openGauss 安全体系创新

实践课



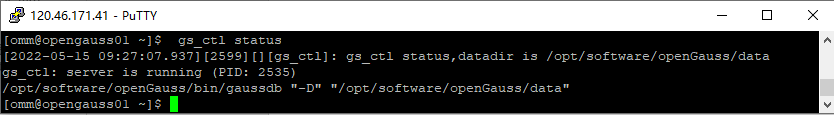
华为技术有限公司

# 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：

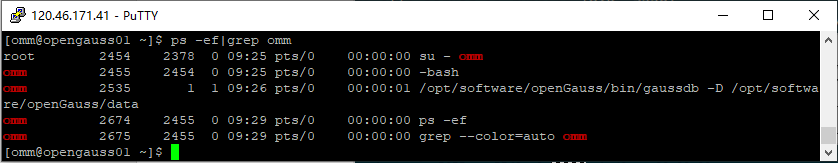
任务一：数据库状态验证

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？

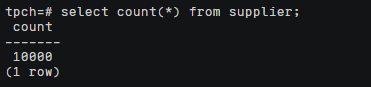
1. 可以兼容更多系统，满足不同的平台。例如，Linux版本众多，但是每个版本采用的软件或者内核版本都不一样，而我们的二进制包所依赖的环境不一定能够正常运行，所以大部分软件直接提供源码。
2. 在编译过程中，某些程序会根据CPU的型号进行优化，如Nginx，MySQL等，所以编译好的程序效率更高，速度更快。
3. 可以定制，满足不同的要求。编译安装给了用户最大的选择权和修改权，如：要增加PHP的模块，配置文件，然后再编译便可。
4. 方便运维，开发人员维护。

# 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：

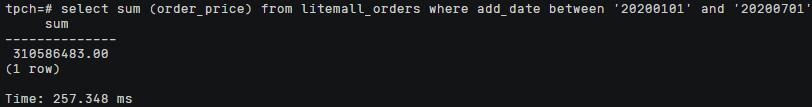
select count(\*) from supplier;;



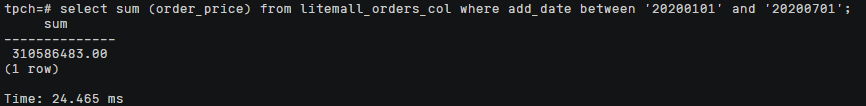
任务二：行存表与列存表执行效率对比

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';

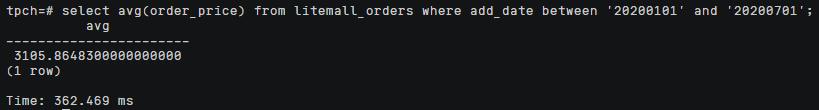


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

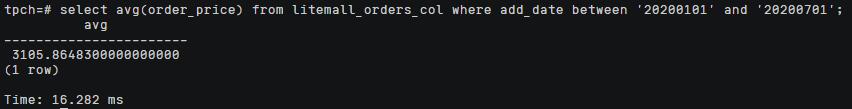


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

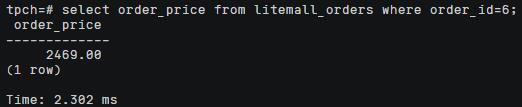


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

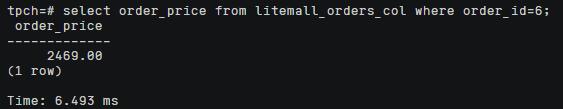


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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？

数据写入方面：

行存储的写入是一次完成，并且能够保证数据的完整性。列存储由于需要把一行记录拆分成单列保存，写入次数明显比行存储多实际时间消耗会更大。所以，行存储在写入上占有很大的优势。

数据读取方面：

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，列存储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

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？

执行插入更新等SQL语句时，行存表效率更高。

执行查询等SQL语句时，列存表效率比行存表效率更高。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

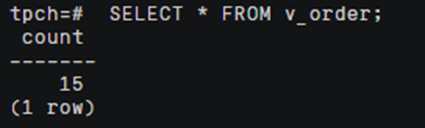
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



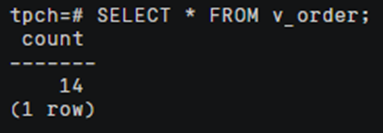
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



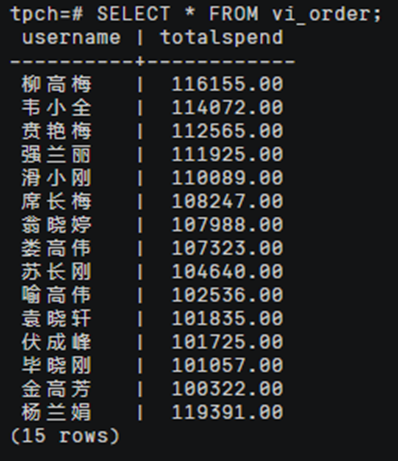
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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已创建的物化视图进行全量更新，而不支持进行增量更新。创建全量物化视图语法和CREATE TABLE AS语法类似。物化视图不支持增删改操作，只支持查询语句。

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刷新。与全量创建物化视图的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。物化视图定义如果为UNION ALL，则其中每个子查询需使用不同的基表。

思考题2：物化视图适用那些使用场景？

全量物化视图：

1. 通常全量物化视图所支持的查询范围与CREATE TABLE AS语句一致。
2. 全量物化视图上支持创建索引。
3. 支持analyze、explain。

增量物化视图：

1. 单表查询语句。
2. 多个单表查询的UNION ALL。
3. 物化视图上支持创建索引。
4. 物化视图支持Analyze操作。

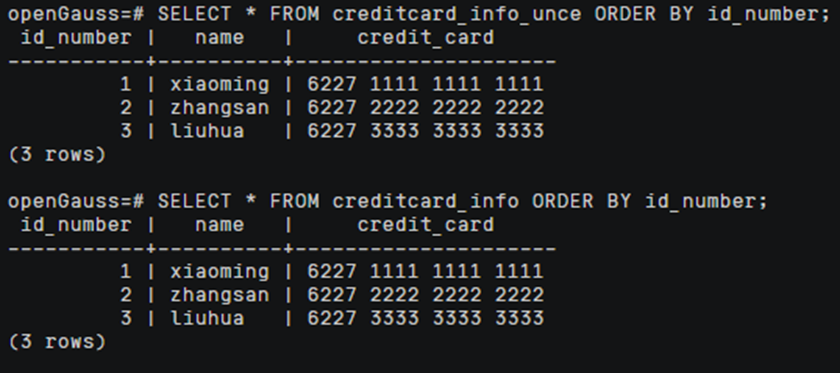
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

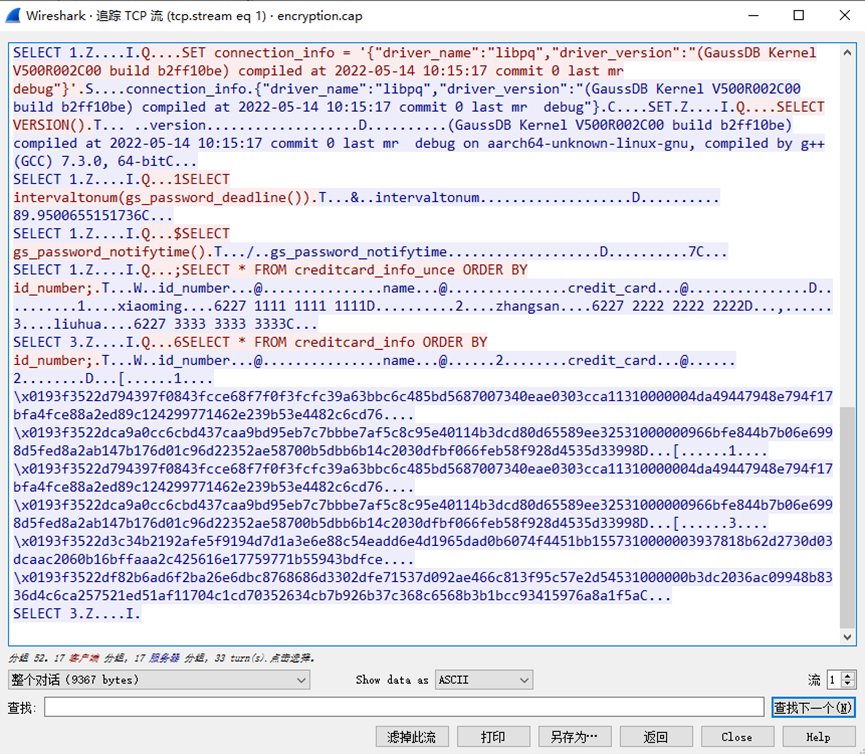
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



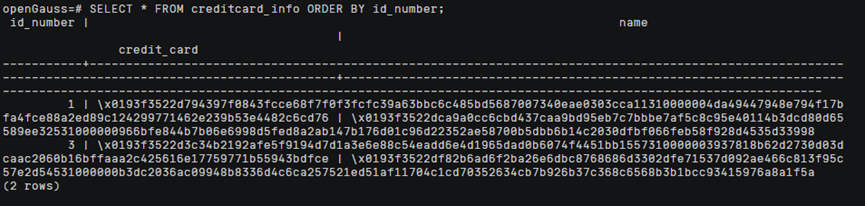
1. 将加密表和非加密表查询结果截图：



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。openGauss全密态数据库会判断返回的数据是否为加密类型，如果是加密类型，则调用加密驱动，根据原始数据类型对数据进行解密。

数据的加解密动作是在客户端完成的。