基于openGauss的场景化综合实验（金融场景）

姓名： 冯思程 学号： 2112213

实验步骤：

* 创建需要符合金融场景的数据表
* 插入表数据
* 手工插入一条数据
* 添加约束
* 查询数据（自行完成查询语句编写）
* 对视图进行操作
* 对索引进行操作
* 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

1. 实验环境说明

**环境说明**：

openGauss是国产的开源关系型数据库，具有高可靠性、高安全性和高性能等特点，适用于处理高负载和大数据量的应用场景。openGauss支持多种操作系统，本次实验我是在MacOS系统下进行实验。而且在先前我已经完成了openGauss的所有安装进程，而且我利用华为云服务器可以进行连接和管理数据库的所有操作。总体上来说，本次实验是在MacOS系统下利用华为云的ESC利用openGauss数据库进行实验。

**选择 openGauss 环境进行实现的原因**：

1）安全性：作为国产开源数据库，openGauss 有着可靠的安全保障机制，包括加密、审计和安全控制等功能，适合处理金融数据等敏感信息。

2）性能：openGauss 具有高性能和可扩展性，支持并行处理和多实例部署等功能，适用于处理高负载和大数据量的金融应用场景。

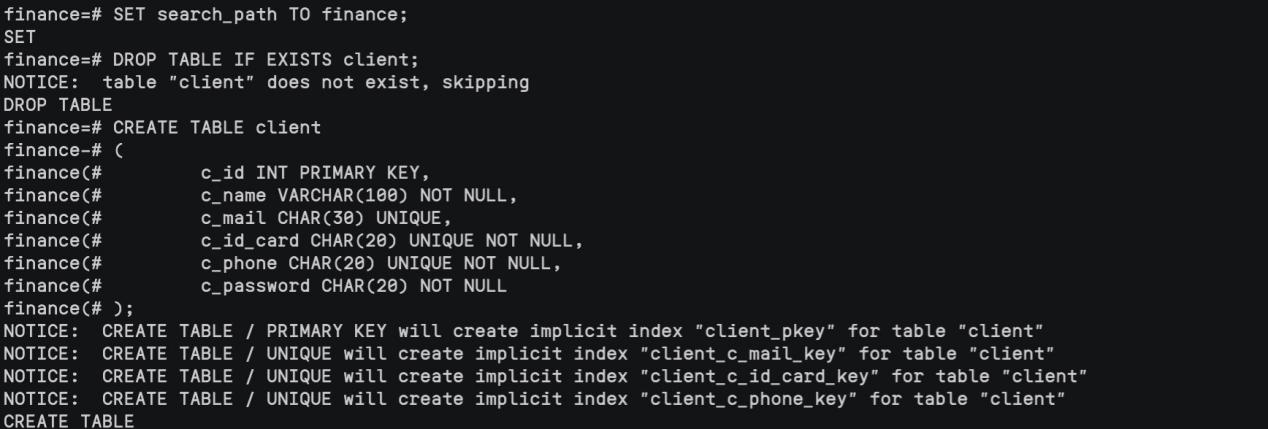
3）开源性：openGauss 是一款开源的数据库，具有丰富的社区资源和生态环境，方便用户定制和扩展功能。

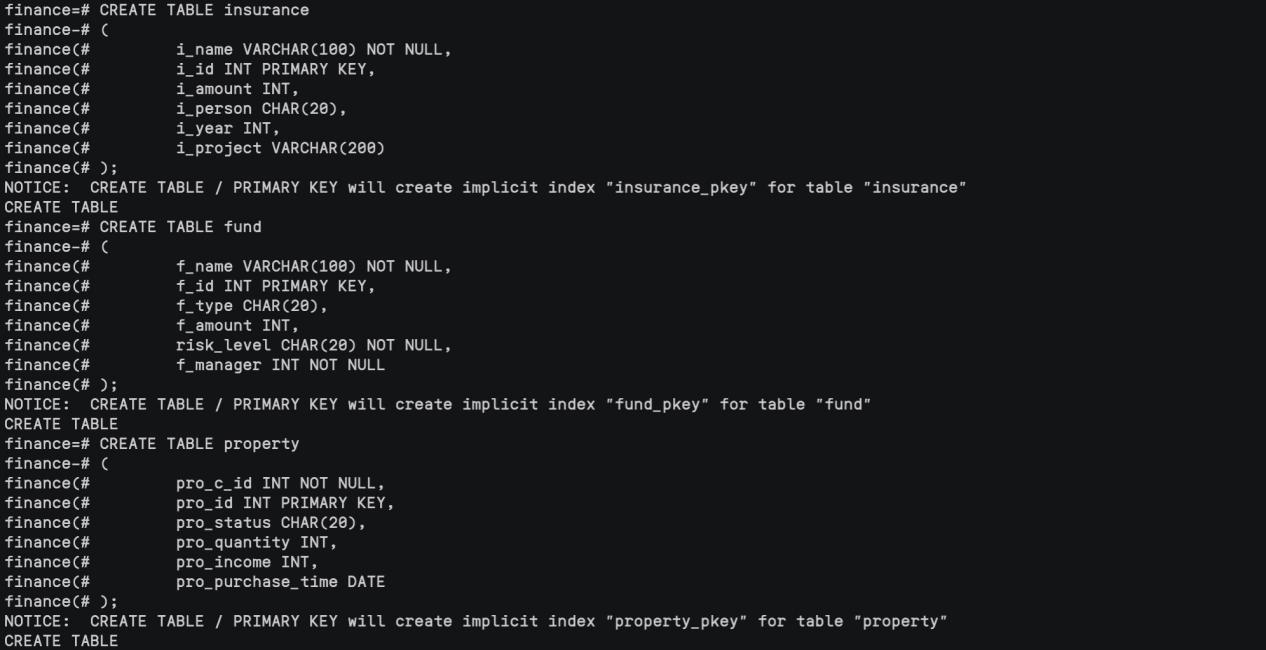
4）支持性：openGauss 具有完善的文档和支持团队，用户可以获得专业的技术支持和服务。

5）便利性：已经在华为云上注册激活了服务器，方便进行实验以及使用。

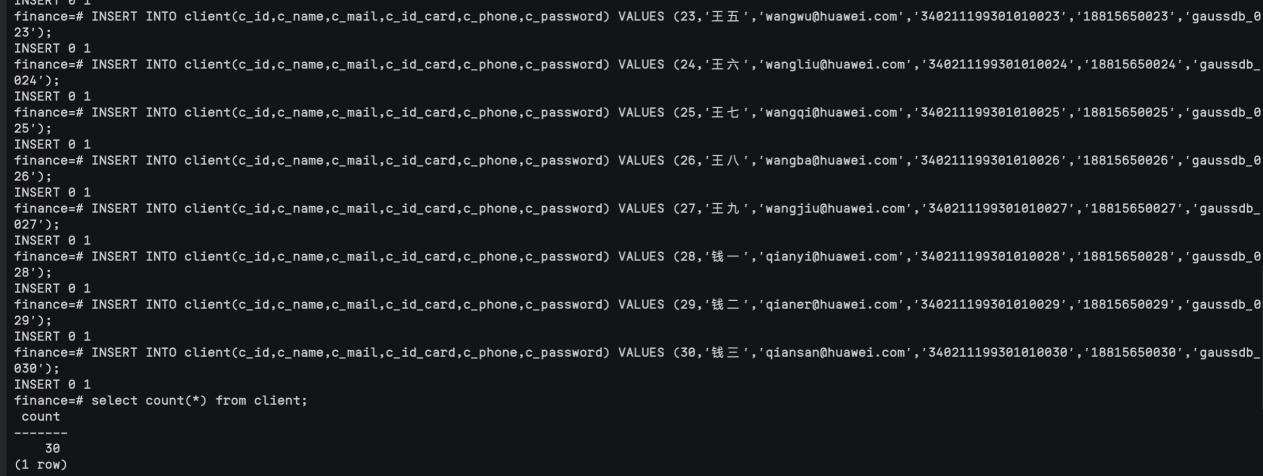
1. 主要步骤截图

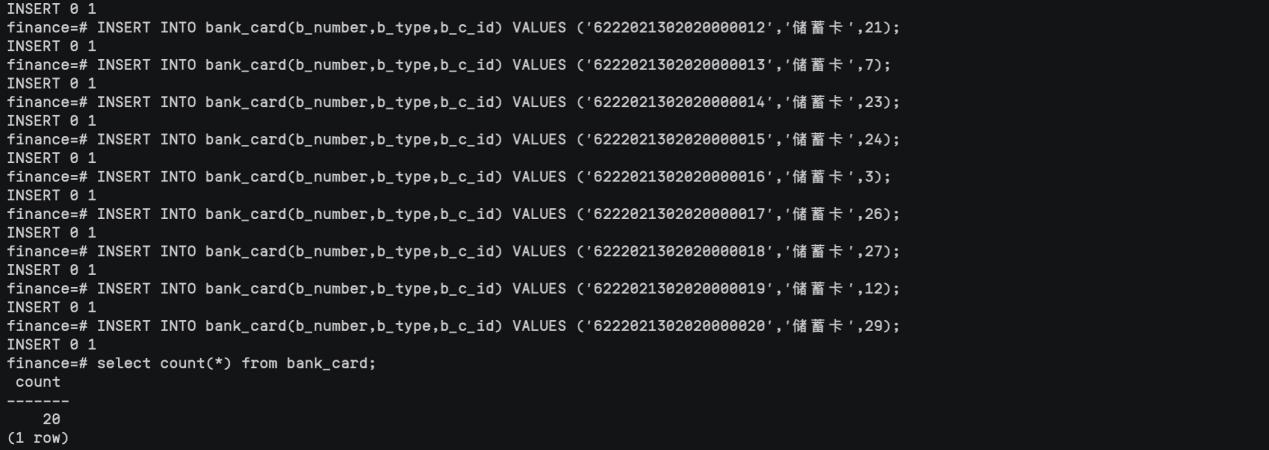
1.1.3创建数据表



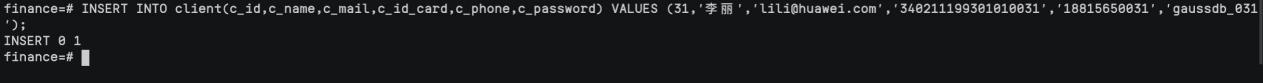
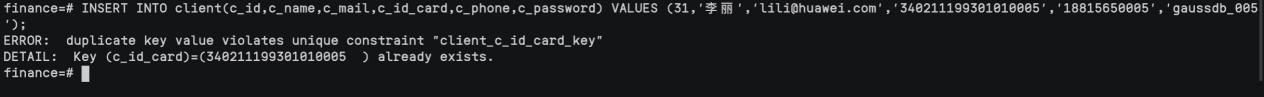


1.1.4插入表数据



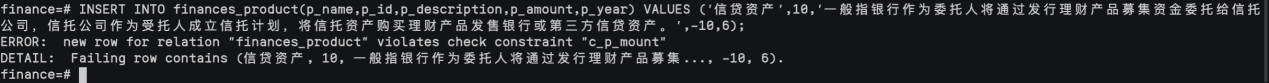


1.1.5手工插入一条数据

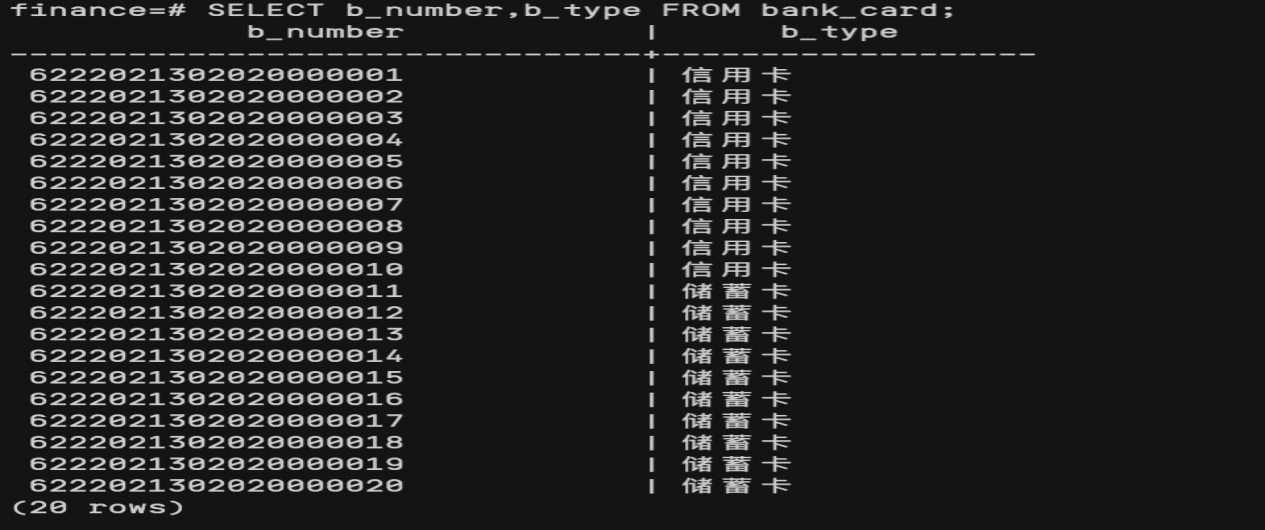


1.1.6添加约束

43821682560120_.pic



1.1.7查询数据



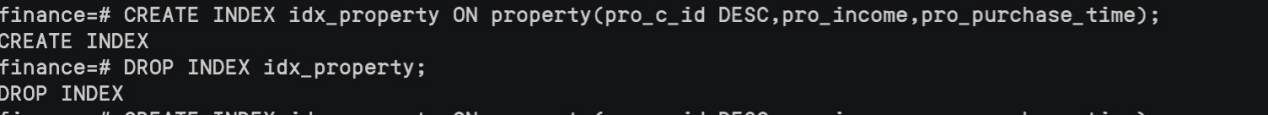


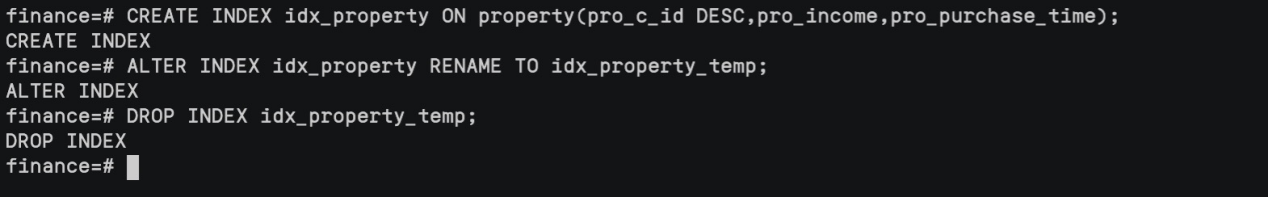
1.1.8视图



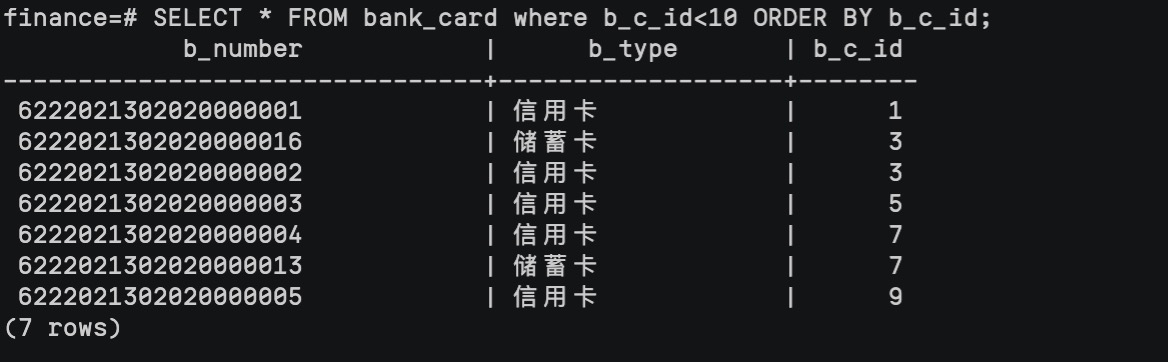


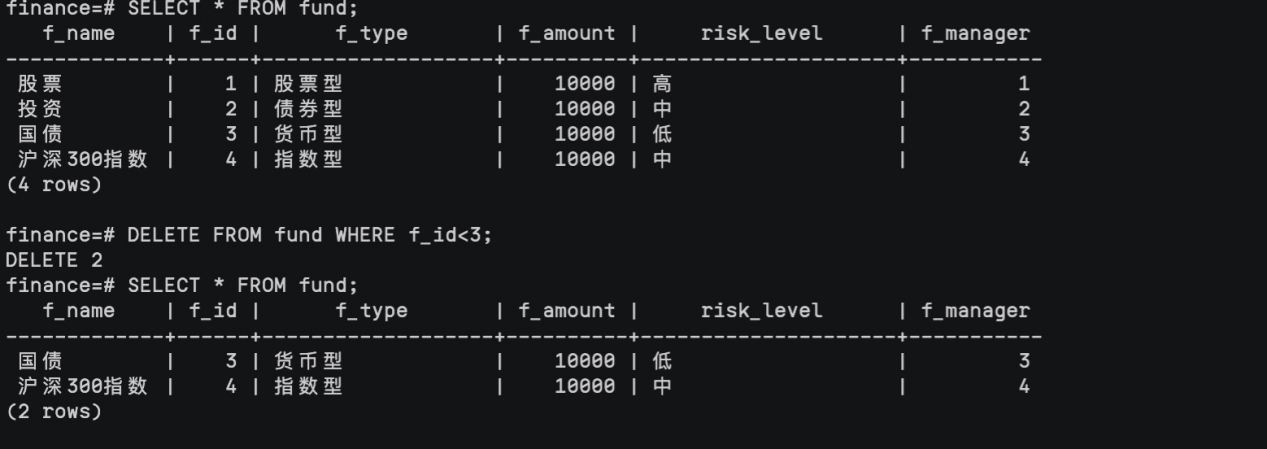
1.1.9索引



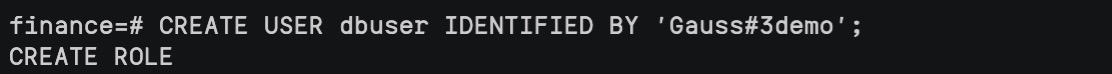


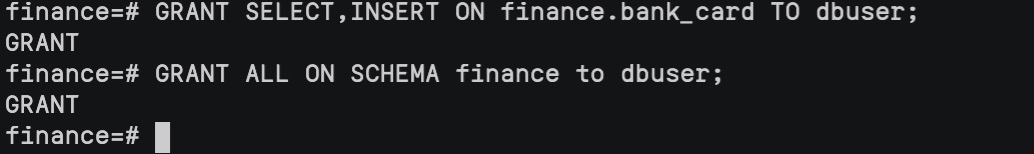
1.1.10数据的修改和删除



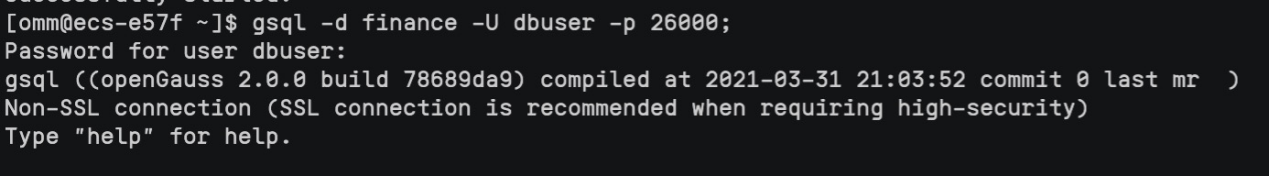


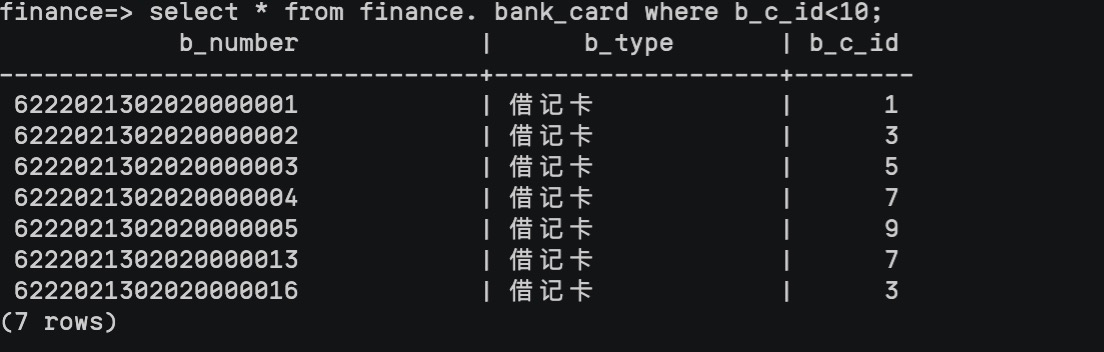
1.1.11新用户的创建和授权



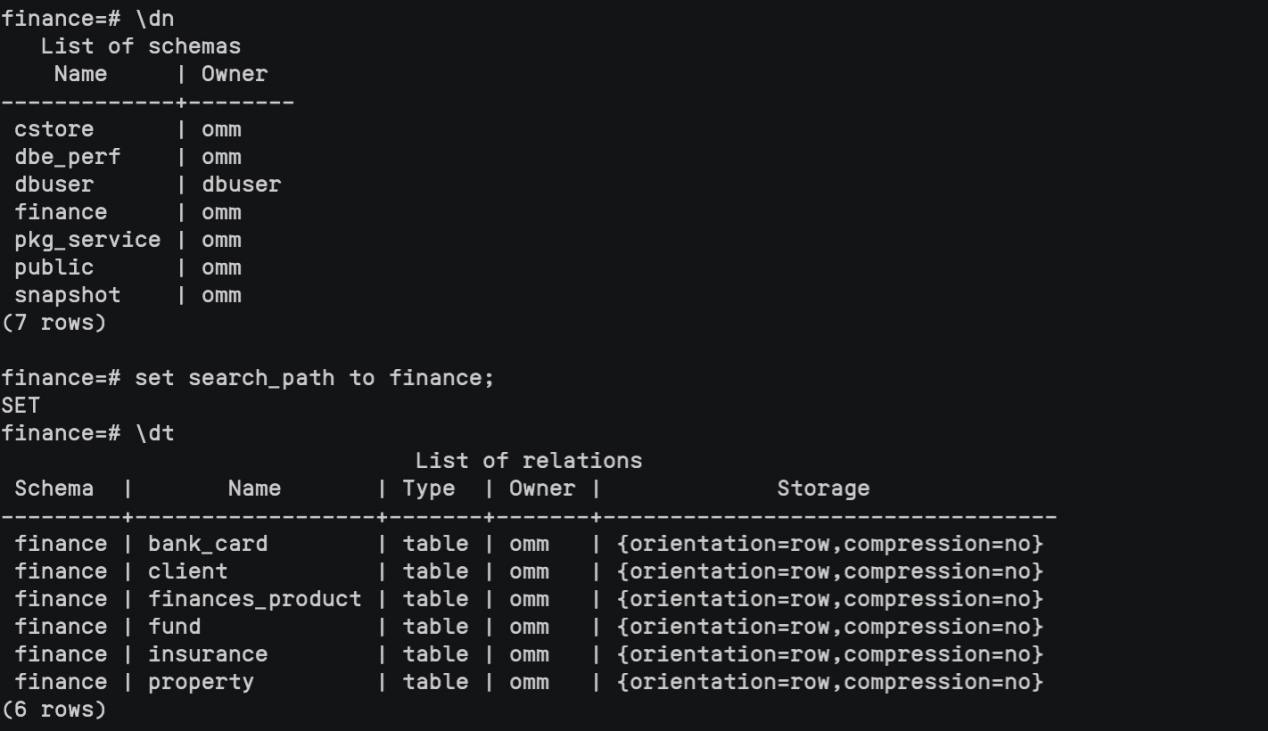


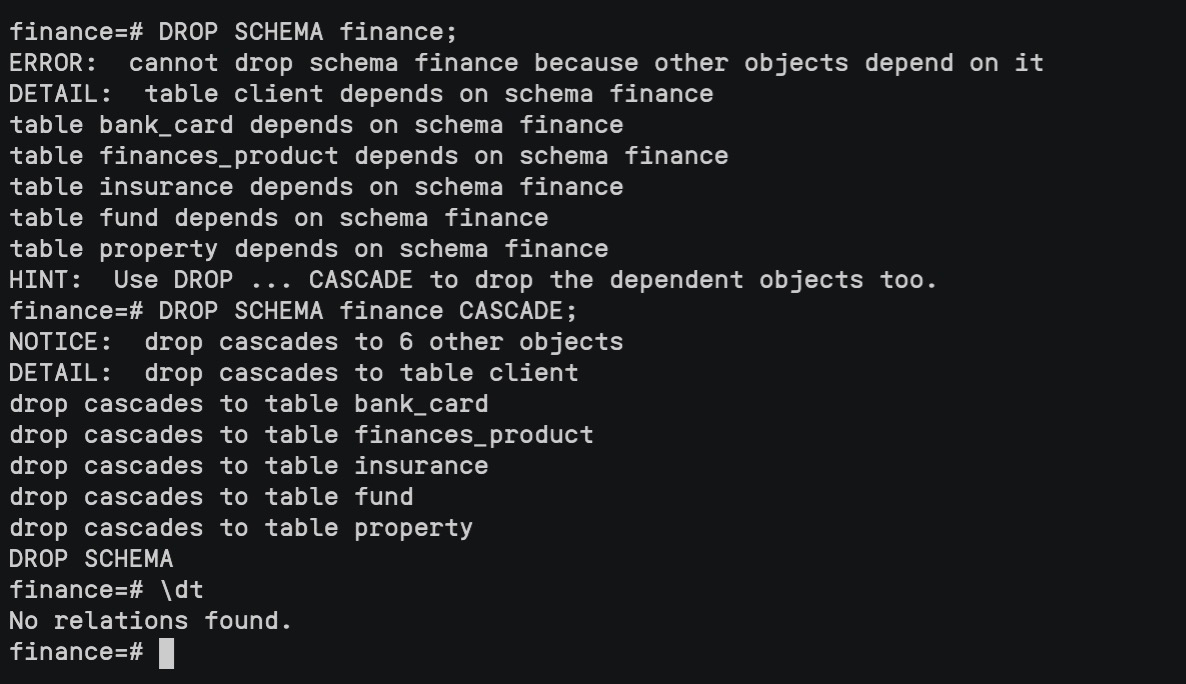
1.1.12新用户连接数据库





1.1.13删除Schema



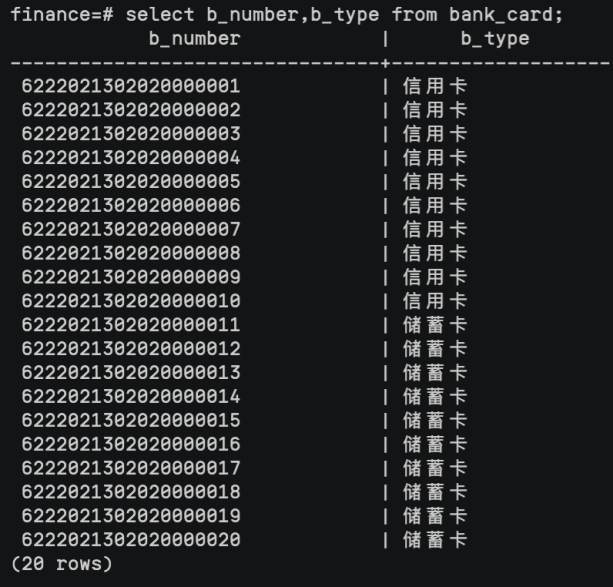


1. 对于1.1.7中的每个查询需求，提供对应的SQL查询语句和能够满足查询需求的关系代数表达式
2. 单表查询：查询银行卡信息表

语句：select b\_number,b\_type from bank\_card;

关系代数式：

截屏2023-05-04 下午12.17.53

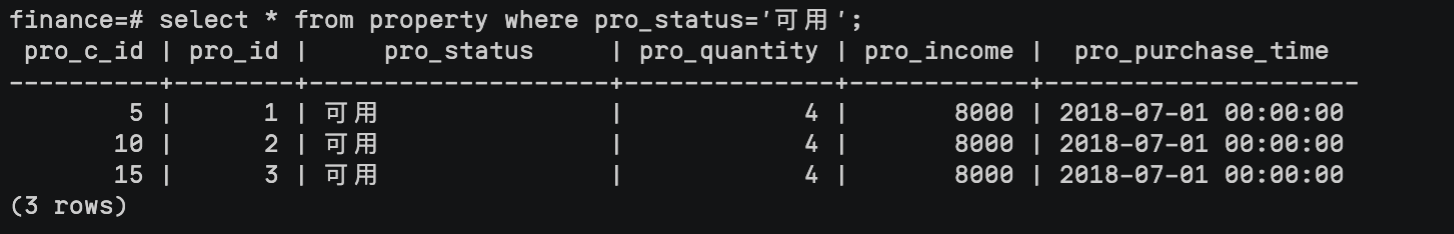


1. 条件查询：查询资产信息中‘可用’的资产数据

语句：select \* from property where pro\_status='可用';

关系代数式：

截屏2023-05-04 下午12.16.18

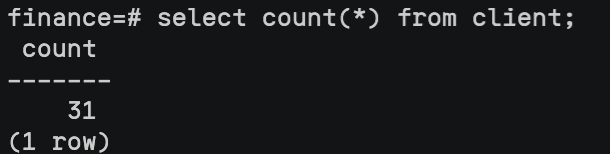


1. 聚合查询：查询用户表中有多少个用户

语句：select count(\*) from client;

关系代数式：

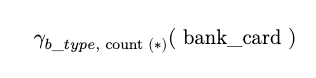
截屏2023-05-04 下午12.21.51



1. 聚合查询：查询银行卡信息表中，储蓄卡和信用卡的个数

语句：select b\_type,count(\*) from bank\_card group by b\_type;

关系代数式：



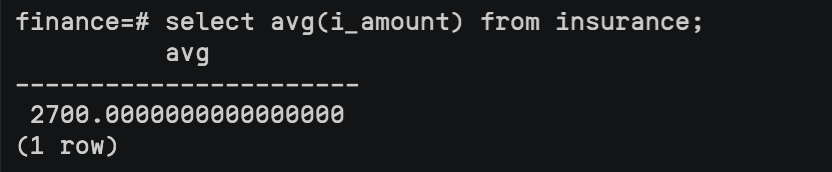


1. 聚合查询：查询保险信息表中，保险金融的平均值

语句：select avg(i\_amount) from insurance;

关系代数式:

截屏2023-05-04 下午4.25.16

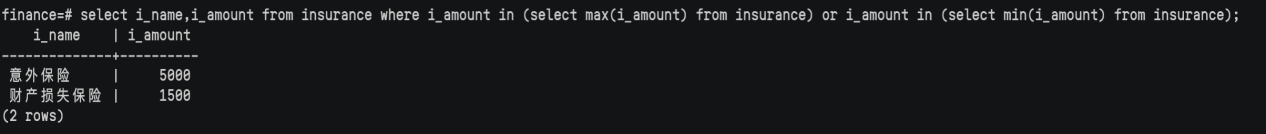


1. 聚合查询：查询保险信息表中保险金额的最大值和最小值所对应的险种和金额

语句：select i\_name,i\_amount from insurance where i\_amount in (select max(i\_amount) from insurance) or i\_amount in (select min(i\_amount) from insurance);

关系代数式：

P49VLTHEmu



1. 连接查询：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

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

关系代数式：

截屏2023-05-04 下午4.49.27



1. 连接查询：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

语句：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\_');

关系代数式：

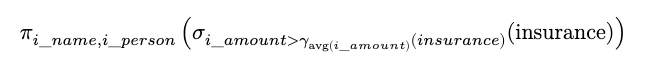
截屏2023-05-09 下午12.02.12

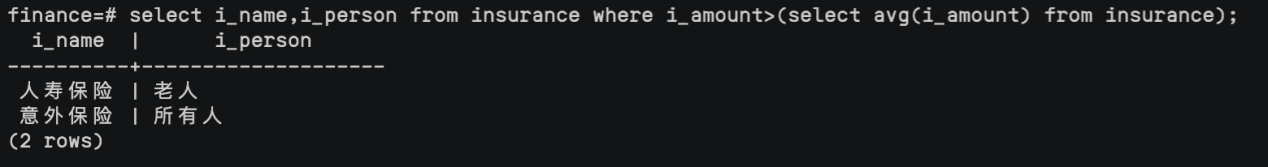


1. 子查询：通过子查询，查询保险产品中保险金额大于平均值的保险名称和适用人群

语句：select i\_name,i\_person from insurance where i\_amount>(select avg(i\_amount) from insurance);（由于题目中没有写要查询保额，所以我选择不查询保额，这会与结果有差别，但这是无关紧要的。）

关系代数式：

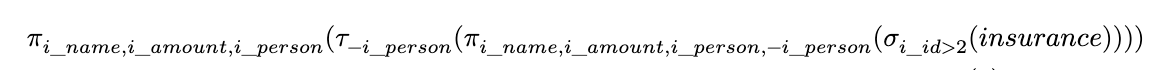


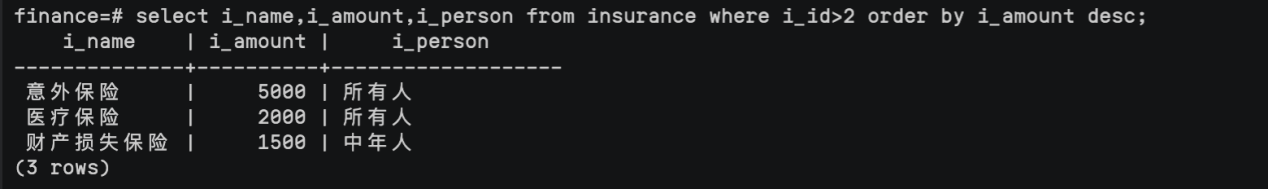


1. ORDER BY子句：按照降序查询保险编号大于2的保险名称、保额和适用人群

语句：select i\_name,i\_amount,i\_person from insurance where i\_id>2 order by i\_amount desc;

关系代数式：

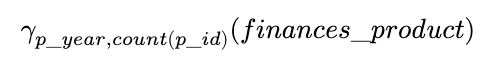




1. GROUP BY子句：查询各保险信息总数，按照p\_year分组

语句：select p\_year,count(p\_id) from finances\_product group by p\_year;

关系代数式：



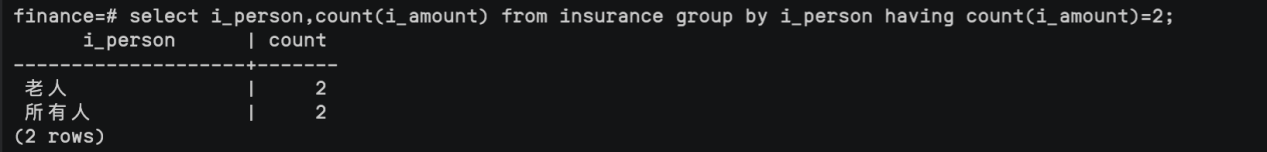


1. HAVING子句：查询保险金额统计数量等于2的适用人群

语句：select i\_person,count(i\_amount) from insurance group by i\_person having count(i\_amount)=2;

关系代数式：



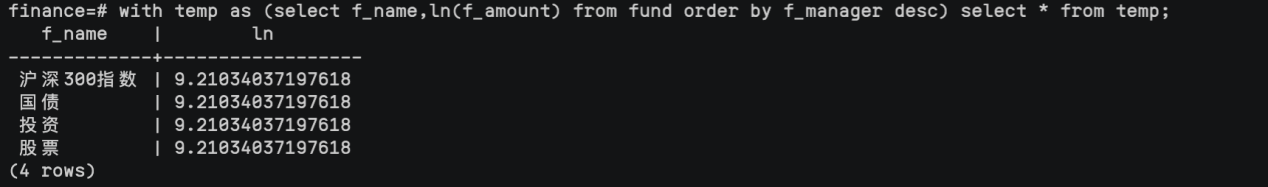


1. WITH AS子句：使用WITH AS查询基金信息表

语句：with temp as (select f\_name,ln(f\_amount) from fund order by f\_manager desc) select \* from temp;

关系代数式：

截屏2023-05-09 下午1.43.37



1. 初始SQL执行结果和要求的执行结果不符原因

1）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

不符：没有查询保额的信息

原因：题干中没有要求要查询到保额信息，所以select关键字后没有加入i\_amount的查询。加入后即可。

1. 聚合查询：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

不符：查询结果是相同的，但是所用的语句有不同。

原因：原句中用了union联合查询，将两个查询结果合并成一个结果集，我自己用的是or关键字分别查询到max和min所对应的目标信息。两者都是正确的查询语句。

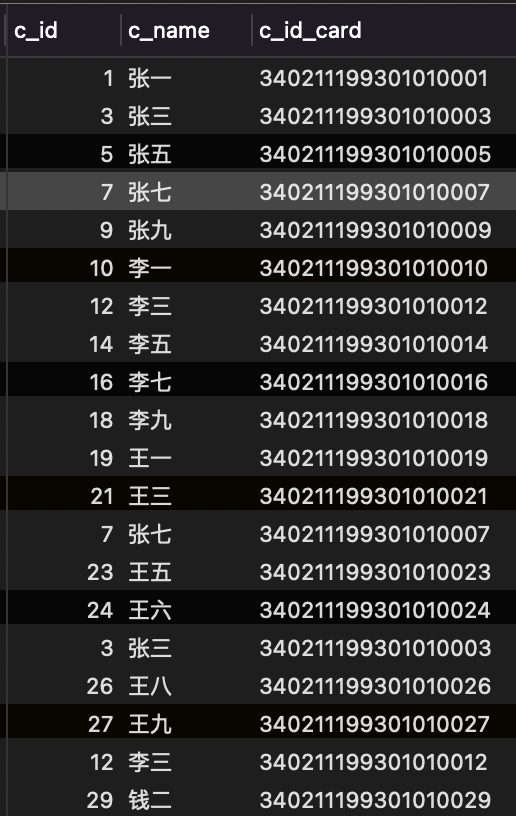
1. WITH AS子句：使用WITH AS查询基金信息表

不符：原来不会这种写法

原因：这个SQL语句从表"fund"中选择基金的名字（f\_name）和基金金额的自然对数（ln(f\_amount)），并按照基金经理（f\_manager）的降序排列结果。查询结果暂时存储在一个名为"temp"的临时表中。最后，返回临时表中的所有记录。这个查询的特点是使用了子查询和临时表，将多个操作组合在一起，可以简化查询语句并提高查询效率。

1. 连接查询：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。

不符：查询结果不一致，结果如下图：



原因：我这里用的语句是：select client.c\_id, client.c\_name, client.c\_id\_card

from client left outer join bank\_card on client.c\_id = bank\_card.b\_c\_id

where bank\_card.b\_c\_id is not null;这里出现的问题经过我对照client表和bank\_card表的元组发现了问题，原因是在bank\_card表上存在重复元组，而在连接的时候没有考虑到这种情况，没有过滤掉重复的元组，于是将重复的元组也select到了，产生了错误。所以又修改成了利用exists来写查询语句。

1. 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

**实验总时长**：4小时左右

我学习了很多关于数据库的基础知识和操作，包括创建表、插入数据、添加约束、查询数据、创建视图、创建索引、修改和删除数据、创建用户和授权、以及连接数据库等。这些操作涵盖了数据库的多个方面，帮助我更好地理解数据库的概念、特点和使用方法。在关系代数式的写法上耗费了较多的时间，也存在不少的疑问，但在老师的帮助下，最后都确定了关系代数式的正确性。

**遇到的问题包括**：

数据类型和约束问题：在创建表时，需要仔细考虑每个属性的数据类型、长度、约束等信息，否则可能会导致后续操作出现问题。例如，如果定义一个属性为整数类型，而实际插入的数据是字符型，就会出现数据类型不匹配的错误。

外键关系问题：在插入数据时，需要特别注意数据的完整性，例如对应外键关系的数据必须先插入到对应的表中。否则，就会出现无法插入数据的情况。

SQL语法问题：在编写查询语句时，需要熟悉SQL语法和常用操作符，否则可能会出现语法错误或者结果不正确的情况。例如，在使用WITH AS操作时，需要注意使用方法的正确性，否则可能会出现查询结果不正确的情况。在上文的问题分析中具体说明了这些问题

在学习这个实验的过程中，学到了很多关于数据库的**知识点**包括：

数据库的基本概念、术语、特点和分类。

关系型数据库的设计原则、范式和关系模型。

SQL语法和常用操作符，包括SELECT、FROM、WHERE、ORDER BY、WITH AS等。

数据库的表、视图、索引、约束等基本操作，以及使用ALTER TABLE命令修改表结构。

数据库用户和授权，包括CREATE USER、GRANT等命令，以及使用REVOKE命令收回权限。

对关系代数表达式的掌握更加熟练，运用也更加自如，了解了多种关系代数运算：union、intersection、difference、product、selection、projection、多种join、还有一些扩展关系代数（扩展投影等）