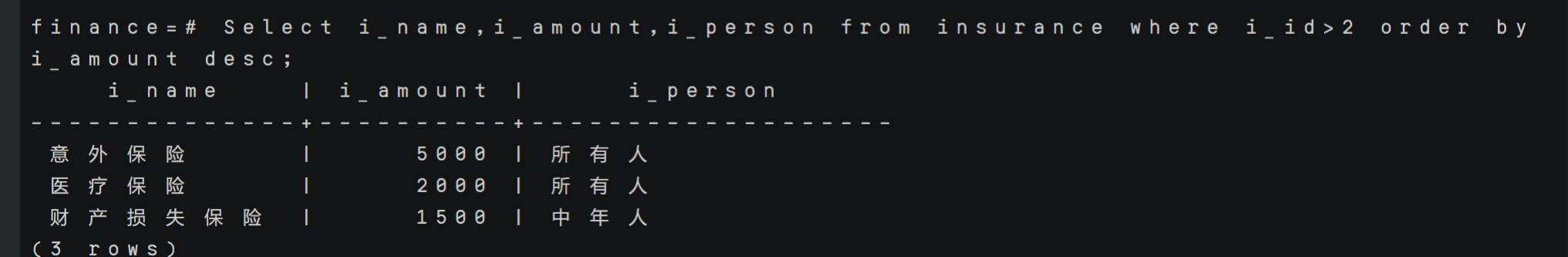
**Select i\_name,i\_person from insurance where i\_amount > (select AVG(i\_amount) from insurance );**



1. 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

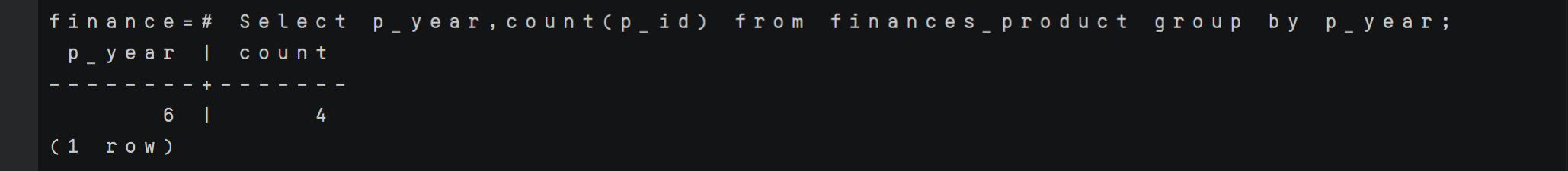
**Select i\_name,i\_amount,i\_person from insurance where i\_id>2 order by i\_amount desc;**

无法使用关系代数实现“降序”操作，仅查询操作实现如下：



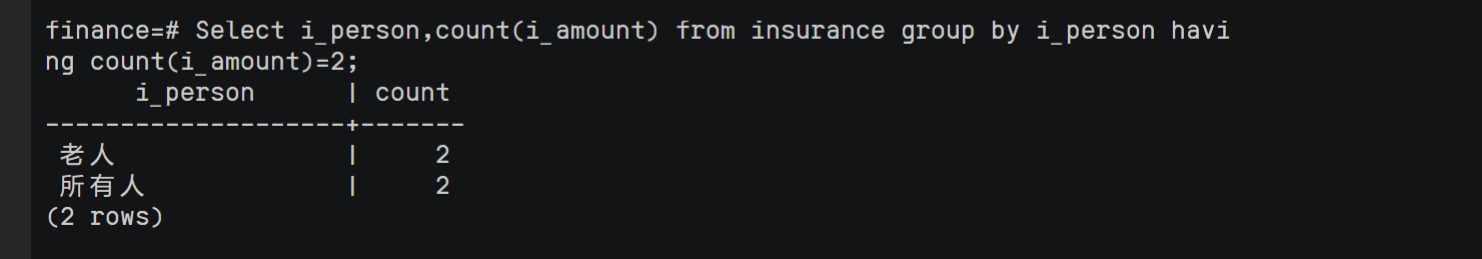
1. 查询各保险信息总数，按pyear分组：

**Select p\_year,count(p\_id) from finances\_product group by p\_year;**



1. 查询保险金额统计数量=2的适用人群数：

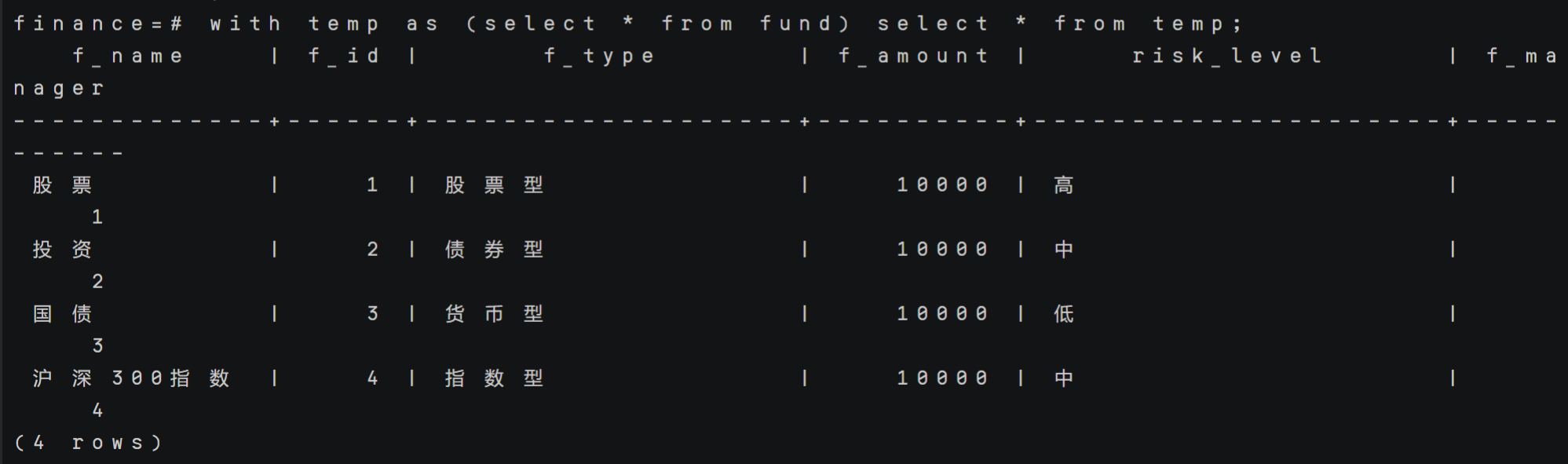
**Select i\_person,count(i\_amount) from insurance group by i\_person having count(i\_amount)=2;**



1. 使用with as查询基金信息表：

**WITH temp AS (SELECT \* FROM fund)**

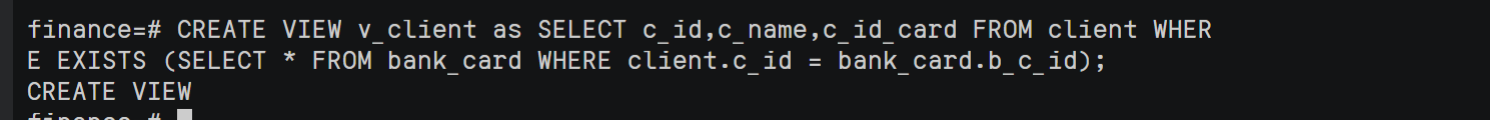
**SELECT \* FROM temp;**

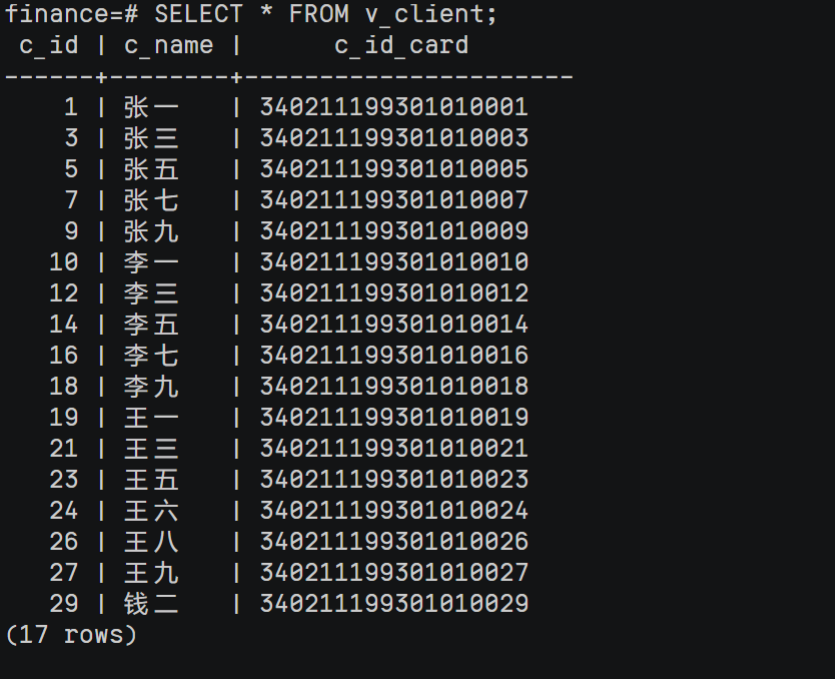


1. 创建视图：

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。

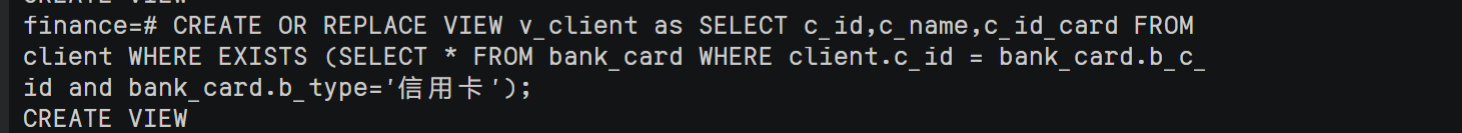
**CREATE VIEW v\_client as SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id);**

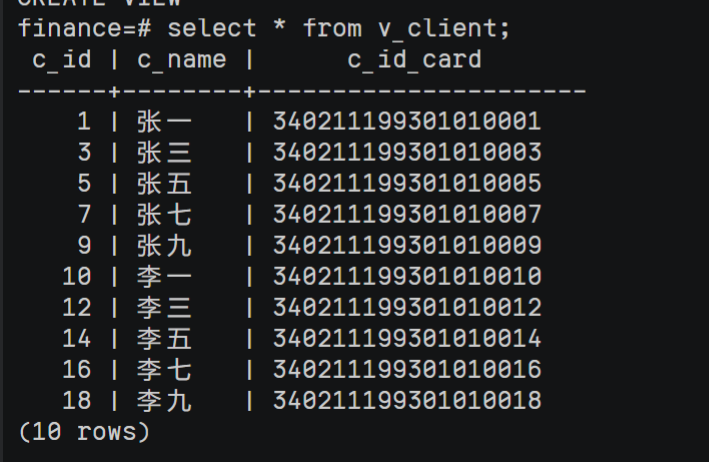




修改视图，在原有查询的基础上，过滤出信用卡用户。

**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');**



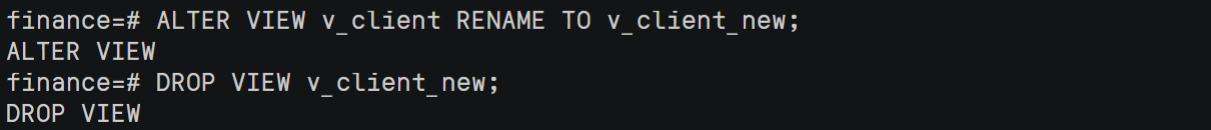


修改视图名称：

**ALTER VIEW v\_client RENAME TO v\_client\_new;**

删除视图：

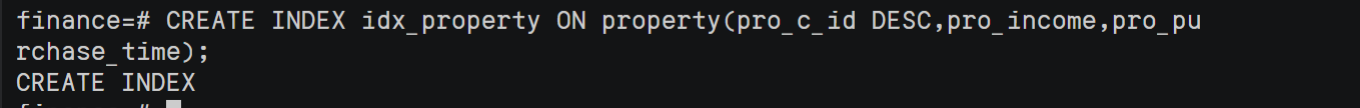
**DROP VIEW v\_client\_new;**



1. 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);**

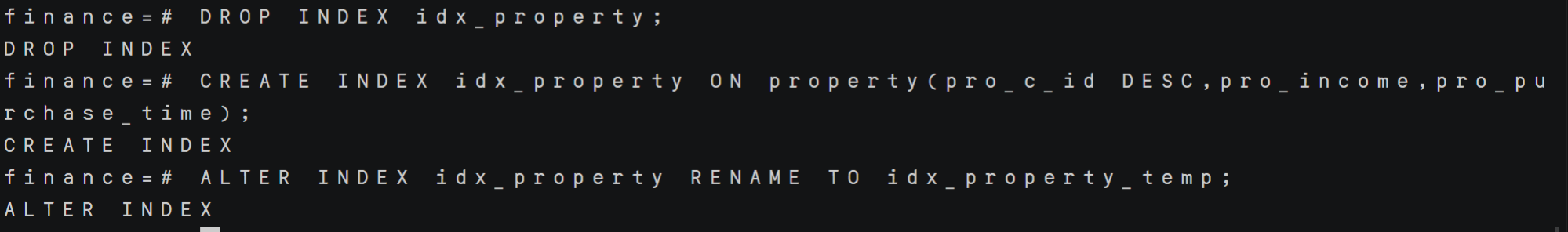


重命名索引：

**DROP INDEX idx\_property;**

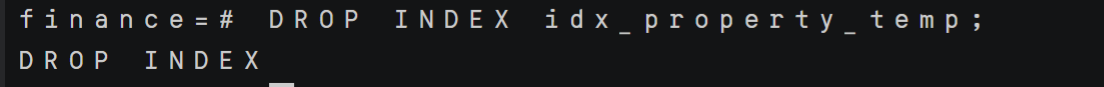
**CREATE INDEX idx\_property ON property(pro\_c\_id DESC,pro\_income,pro\_purchase\_time);**

**ALTER INDEX idx\_property RENAME TO idx\_property\_temp;**



删除索引：

**DROP INDEX idx\_property\_temp;**

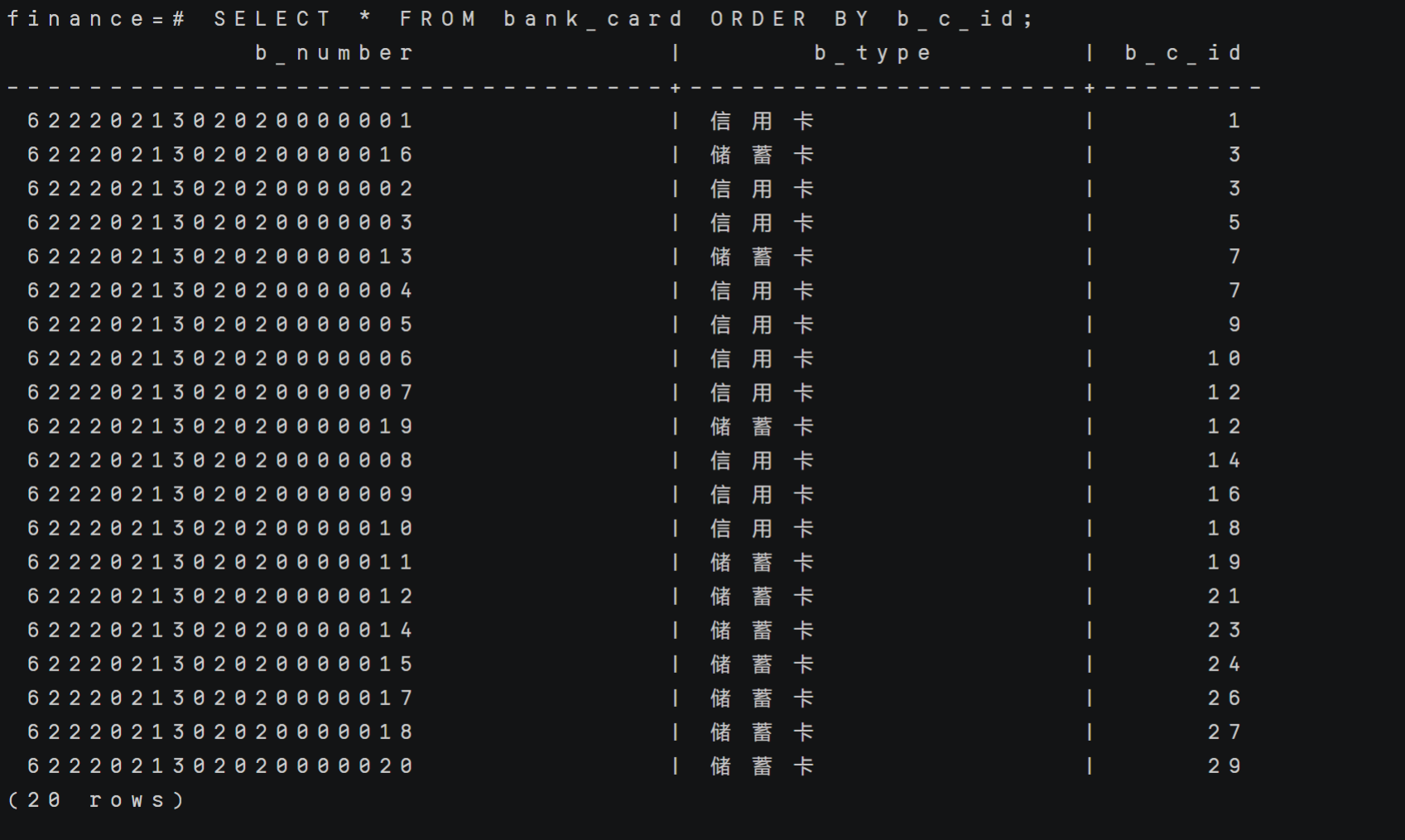


1. 数据的修改和删除

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。

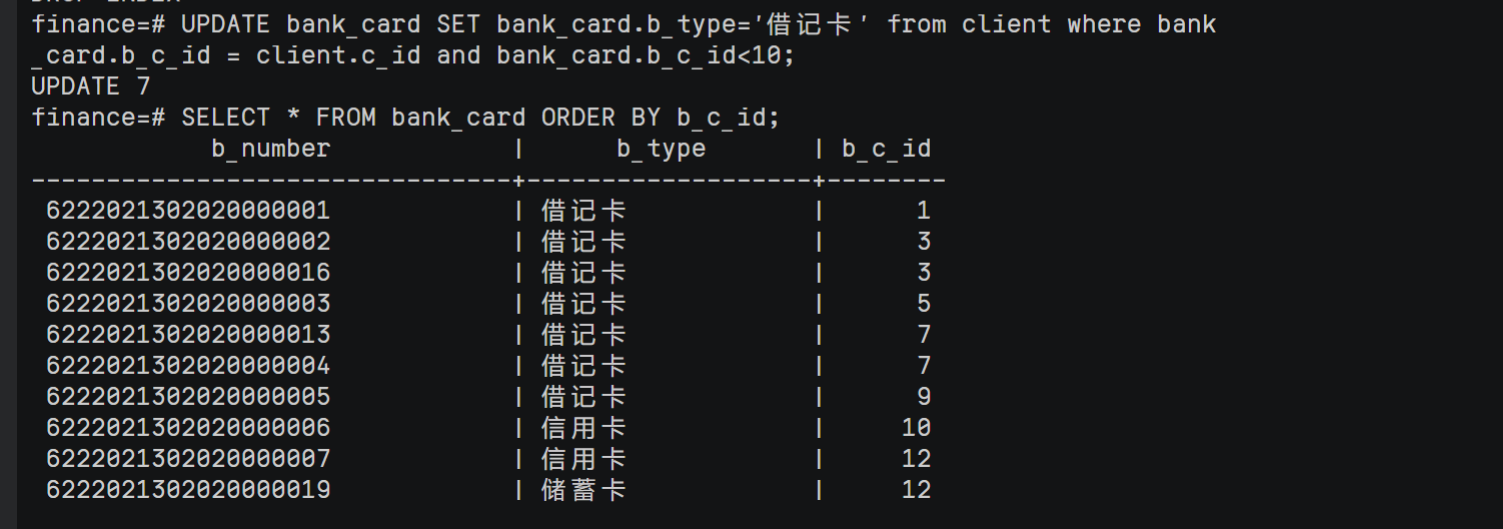
**UPDATE bank\_card SET bank\_card.b\_type='借记卡' from client where bank\_card.b\_c\_id = client.c\_id and bank\_card.b\_c\_id<10;**

修改前：



修改后：

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

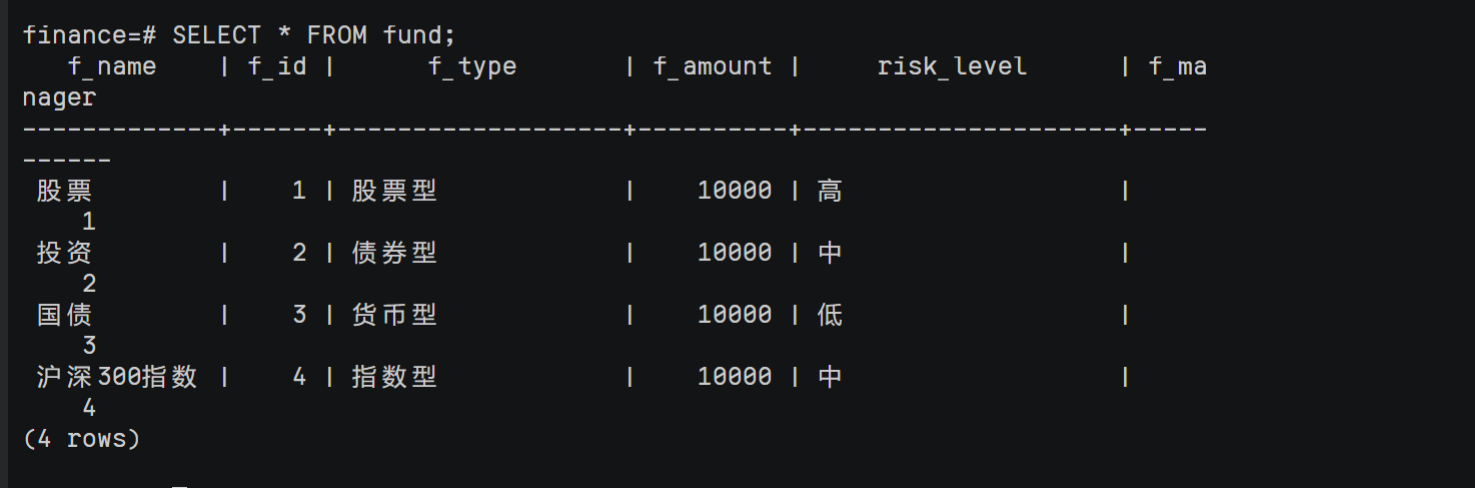


删除基金信息表中编号小于3的行：

**DELETE FROM fund WHERE f\_id<3;**

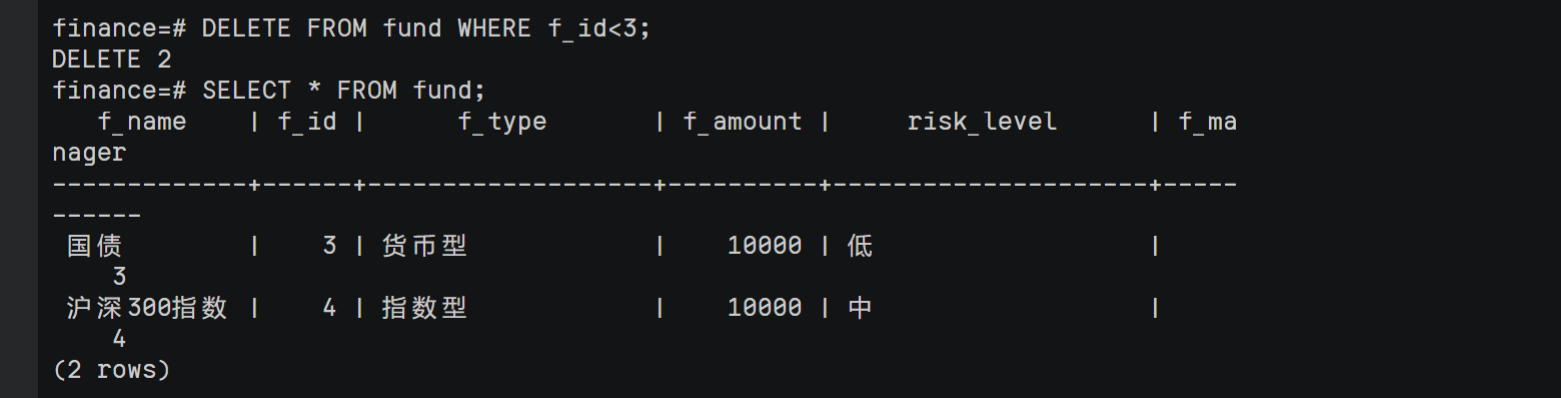
删除前：

**SELECT \* FROM fund;**



删除后：

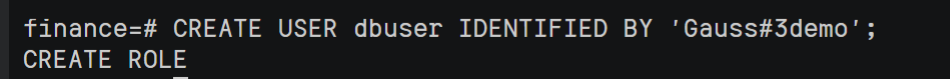
**SELECT \* FROM fund;**



1. 选做部分：
2. 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。

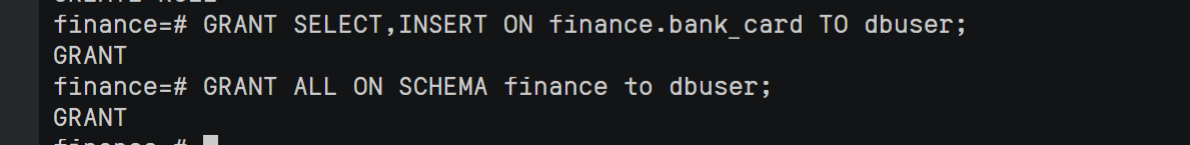
CREATE USER dbuser IDENTIFIED BY 'Gauss#3demo';



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;

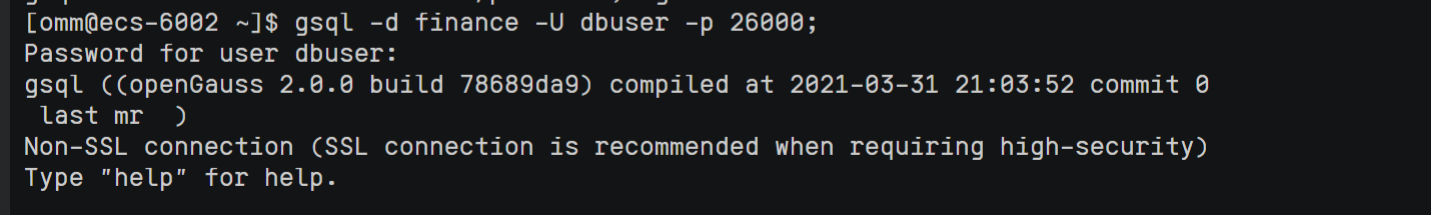


1. 新用户连接数据库：

在gsql登录数据库，使用新用户连接。

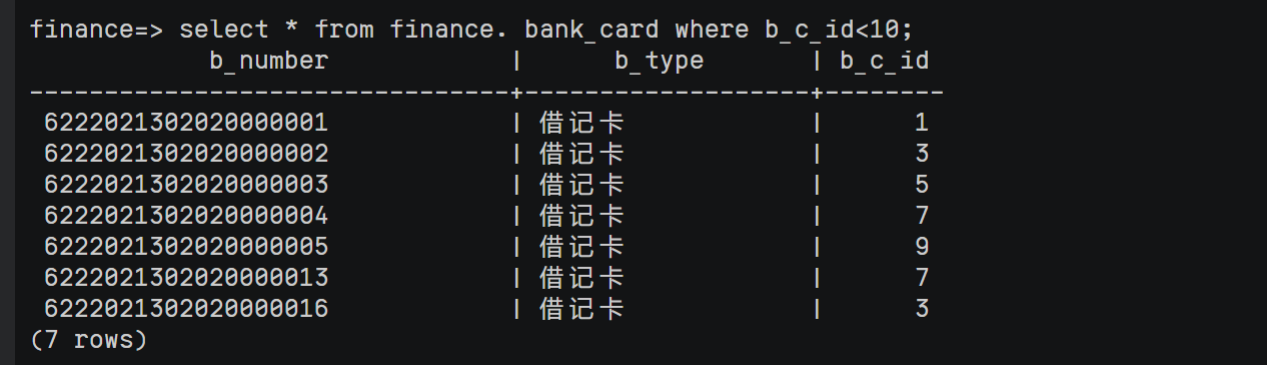
使用操作系统omm用户在新的窗口登陆并执行以下命令，并输入对应的密码。

gsql -d finance -U dbuser -p 26000;



访问finance数据库的表bank\_card。

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



1. 删除schema操作：

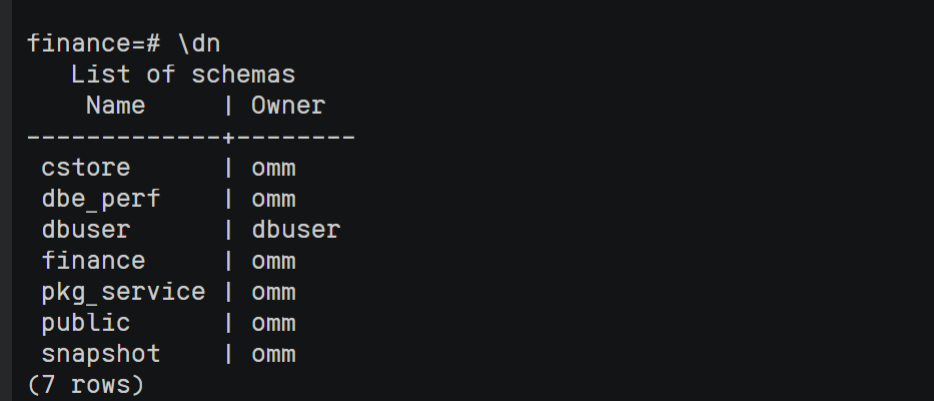
使用管理员用户登陆finance数据库。

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

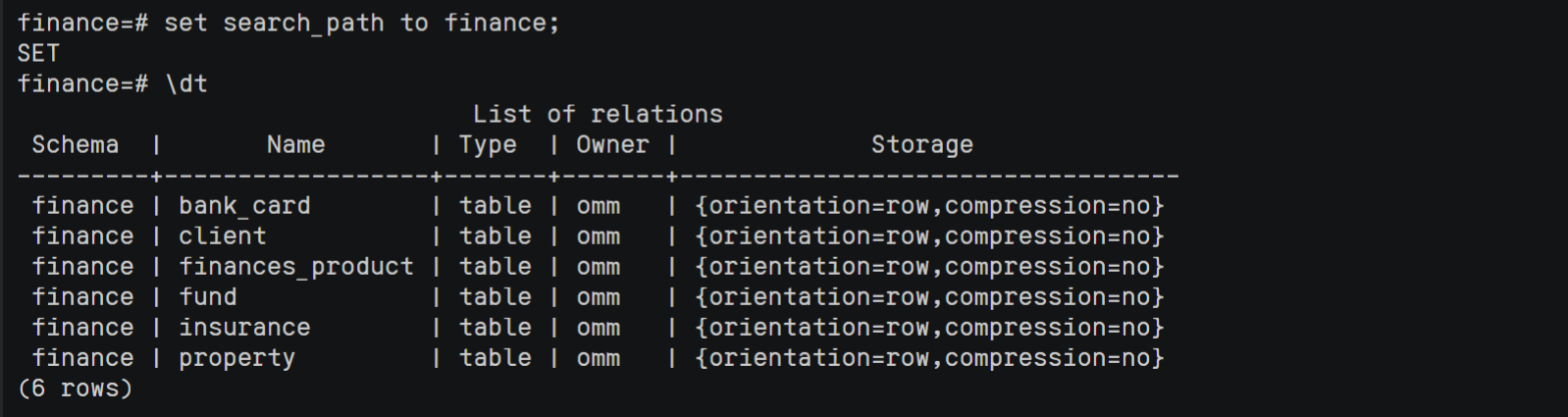


设置默认查询为finance。

set search\_path to finance;

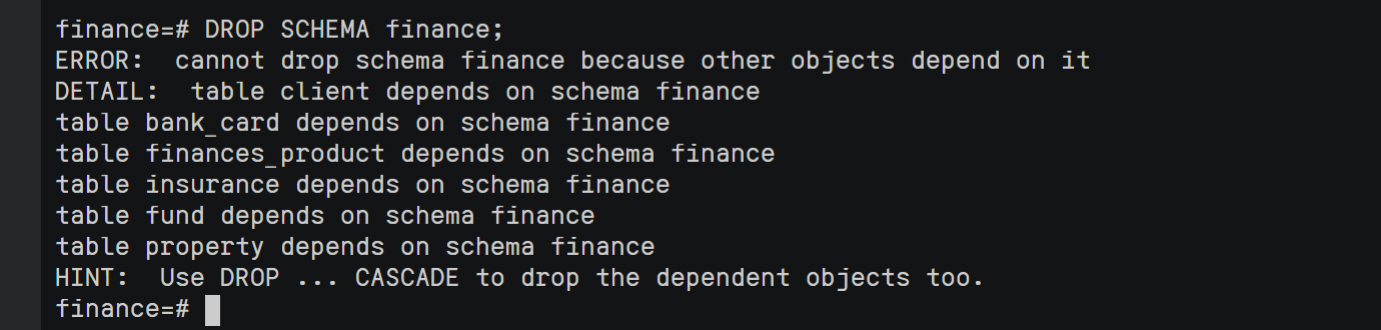
使用“\dt”命令可以看到在finance中的对象。

\dt



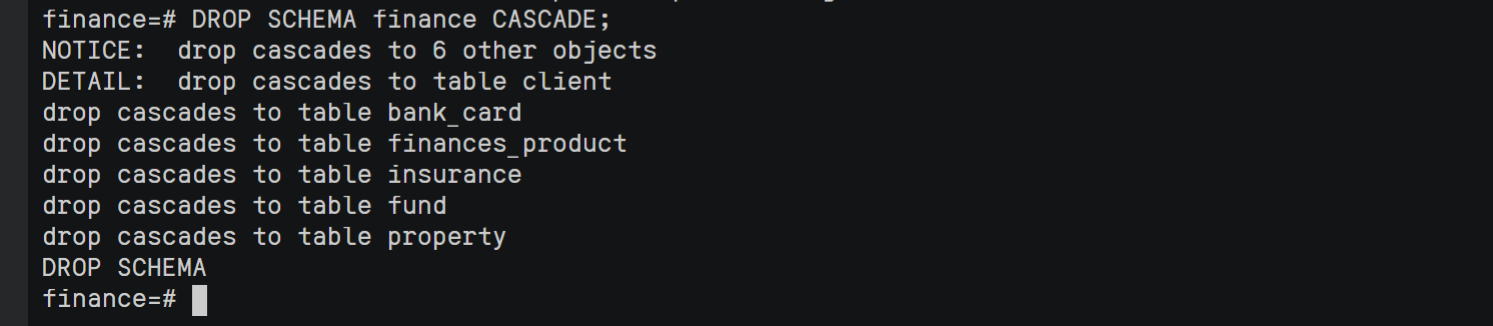
使用DROP SCHEMA 命令删除finance会有报错，因为finance下存在对象。

DROP SCHEMA 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;



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

\dt



1. 相关说明：
2. 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。

1. 执行结果截图附在每条语句后。
2. 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。
3. 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**

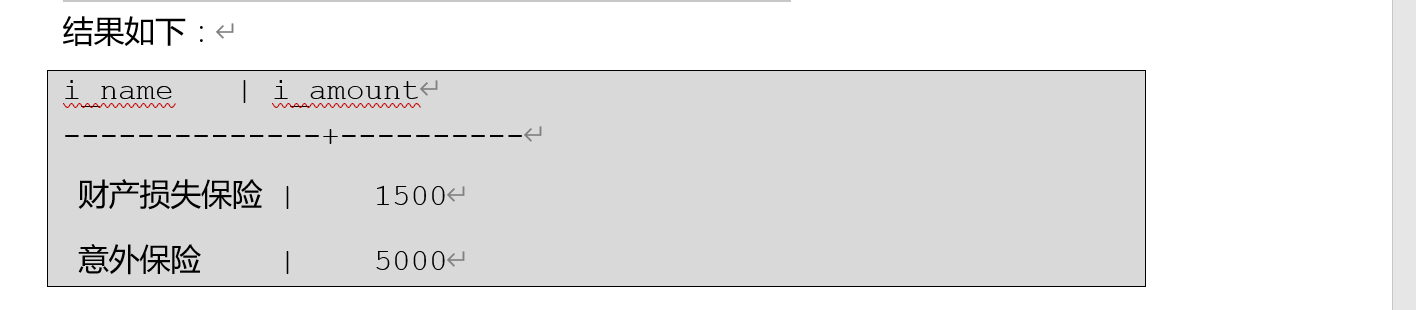
**select i\_name,i\_amount from insurance where i\_amount in (select min(i\_amount) from insurance);**

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。



1. 实验总时长为：前后多次进行实验并截图，共约4h
2. 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。
3. 实验中学习到的知识点：关系代数中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。