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

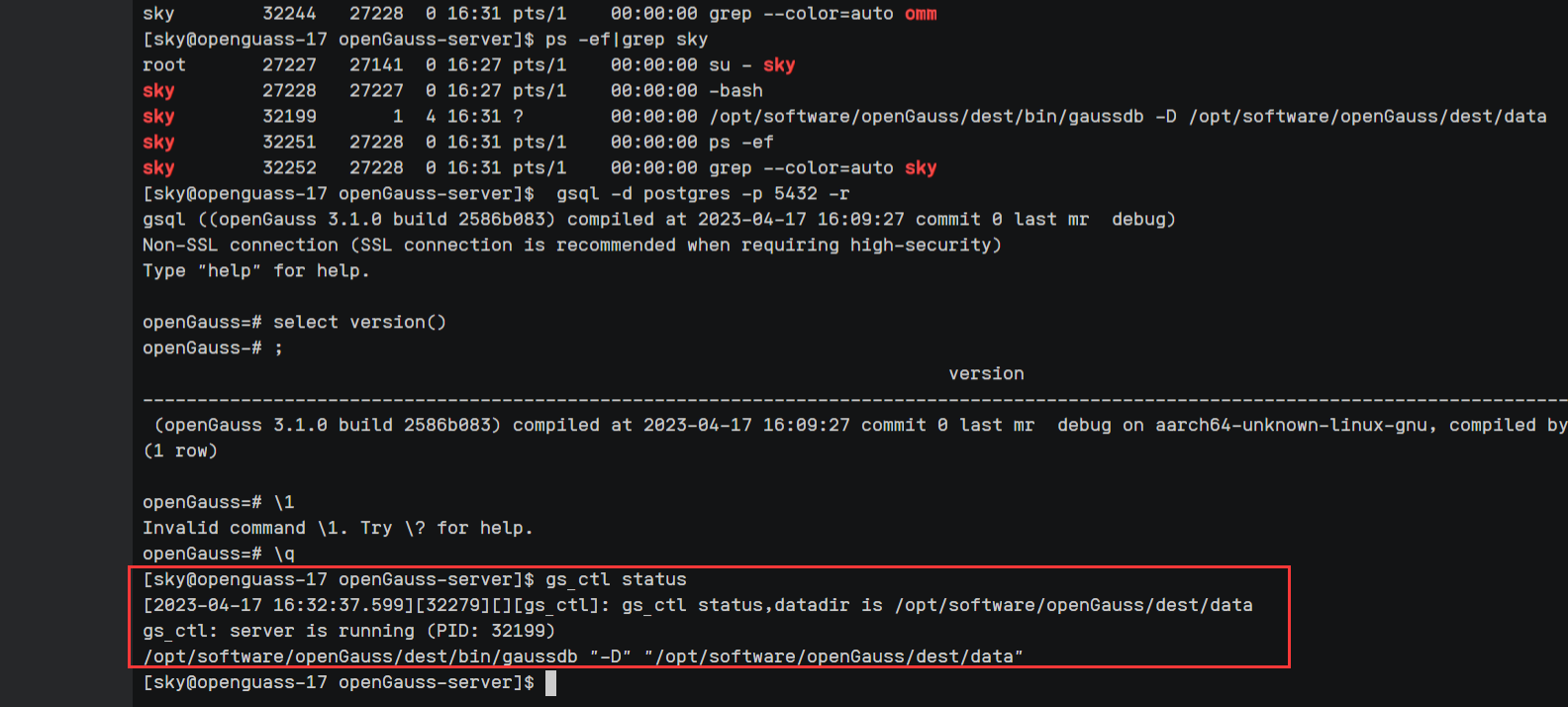


图 1 查询数据库状态成功截图

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）

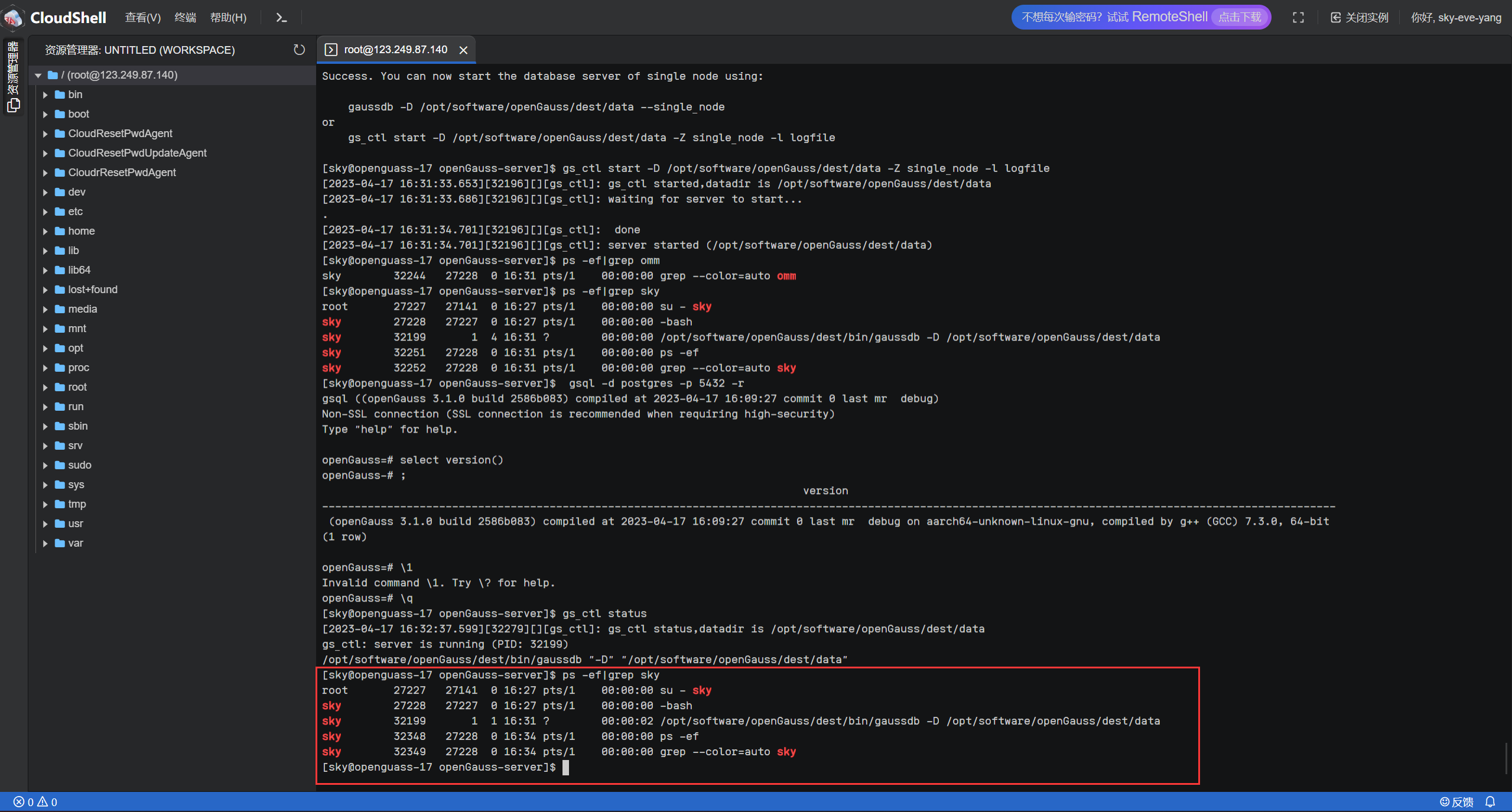


图 2 数据库服务进程截图

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？

**源码编译**是指将软件的源代码转换成可执行程序的过程。在源码编译过程中，用户需要先获取软件的源代码，然后通过编译器将其转换成机器语言，最终生成可执行文件。与之相对应的是二进制包安装，即直接使用已经编译好的二进制文件进行安装。相比之下，源码编译需要更多的时间和精力，但是可以提供更高度定制化和灵活性。

**安装数据库**的方式有以下几种：

1. 二进制包安装：即直接使用已经编译好的二进制文件进行安装。这种方式通常比较简单快捷，适合于一般用户或者初学者。

2. 源码编译安装：即将软件的源代码转换成可执行程序的过程。这种方式需要更多的时间和精力，但是可以提供更高度定制化和灵活性。

3. 容器化部署：即使用容器技术（如Docker）将数据库打包成镜像，然后在不同的环境中进行部署。这种方式可以提供更好的可移植性和隔离性。

4. 云服务部署：即使用云服务提供商（如AWS、阿里云等）提供的数据库服务进行部署。这种方式可以免去用户自行搭建硬件环境和维护系统的繁琐工作，同时也可以根据实际需求灵活调整资源规模。

**源码编译安装数据库**的好处在于用户可以更加灵活地控制openGauss的安装和配置过程，以满足不同场景下的需求。具体来说，源码编译安装可以带来以下几个优点：

1. 定制化：源码编译安装可以让用户根据自己的需求进行定制和优化。例如，用户可以根据自己的硬件环境和应用场景选择不同的编译选项，以达到最佳性能。

2. 安全性：源码编译安装可以避免使用第三方二进制包带来的潜在风险。用户可以自行验证代码完整性和可信度，从而提高系统的安全性。

3. 可维护性：源码编译安装可以让用户更好地理解系统架构和运行机制，从而更好地进行维护和升级。

4. 社区支持：开源软件通常有一个庞大的社区支持体系，用户可以通过参与社区活动获取技术支持、分享经验等。

即通过源码编译安装数据库是一种更加灵活、可定制、可控、可维护、可靠、可扩展的方式。虽然相对于二进制包安装来说需要花费更多时间和精力，但是对于需要高度定制化的应用场景来说，源码编译安装是一个更好的选择。

# 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;;

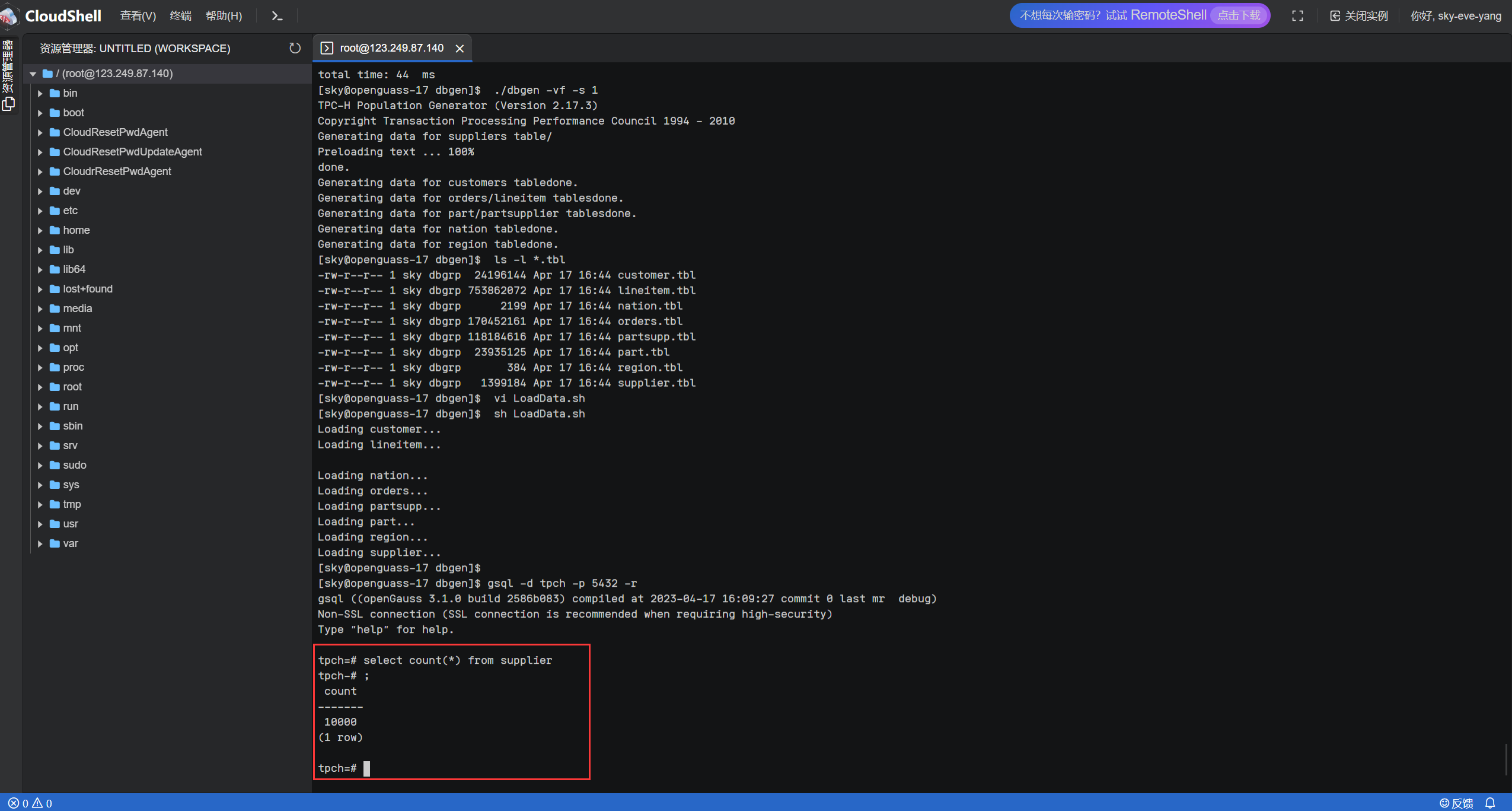


图 3 查询supplier表的行数

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';

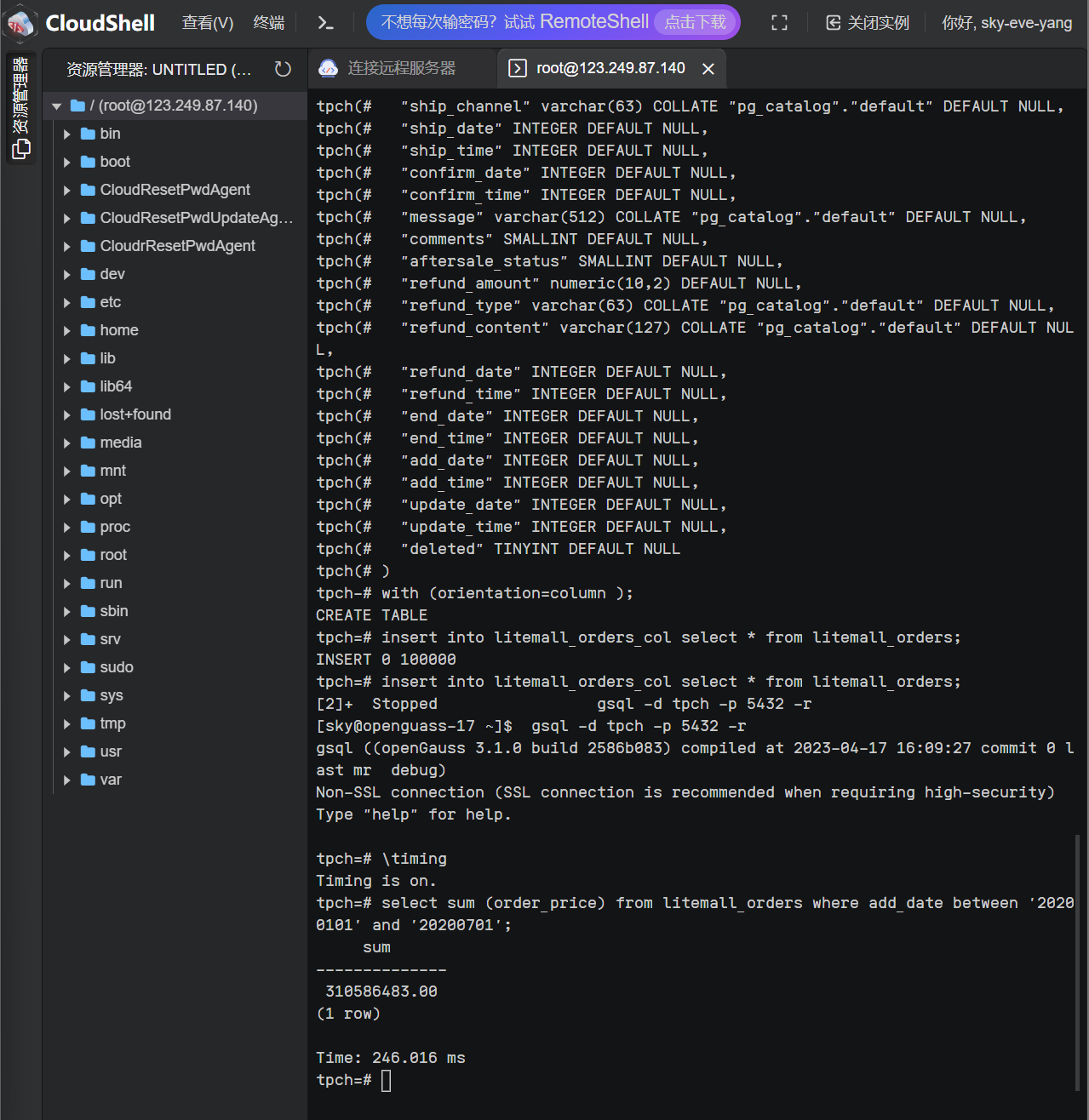


图 4 行存表查询order\_price总和的执行效率

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

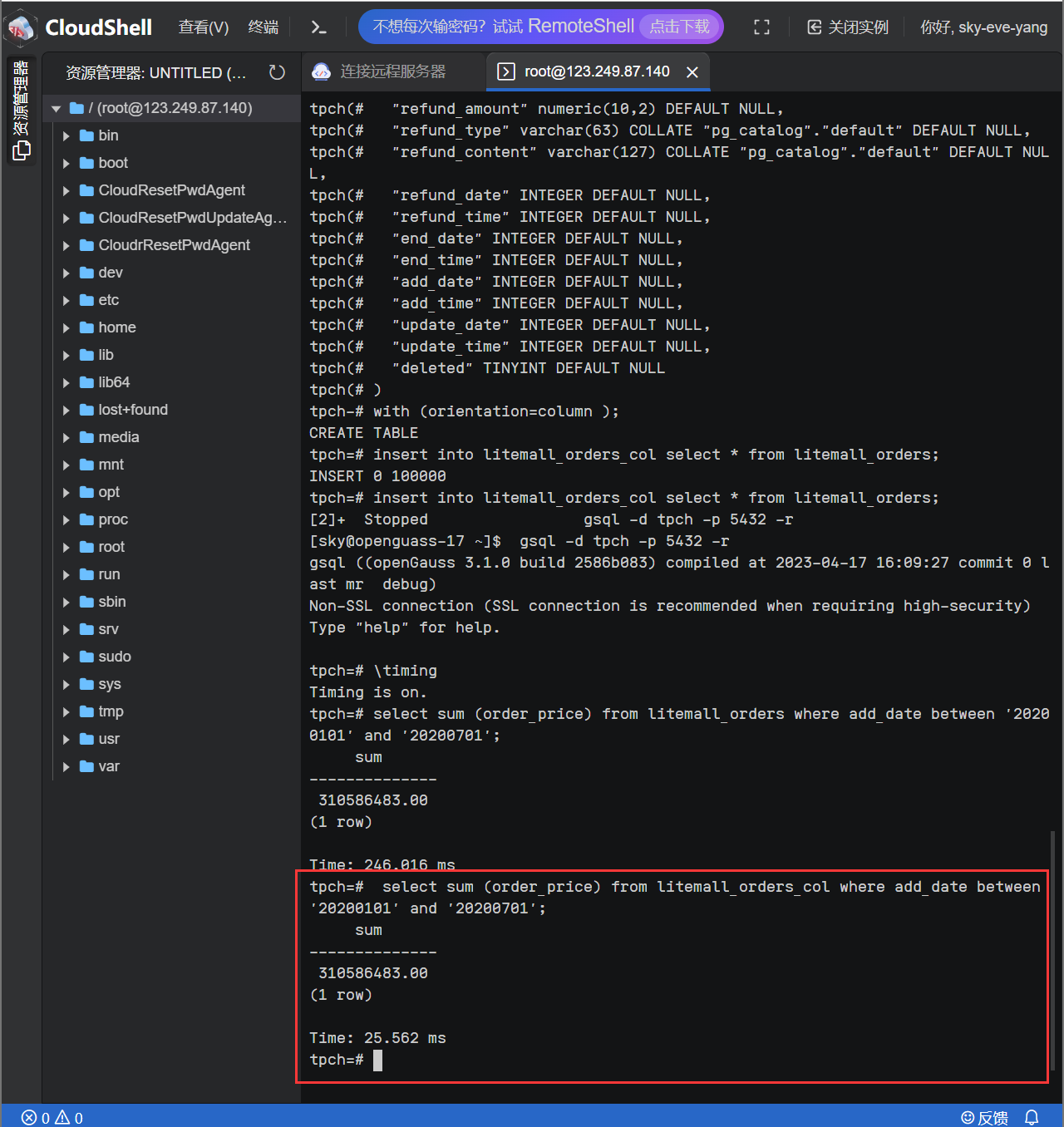


图 5 列存表查询order\_price总和的执行效率

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';

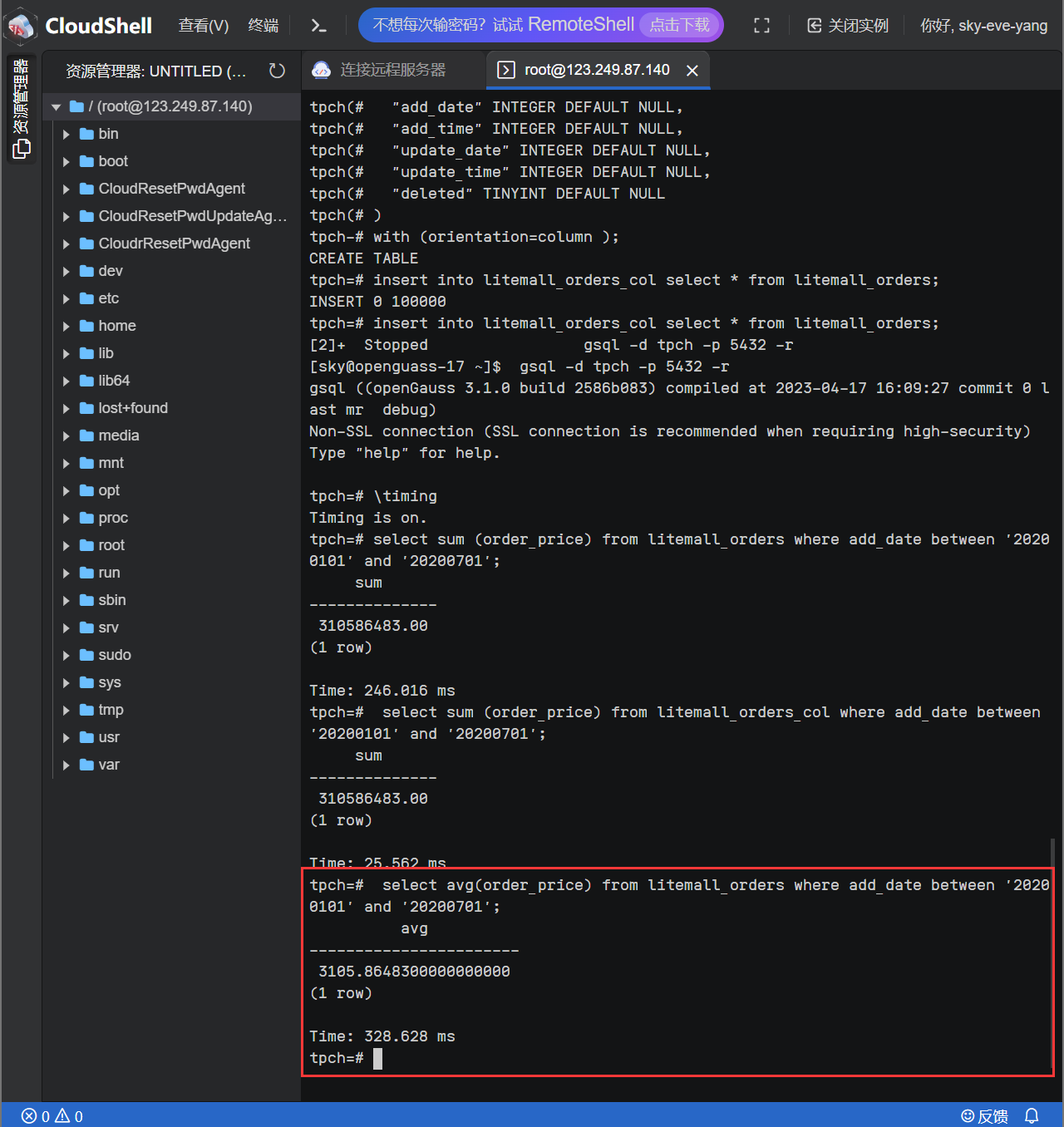


图 6 行存表查询order\_price平均值的执行效率

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

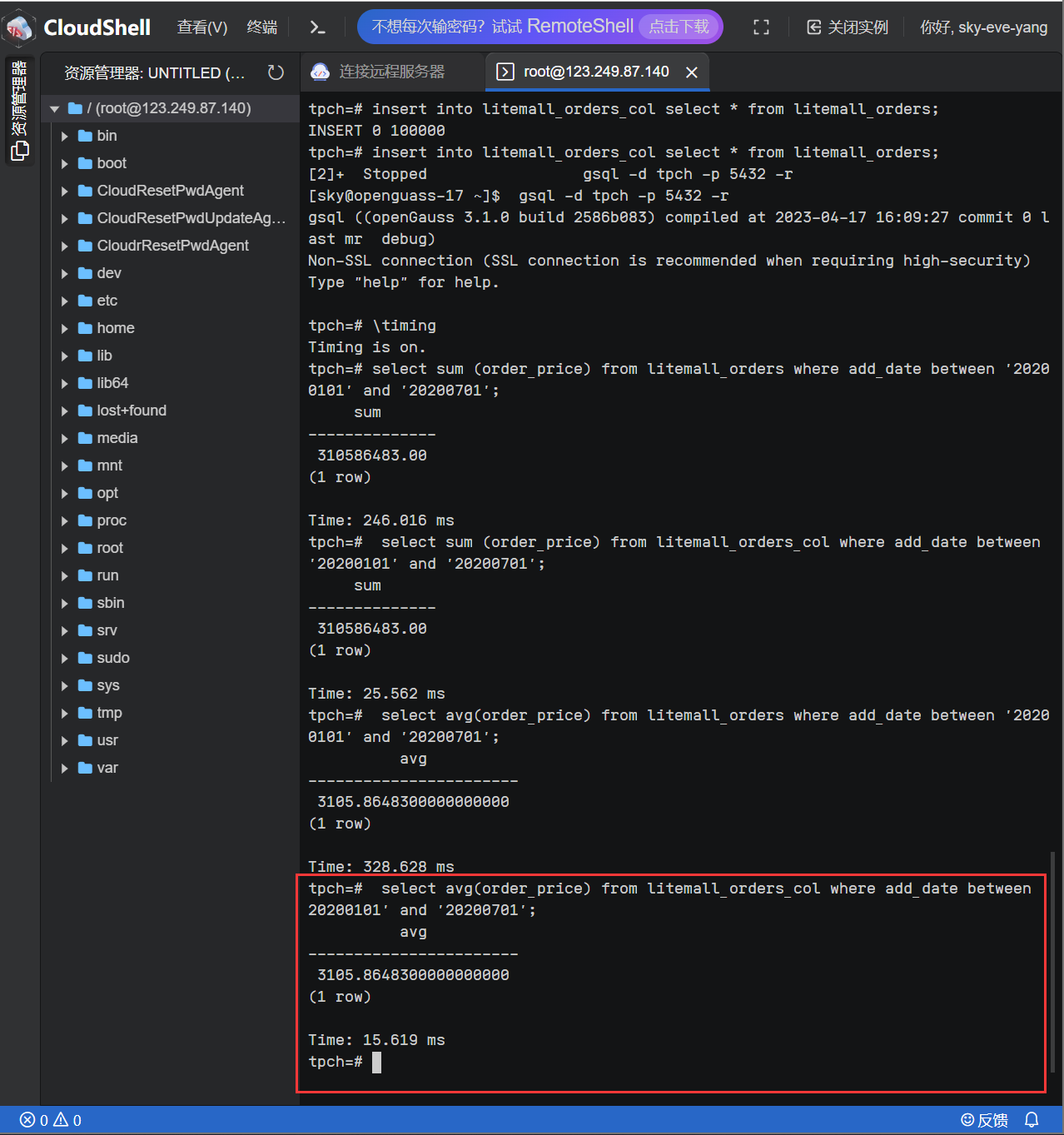


图 7 列存表查询order\_price平均值的执行效率

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;

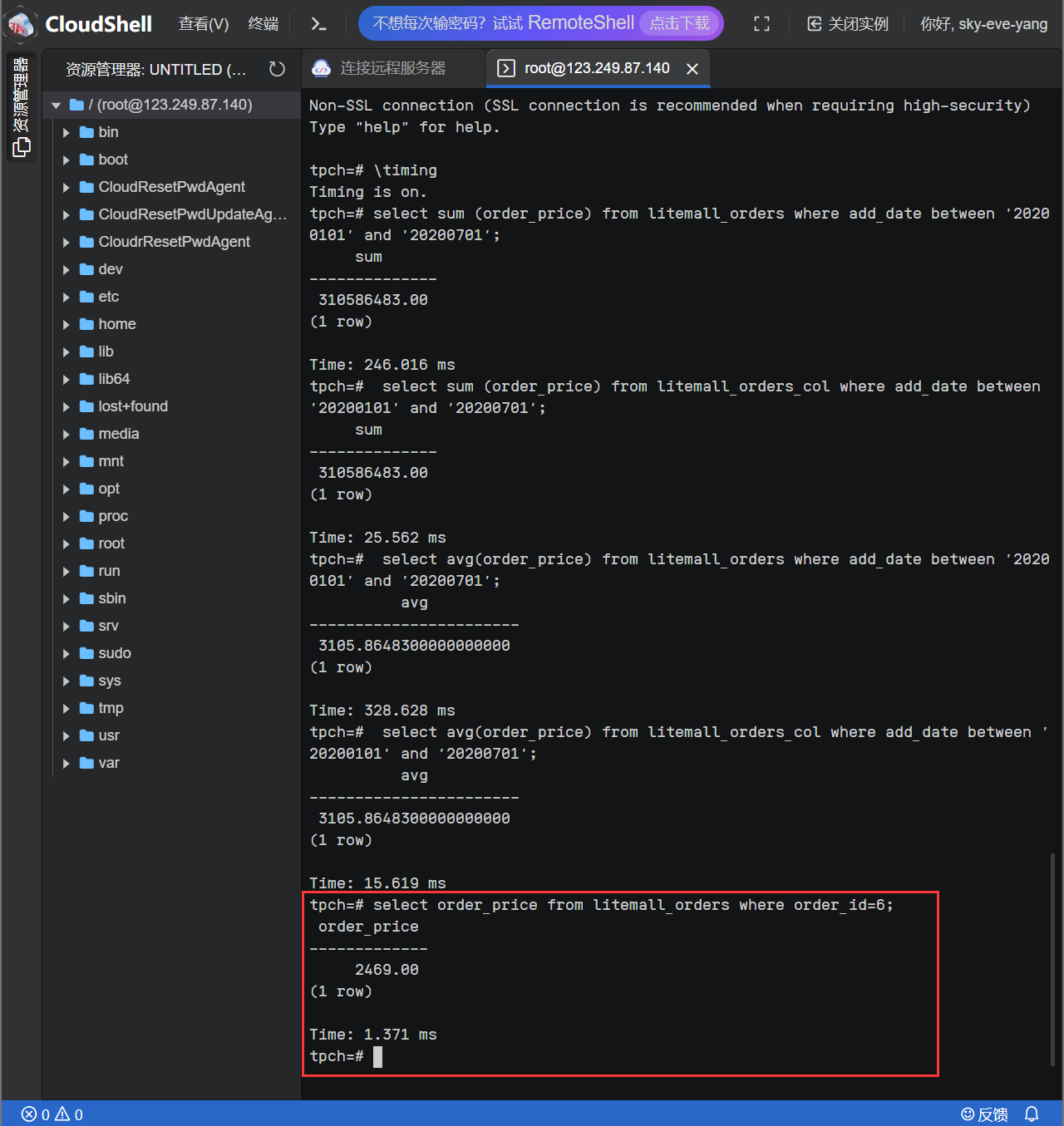


图 8 行存表查询order\_id为6时order\_price值的执行效率

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

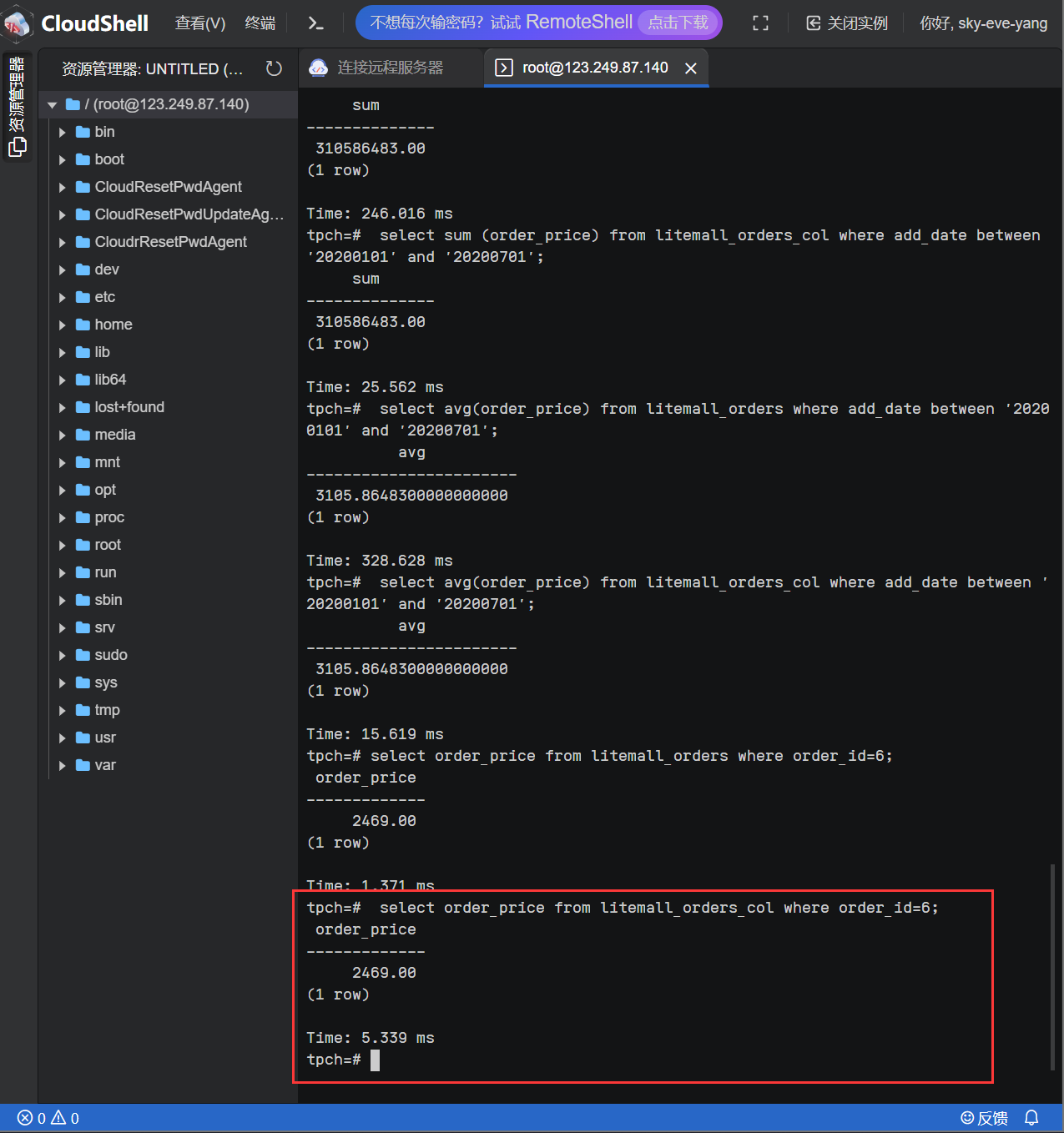


图 9 列存表查询order\_id为6时order\_price值的执行效率

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;

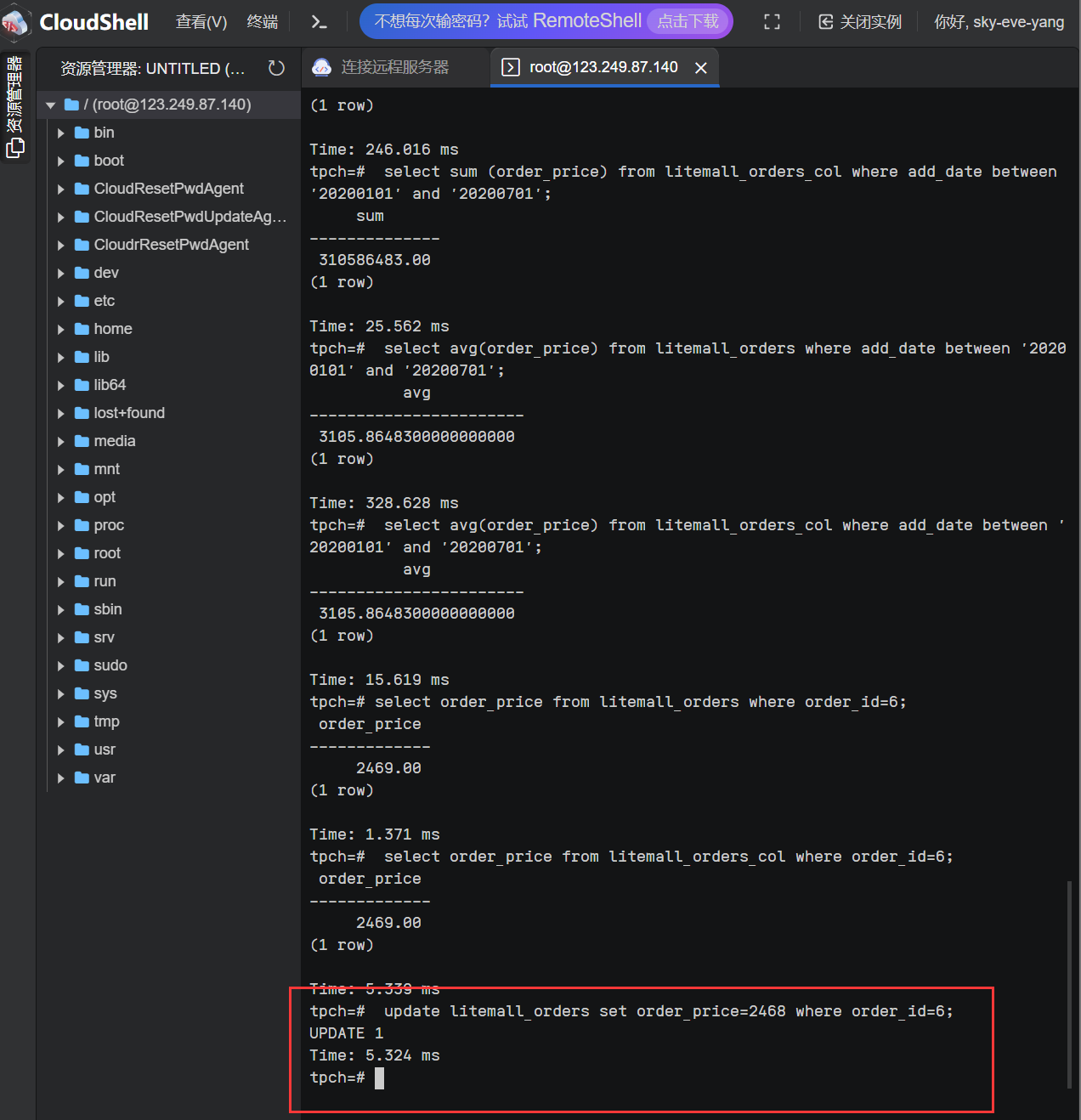


图 10 行存表修改order\_id为6时order\_price值的执行效率

update litemall\_orders\_col set order\_price=2468 where order\_id=6;

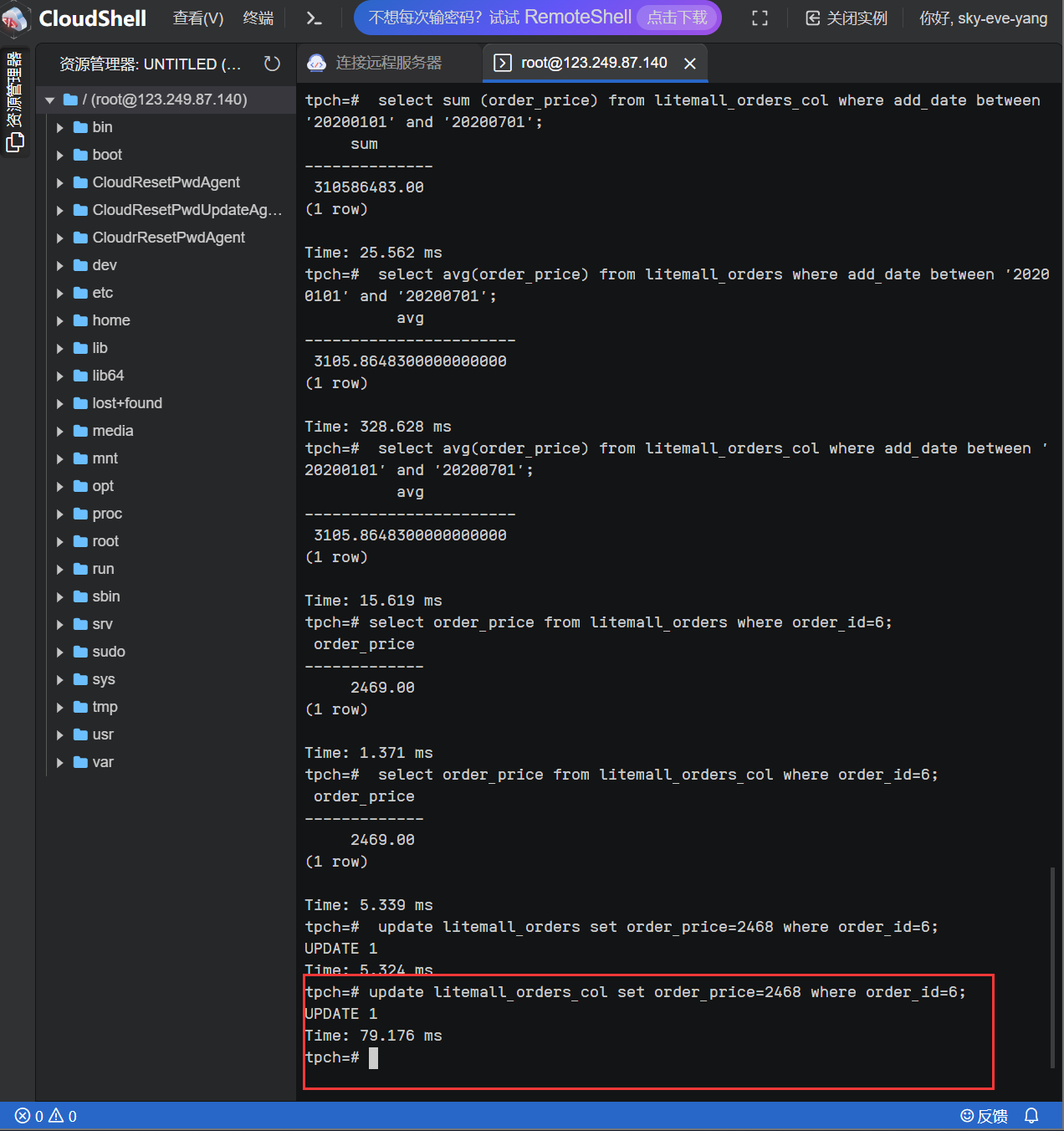


图 11 列存表修改order\_id为6时order\_price值的执行效率

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. 数据访问模式不同：行存表和列存表在数据访问模式上存在差异。行存表是按照行的方式进行存储，即将一条记录的所有字段都存储在一起。而列存表则是按照列的方式进行存储，即将同一列的数据都放在一起。因此，在执行查询操作时，如果需要访问多个字段，则行存表需要读取整个记录，而列存表只需要读取所需字段对应的列。这种数据访问模式上的差异会导致在某些情况下，列存表比行存表更快地执行查询操作。

2. 存储结构不同：行存表和列存表在物理上的存储结构也存在差异。行存表采用堆积式结构进行数据组织，而列存表采用压缩式结构进行数据组织。由于压缩式结构可以减少磁盘I/O次数和网络传输量，因此，在某些情况下，列存表比行存表更快地执行查询操作。

3. SQL语句类型不同：不同类型的SQL语句对于存表和列存表的执行效率也有影响。例如，在查询某个字段时，如果该字段在列存表中被压缩了，则需要先解压缩才能进行查询操作。这会增加查询时间。而对于一些聚合操作（如SUM、AVG等），由于列存表中每个字段都单独保存，因此可以更快地进行计算。

综上所述，行存表和列存表在执行相同SQL语句时所需时间不同主要是由于它们采用了不同的数据访问模式、存储结构和SQL语句类型。对于需要频繁进行字段处理的场景，建议采用列存表的方式，见图4和图5、图6和图7的对比；对于需要频繁进行表记录中多个字段处理的场景，建议采用行存表的方式，见图8和图9、图10和图11的对比。

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？

在执行以下类型的SQL语句时，行存表的效率更高：

1. 需要查询所有字段或大部分字段的SQL语句。因为在行存表中，每条记录的所有字段值都存储在一起，因此可以更快地获取所有字段的值。

2. 需要进行频繁的更新操作或者插入操作的SQL语句。因为在行存表中，每条记录都是连续存储的，因此可以更快地进行插入和更新操作。

而在执行以下类型的SQL语句时，列存表的效率更高：

1. 需要查询特定字段或少量字段的SQL语句。因为在列存表中，每个字段都单独保存，并且被压缩了，因此可以更快地获取特定字段或少量字段的值。

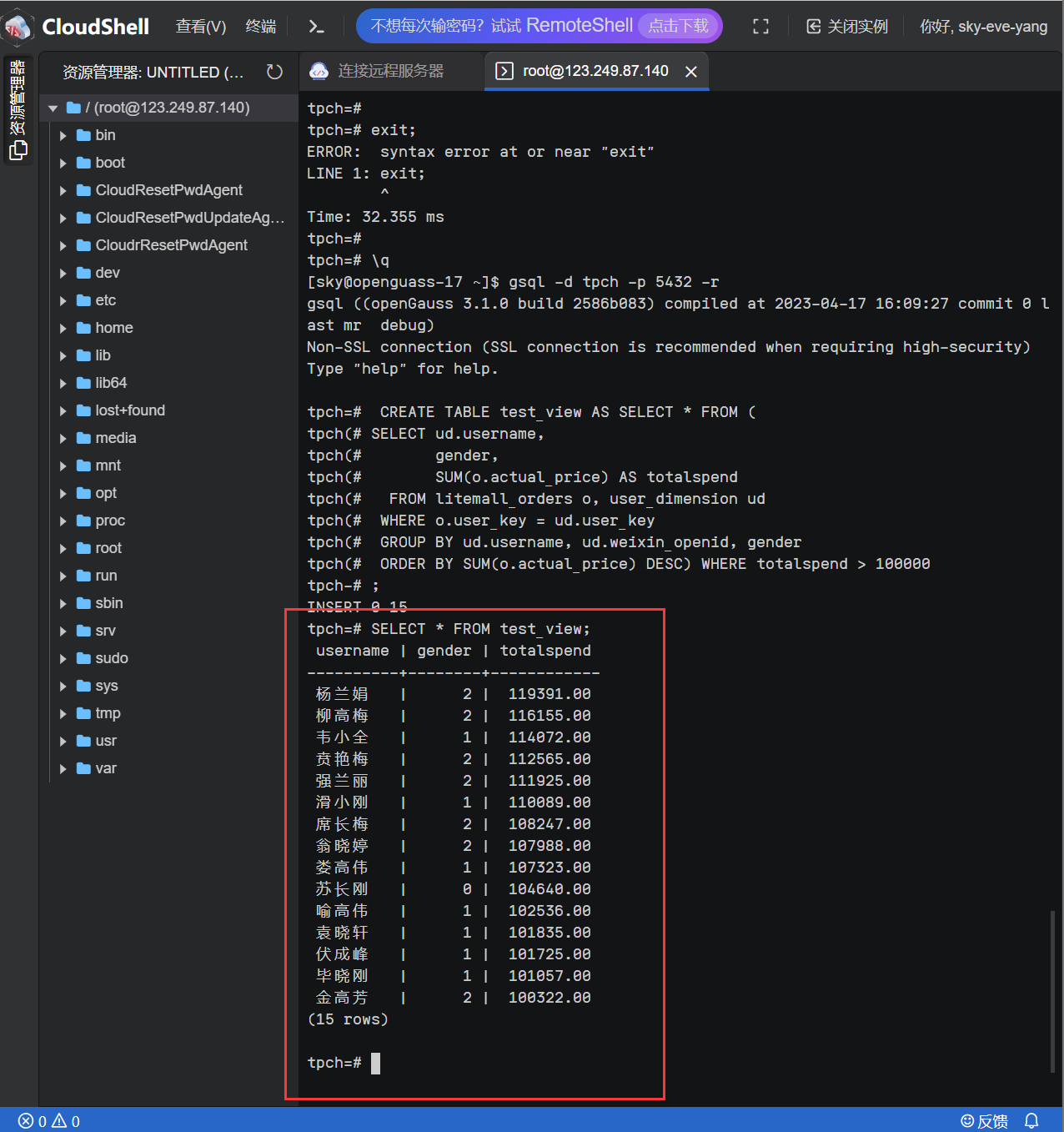
2. 需要进行聚合操作（如SUM、AVG等）或者统计操作（如COUNT）的SQL语句。因为在列存表中，每个字段都单独保存，并且被压缩了，因此可以更快地进行计算和统计操作。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

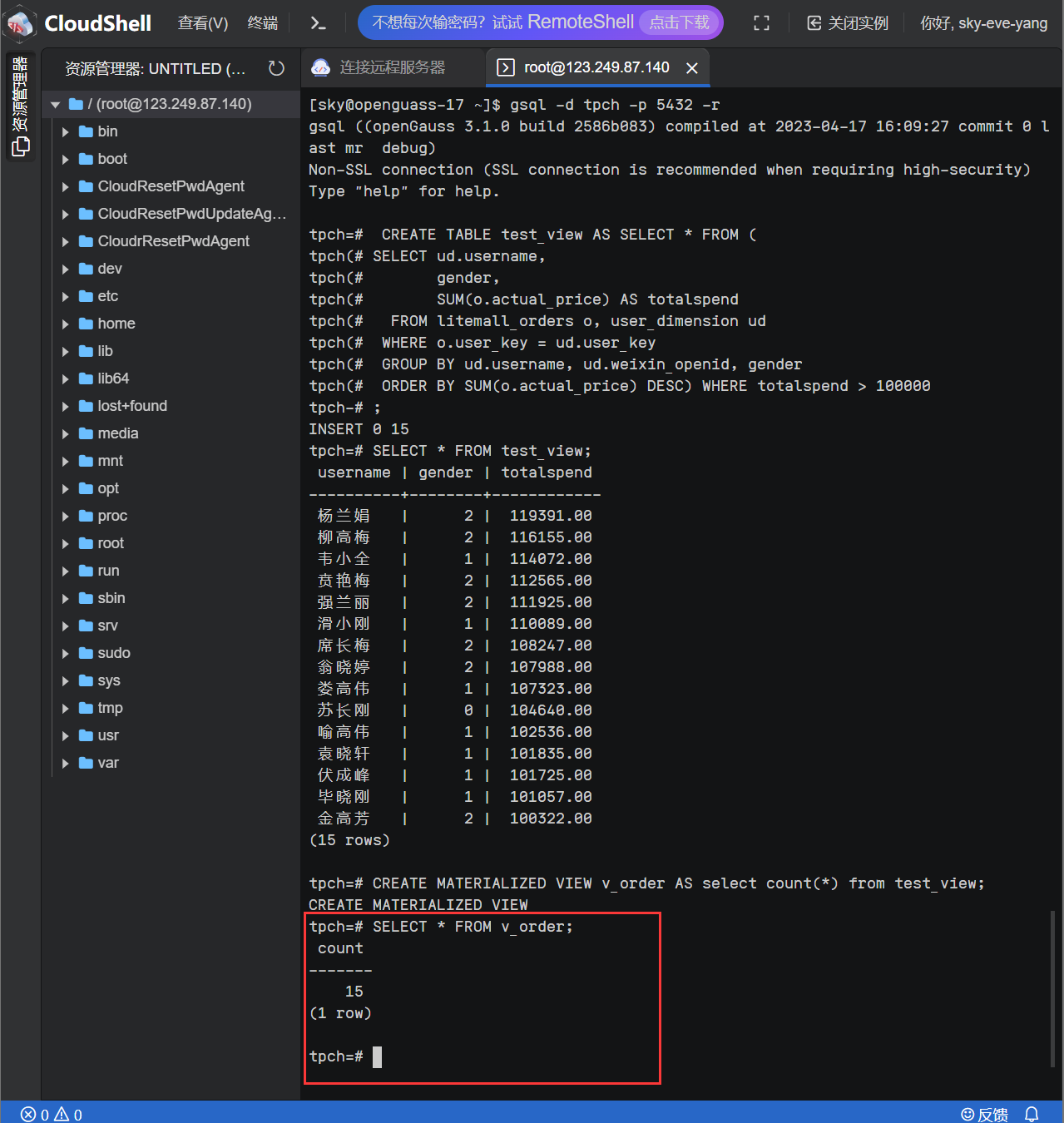
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



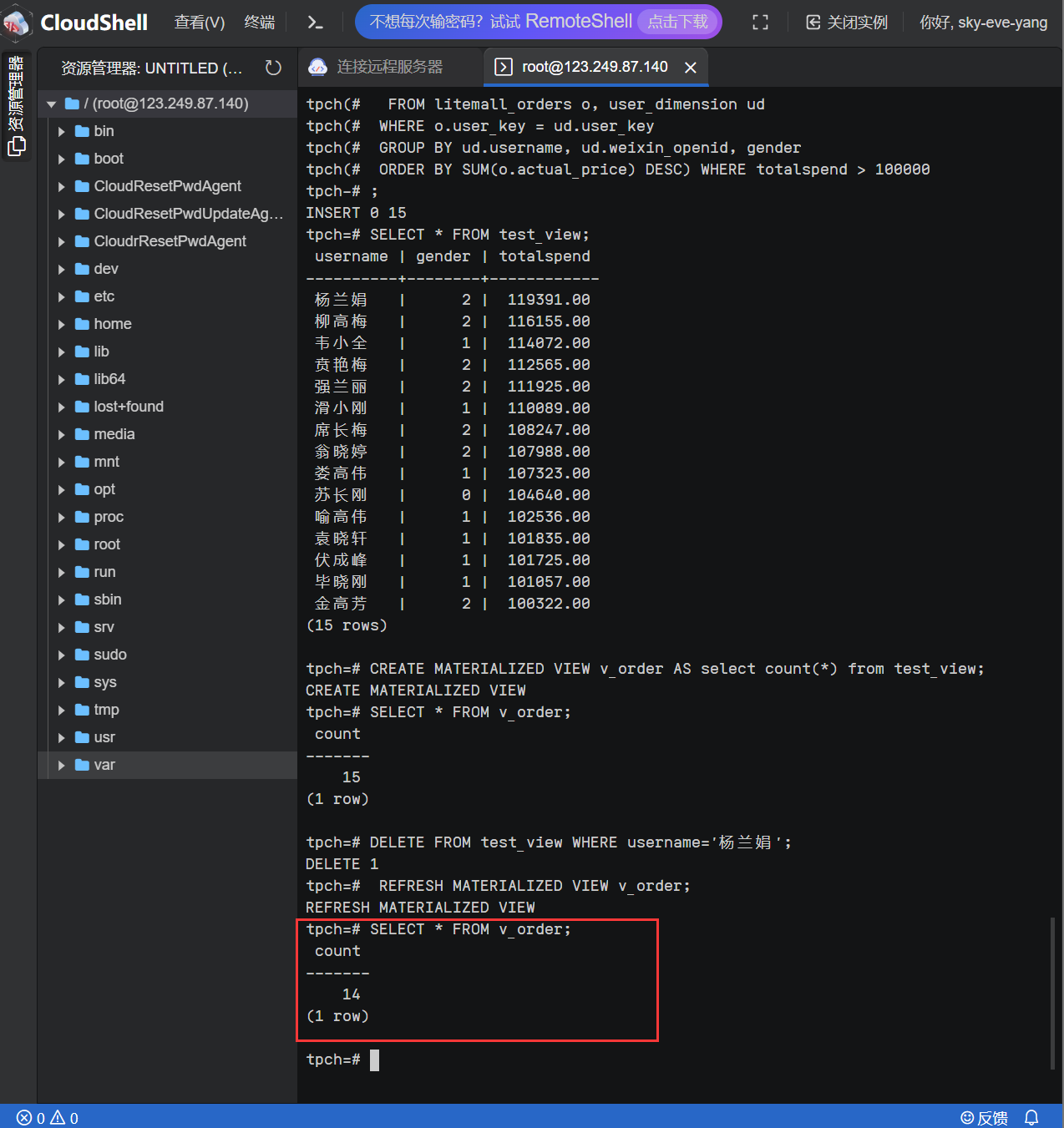
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



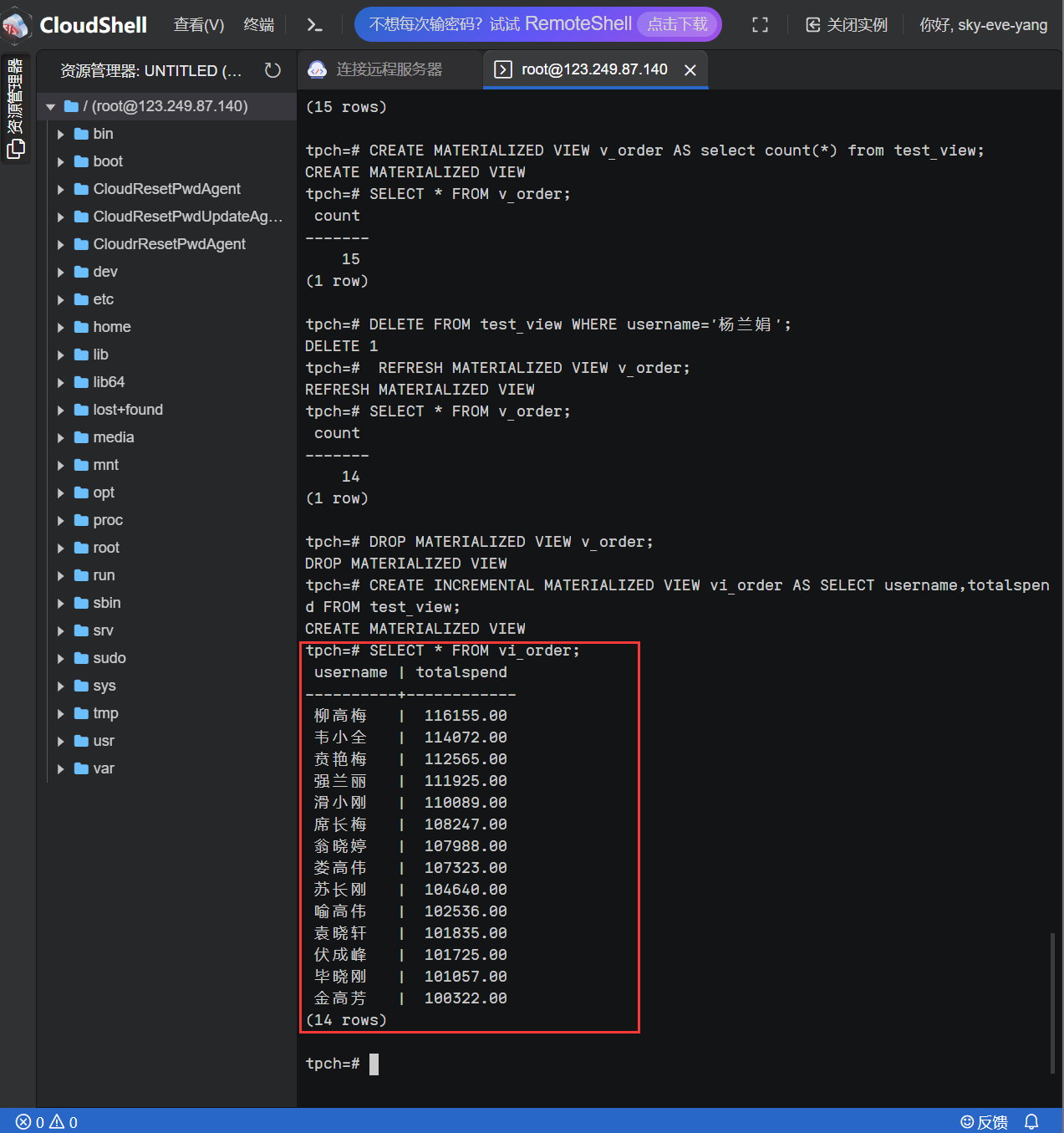
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



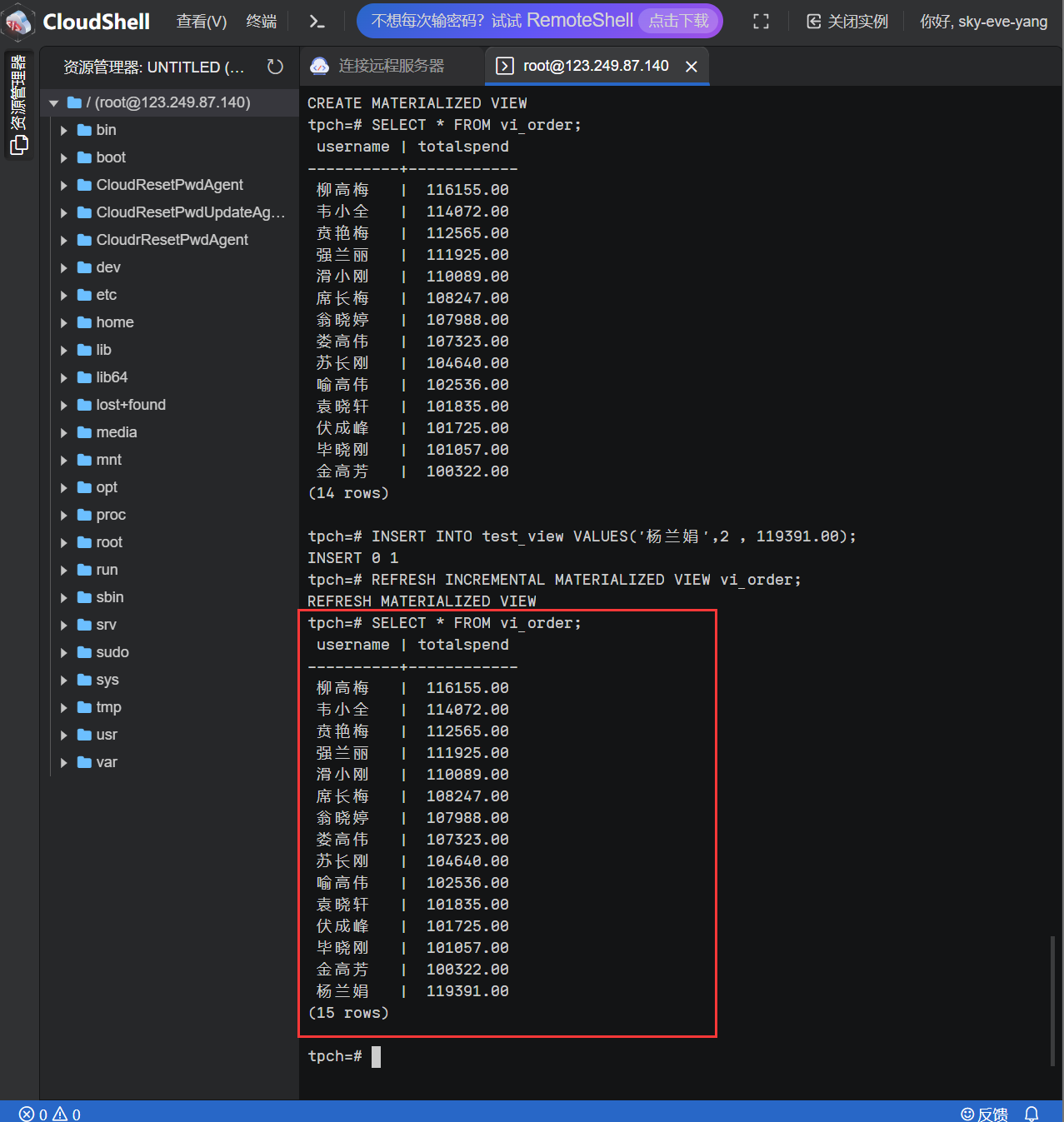
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

全量物化视图是将查询结果存储在磁盘上，每次查询时都会重新计算并更新视图数据。因此，全量物化视图适用于数据更新较少的场景。

而增量物化视图则是将查询结果存储在内存中，并且只在需要时才会进行计算和更新。因此，增量物化视图适用于数据更新频繁的场景。

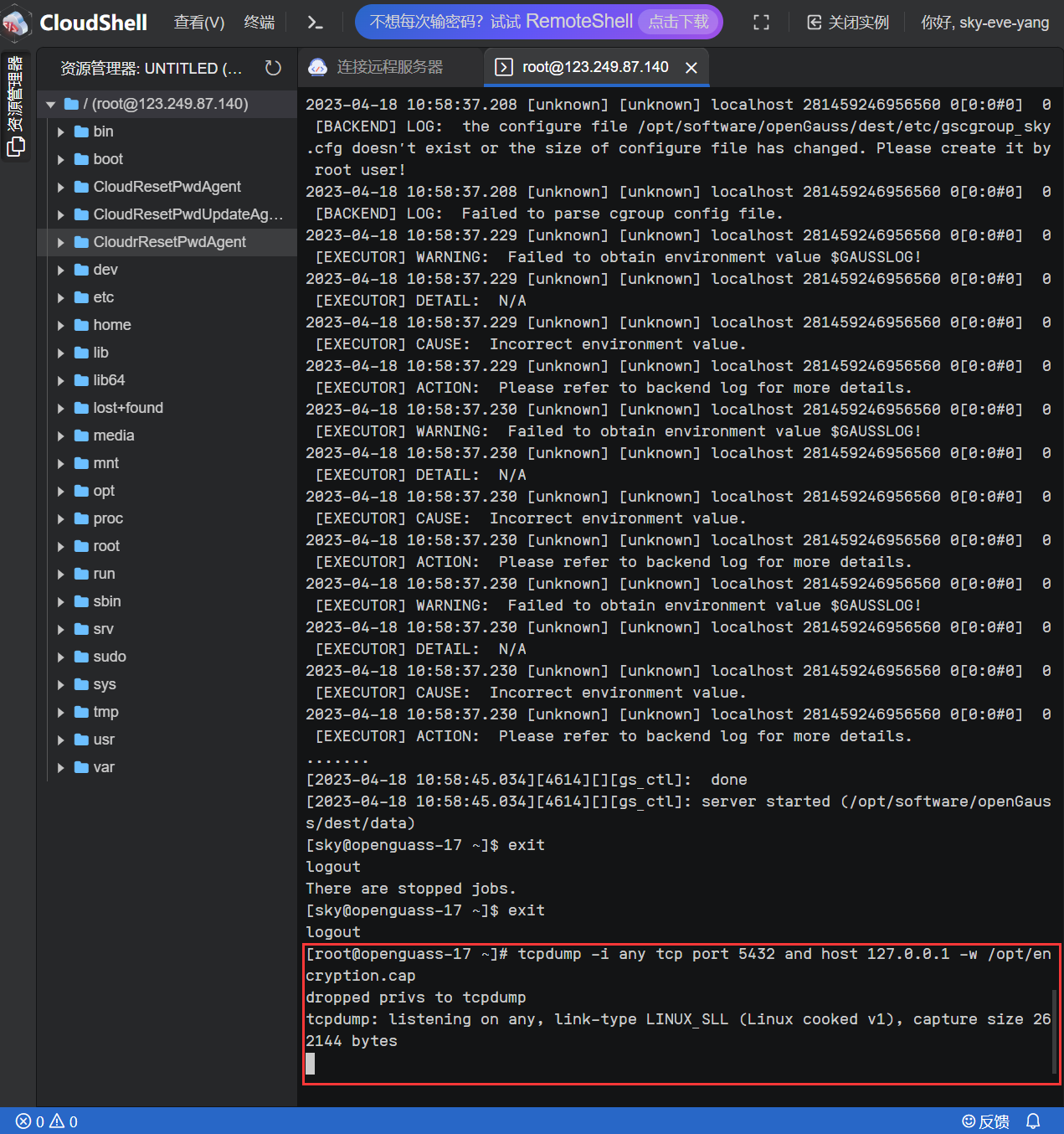
思考题2：物化视图适用那些使用场景？

适用于需要进行复杂或频繁查询的场景，这是由于物化视图是一种预先计算并存储结果的表，其预先计算是在创建或刷新物化视图时进行的，并将结果缓存至磁盘或内存中，可以有效减少对底层表的重复访问和计算，提升查询性能，降低系统负载。

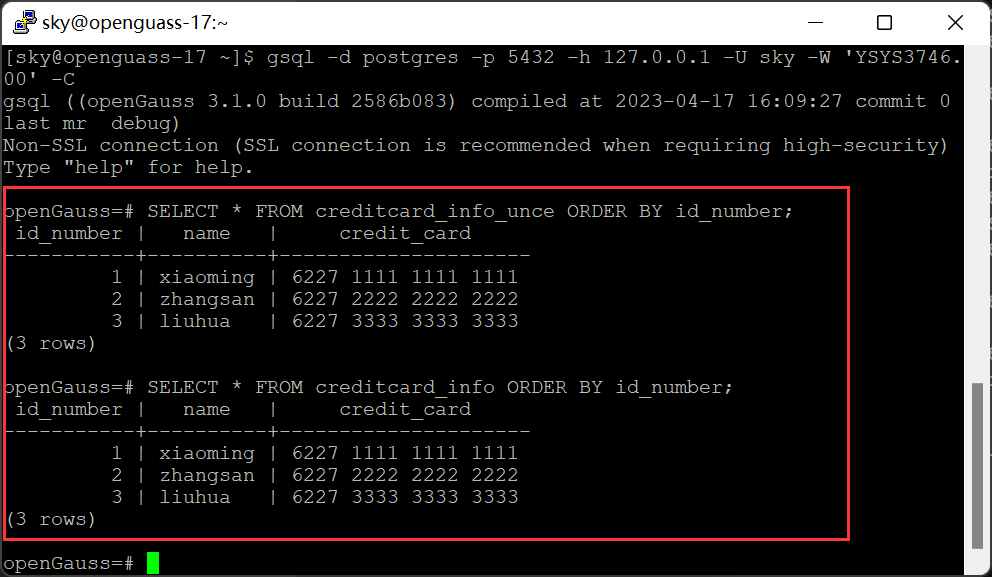
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

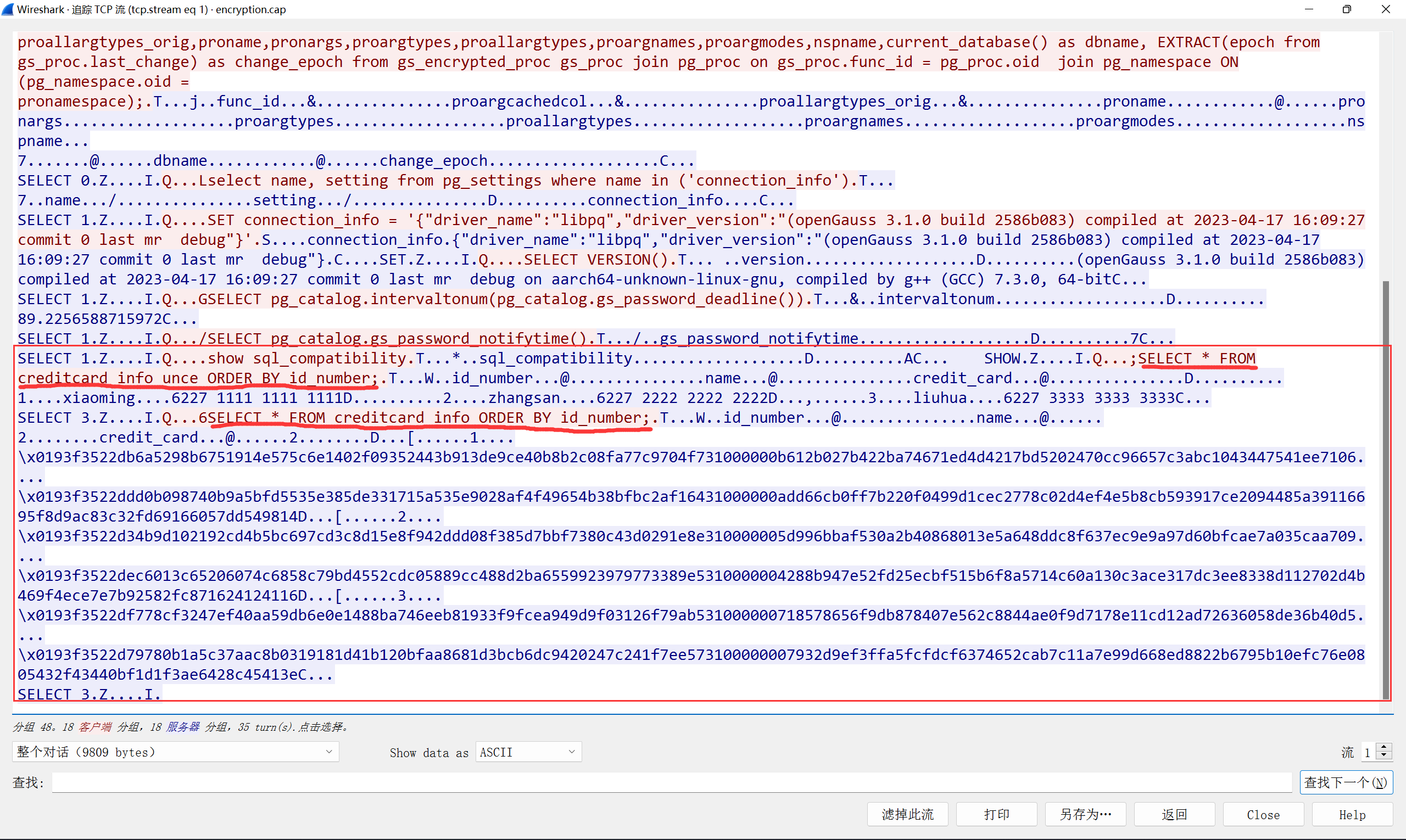
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



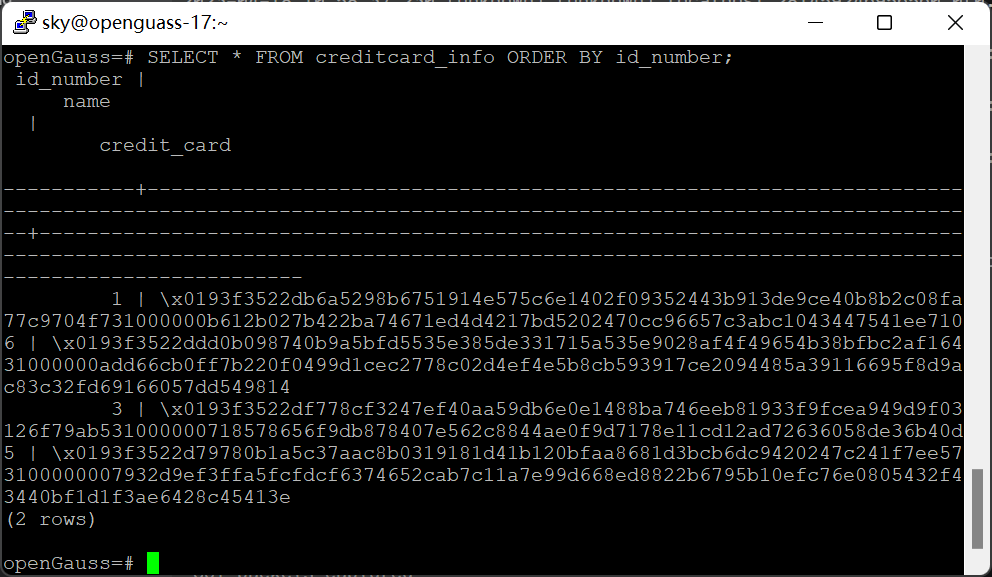
1. 将加密表和非加密表查询结果截图：



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？

对于密态数据库，数据实际存储在物理磁盘上的时候是密文。在客户端进行加密处理后，数据通过网络传输到数据库内核，在内核中完成查询运算后再返回给客户端，**数据始终处于加密状态**。因此，即使磁盘被窃取也无法获取到明文数据。

数据的加解密动作是在客户端完成的，或者说是通过客户端驱动进行的。在全密态数据库中，用户首先需要通过客户端密钥对需要存储在数据库中的数据进行加密处理，并将加密后的数据传输到数据库内核。当用户需要查询某个字段时，在数据库内核中执行相应的查询操作。查询结果返回给客户端时，需要对查询结果进行解密。**只有持有相应密钥的用户才能够对加密数据进行解密**，从而获取到明文数据。