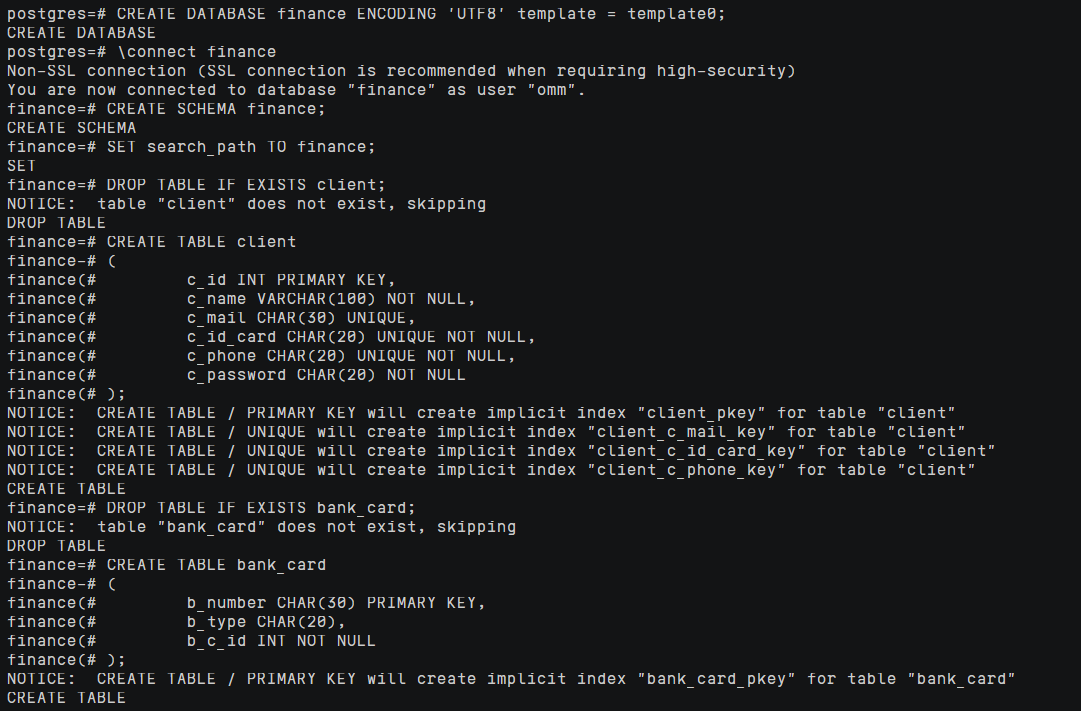
**场景化综合实验报告**

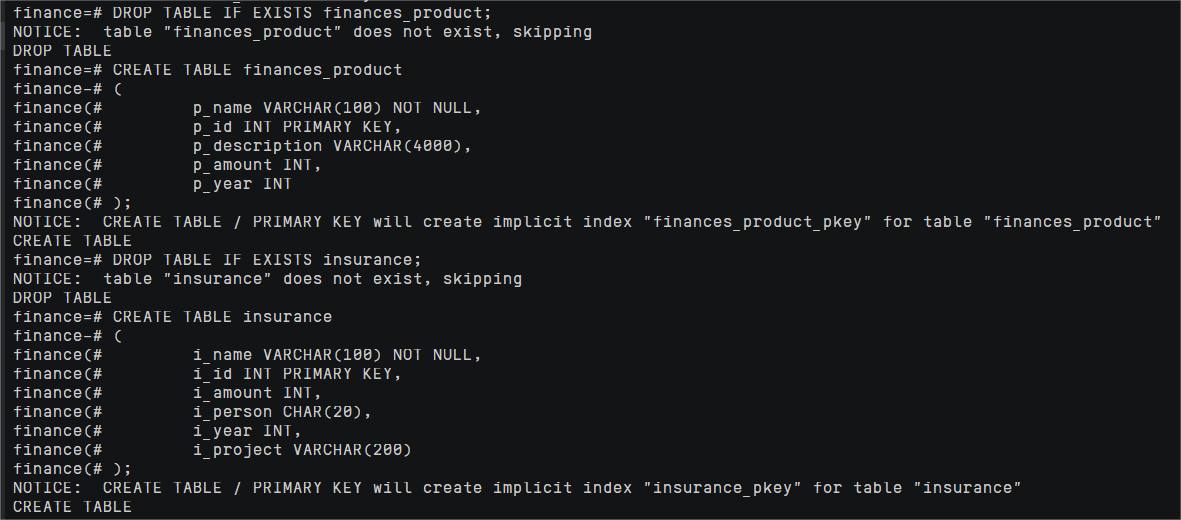
1. **实验环境说明**

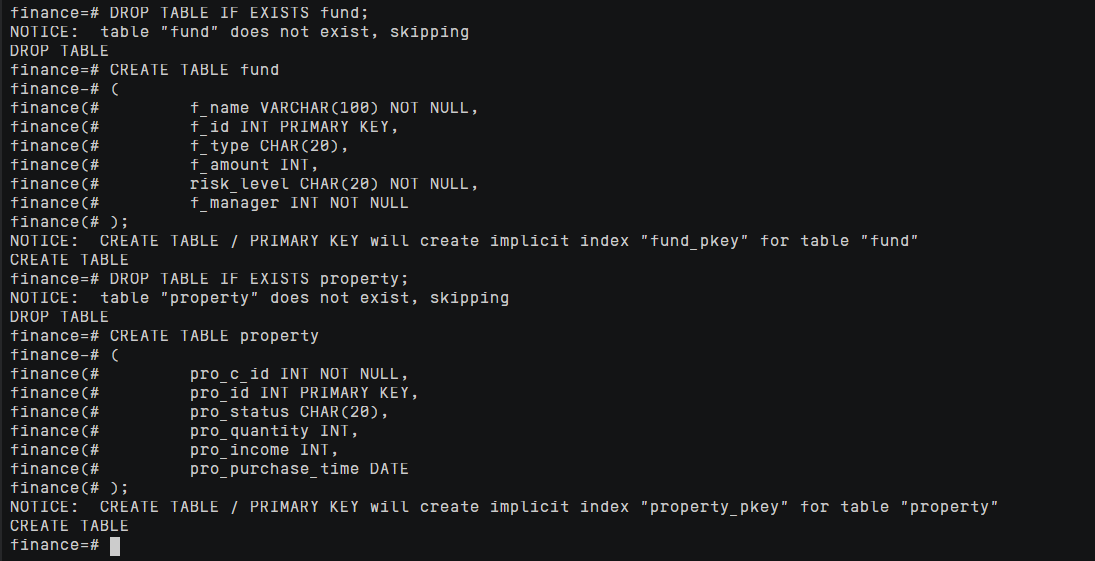
本实验是在金融场景下进行的，基于openGauss开始的实验。选择openGauss是因为openGuase基于多核架构的并发控制技术、NUMA-Aware存储引擎、SQL-Bypass智能选路执行技术，释放处理器多核扩展能力，实现两路鲲鹏128核场景150万tpmC性能支持RTO<10S的快速故障倒换，全链路数据保护，满足安全及可靠性要求通过智能参数调优、慢SQL诊断、多维性能自监控、在线SQL时间预测等能力，让运维由繁至简，并且之前一直使用更熟悉其环境和使用过程。

1. **执行结果截图**

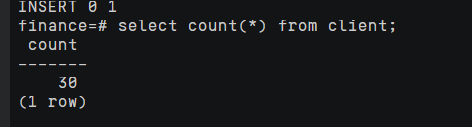
**1.1.3**

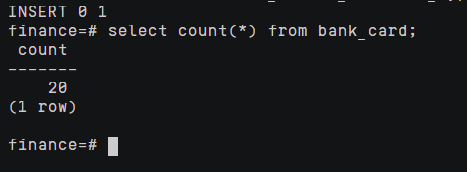


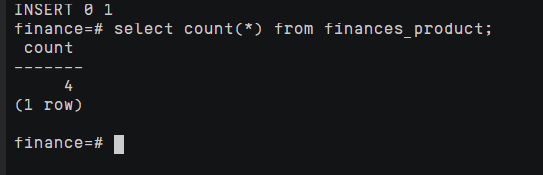


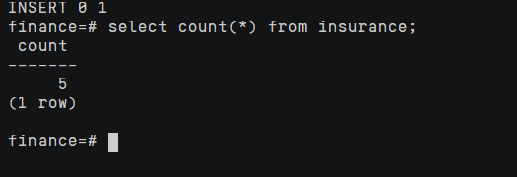


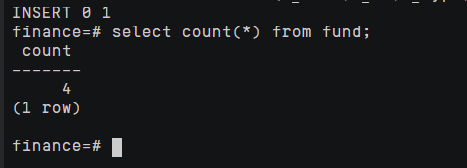
**1.1.4**

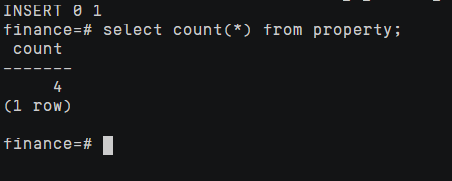




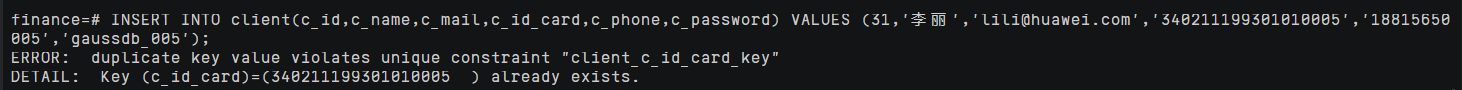








**1.1.5插入失败示例如下：**



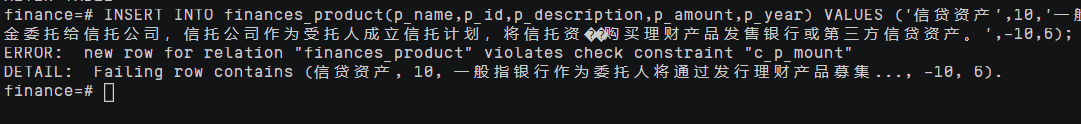
**插入成功示例如下：**



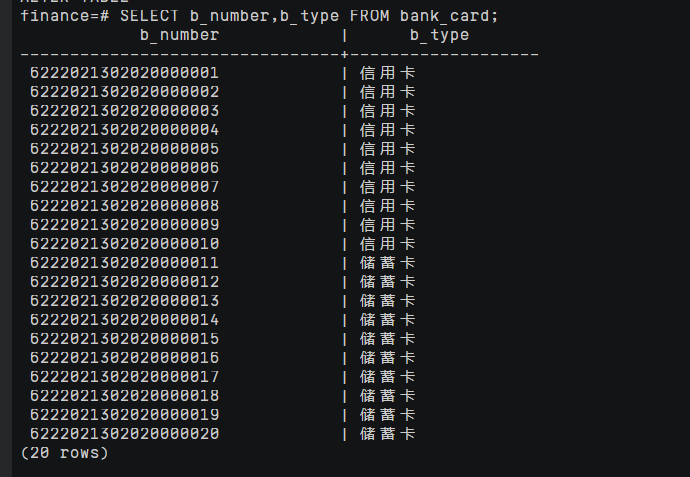
**1.1.6添加约束：**

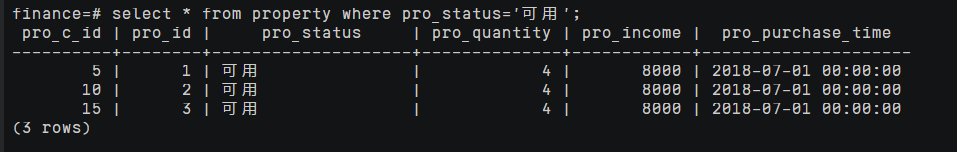


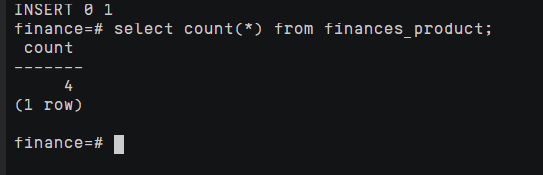
**添加约束后插入不合法的数据报错如下：**

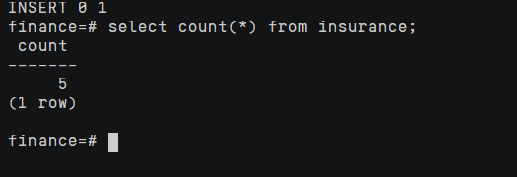


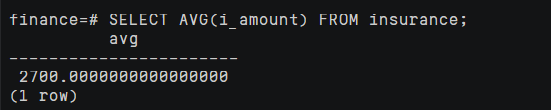
**1.1.7查询数据如下**

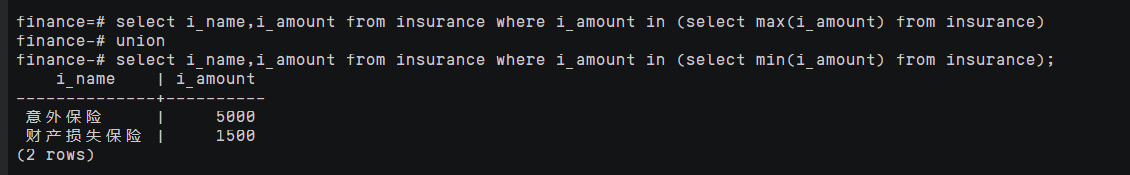
****

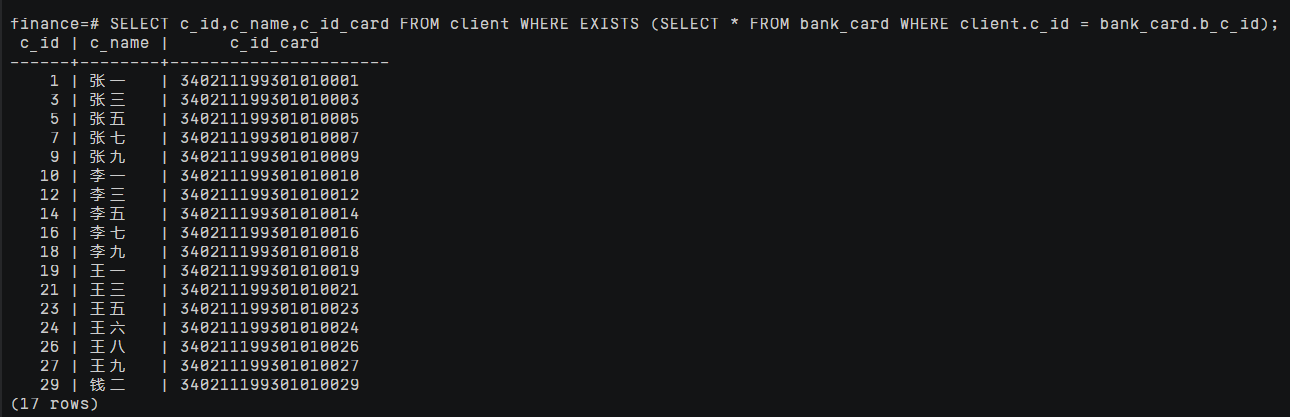
****

****

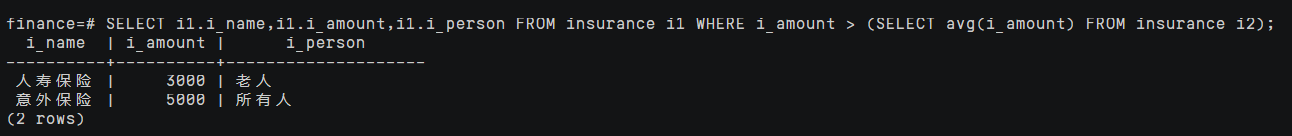
****

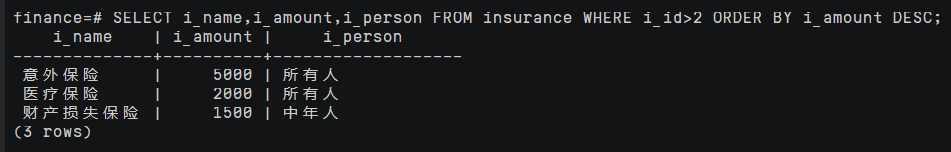
****

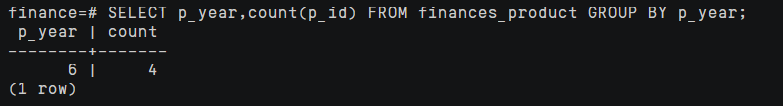
****

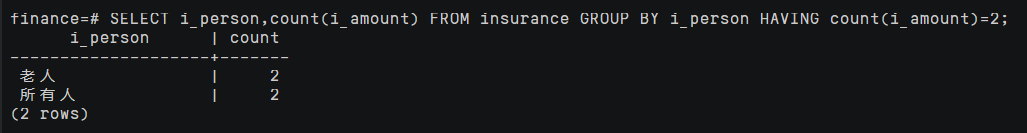
****

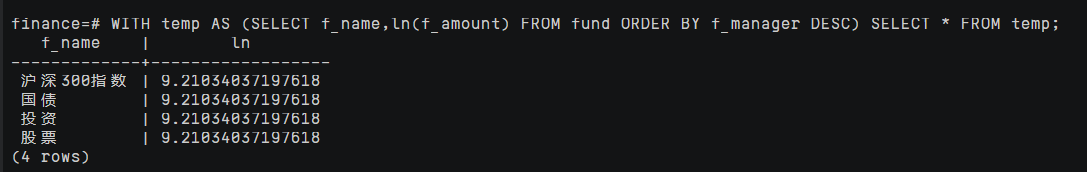
****

****

****

****

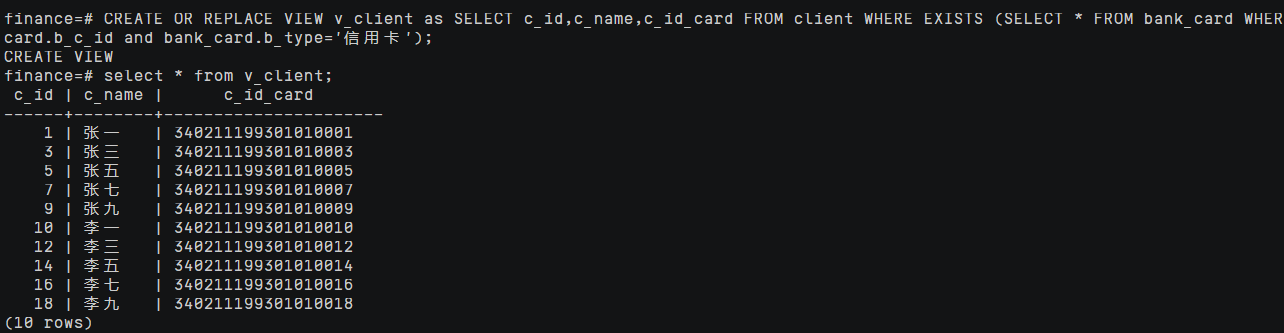
****

****

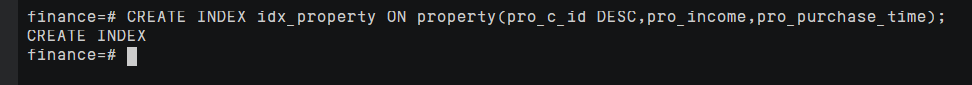
**1.1.8创建视图并查询**



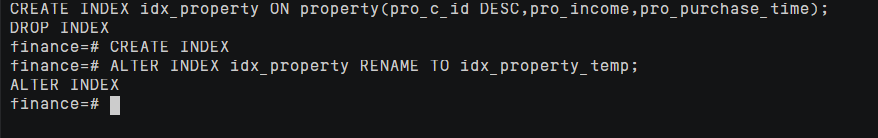
**修改视图，在原有查询的基础上，过滤出信用卡用户并进行查询**



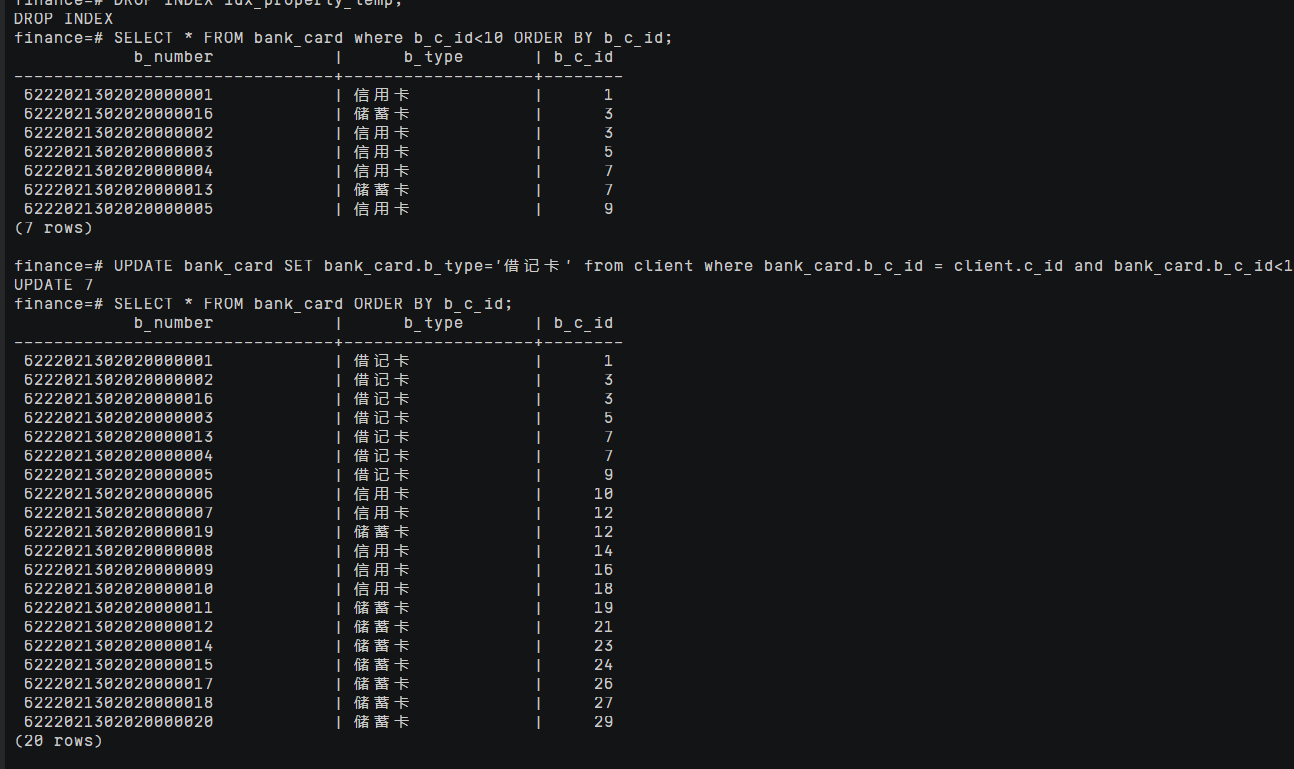
**1.1.9 在普通表property上创建索引。**



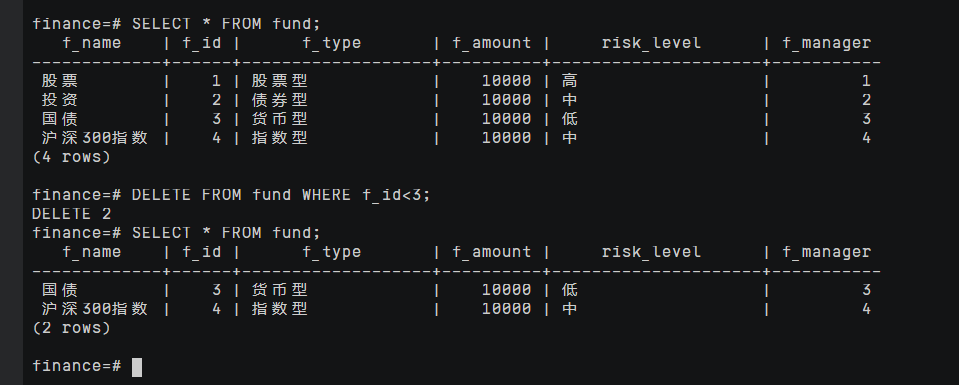
**重建索引并重命名索引**



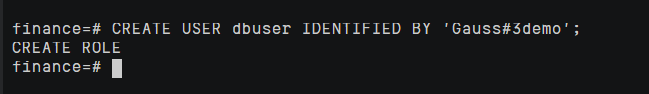
**1.1.10 对表数据进行更新**



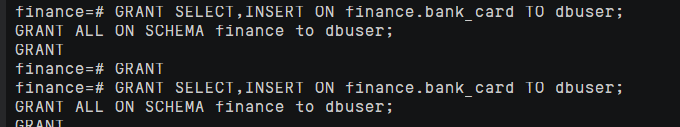
**删除指定数据**



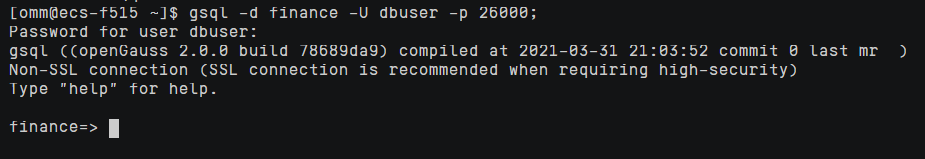
**1.1.11 创建新用户，密码为Gauss#3demo**



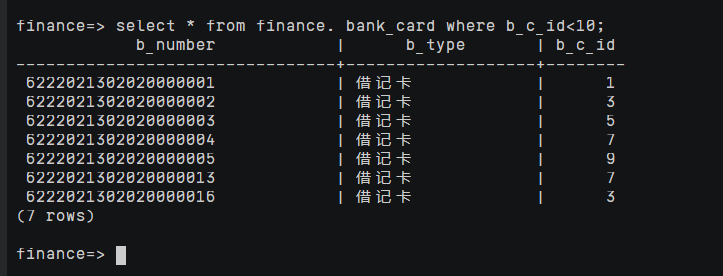
**赋予权限**



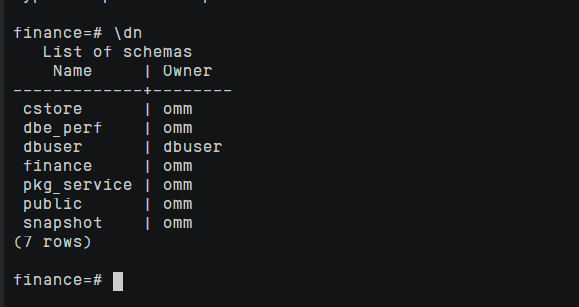
**1.1.12 用新用户登录finance数据库**



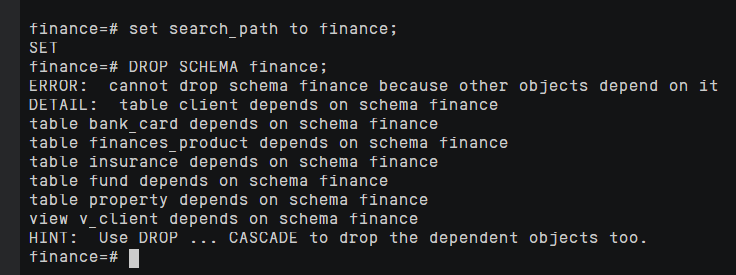
**访问finance数据库的表bank\_card。**



**1.1.13 使用“\dn”查看数据库下的schema。**



**使用DROP SCHEMA 命令删除finance报错如下：**



1. **sql语句和关系代数表达式**

1.查询银行卡信息表

SELECT b\_number,b\_type FROM bank\_card;

2.查询资产信息中‘可用’的资产数据。

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

3. 查询用户表中有多少个用户

SELECT count(\*) FROM client;

4. 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;

5. 查询保险信息表中，保险金额的平均值

SELECT AVG(i\_amount) FROM insurance;

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 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);

7．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);

8. 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\_');

9.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 i\_name,i\_amount,i\_person FROM insurance WHERE i\_amount > (SELECT avg(i\_amount) FROM insurance);

10.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;

11.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;

12.查询保险金额统计数量等于2的适用人群数。

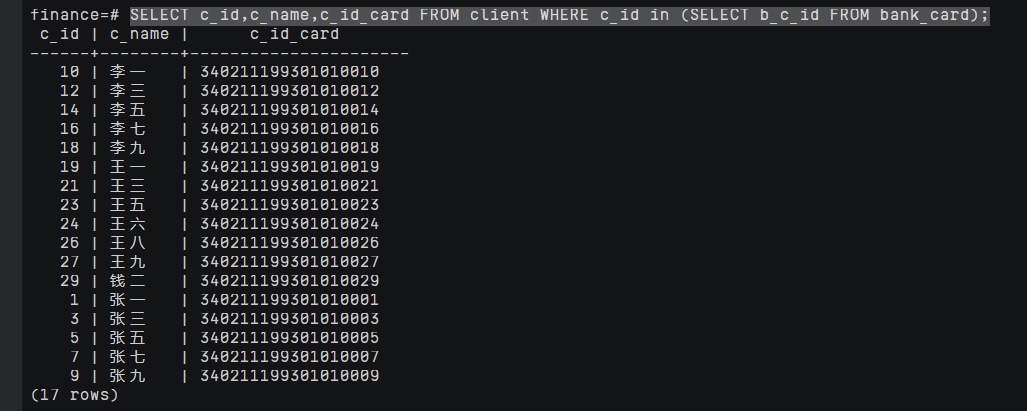
SELECT i\_person,count(i\_amount) FROM insurance GROUP BY i\_person HAVING count(i\_amount)=2;

13.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;

1. **与标准结果不符的场景**

1.在1.1.7.4中使用半连接查询使用的方法是在where后面用in子查询实验，而不是where后面跟exists查询，使用的查询语句为：SELECT c\_id,c\_name,c\_id\_card FROM client WHERE c\_id in (SELECT b\_c\_id FROM bank\_card); 但查询结果相同。



1. **总结**

本实验共花费四个小时。自身在部分查询操作如WITH AS查询和EXISTS查询的掌握和使用还很薄弱，对添加约束也很不熟悉,以及在写代数表达式方面很艰难。

本次实验使我在查询、更新、创建视图等方面更加熟练，并且还了解了半连接、反连接等理论知识，之后也会在自己薄弱的方面多加练习，并且牢记理论基础。