openGauss 安全体系创新

实践课



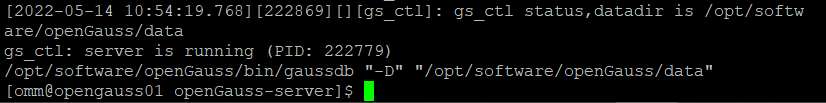
华为技术有限公司

# 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：

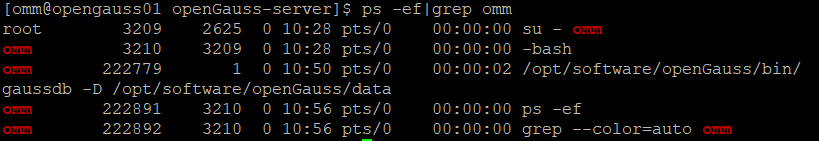
任务一：数据库状态验证

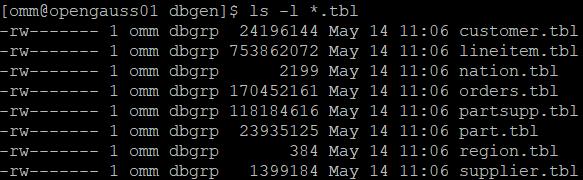
1. 查询数据库状态成功截图



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？

**优点**：

源码安装用户可以根据自己的需要对源码进行修改，然后再编译生成二进制文件。方便于对数据库进行修改和更新。如果采用二进制文件安装，用户则无法对程序作任何修改。编译安装过程，可以设定参数，按照需求进行安装并且安装的版本，可以自己选择，灵活性比较大。

但是源码编译有相应的**缺点**：

由于安装包过新或者是其他问题，导致依赖的包没有，或者版本过低。这个时候就要解决包的依赖问题，linux系统中有的包，一个依赖一个，可能装一个小东西，就要解决一堆包的依赖问题，花很多时间解决包的依赖问题，得不尝失。

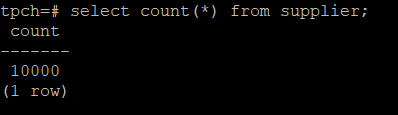
我认为，在此处安装数据库的时候，基本不需要解决包依赖的相关问题，反而能够灵活设定参数更能适应我们的需求。

# 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：

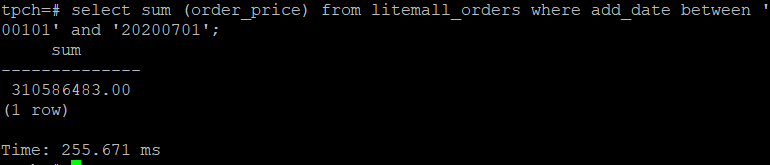
select count(\*) from supplier;;



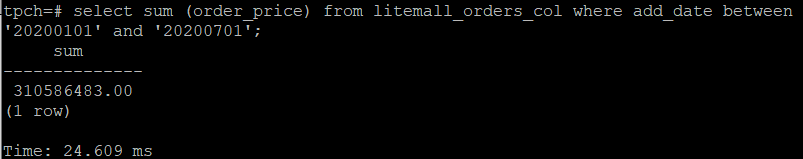
任务二：行存表与列存表执行效率对比

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';

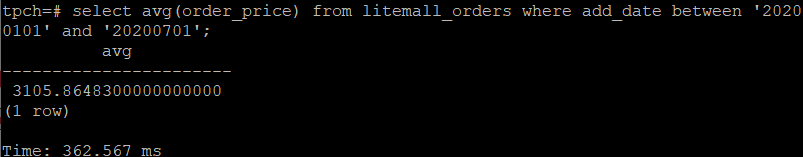


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

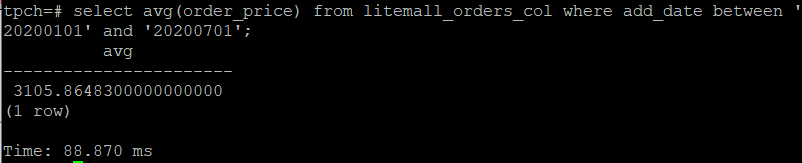


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

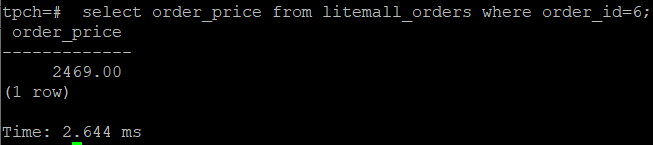


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

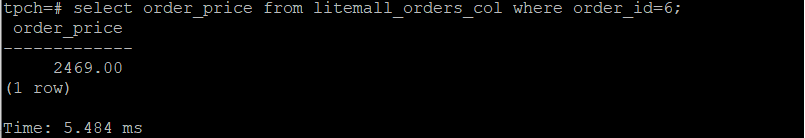


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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？

传统的关系型数据库，如 Oracle、DB2、MySQL、SQL SERVER 等采用行式存储法(Row-based)，在基于行式存储的数据库中， 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。

列式存储(Column-based)是相对于行式存储来说的，新兴的 Hbase、HP Vertica、EMC Greenplum 等分布式数据库均采用列式存储。在基于列式存储的数据库中， 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

因此，在执行相同的SQL语句中执行时间不同的主要原因：

数据读取时，行存储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，出于缩短处理时间的考量， 这时列存表就会快很多。同理适用于行存表。

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？

1）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。

2）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，减少了处理的时间

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

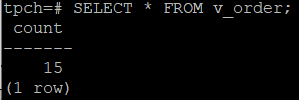
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



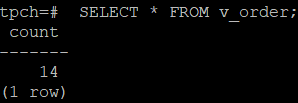
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



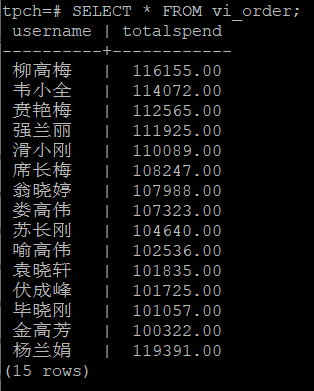
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

1）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。

2）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。

3）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？

用途：

视图根本用途在我看来就一个：简化sql查询，提高开发效率。如果说还有另外一个用途那就是兼容老的表结构。

使用场景：

1）计算列的需要，数据库设计范式要求我们减少冗余字段，因此现在很多数据表都没有计算列字段，如采购单：有价格、数量、税率、含税金额，多半没有不含税金额、税额，而这些字段在很多报表中有都会用到，所以我们可以创建一个含有计算列字段的视图来解决这个问题。

2）不同表字段聚合，信息重组，如：经销商通常有业务员，业务员通常有上下级关系（客户经理、区域经理、大区经理等）因此我们可以联合经销商表、业务员信息表、业务员上下级关系表定义一个经销商视图。

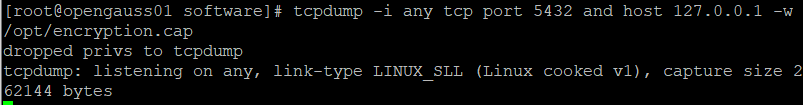
3）安全性需要，合理利用视图则可以减少很多授权工作和保证数据安全性。当下新构建的系统几乎都是暴露api接口，因此数据安全性更多关注在接口的身份认证和数据粒度方面。

4）兼容老的数据表。

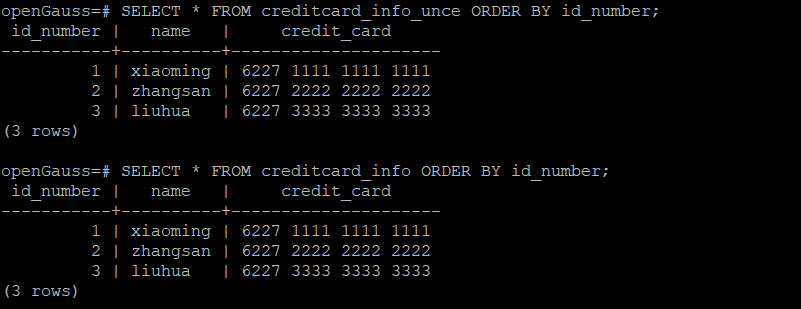
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

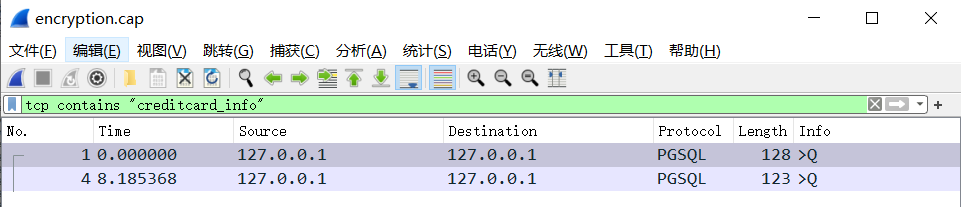
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



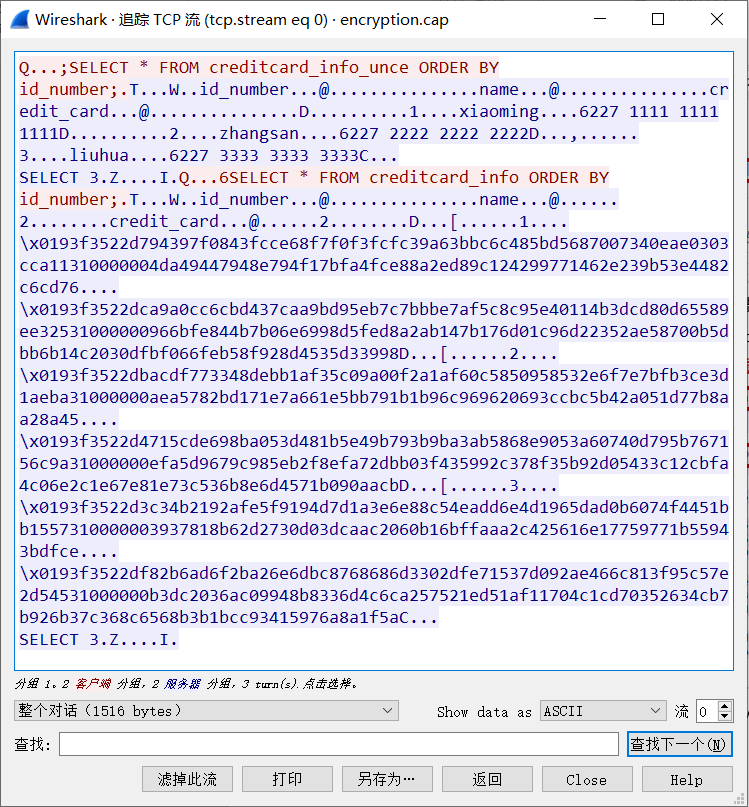
1. 将加密表和非加密表查询结果截图：



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？

是密文，加解密动作在客户端完成