openGauss 安全体系创新

实践课



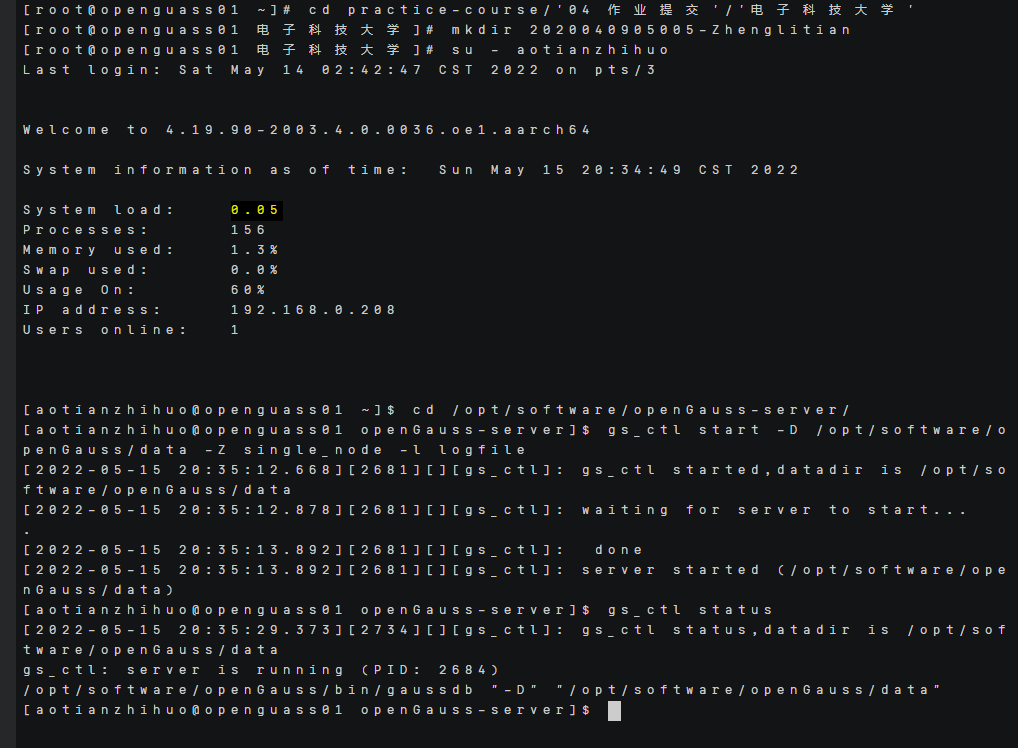
华为技术有限公司

# 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：

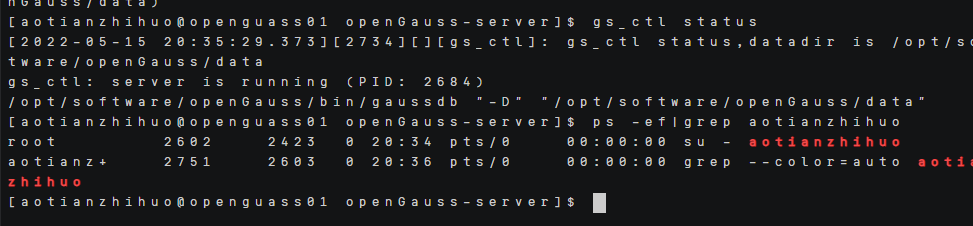
任务一：数据库状态验证

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？

在进行本次实验前，已经进行过一次大部分关卡的练习，所以数据库状态可能会与第一次运行时不完全一样。

本关卡先下载了opengauss数据库，随后进行了一些依赖环境的下载，使用了软连接替换python版本，进行了linux用户组间的切换和环境变量的编写，随后进行了数据的初始化操作。

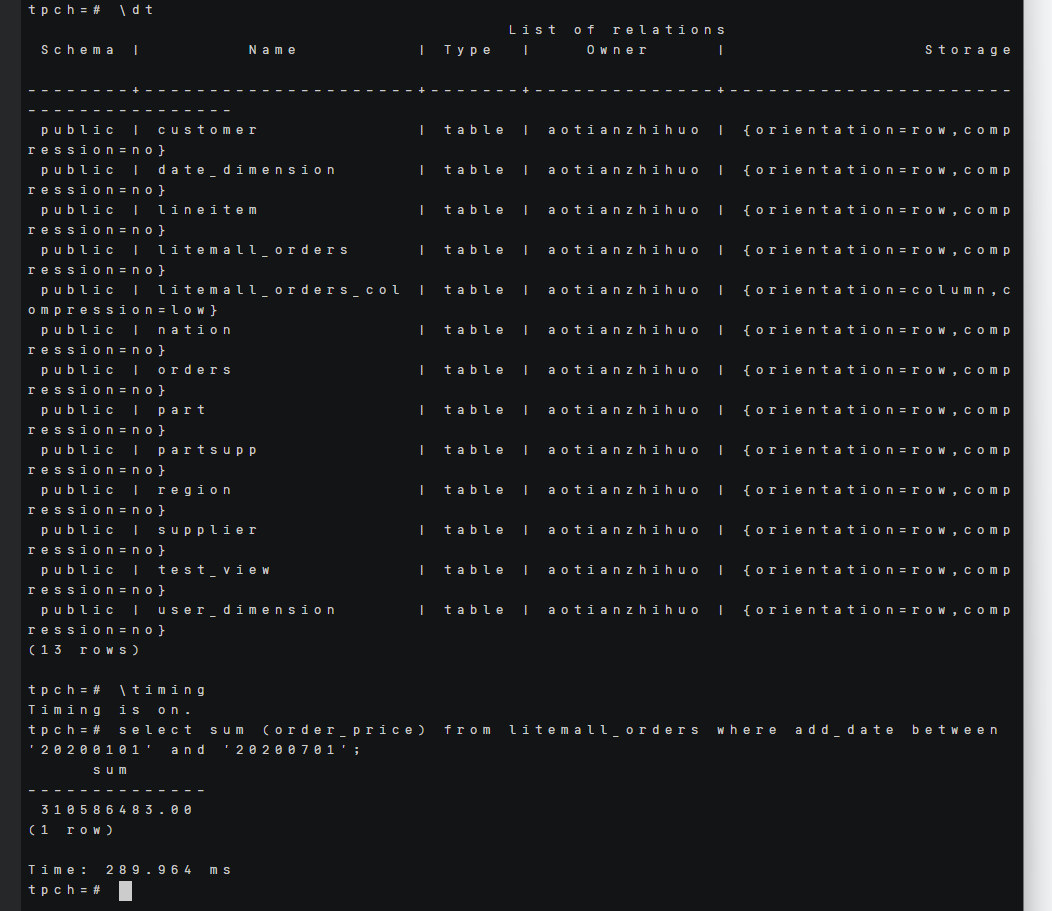
思考题：源码编译虽耗费时间长，但可移植性很强，不会受环境影响，出现特别奇怪的bug，数据库安装步骤见上文。

# 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：

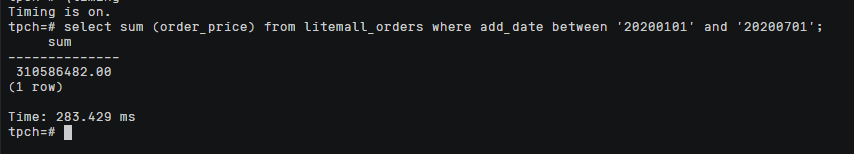
select count(\*) from supplier;;



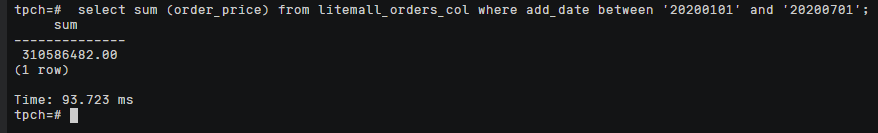
任务二：行存表与列存表执行效率对比

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';

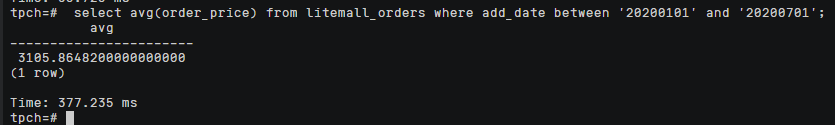


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

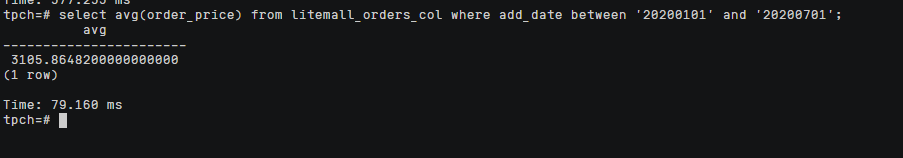


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

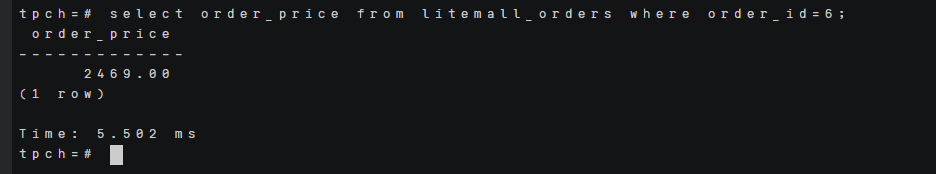


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

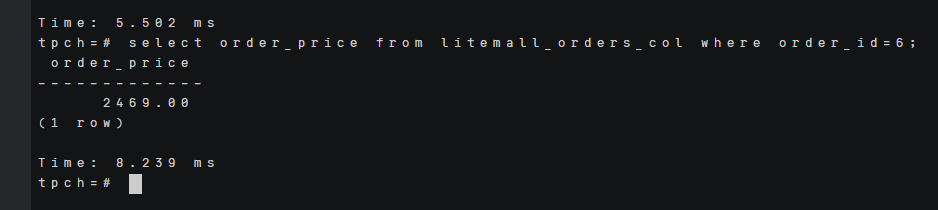


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

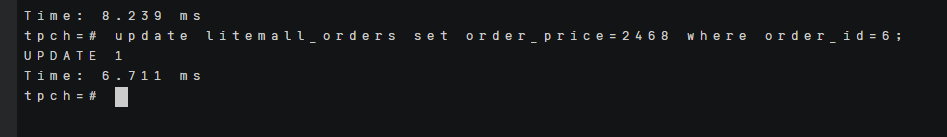


select order\_price from litemall\_orders\_col where order\_id=6;

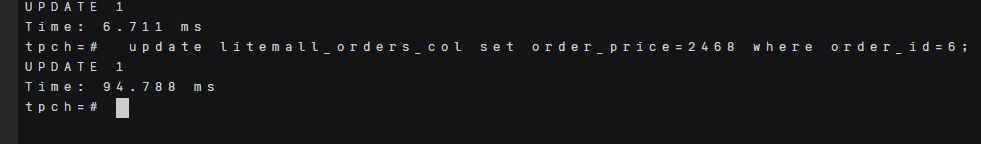


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



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



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？

行存储的写入是一次性完成，消耗的时间比列存储少，并且能够保证数据的完整性，缺点是数据读取过程中会产生冗余数据，如果只有少量数据，此影响可以忽略；数量大可能会影响到数据的处理效率。

　　列存储在写入效率、保证数据完整性上都不如行存储，它的优势是在读取过程，不会产生冗余数据，这对数据完整性要求不高的大数据处理领域，比如互联网，犹为重要。

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？

如果大部分时间都是关注整张表的内容，而不是单独某几列，并且所关注的内容是不需要通过任何聚集运算的，那么推荐使用行式存储。原因是重构每一行数据（即解压缩过程）对于HANA来说，是一个不小的负担。

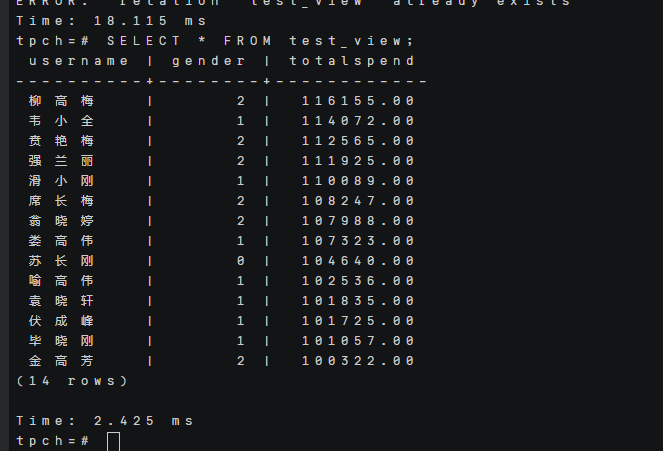
列式存储的话，比如比较关注的都是某几列的内容，或者有频繁聚集需要的，通过聚集之后进行数据分析的表。

# 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：

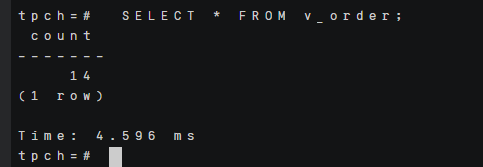
SELECT \* FROM test\_view;



由于之前提前执行了步骤七，删除了杨兰娟，因此本表只有十四行而不是十五行，希望老师能够不要介意

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。

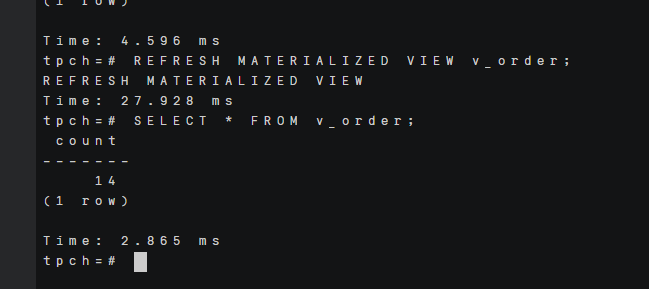
SELECT \* FROM v\_order;



由于之前提前执行了步骤七，删除了杨兰娟，因此本表只有十四行而不是十五行，希望老师能够不要介意

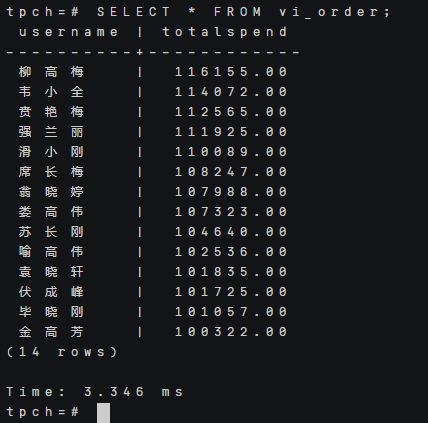
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



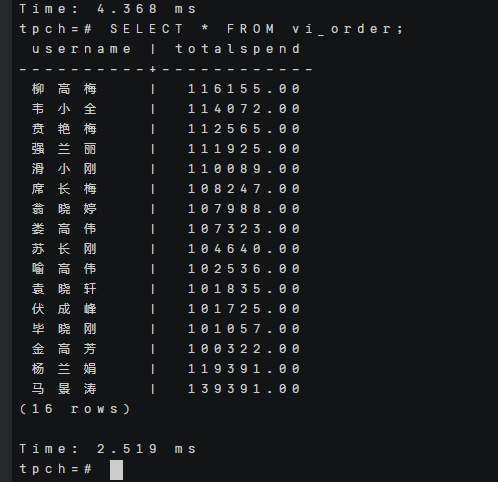
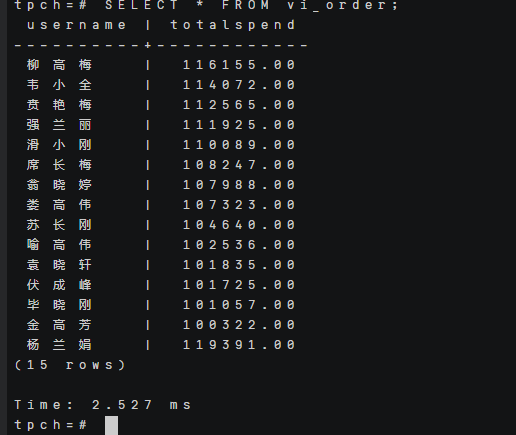
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

增量物化视图顾名思义就是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，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

思考题2：物化视图适用那些使用场景？

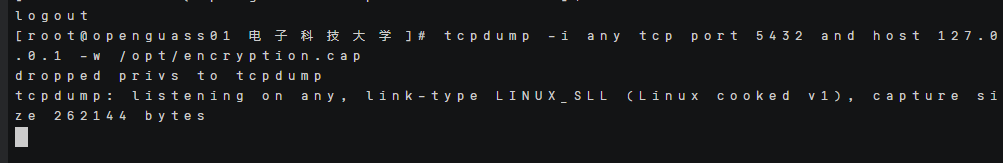
物化视图实际上就是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对SQL语句的查询，性能并没有实际上提高。而物化视图实际上就是存储SQL所执行语句的结果，起到缓存的效果。

适用场景：1、用于查询优化 2、用于高级复制

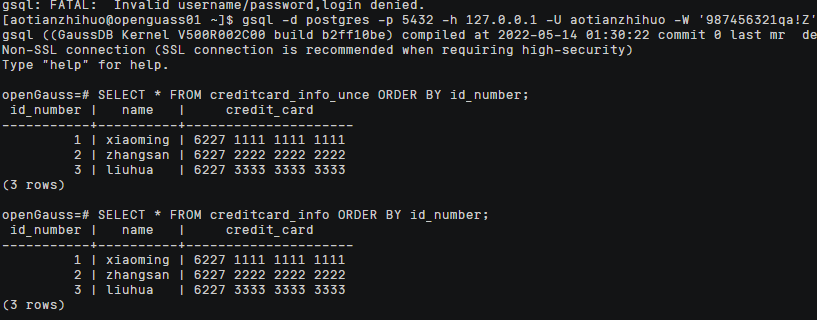
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

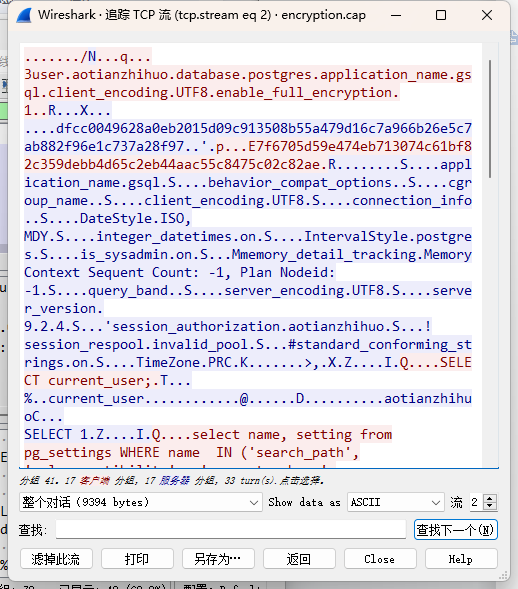
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



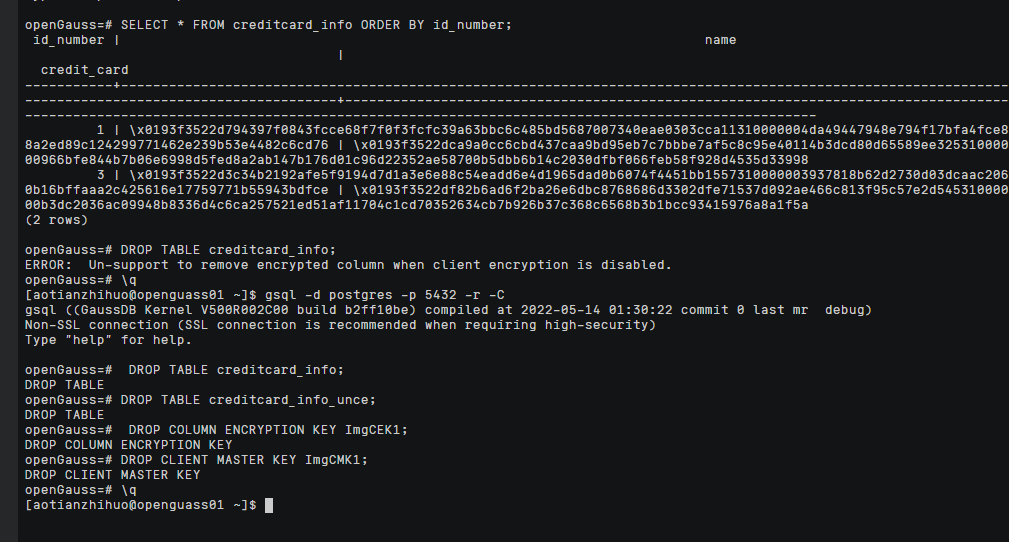
1. 将加密表和非加密表查询结果截图：



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