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

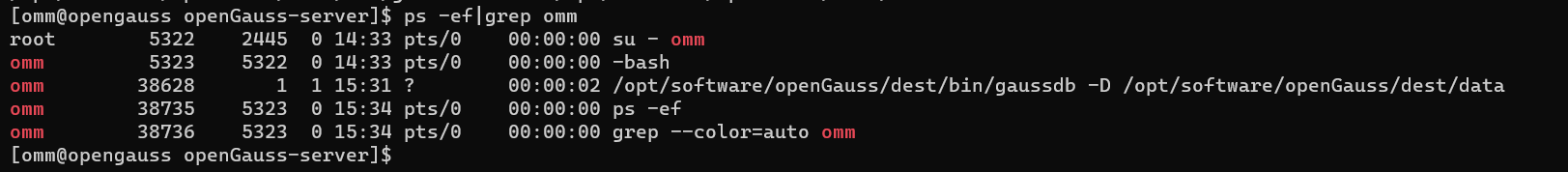
1. 查询数据库状态成功截图



gs\_ctl status 命令

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）



ps -ef|grep omm 命令

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.能够定制化需求：通过源码编译，可以对数据库进行定制化设置，比如更改默认的配置选项、添加或删除一些功能或插件等，以满足特定的需求。

2.能够有更高的性能：通过编译可以将数据库的代码与特定的系统和硬件架构进行优化，从而提高数据库的性能。

3.安全性高：通过源码编译和安装可以确保数据库的安全性，因为用户可以对源代码进行审查和验证，从而保证安装的是可靠的代码。

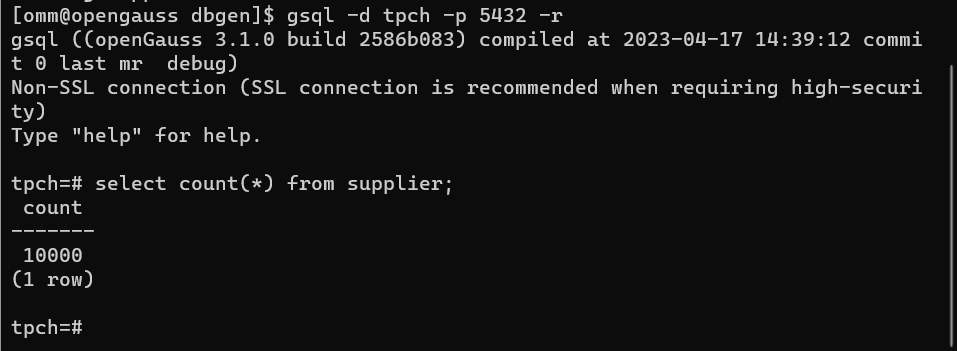
4.学习和探索：通过源码编译和安装数据库可以学习和探索数据库内部工作原理，从而更深入地了解数据库的结构和运作方式。

# 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：

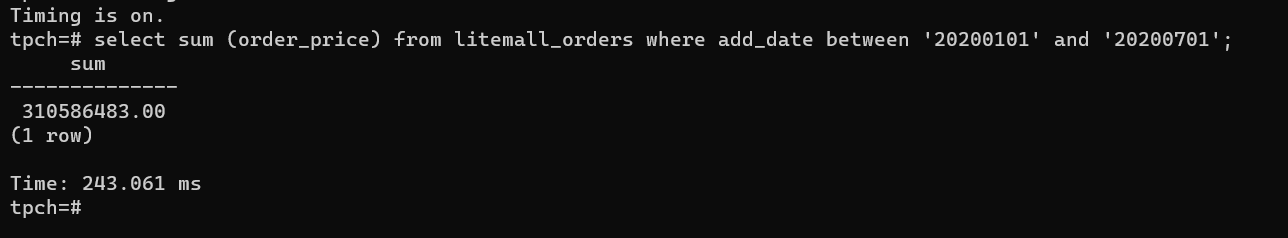
select count(\*) from supplier;;



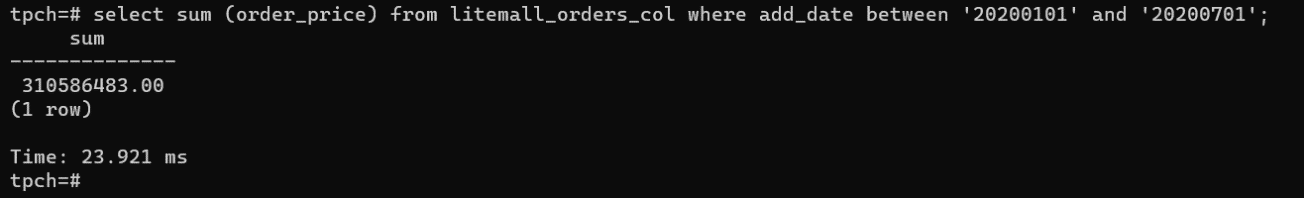
任务二：行存表与列存表执行效率对比

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';

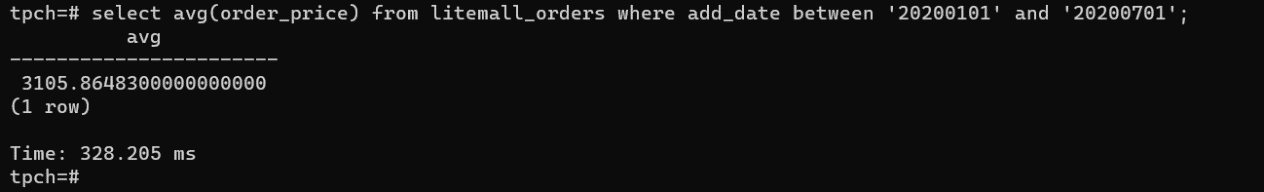


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

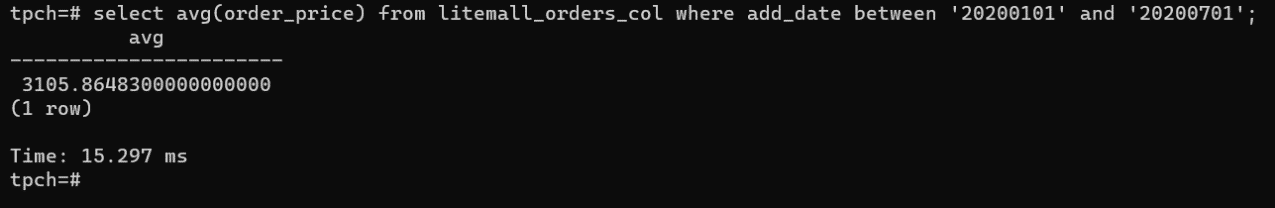


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

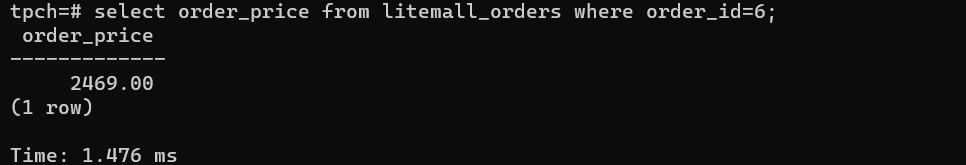


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

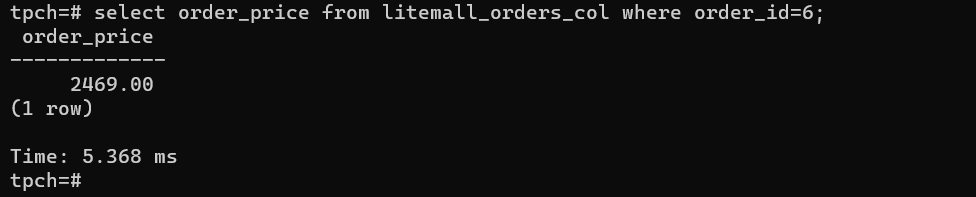


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

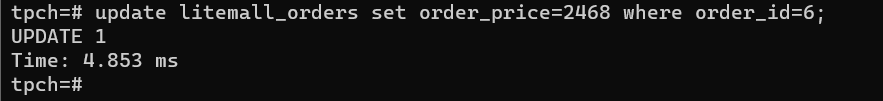


select order\_price from litemall\_orders\_col where order\_id=6;

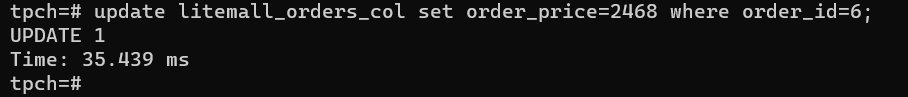


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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？

答：因为它们在存储和访问数据的方式上有所不同。

1. 行存表将每一行的数据都存储在一起，因此在执行查询时需要读取整行数据，即使只需要其中的一部分数据也要将整行数据都读取出来，这会导致查询时需要读取的数据量较大，而且在数据量较大时性能会更差。此外，由于行存表的数据存储方式导致每一行的数据都会占用一定的存储空间，因此在存储大量数据时，行存表需要占用更多的存储空间。
2. 列存表将同一列的数据存储在一起，查询时只需要读取所需要的列数据，可以减少需要读取的数据量，从而提高查询性能。此外，列存表存储数据时只需要存储列数据，可以减少存储空间的占用，因此在存储大量数据时，列存表需要的存储空间相对较少。

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：根据行存表和列存表存储和查询方式的不同：

行存表在执行以下类型的 SQL 语句时效率更高：

1. 事务处理：行存表可以很好地支持事务处理，因为它们存储数据的方式可以确保事务的一致性和完整性。
2. 单行查询：当只需要查询一行数据时，行存表的效率往往更高，因为只需要读取一行数据，不需要扫描整个表格。
3. 数据更新：当需要频繁更新表格中的数据时，行存表的效率往往更高，因为更新单行数据时只需要修改一行数据即可，不需要修改整个列的数据。

列存表在执行以下类型的 SQL 语句时效率更高：

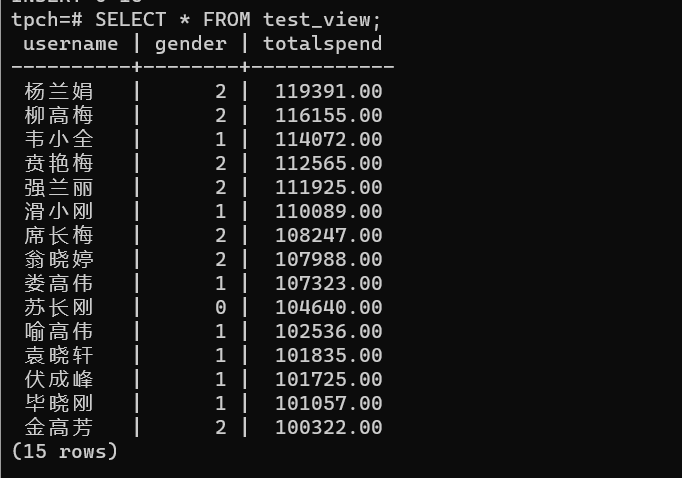
1. 复杂查询：当需要查询大量数据时，列存表的效率往往更高，因为它们只需要读取所需的列数据，不需要读取整个行的数据。
2. 聚合查询：当需要对数据进行聚合计算时，列存表的效率往往更高，因为它们可以利用列数据的存储方式快速进行聚合计算。
3. 数据分析：当需要进行数据分析时，列存表的效率往往更高，因为它们可以快速扫描列数据进行分析，而不需要读取整个行的数据。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

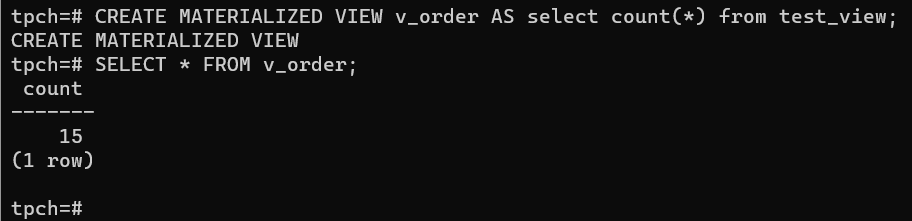
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



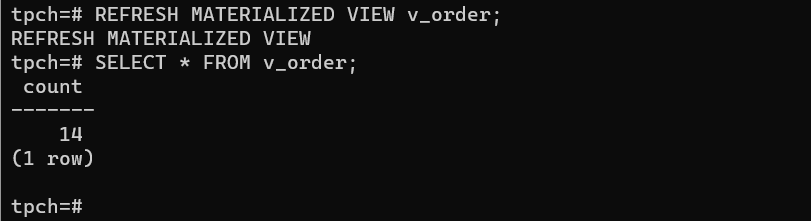
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



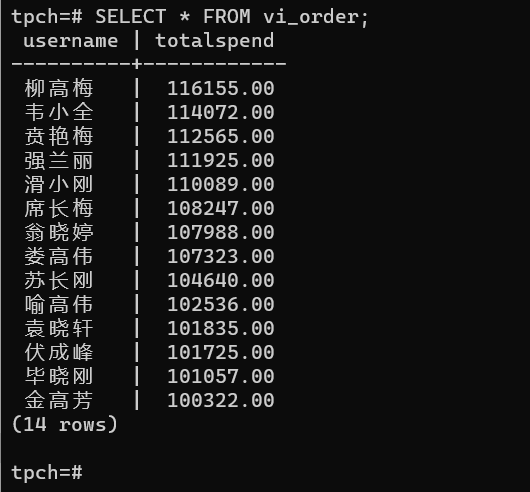
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



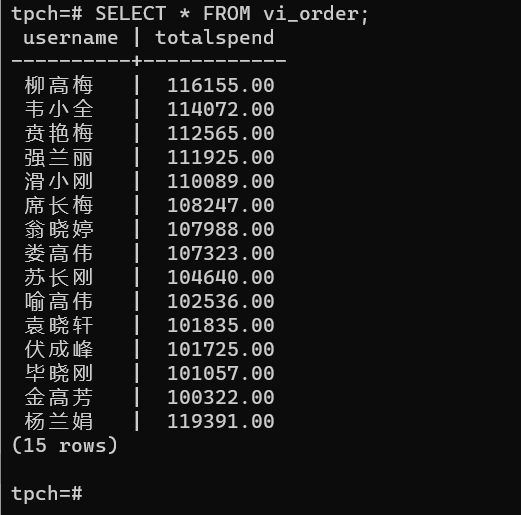
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

答：全量物化视图和增量物化视图的主要差别在刷新方式和刷新时间。

1. 全量物化视图是指将查询结果全部存储在物化视图中的视图。全量物化视图在创建时需要将查询结果全部计算并存储到物化视图中，因此创建时间可能会较长。当源表的数据发生变化时，需要重新计算全量物化视图的所有数据，因此刷新时间相对较长，这可能会对性能产生一定的影响。
2. 增量物化视图只将源表中新增、修改或删除的数据存储在物化视图。增量物化视图在创建时不需要将整个查询结果计算并存储到物化视图中，只需要存储源表中的部分数据，因此创建时间相对较短。当源表的数据发生变化时，只需要计算新增、修改或删除的数据，并更新物化视图中相应的数据，因此刷新时间相对较短，这可以提高性能并减少资源占用。

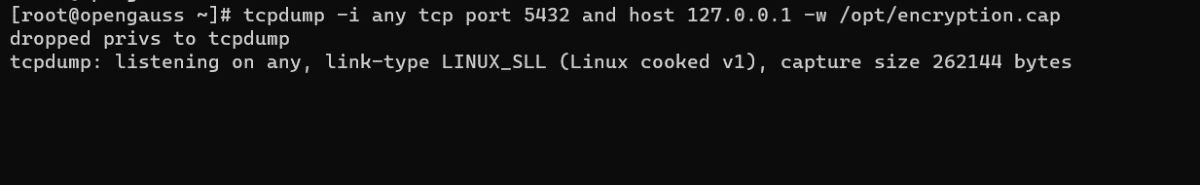
思考题2：物化视图适用那些使用场景？

1. 查询结果稳定且数据变化较少的情况：如果某个查询的结果在一段时间内基本不会变化，而且查询结果较大，这时候可以使用物化视图来缓存查询结果，从而提高查询性能。
2. 复杂查询的情况：当需要进行复杂的多表联接或者聚合操作时，查询语句往往会非常复杂，而且执行速度也比较慢。这时候可以使用物化视图来缓存这个查询的结果，以提高查询性能。
3. 分布式环境下的查询：当需要在分布式环境下查询数据时，由于数据分布在多个节点上，查询速度往往较慢。这时候可以在每个节点上创建物化视图，以缓存查询结果，从而提高查询性能。

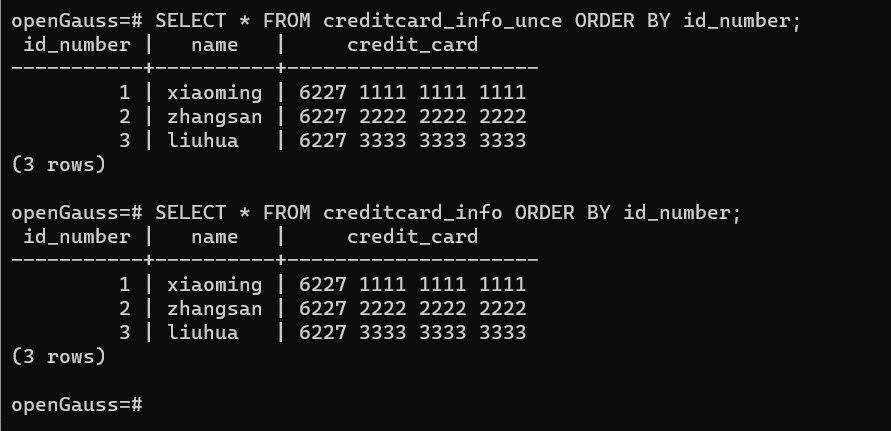
# 关卡四：openGauss密态数据库特性应用

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：为了保证数据的安全性，数据实际以密文的形式进行存储。对于数据的加解密动作，这通常取决于使用的加密技术以及加密的具体实现方式。在一些场景下，数据的加解密动作可能是在客户端完成的，例如在客户端使用 SSL/TLS 加密传输数据时，客户端会对数据进行加密，然后再将加密后的数据传输给服务端。在其他一些场景下，加解密动作可能是在服务端完成的，例如在数据库层面使用加密技术对数据进行加密存储时，加解密动作通常是在服务端完成的。