openGauss 安全体系创新

实践课



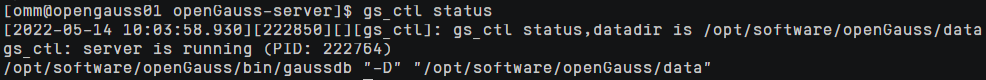
华为技术有限公司

# 关卡一、openGauss数据安装及基本操作

openGauss数据安装及基本操作, 作业提交任务如下：

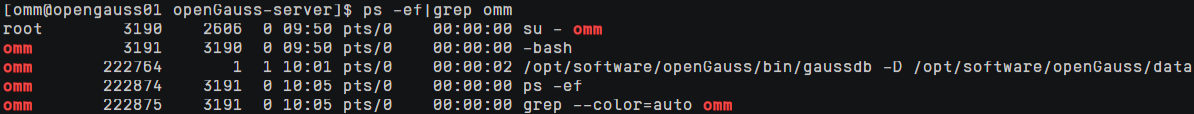
任务一：数据库状态验证

1. 查询数据库状态成功截图



任务二：数据库服务进程验证

1. 查看数据库服务进程截图（包含数据库服务器的主机名）



任务三：实践思考题

思考题1：为什么需要通过源码编译，安装数据库？

1.通过源码编译可以更为直观地了解到具体每步的作用，方便我知晓安装数据库的基本原理与基础的操作步骤，并且方便我在遇到错误时快速的定位源头，从而直观地解决错误，更好地促进作为初学者的我的学习。

2.更好地适应opengauss的环境，更自由地个性化定制以满足运行的需求，确保了在大多数情况下的正常运行。

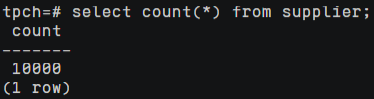
3.更方便后期的更新，维护等操作，对于需要修改的地方无需进行大面积的重新处理，更易于长期的使用。

# 关卡二、openGauss数据导入及行存列存

任务一：数据初始化验证

1. 查询supplier表的行数，并将结果进行图：

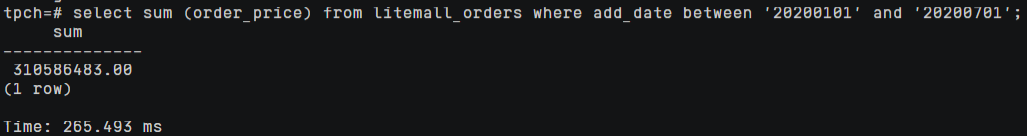
select count(\*) from supplier;;



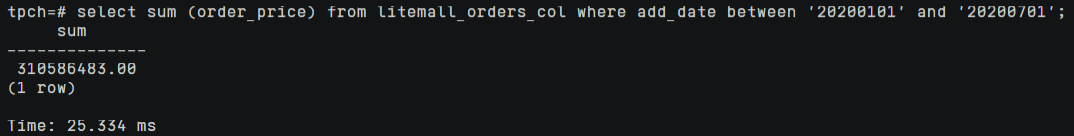
任务二：行存表与列存表执行效率对比

1. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的总和查询，并对比执行效率截图

select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



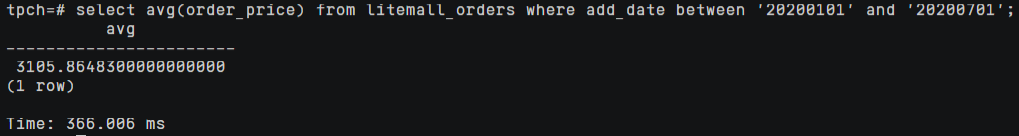
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



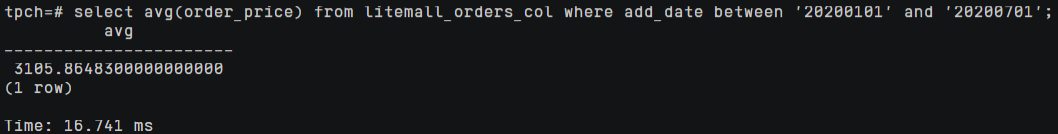
列存表执行效率高

2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

select avg (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



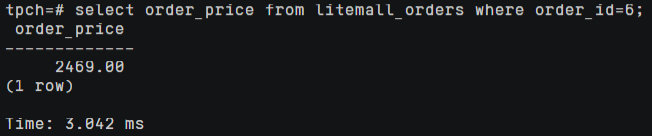
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



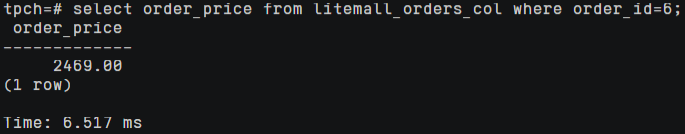
列存表执行效率高

3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

select order\_price from litemall\_orders where order\_id=6;



select order\_price from litemall\_orders\_col where order\_id=6;



行存表执行效率高

4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

update litemall\_orders set order\_price=2468 where order\_id=6;

litemall_orders行存表中order_id为6的order_price修改为2468

update litemall\_orders\_col set order\_price=2468 where order\_id=6;

litemall_orders_col列存表中order_id为6的order_price修改为2468

行存表执行效率高

任务三：实践思考题

思考题1：

行存表与列存表在执行相同的SQL语句时，为何执行的时间不同？

在行式表中， 数据是按照行数据为基础逻辑存储单元进行存储的， 一行中的数据在存储介质中以连续存储形式存在；在列存表中， 数据是按照列为基础的逻辑存储单元进行存储的，一列中的数据在存储介质中以连续存储形式存在。在行式表中，所有数据都是连续的，只需定位一次便可以找到想写入的数据的表的位置；而列式表只有一列数据是连续的，若想写入则需定位多次表头的位置才能完成写入的操作，效率较低。行式表在读取的时候只能读取全部的数据，存在大量的冗余项；而列式表每次读取的只有集合中想要的部分，效率上更高。

思考题2：

在执行哪些类型SQL时，行存表效率更高？在执行哪些类型SQL时，列存表效率更高？

行式表效率高：数据写入，数据修改，对于特定数据的查询，随机的增删改查操作，需要在行中选取所有属性的查询等操作。

列式表效率高：对于只涉及少数同类数据的大面积增删改查等操作。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

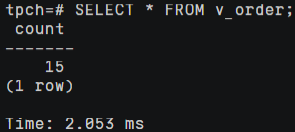
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



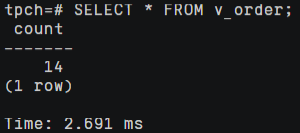
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



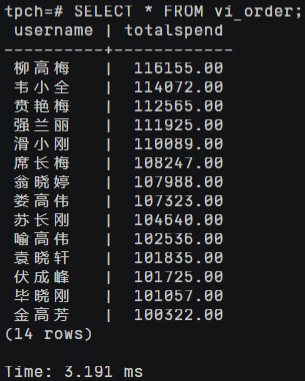
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



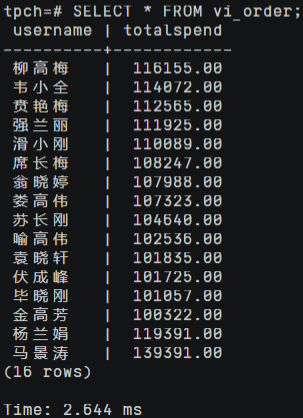
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



任务二：实践思考题

思考题1：全量物化视图与增量物化视图有哪些差别？

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新；增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新，与全量创建物化视图不同在于目前增量物化视图所支持场景较小。全量物化视图不支持NodeGroup；而增量物化视图会继承基表NodeGroup创建。创建全量物化视图可以指定分布列。而量物化视图在创建物化视图时不可指定物化视图分布列。

思考题2：物化视图适用那些使用场景？

1.用于优化sql查询，提升开发的效率。

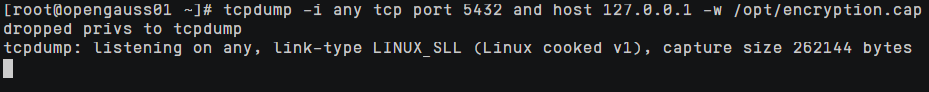
2.融合不同表中的字段，更高效的增删改查。

3.在对安全性有需求的情况。

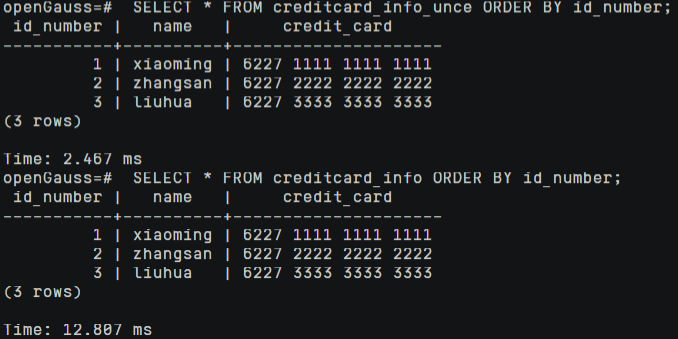
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

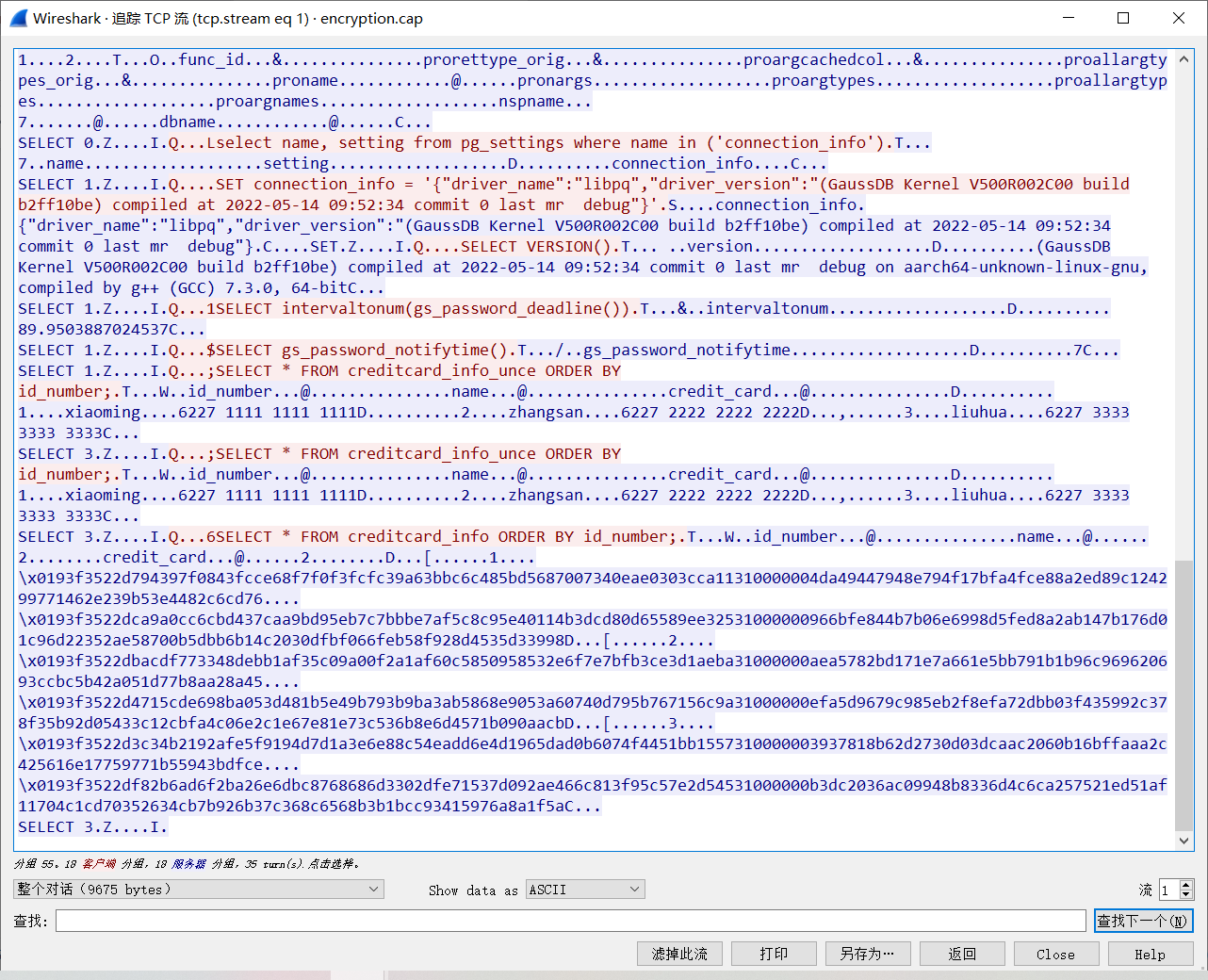
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



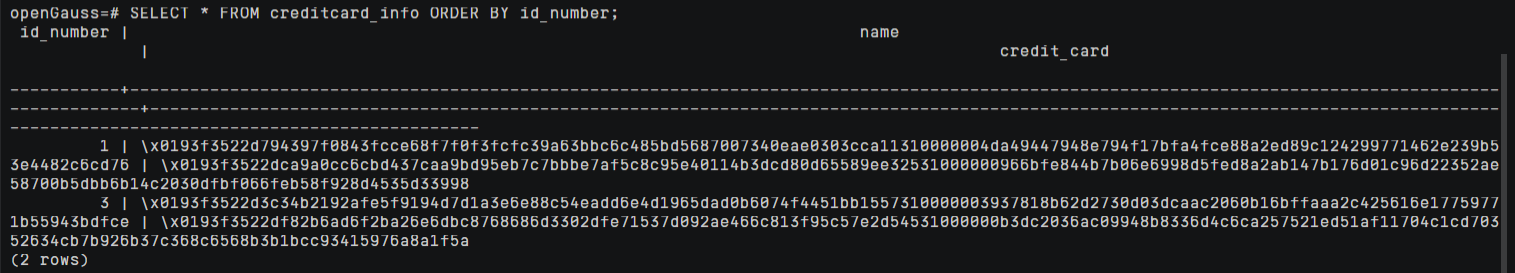
1. 将加密表和非加密表查询结果截图：



1. 用wireshark解析加密表和非加密表的差异时，非加密表name列和credit\_card列是明文，加密表name列和credit\_card列均是密文，将执行结果截图：



1. 查询加密表，查询到的结果为密文，将执行结果截图：



任务二：实践思考题

思考题1：

数据实际存储在物理磁盘上的时候是明文还是密文？数据的加解密的动作是在客户端完成的还是服务端完成的？

密文；客户端