场景化综合实验(金融场景)

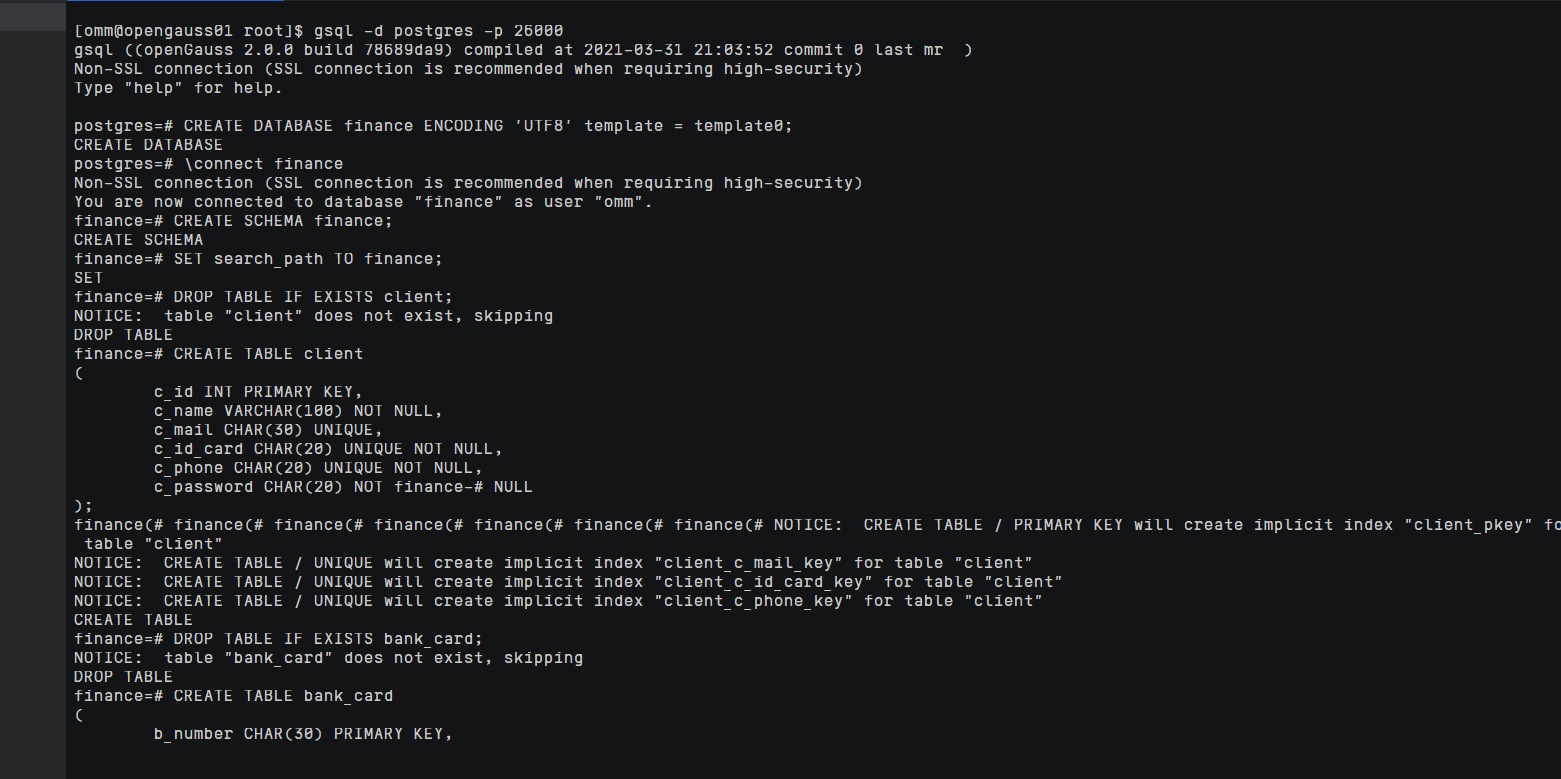
姓名： 齐明杰 学号： 2113997

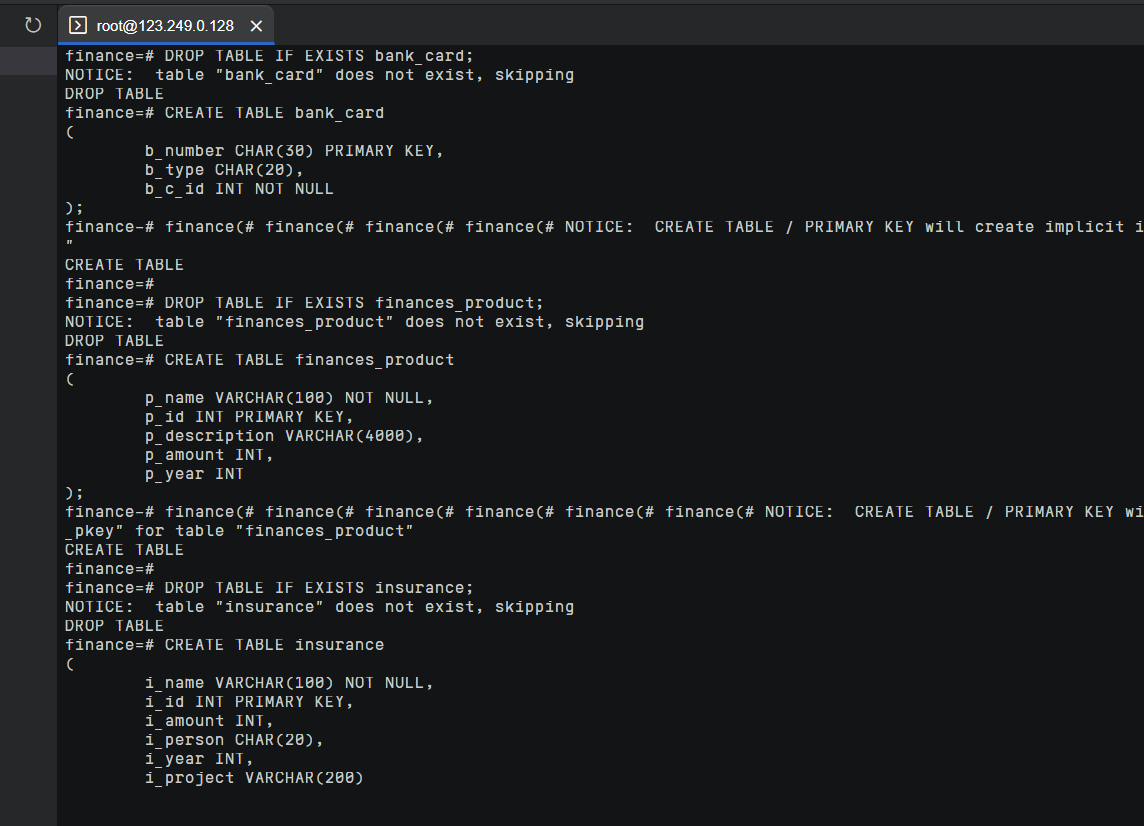
1. **实验环境说明**

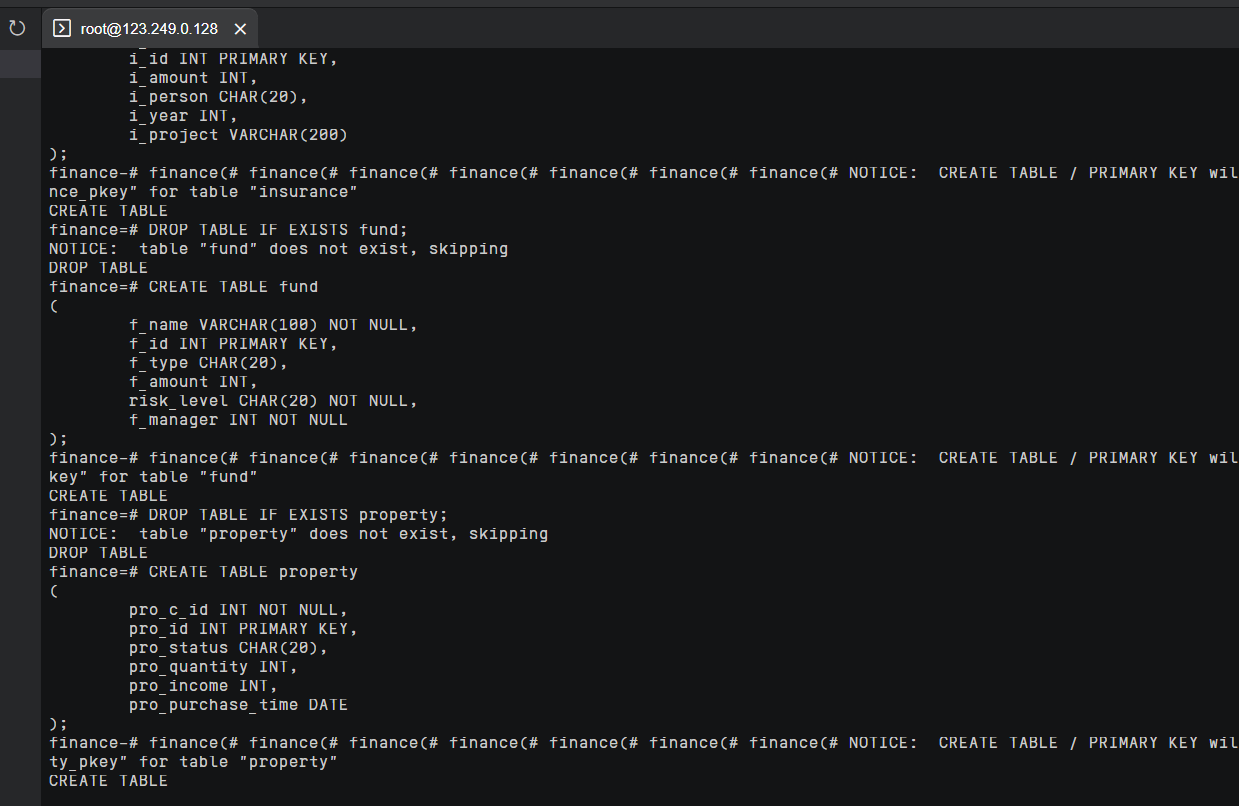
本次实验在 openGauss 数据库管理系统环境下进行。openGauss 是一款开源的企业级关系型数据库管理系统，由华为公司发起并贡献，广泛应用于各种场景，如金融、互联网、大数据等。该数据库系统遵循 SQL 标准，同时具有高性能、高可用、高安全等特性。openGauss 具有出色的性能，对于复杂的查询和大型数据集处理具有较强的能力。这使得在实验过程中可以更好地了解如何优化查询性能，提高数据库系统的运行效率，因此我选择在openGauss中进行。

1. **1.1.3-1.1.13中完成主要步骤后的执行结果截图**

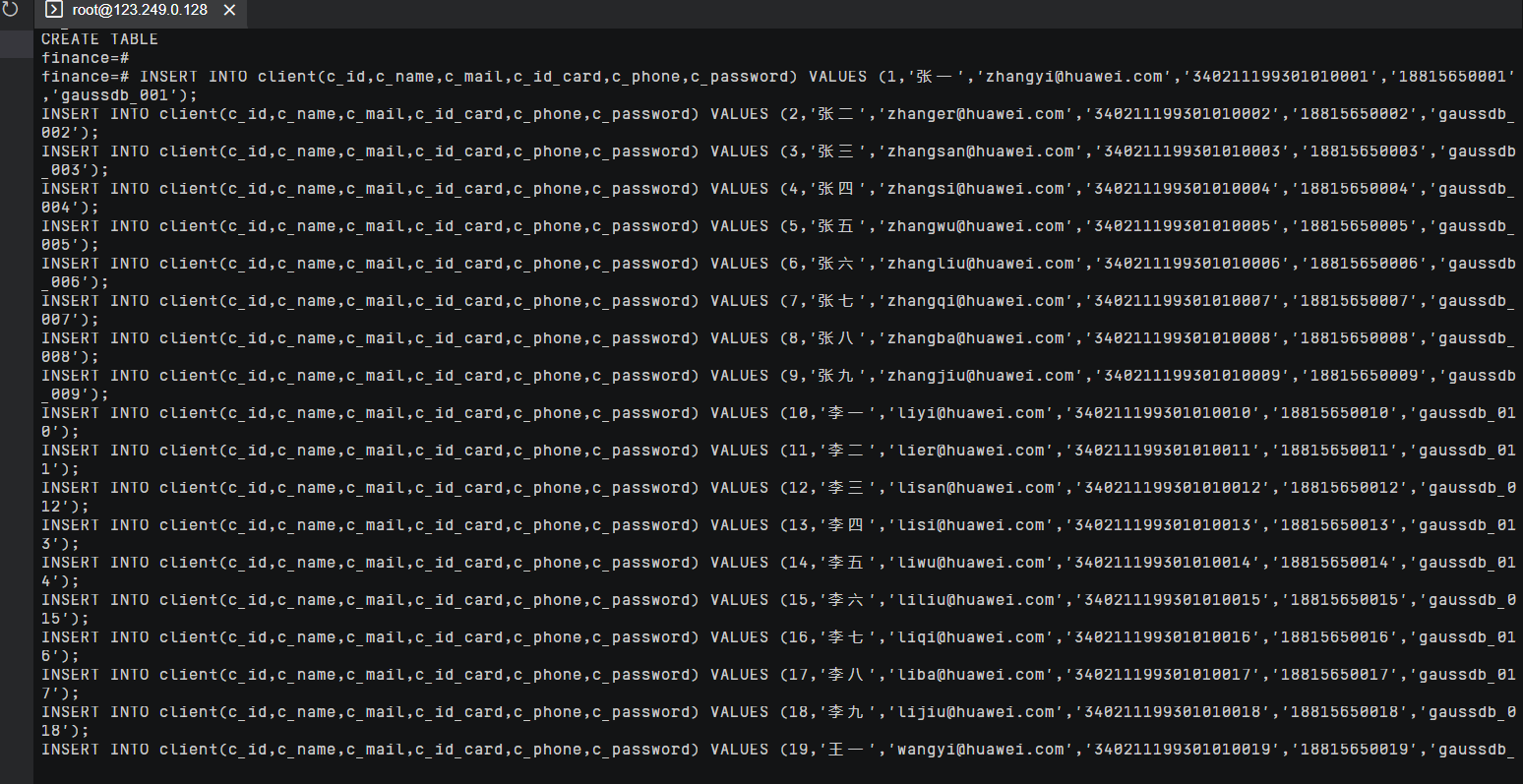
1.1.3

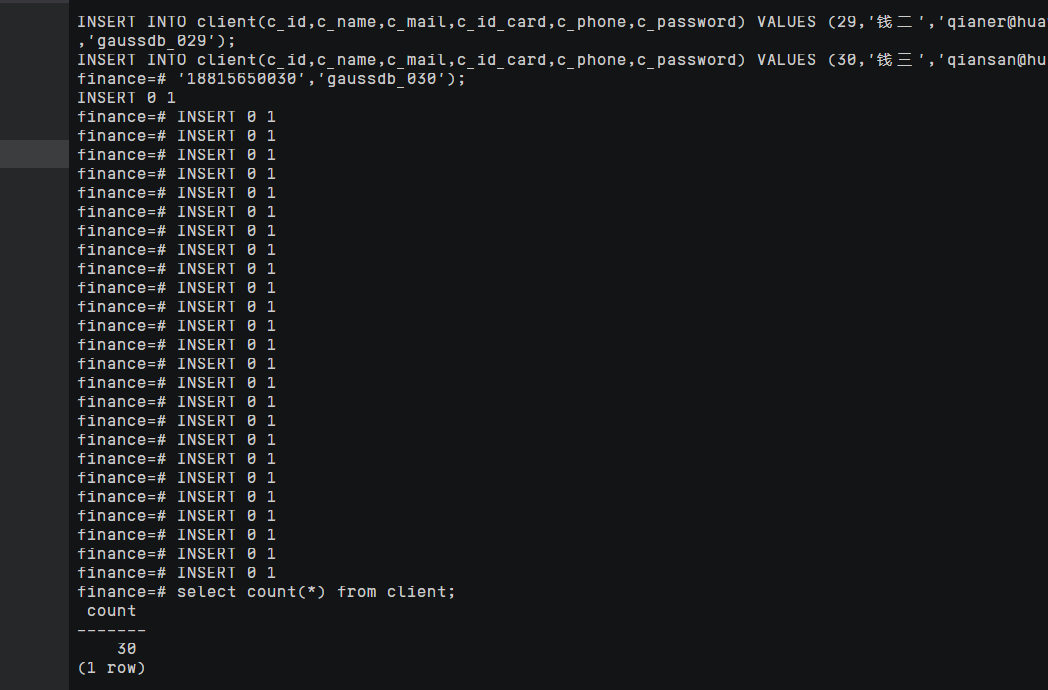


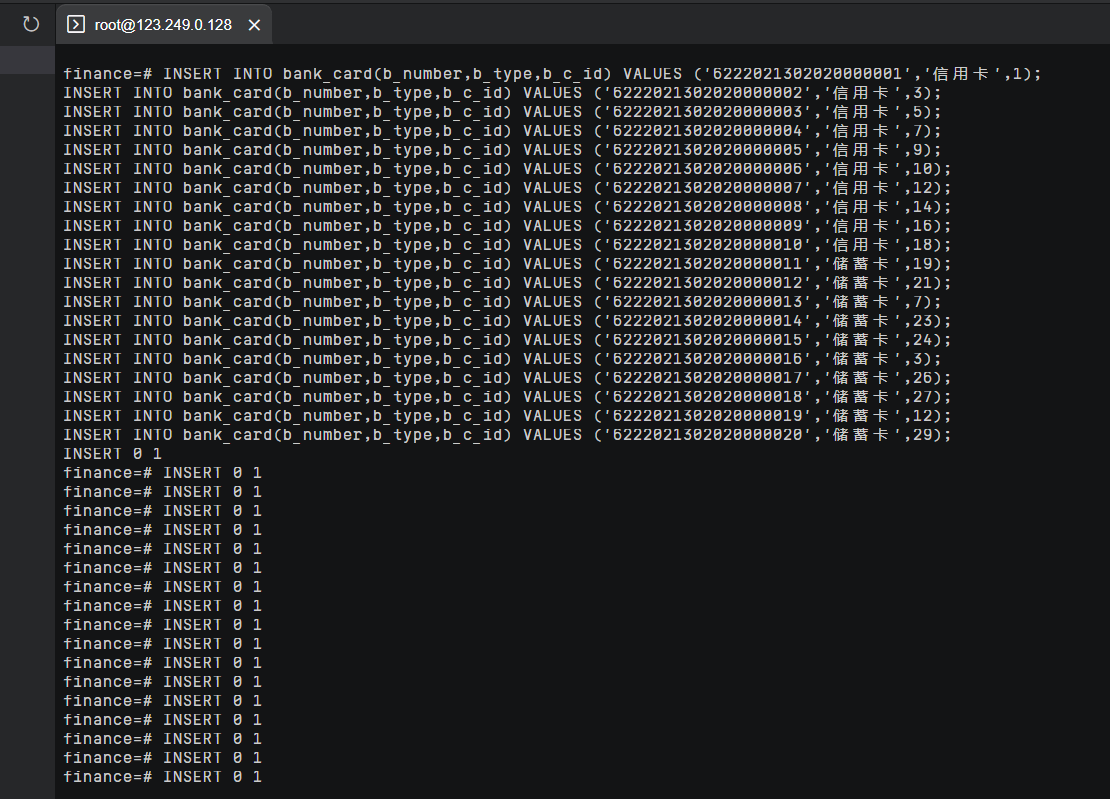


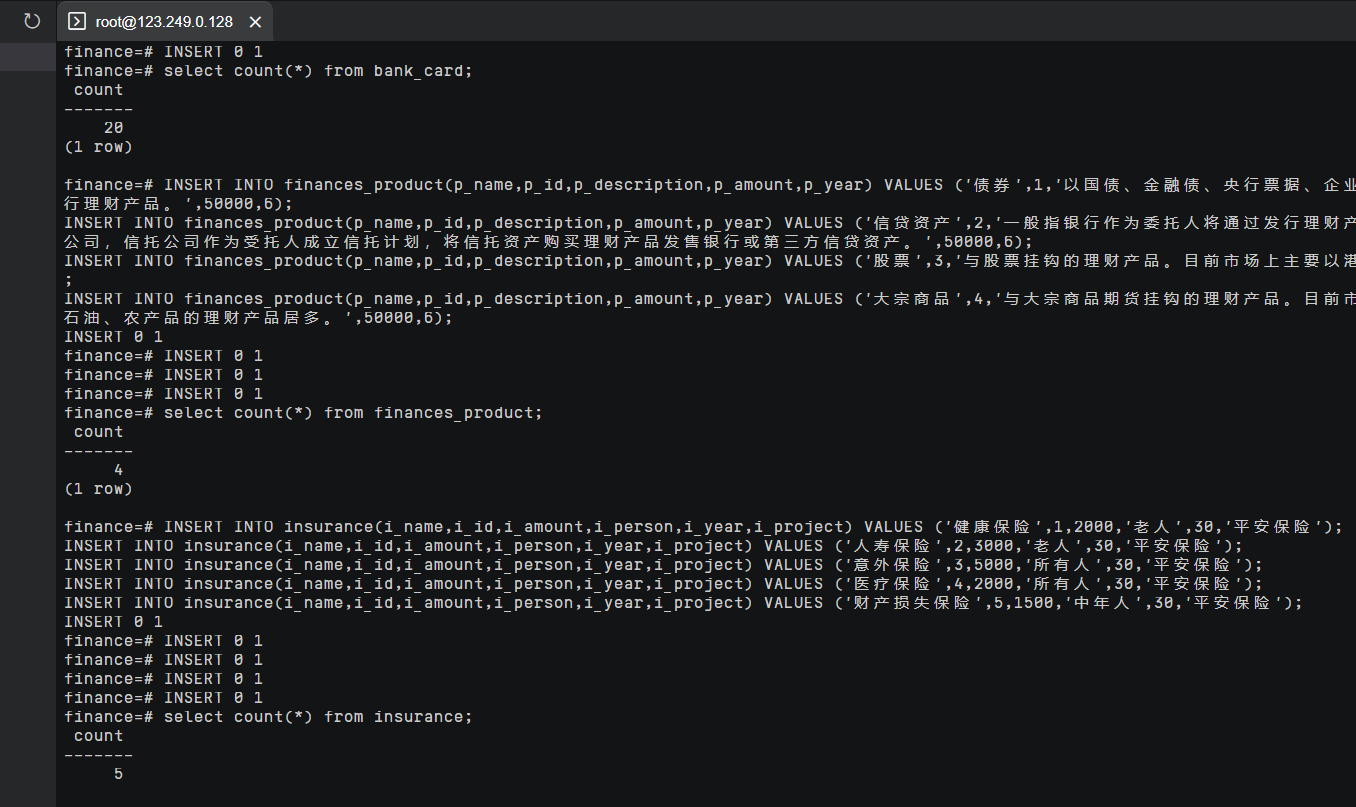


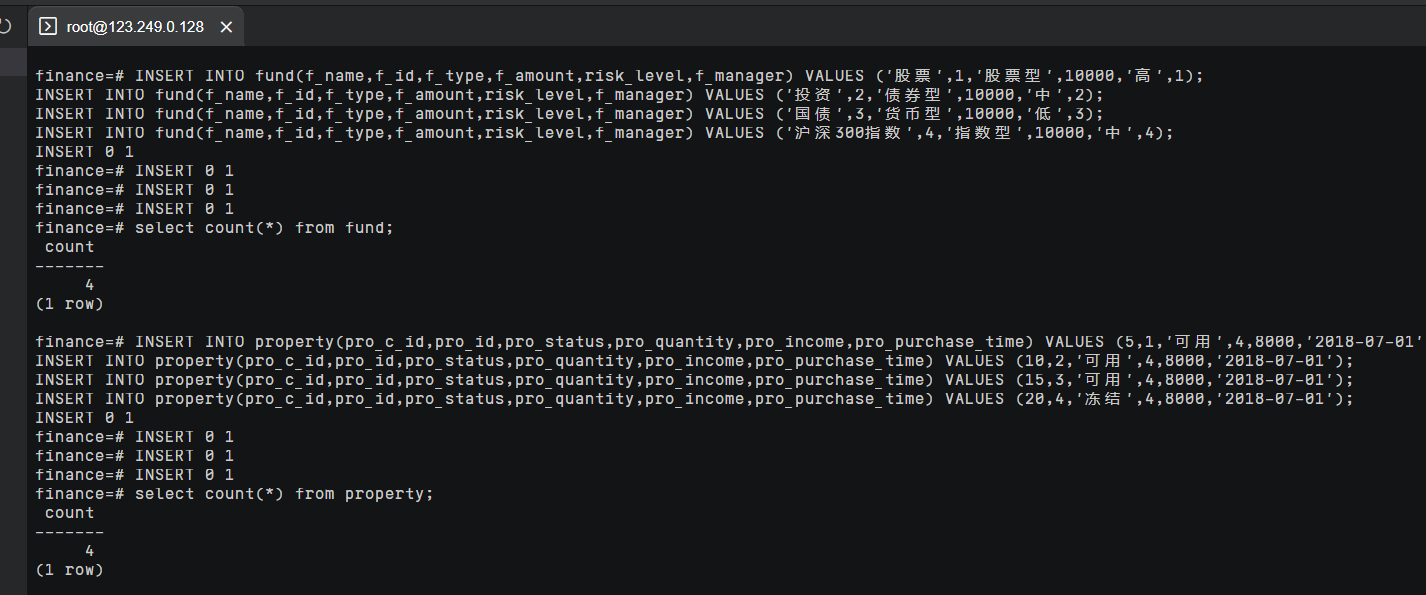
1.1.4



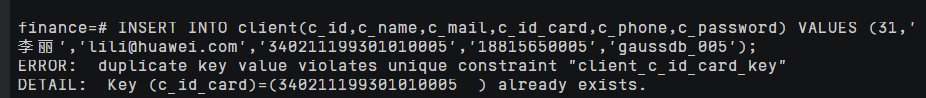


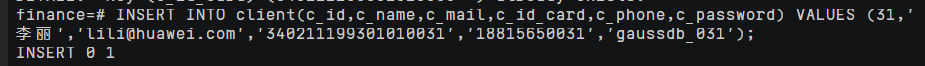




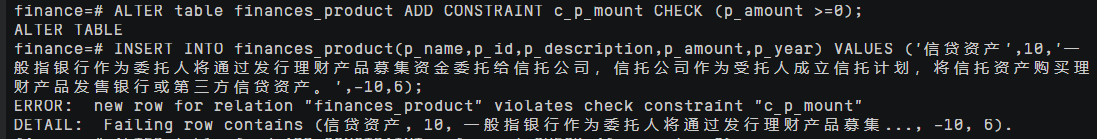


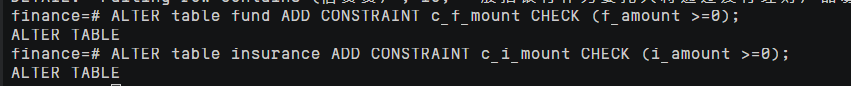
1.1.5



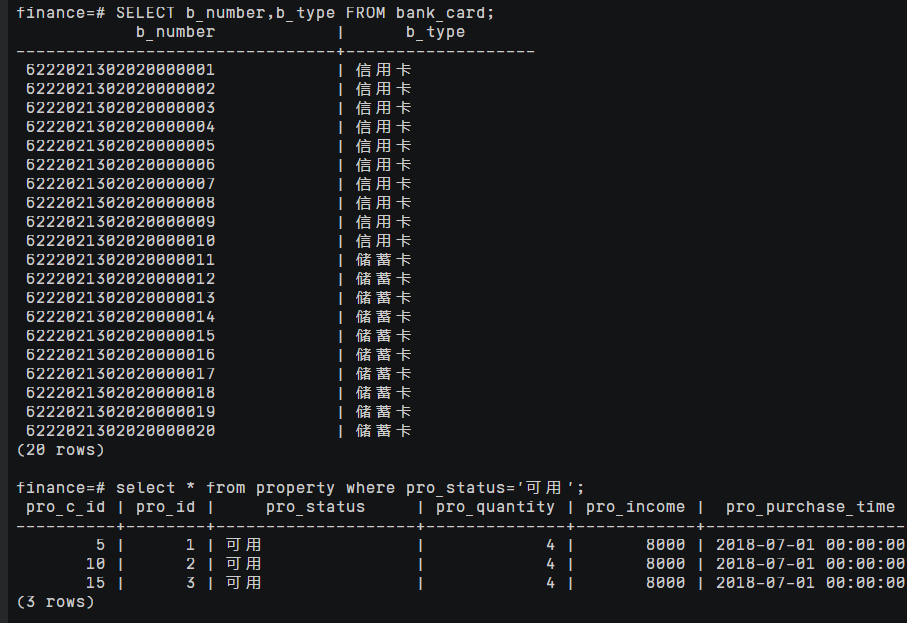


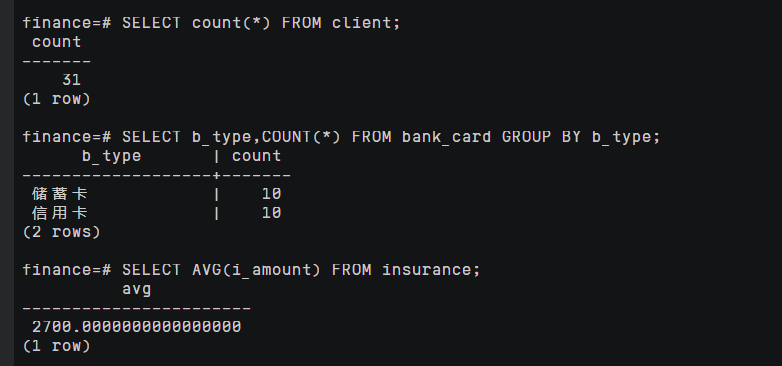
1.1.6

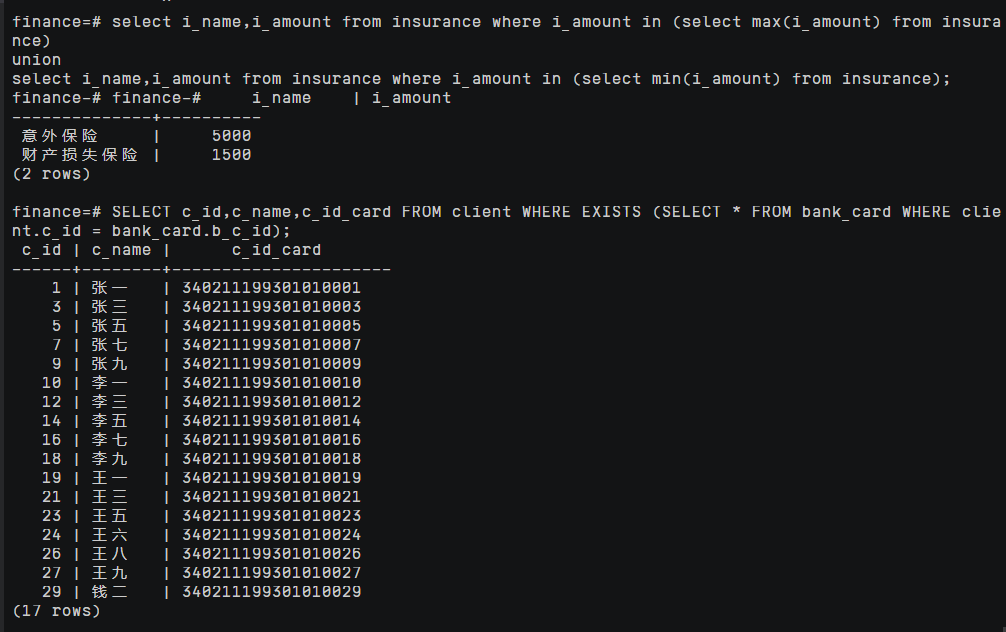


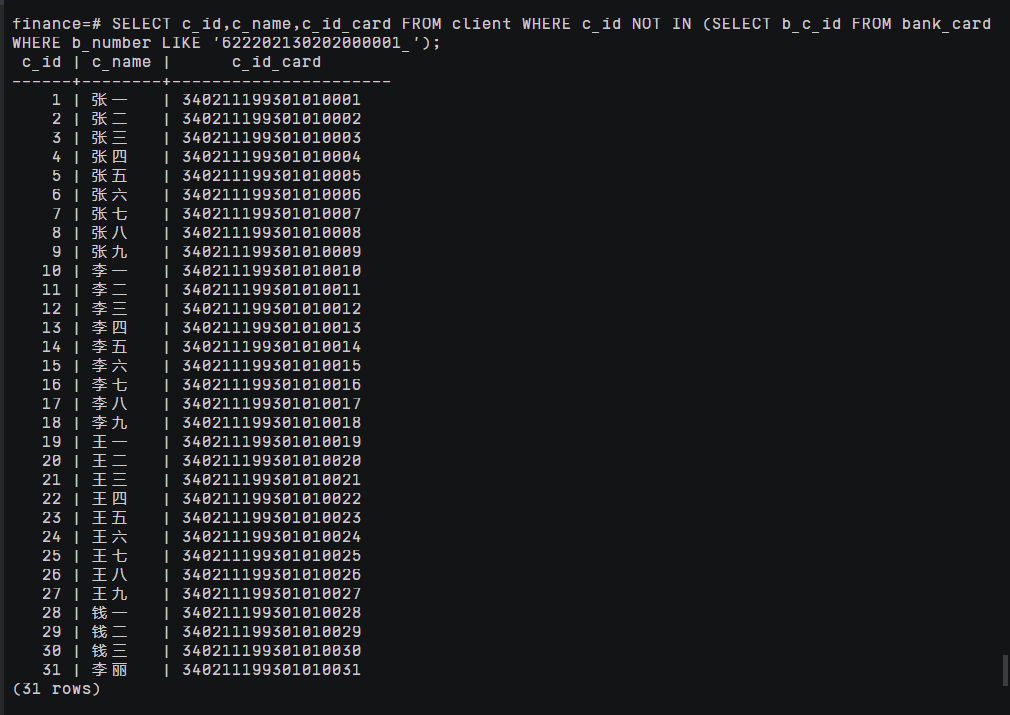


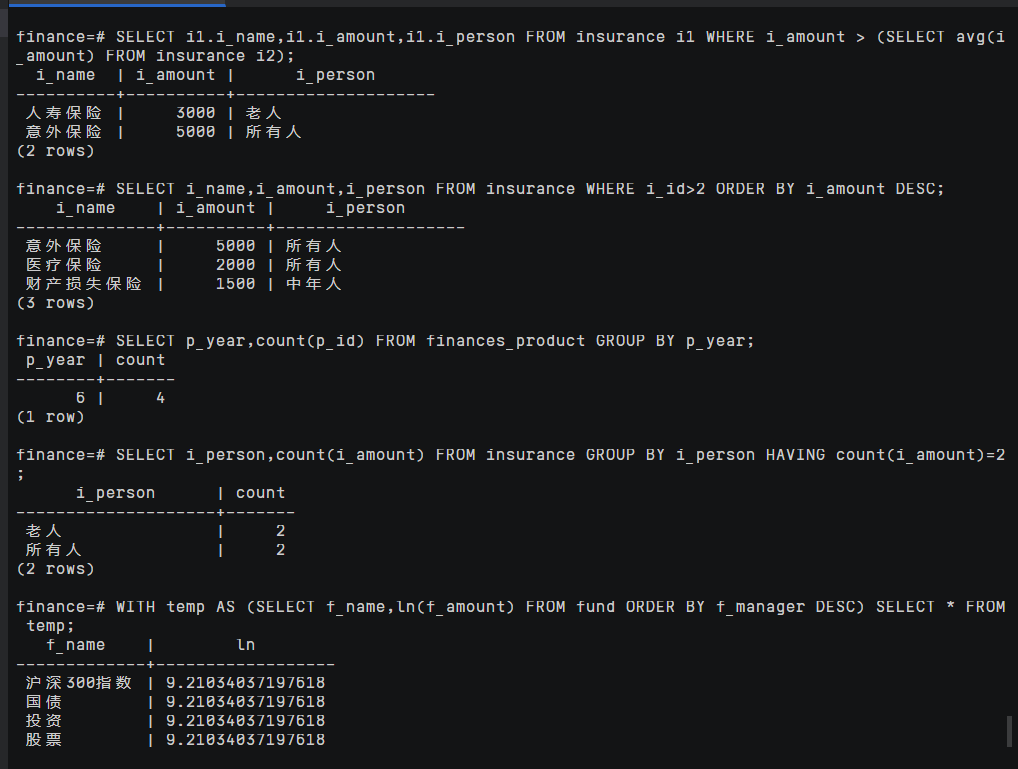
1.1.7



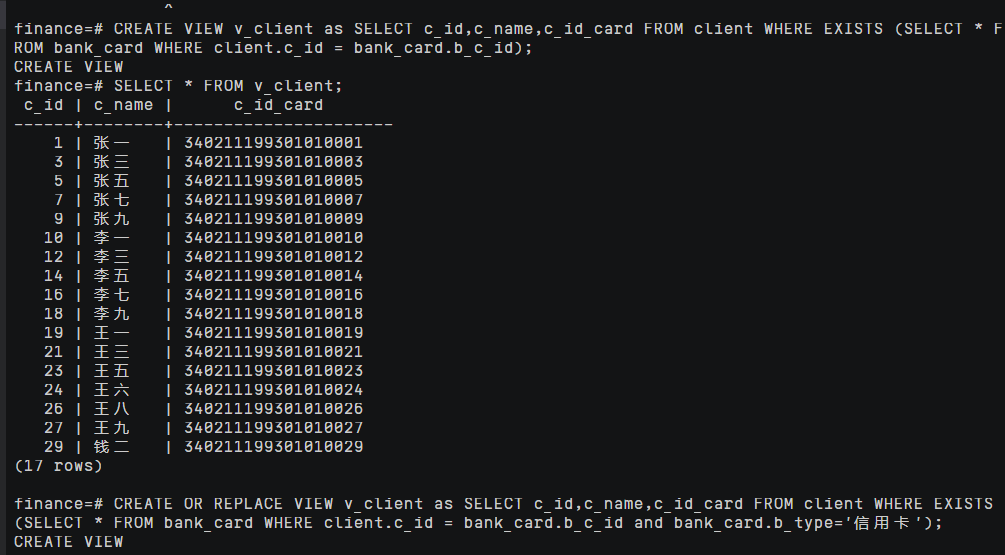


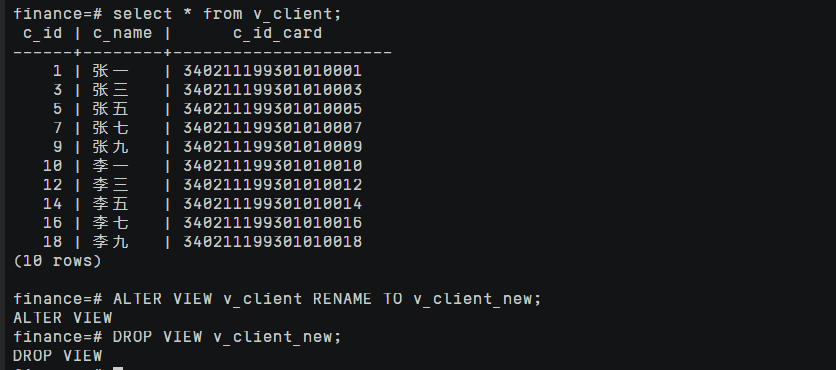




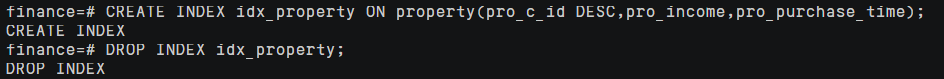


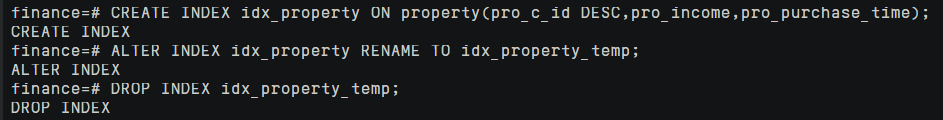
1.1.8



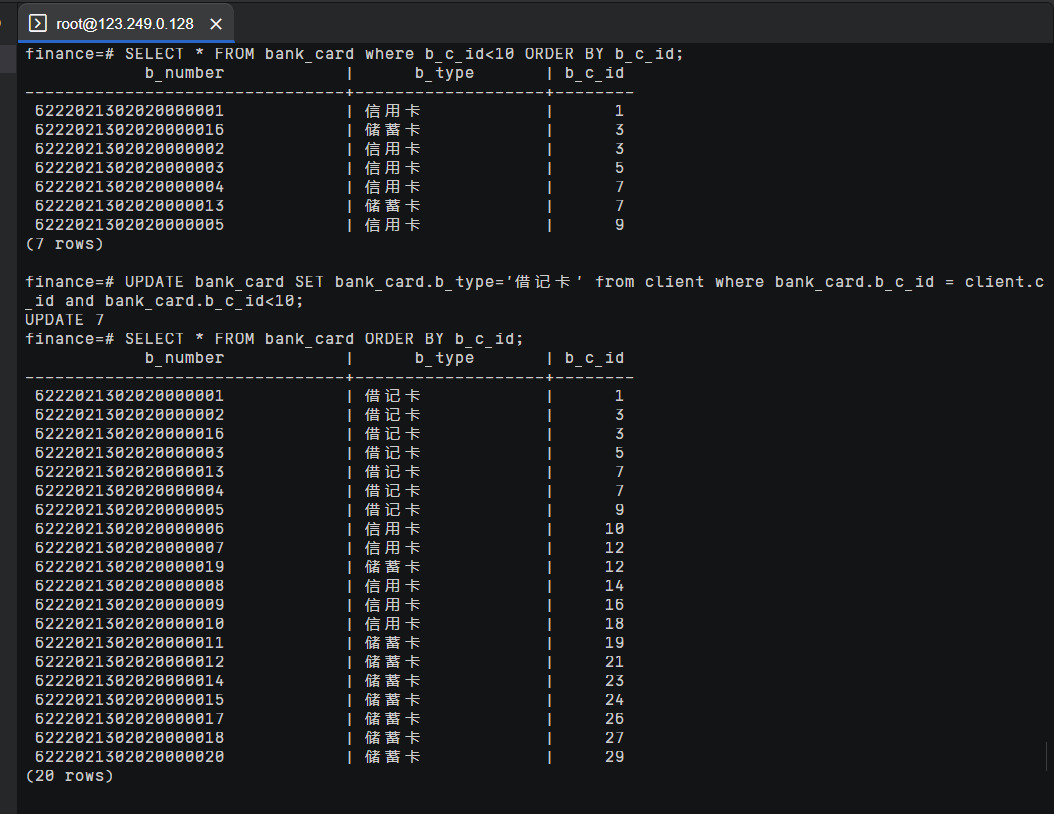


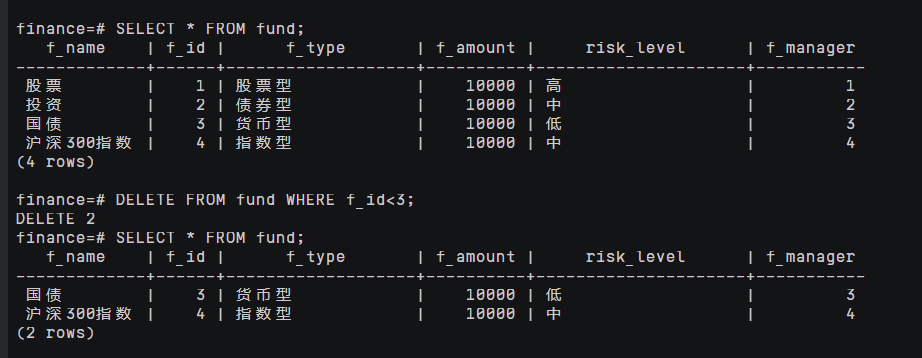
1.1.9





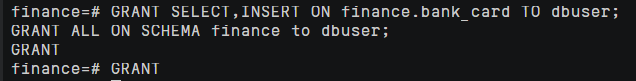
1.1.10



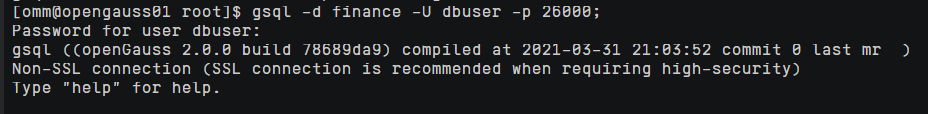


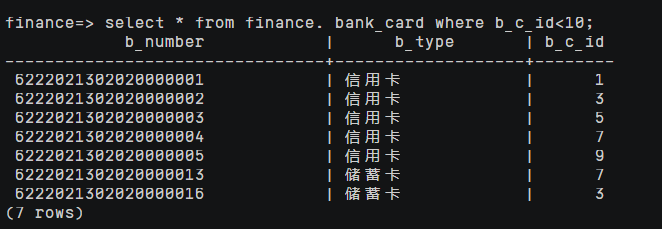
1.1.11



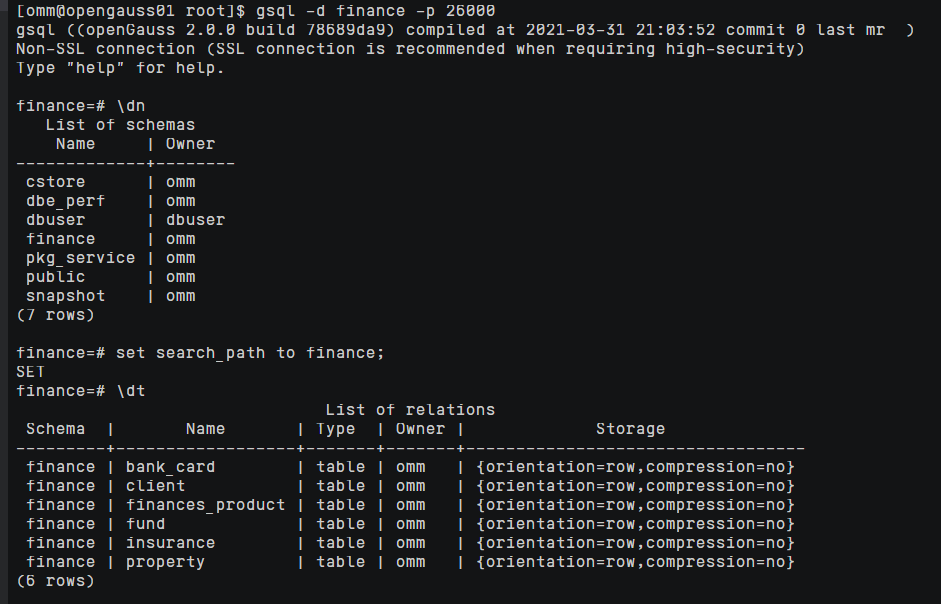


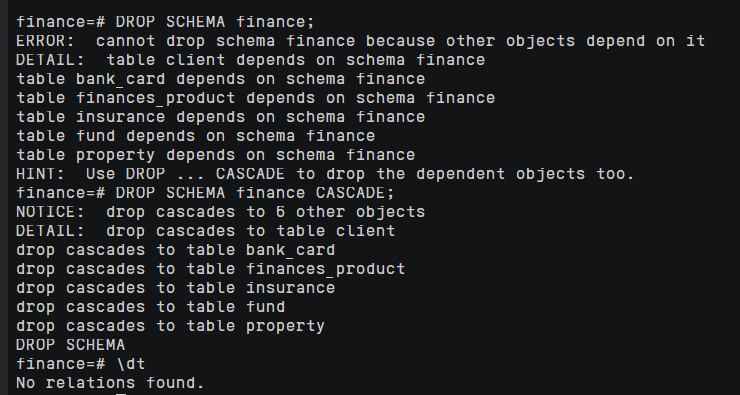
1.1.12





1.1.13





1. **1.1.7节中SQL查询语句和能够满足查询需求的关系代数表达式**

1、查询银行卡信息表。

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

π b\_number, b\_type (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';

σ pro\_status='可用' (property)

3、查询用户表中有多少个用户。

SELECT count(\*) FROM client;

γ count(\*) (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;

γ b\_type, count(\*) (bank\_card)

5、查询保险信息表中，保险金额的平均值。

SELECT AVG(i\_amount) FROM insurance;

γ AVG(i\_amount) (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);

(π i\_name, i\_amount (σ i\_amount=max(i\_amount) (insurance))) ∪ (π i\_name, i\_amount (σ i\_amount=min(i\_amount) (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);

π c\_id, c\_name, c\_id\_card (client ⨝ client.c\_id = bank\_card.b\_c\_id (bank\_card))

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\_');

π c\_id, c\_name, c\_id\_card (client ⨝ client.c\_id ≠ σ b\_number LIKE '622202130202000001\_' (bank\_card))

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 i1.i\_name,i1.i\_amount,i1.i\_person FROM insurance i1 WHERE i\_amount > (SELECT avg(i\_amount) FROM insurance i2);

π i\_name, i\_amount, i\_person (σ i\_amount > γ AVG(i\_amount) (insurance) (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;

τ i\_amount DESC (π i\_name, i\_amount, i\_person (σ i\_id > 2 (insurance)))

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;

γ p\_year, count(p\_id) (finances\_product)

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;

γ i\_person, count(i\_amount) (σ count(i\_amount) = 2 (insurance))

13、使用WITH AS查询基金信息表。

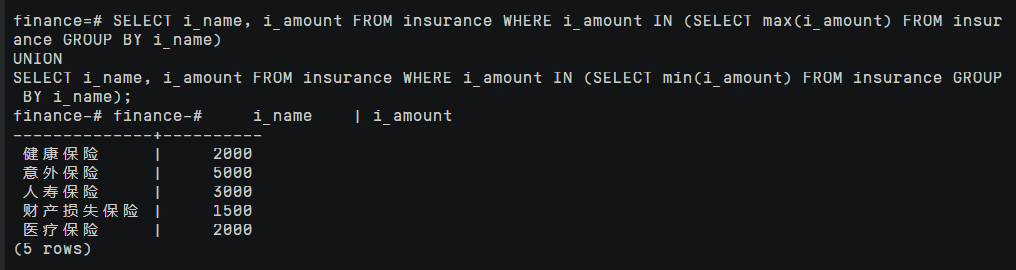
WITH temp AS (SELECT f\_name,ln(f\_amount) FROM fund ORDER BY f\_manager DESC) SELECT \* FROM temp;

WITH temp AS (τ f\_manager DESC (π f\_name, ln(f\_amount) (fund))) (temp)

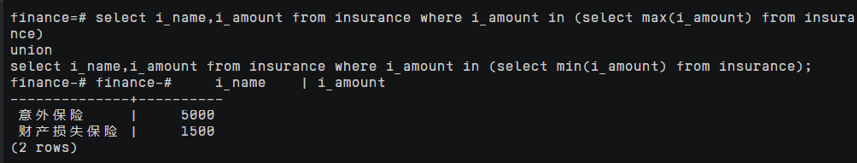
1. **初始SQL执行结果不符的原因以及其内容**

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

错误结果：



正确结果:



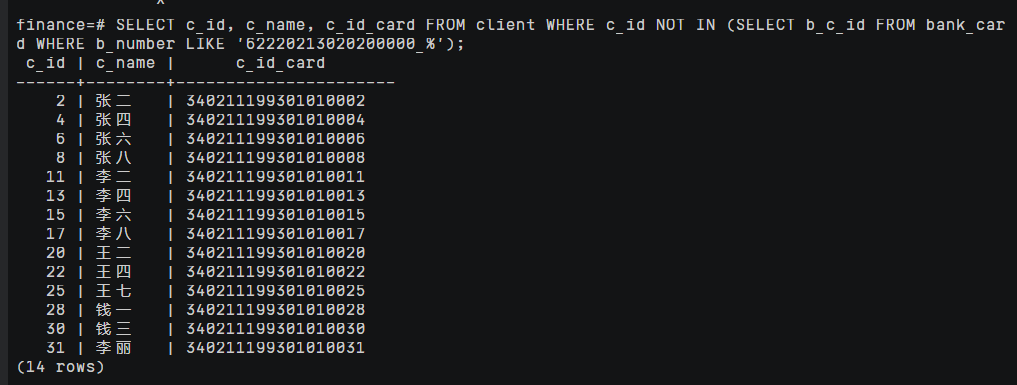
错误原因分析：

错误的 SQL 查询语句在子查询中使用了 GROUP BY i\_name，这导致子查询分别返回了每个 i\_name 对应的最大和最小 i\_amount 值。这会导致主查询检索所有 i\_name 对应的最大和最小 i\_amount 值，而不是整个表中的最大和最小 i\_amount 值。

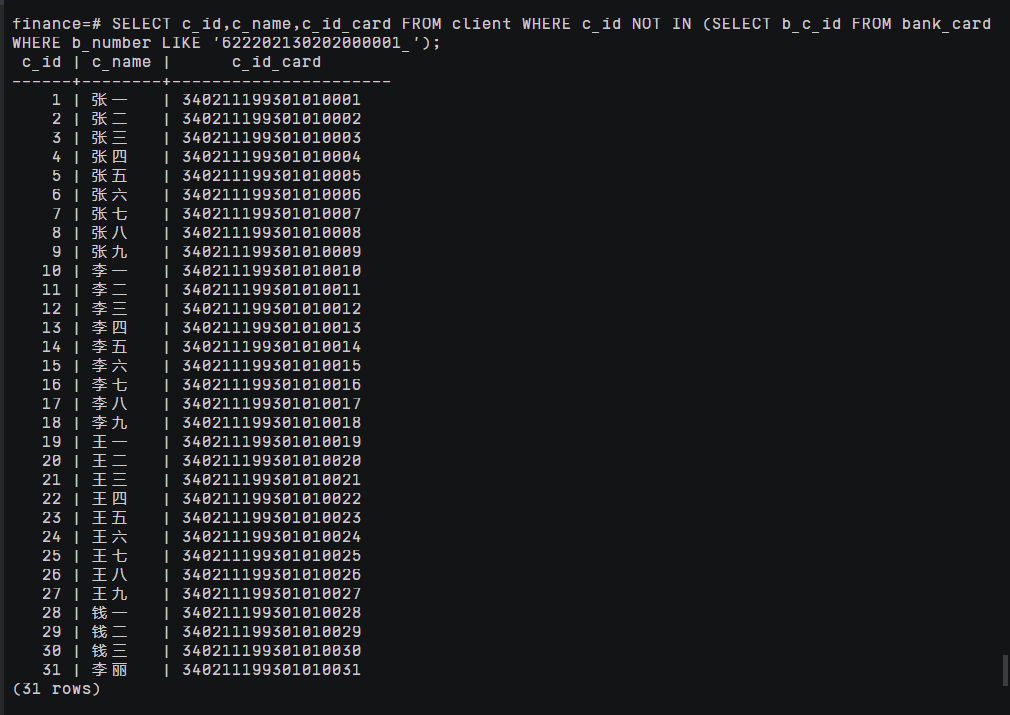
正确的 SQL 查询语句不包含 GROUP BY i\_name 子句，因此子查询将返回整个 insurance 表中的最大和最小 i\_amount 值。主查询将根据这些值检索对应的 i\_name 和 i\_amount。这是预期的查询行为，以找出具有最高和最低保额的保险记录。

反连接查询

错误结果：



正确结果:

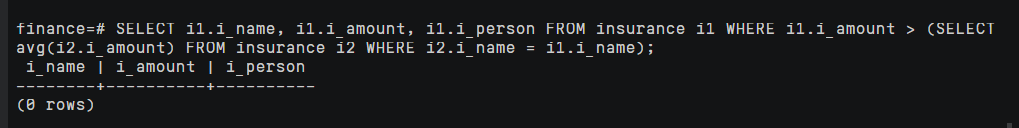


错误原因分析：

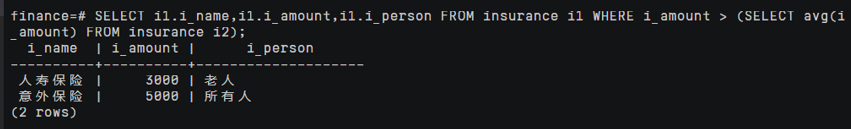
错误的 SQL 查询语句中，LIKE 子句的通配符从正确的 '622202130202000001\_' 更改为 '62220213020200000\_%'。这将导致子查询匹配以 '62220213020200000' 开头的所有 b\_number，而不是仅匹配以 '622202130202000001' 开头的 b\_number。这会改变主查询的筛选条件，从而导致返回的结果不符合预期。

子查询

错误结果：



正确结果:

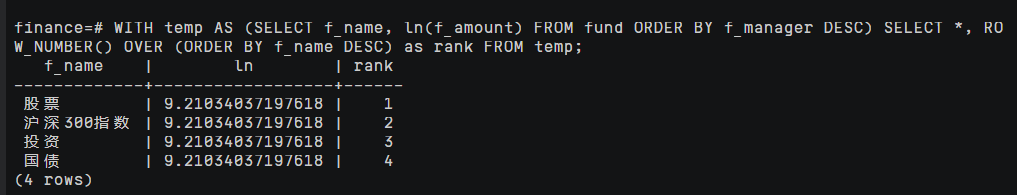


错误原因分析：

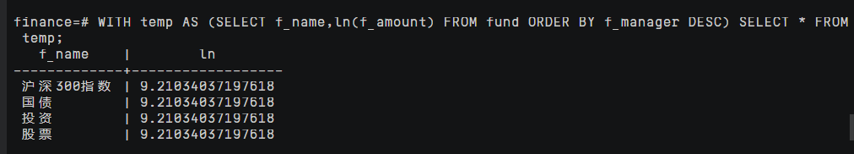
错误的 SQL 查询语句在子查询的 WHERE 子句中添加了条件 i2.i\_name = i1.i\_name。这导致子查询计算的平均值是基于相同 i\_name 的 i\_amount，而不是整个 insurance 表中的所有 i\_amount。这会改变主查询的筛选条件，从而导致返回的结果不符合预期。

WITH AS子句

错误结果：



正确结果:



错误原因分析：

错误的 SQL 查询语句在最后的 SELECT 语句中添加了一个 ROW\_NUMBER() 函数，对结果进行额外的排序。虽然这不会导致结果集发生变化，但会在结果集中添加一个额外的列 rank，这可能会导致混乱。这种错误虽然不影响结果集的内容，但在某些情况下可能会对进一步分析造成困扰。

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**

1、实验总时长分析：本次实验共计耗时8小时。实验主要分为以下几个阶段：实验环境准备、编写 SQL 查询语句、测试与优化、撰写实验报告。实验过程中花费时间较多的部分在于编写 SQL 语句以及测试与优化阶段，以确保实现预期的功能并达到较好的性能。

2、遇到的问题：

在编写 SQL 查询语句时，遇到了一些逻辑错误，例如子查询的使用不当，导致查询结果与预期不符。通过仔细阅读文档和参考示例，最终解决了这些问题。

在测试与优化阶段，发现部分查询性能较低。为了提高查询性能，对查询语句进行了优化，例如添加索引、调整查询条件等。

在实验过程中，对 openGauss 的某些特性和函数的使用不够熟练，需要查阅文档和在线资源以获取帮助。

3、实验中学习到的知识点：

学习了 openGauss 数据库管理系统的基本概念和功能，以及如何在实验环境中配置和使用 openGauss。

熟练掌握了编写 SQL 查询语句的技巧，包括对表进行连接、过滤、聚合等操作，以实现各种查询需求。

学习了如何对查询性能进行优化，包括添加索引、调整查询条件、利用 openGauss 的性能优化特性等。

深入了解了数据库系统的高可用性、安全性等方面的知识，以及如何在 openGauss 中实现这些特性。

通过解决实验过程中遇到的问题，提高了自己在数据库开发和查询方面的问题解决能力。

通过本次实验，我们对关系型数据库管理系统和 SQL 查询语句有了更深入的理解，同时提高了数据库开发和查询的实践能力。