openGauss 安全体系创新

实践课



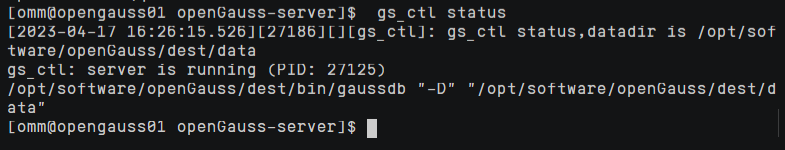
华为技术有限公司

# 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：

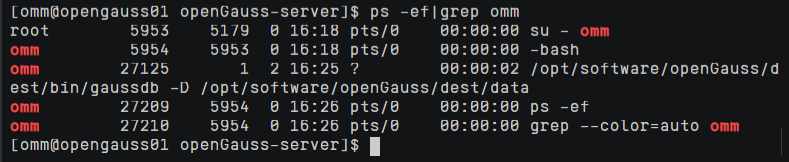
任务一：数据库状态验证

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？

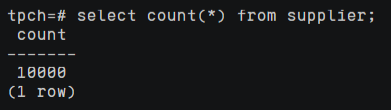
通过源码编译安装数据库可以提供更好的灵活性和可调试性。可以根据自己的需求进行灵活的配置和编译选项，从而生成定制化的数据库安装。可以根据需要进行调试符号的设置、启用调试信息等，以便更好地进行数据库的开发和调试工作。

# 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：

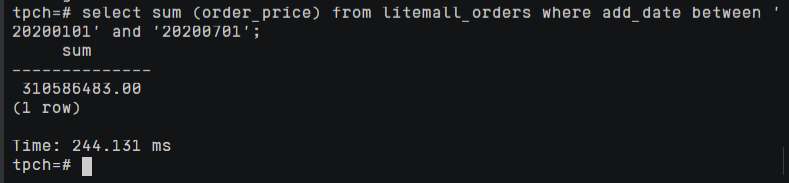
select count(\*) from supplier;;



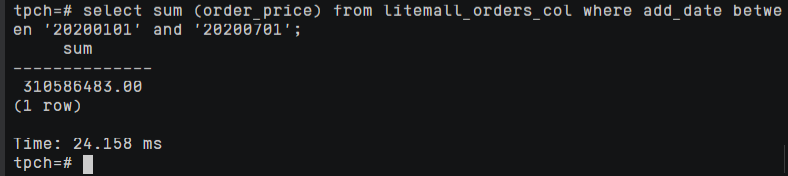
任务二：行存表与列存表执行效率对比

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';

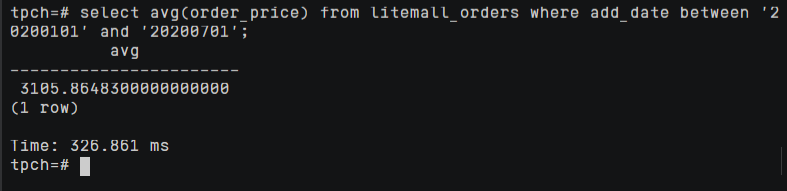


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

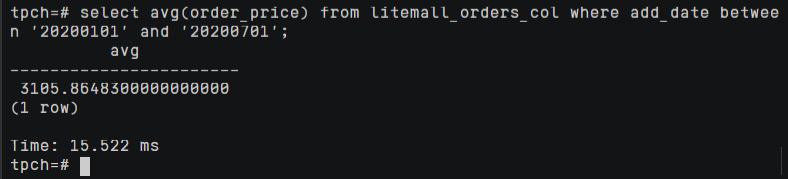


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

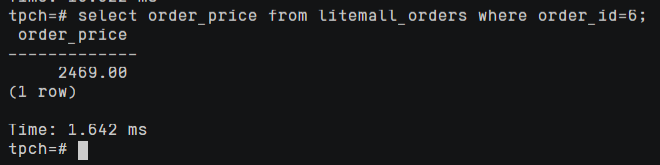


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

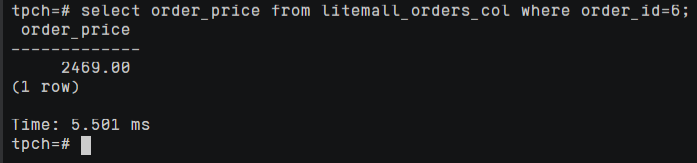


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

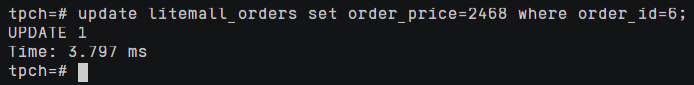


select order\_price from litemall\_orders\_col where order\_id=6;

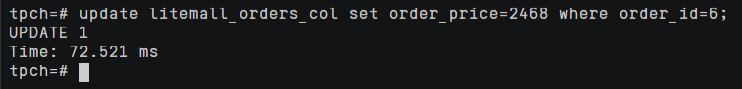


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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？

行存表，数据读取时通常将一行数据完全读出，如果只需要其中几列数据的情况，就会存在冗余列。而列存表每次读取的数据是集合的一段或者全部，不存在冗余性问题。在写入和修改数据时，行存表是在指定位置写入一次，而列存表是将磁盘定位到多个列上分别写入。因此，在执行相同的SQL语句时，行存表和列存表的执行时间可能不同。

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？

行存表效率可能较高的情况包括：需要读取整行数据的查询：对于需要读取整行数据的查询，如根据主键查询某一行数据，行存表由于将每一行的数据连续存储，可以直接通过行号或者主键进行快速访问，因此可能具有较高的查询效率。小规模数据的查询：对于数据规模较小的情况，行存表由于不需要进行列级别的操作和压缩，可能在查询效率上具有优势。

列存表（Columnstore Table）效率可能较高的情况包括：聚合和分析查询：对于需要进行聚合、过滤和分析等操作的查询，列存表由于将每一列的数据连续存储在一起，可以仅读取相关的列数据，从而提供更好的性能。大规模数据的查询：对于数据规模较大的情况，列存表由于其较高的数据压缩率和只读取相关列的特点，可以在查询效率上具有优势。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

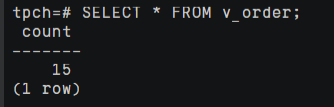
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



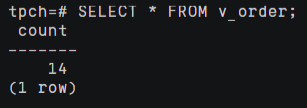
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



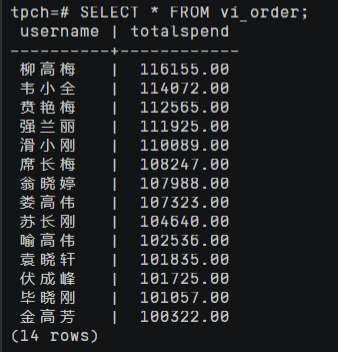
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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？

数据更新方式：全量物化视图会在每次刷新时重新计算和加载完整的视图数据，而增量物化视图则只更新变化的数据部分。全量物化视图在刷新时需要重新计算和加载整个视图的数据，因此在处理大量数据时可能会较慢，而增量物化视图只更新发生变化的数据，因此在处理大量数据时可能更加高效。

存储空间占用：全量物化视图通常会占用较大的存储空间，因为它需要存储完整的视图数据，而增量物化视图只需要存储变化的数据部分，因此占用的存储空间通常较小。

查询性能：全量物化视图在刷新时需要重新计算和加载整个视图的数据，因此在刷新期间可能会导致性能下降，而增量物化视图只需要更新变化的数据部分，因此在刷新期间对查询性能的影响较小。

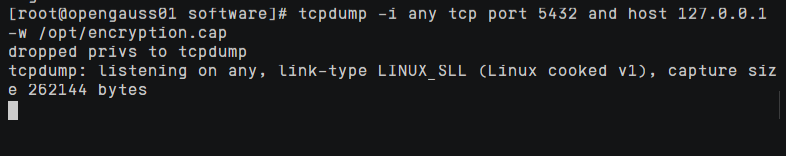
刷新频率：全量物化视图的刷新通常需要较长的时间，因为需要重新计算和加载整个视图的数据，而增量物化视图由于只需要更新变化的数据部分，因此可以更频繁地进行刷新，从而更及时地反映源表的变化。

数据一致性：全量物化视图在刷新时会保持与源表的完全一致性，而增量物化视图在刷新期间可能会出现与源表的稍微不一致的情况，因为只更新变化的数据部分。思考题2：物化视图适用那些使用场景？

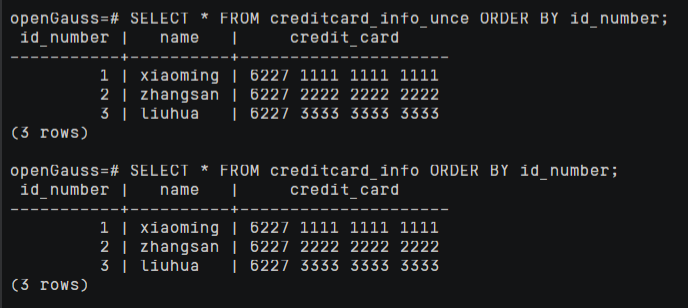
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

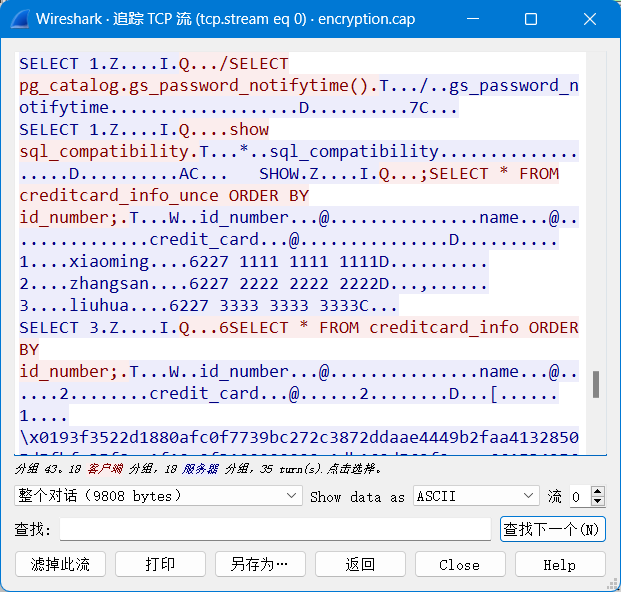
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



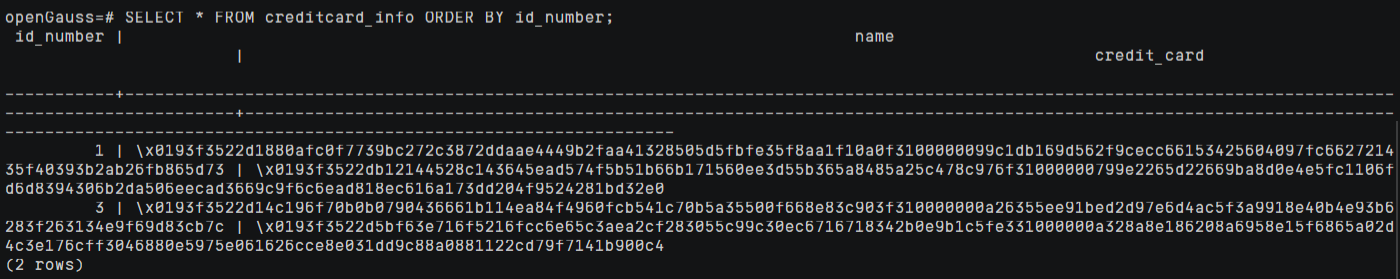
1. 将加密表和非加密表查询结果截图：



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？

数据在物理磁盘上实际存储时是经过加密处理的，加解密的动作是在数据库服务端完成。