openGauss 安全体系创新

实践课



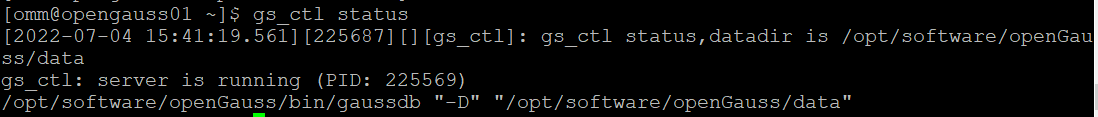
华为技术有限公司

# 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：

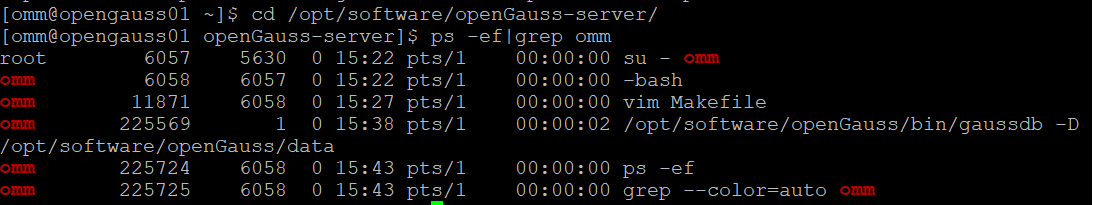
任务一：数据库状态验证

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？

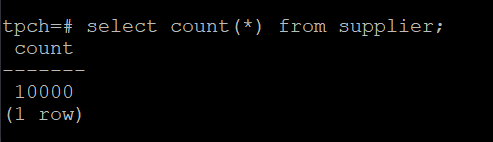
1. 满足不一样的运行平台， Linux发型版本众多，可是每一个版本采用的软件或者内核版本都不同，而二进制包所依赖的环境不一定可以正常运行，因此大部分软件直接提供源码；
2. 方便定制，满足不一样的需求，通常所须要的软件都是能够定制的，需要什么就安装什么，大多数二进制代码都是一键装全，因此自由度并不高；
3. 方便运维、开发人员维护，源码是能够打包二进制的，可是对于这个软件的打包都会有一份代价不小的额外工作，包括维护，因此若是源码的话，软件产商会直接维护，可是若是是二进制的话，通常都是Linux发行商提供。

# 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;;

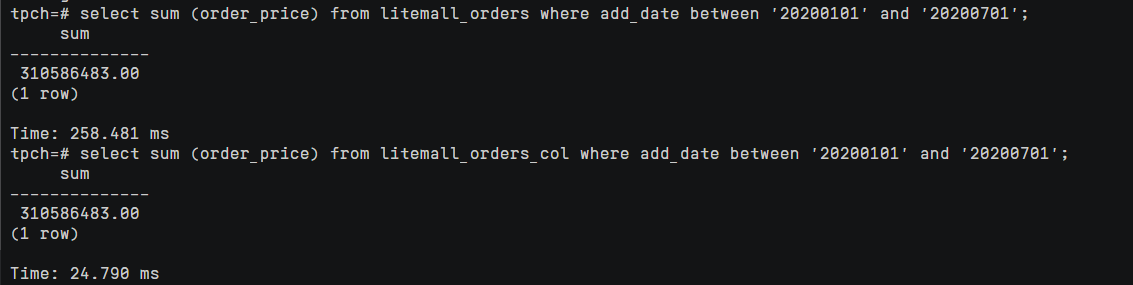


任务二：行存表与列存表执行效率对比

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';

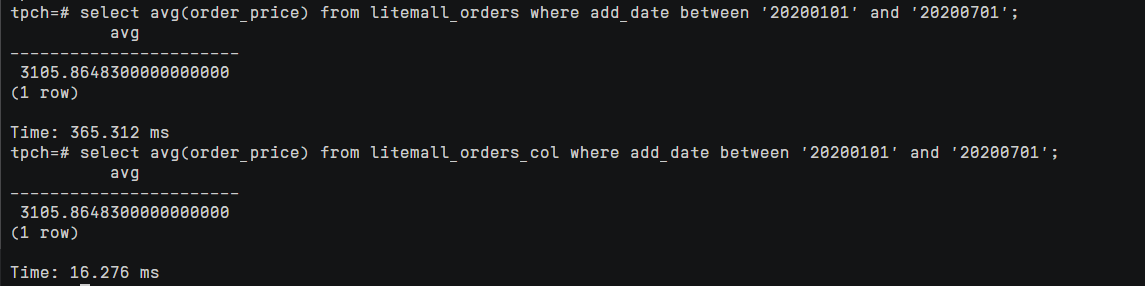
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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';

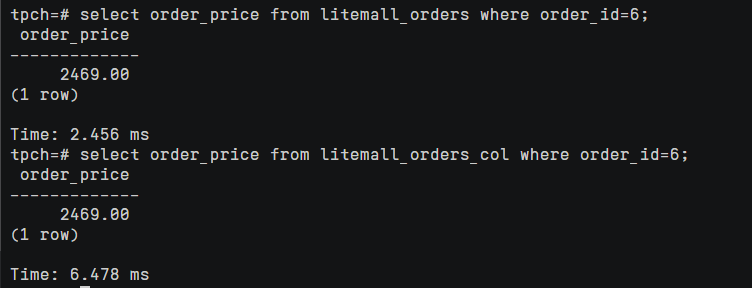
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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;

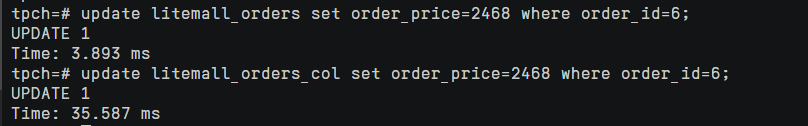
select order\_price from litemall\_orders\_col where order\_id=6;



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？

1. 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。
2. 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。
3. 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；而列存储在写入效率、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？

行存表适用场景：

1. 适合随机的增删改查操作;
2. 需要在行中选取所有属性的查询操作;
3. 需要频繁插入或更新的操作，其操作与索引和行的大小更为相关；

列存表适用场景：

1. 查询往往只关心少数几个数据列；
2. 如果读取的数据列属于相同的列族，列式数据库可以从相同的地方一次性读取多个数据列的值，避免了多个数据列的合并。

# 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;

图片包含 图形用户界面

描述已自动生成

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;

文本

描述已自动生成

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;

文本

描述已自动生成

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;

图片包含 图形用户界面

描述已自动生成

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;

文本

描述已自动生成

文本

描述已自动生成

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？

全量物化视图刷新时要刷新全部数据，而增量物化试图只需要刷新增量的数据；全量物化视图可以支持复杂查询，而增量物化视图，只支持简单查询。

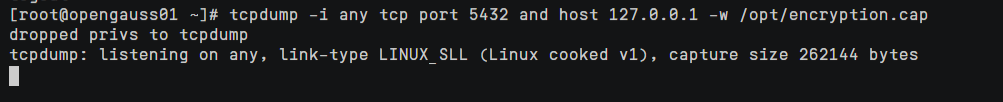
思考题2：物化视图适用那些使用场景？

报表统计、大表统计等，定期固化数据快照，避免对多表重复跑相同的查询

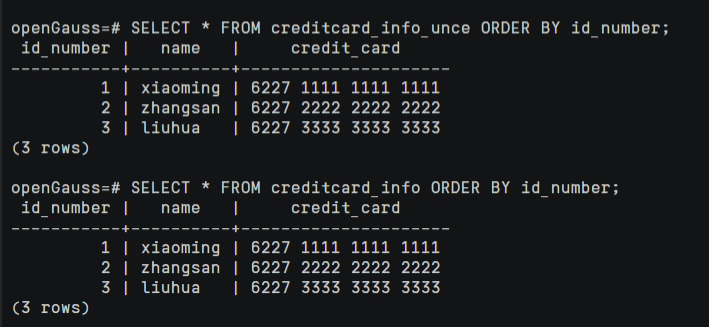
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

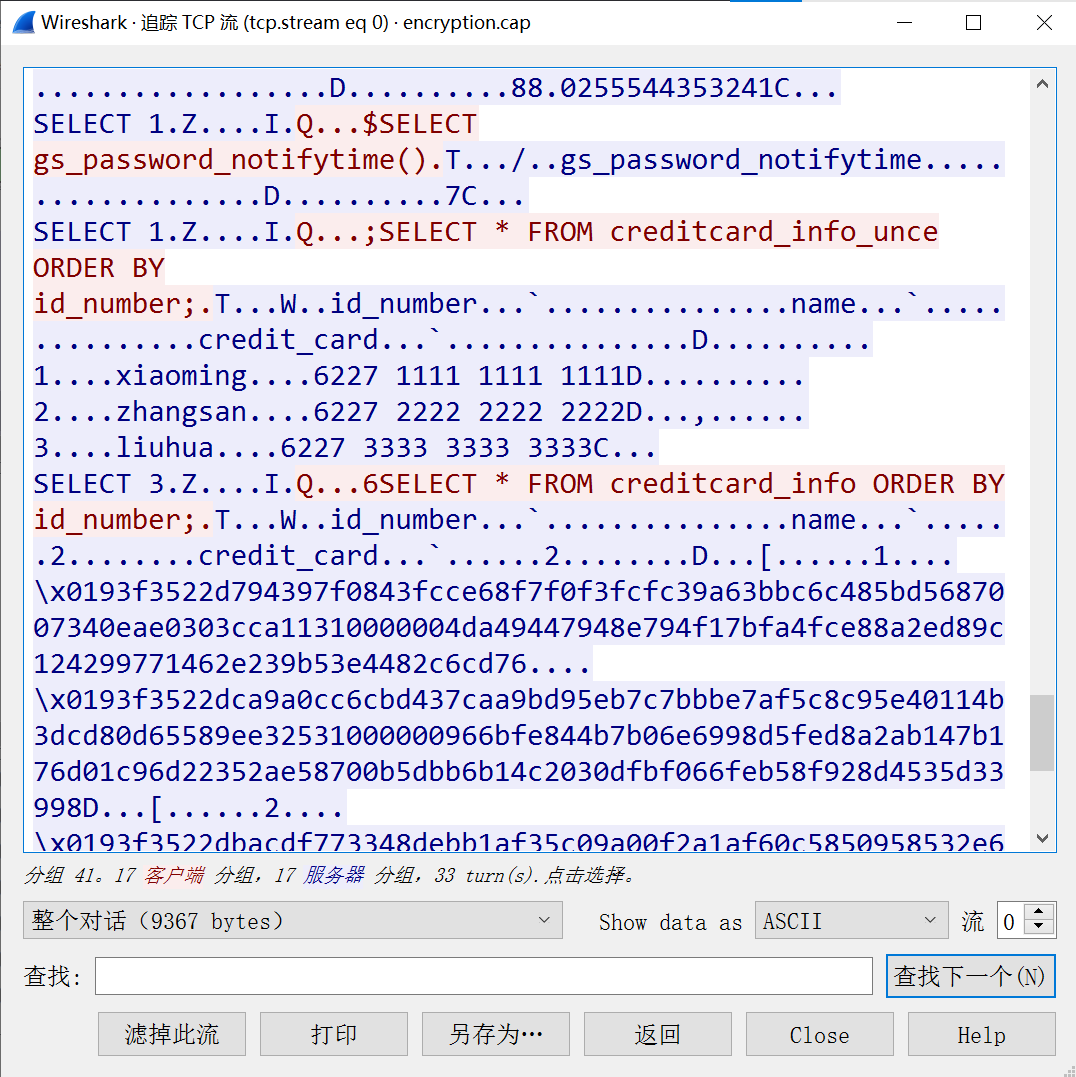
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



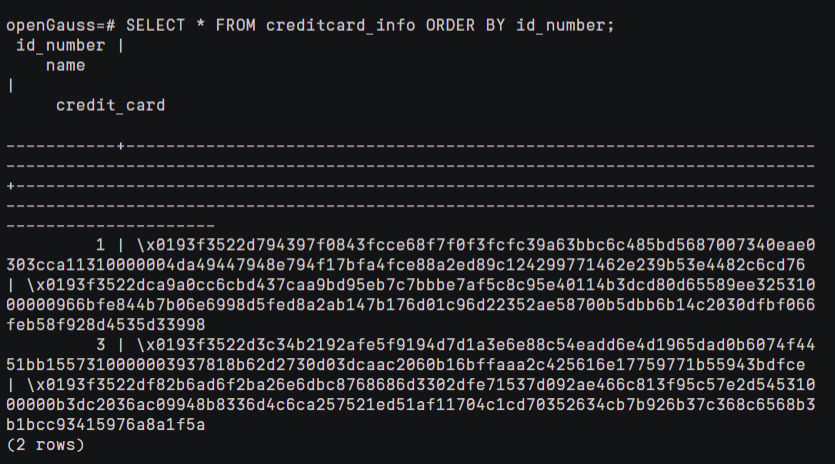
1. 将加密表和非加密表查询结果截图：



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？

数据实际存储的时候是以明文的方式进行存储，可以使用grep命令，在数据目录下查询到对应的明文信息。从总体流程上来看，数据在客户端完成加密，以密文形式发送到GaussDB Kernel数据库服务侧，即需要在客户端构建加解密模块。在数据库服务侧，加密列的数据始终以密文形态存在，整个查询也在密文形态下实现。用户根据业务需要对数据定义加密属性信息(被加密的列被称之为加密列)，对于不需要加密的数据则按照原有明文格式发送至服务端。当查询任务发起后，客户端需要对当前的Query进行解析，如果查询语句中涉及加密列，则对对应的列参数(加密列关联参数)也要进行加密(这里说的加密均需要为确定性加密，否则无法支持对应的查询)，如果查询语句中不涉及加密列，则直接发送至服务端，无需额外的操作。