openGauss 安全体系创新

实践课



华为技术有限公司

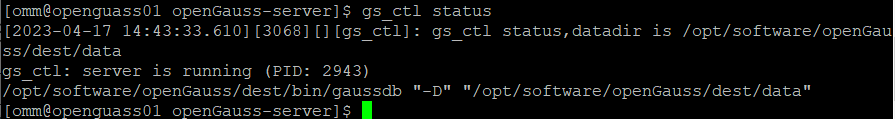
# 关卡一、openGauss数据安装及基本操作

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

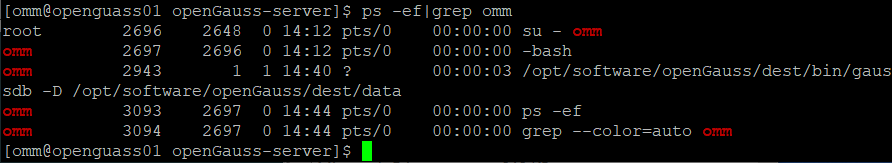
[omm@openguass01 openGauss-server]$ gs\_ctl status



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）

[omm@openguass01 openGauss-server]$ ps -ef|grep omm



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？

答：源码编译、安装数据库是指：自己去网上下载源码包，然后解压安装。

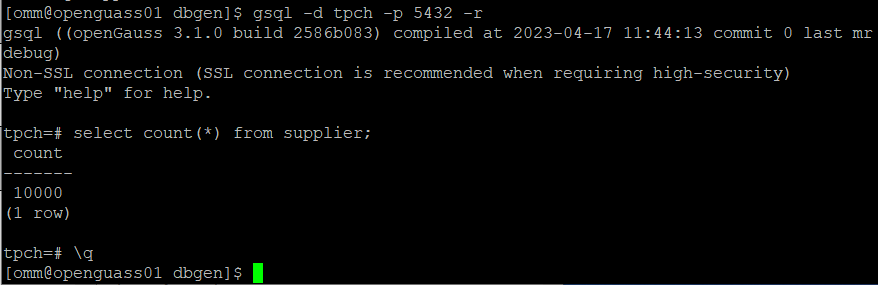
在编译安装过程中，可以设定参数，按照需求，进行安装，并且可以自己选择安装的版本，灵活性比较大，兼容性更强。

与平台无关，可按需定制编译，最灵活、性能最好；可以安装到自己想要安装的目录、并且支持单台服务器安装多个版本的MySQL。

# 关卡二、openGauss数据导入及行存列存

任务一：数据初始化验证

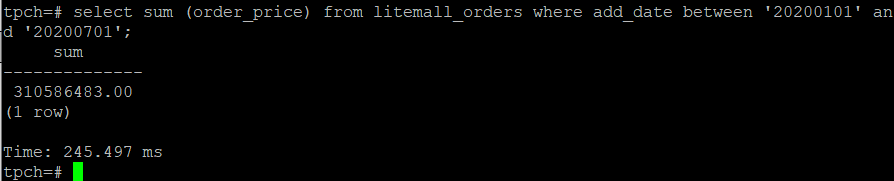
1. 查询supplier表的行数，并将结果进行图：

select count(\*) from supplier;

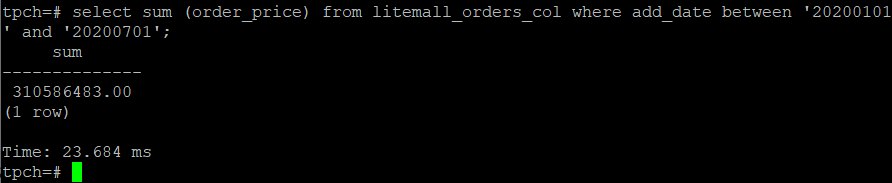
任务二：行存表与列存表执行效率对比

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';

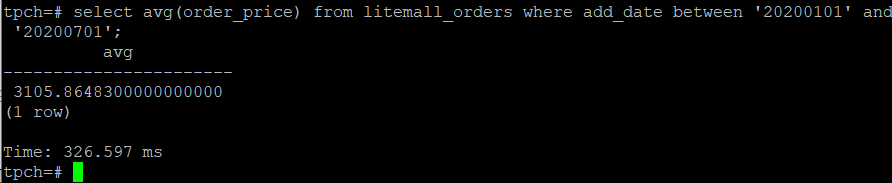


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

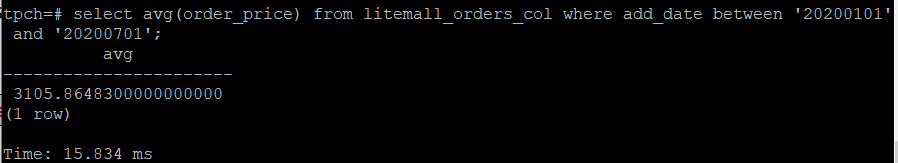


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

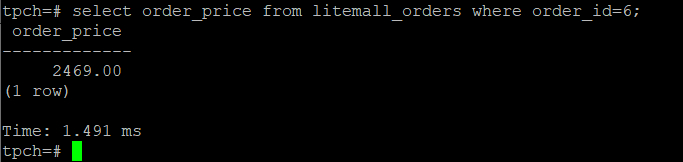


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

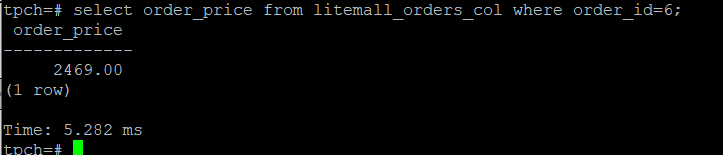


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

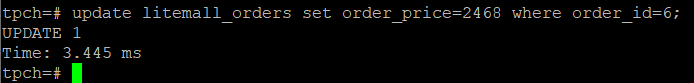


select order\_price from litemall\_orders\_col where order\_id=6;

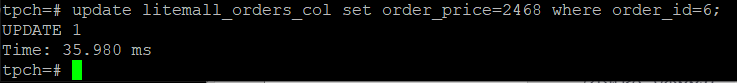


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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？

二者对数据的存储方式不同：行式存储下一张表的数据都放在一起，但列式存储下被分开保存；

在写入时，行存储一次完成，可以确定数据的完整性。列存储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。行存储在写入上占有很大的优势；

在修改数据时，行存储是在指定位置写入一次，列存储是将磁盘定位到多个列上分别写入。行存储在数据修改也是占优的；

在读取数据时，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，则会存在冗余列）。列存储每次读取的数据是集合中的一段或者全部，由于存储的数据同质，使容易数据解析，行存储则复杂的多（在一行记录中保存了多种类型的数据，数据解析需要多种数据类型之间频繁转换），所以列存储的解析过程中更有利于分析大数据，并且在读取过程，不会产生冗余数据。

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？

答：在执行下列SQL操作时，行存表效率更高。

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相关。

在执行下列SQL操作时，列存表效率更高。

1. 在数据列中查找数据；
2. 对数据列进行运算
3. 对大型数据的查询分析；
4. 需要频繁聚集。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

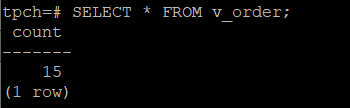
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



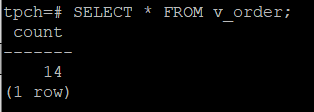
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



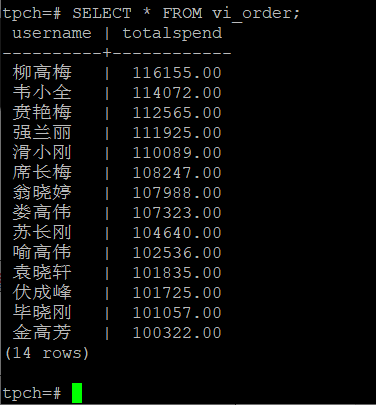
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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创建好的物化视图做全量更新，而不支持做增量更新。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

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。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？

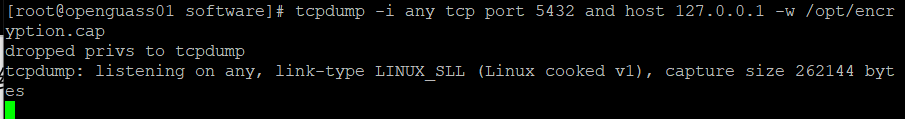
物化视图实际上就是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对SQL语句的查询，性能并没有实际上提高。而物化视图实际上就是存储SQL所执行语句的结果，起到缓存的效果。

物化视图的应用场景有两种：1、用于查询优化 2、用于高级复制。

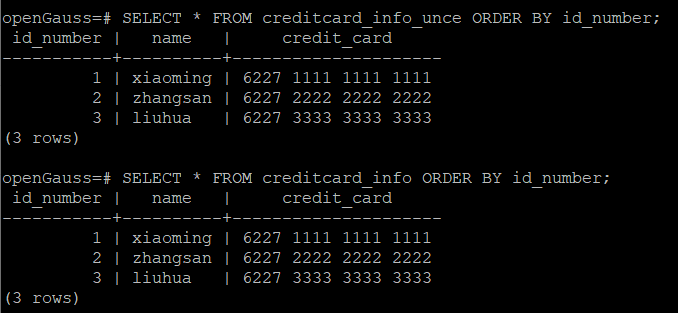
# 关卡四：openGauss密态数据库特性应用

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：



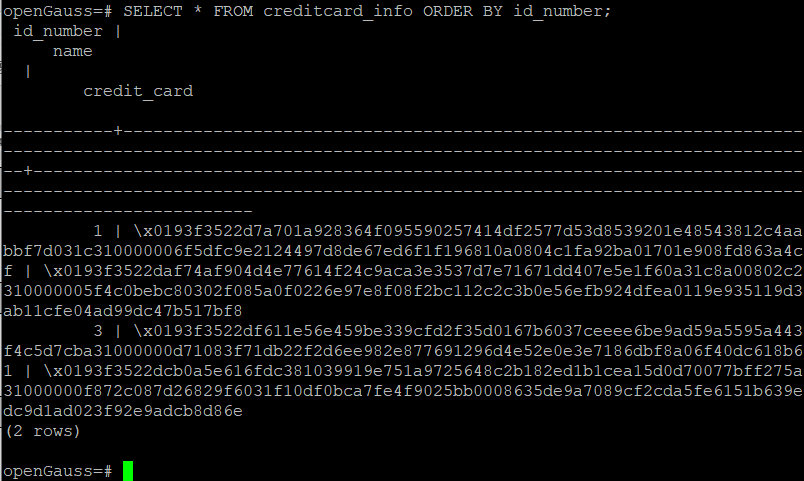
1. 将加密表和非加密表查询结果截图：



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？

答：数据实际存储在物理磁盘上的时候是密文；

数据的加解密的动作是在客户端完成的。