openGauss 安全体系创新

实践课



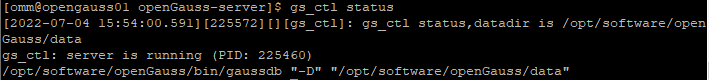
华为技术有限公司

# 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：

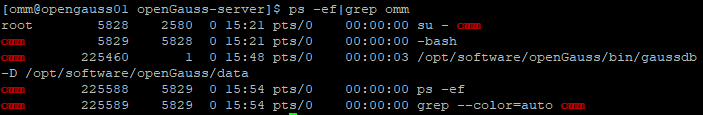
任务一：数据库状态验证

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：满足不同的运行平台，我们Linux发型版本众多，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：方便定制，满足不同的需求，很多时候我们所需要的软件都是可以定制的，我需要什么就安装什么，大多数二进制代码都是一键装全，所以自由度并不高！

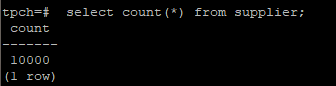
3：方便运维、开发人员维护，我们的源码是可以打包二进制的，但是对于这个软件的打包都会有一份代价不小的额外工作，包括维护，所以如果是源码的话，软件产商会直接维护，但是如果是二进制的话，一般都是Linux发行商提供

# 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：

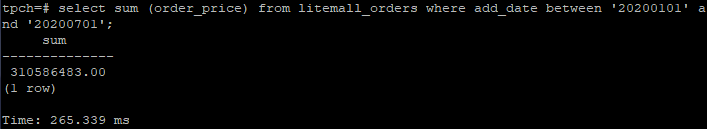
select count(\*) from supplier;;



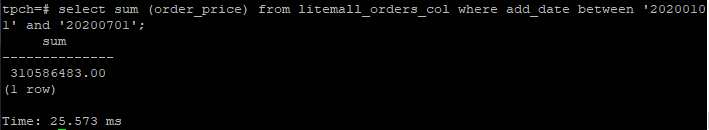
任务二：行存表与列存表执行效率对比

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';

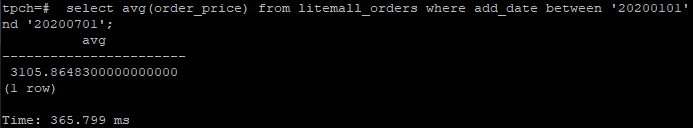


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

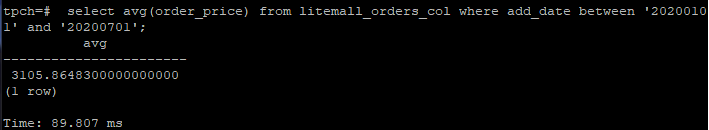


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

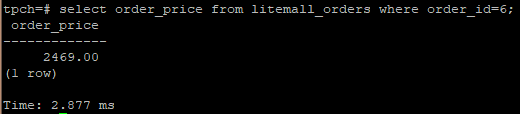


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

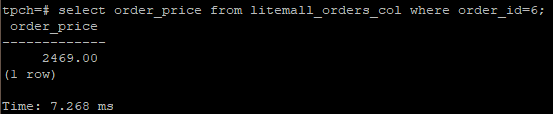


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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？

行存的时候，一行记录的属性值存储在临近的空间，然后接着是下一条记录的属性值。

而列存的时候，单个属性所有的值存储在临近的的空间，即一列的所有数据连续存储的，每个属性有不同的空间。

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？

在数据写入上的对比:

1）行存储的写入是一次完成。写入建立在操作系统的文件系统上，可以保证写入过程的成功或者失败，数据的完整性因此可以确定。

2）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，再加上磁头需要在盘片上移动和定位花费的时间，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。

3）还有数据修改,这实际也是一次写入过程。所以，数据修改也是以行存储占优。

在数据读取上的对比:

1）行存储通常将一行数据完全取出，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，消除冗余列的过程通常是在内存中进行的。

2）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，查找内容连续存储，特别适合投影。

3） 两种存储的数据分布。由于列存储的每一列数据类型是同质的，不存在二义性问题。比如说某列数据类型为整型(int)，那么它的数据集合一定是整型数据。这种情况使数据解析变得十分容易。相比之下，行存储则要复杂得多，因为在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，这个操作很消耗CPU，增加了解析的时间。所以，列存储的解析过程更有利于分析大数据。

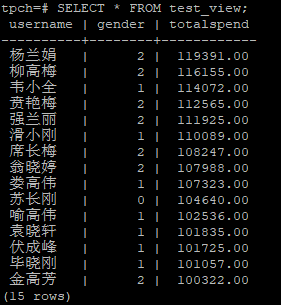
4）从数据的压缩以及更性能的读取来对比。同一列的数据，数据类型一致，列存的模式下就适合数据压缩，不同的列可以采用不同的压缩算法，压缩存储就会带来IO性能的提升。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

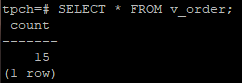
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

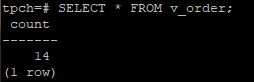
SELECT \* FROM test\_view;



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？

全量物化视图在刷新时会根据全部数据重新生成新的视图，而增量物化视图在刷新时会根据数据的变化在原先视图中进行修改生成新的视图。

思考题2：物化视图适用那些使用场景？

当我们既需要明细查询，又需要固定维度查询的场景时。

仅涉及表中很小一部分列或者行的查询场景。

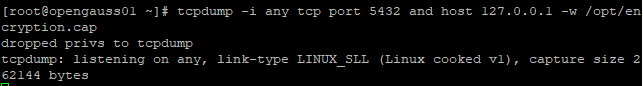
包含耗时的聚合查询。

需要匹配不同的前缀索引。

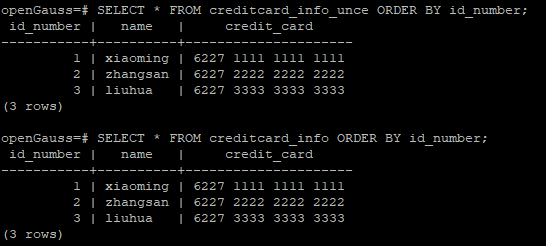
# 关卡四：openGauss密态数据库特性应用

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：



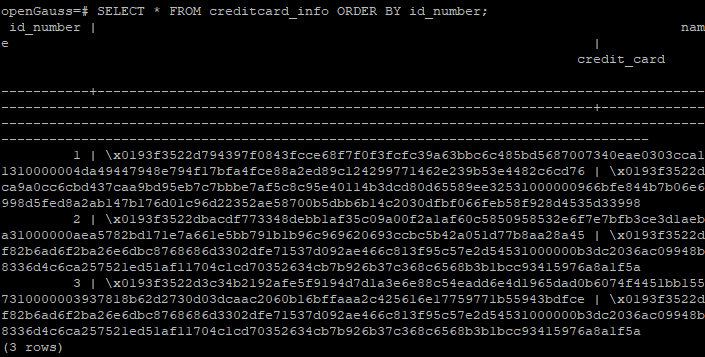
1. 将加密表和非加密表查询结果截图：



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？

数据实际存储在物理磁盘上的时候是明文进行存储的。数据在客户端完成加解密，然后将密文传送到服务端，对加密内容进行查询时，客户端需要将查询内容也进行加密发送至服务端进行查询。