openGauss 安全体系创新

实践课



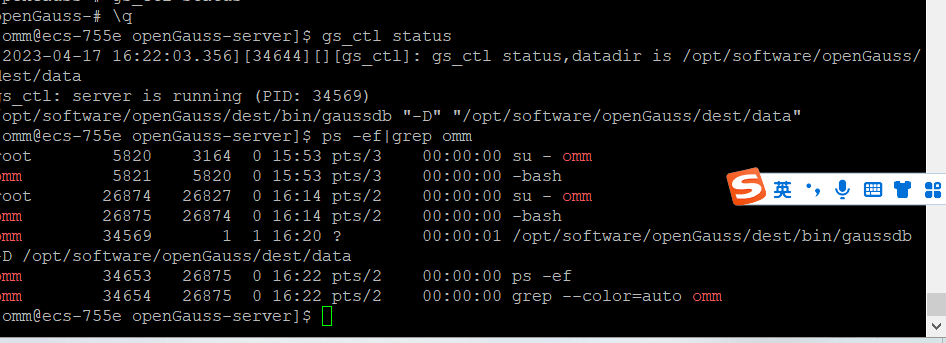
华为技术有限公司

# 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：

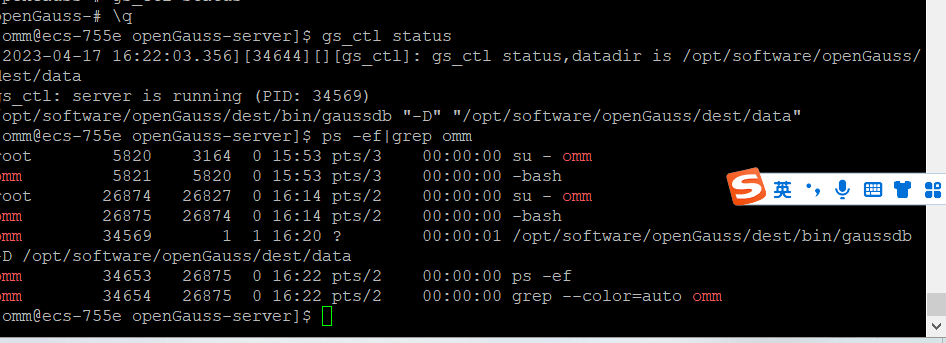
任务一：数据库状态验证

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？

定制配置选项：通过源码编译的方式，可以使用不同的配置选项和自定义编译器选项来构建数据库，以满足自己的需求。  
软件升级：发布版本的数据库通常包含更新的特性和修复的安全漏洞/BUG，为了升级到最新版本，通常需要通过源代码进行编译和安装。  
跨平台支持：通过源码编译，可以在许多平台上构建和安装数据库，而不局限于特定操作系统或体系结构。  
源码可移植性：源码通常可以轻松地移植到其他平台上，因此在需要在不同的平台上运行数据库时，源码编译就变得尤为重要。

# 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：

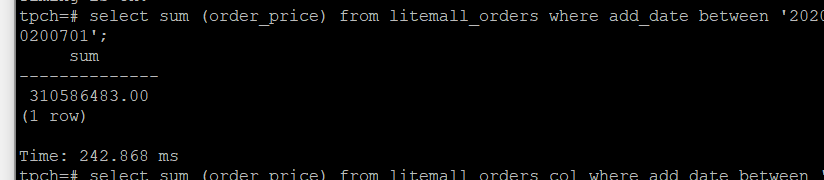
select count(\*) from supplier;;



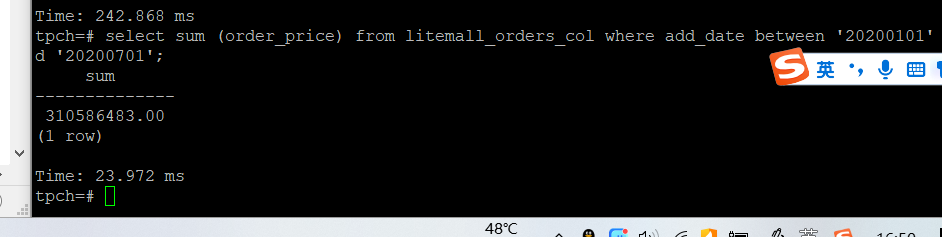
任务二：行存表与列存表执行效率对比

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';

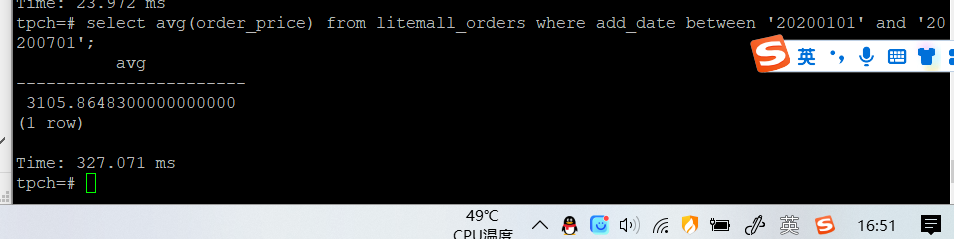


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

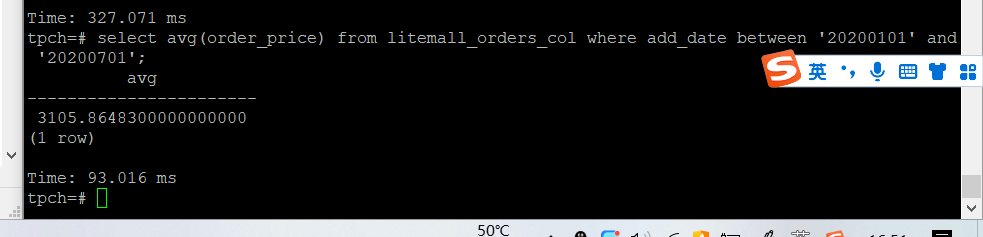


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

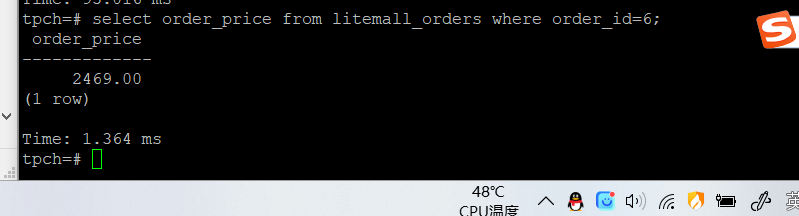


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

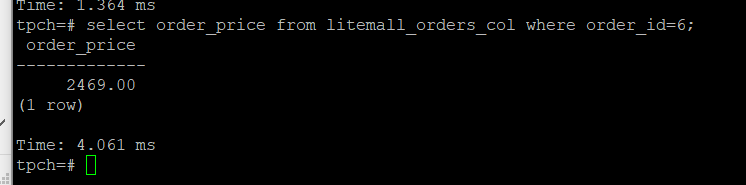


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

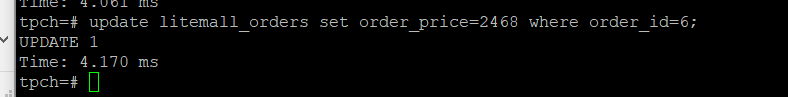


select order\_price from litemall\_orders\_col where order\_id=6;

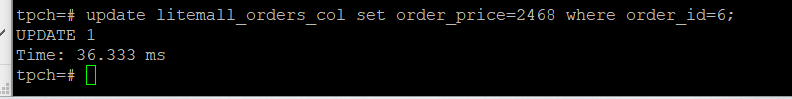


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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？

行存表存储结构将数据以行为单位存储在磁盘上，而列存表则将数据以列为单位存储在磁盘上。这两种存储结构的差异决定了它们在执行查询时的执行情况。

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？

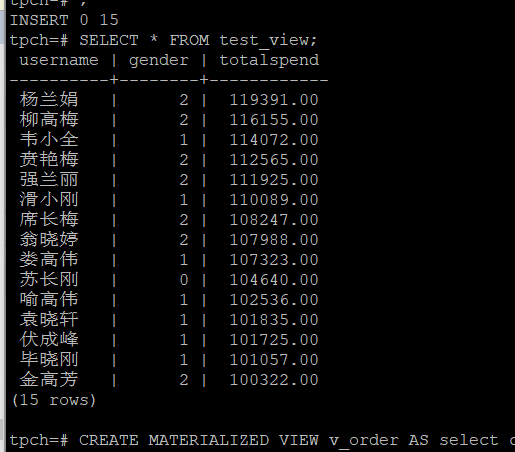
在执行查询类 SQL 时，一般情况下行存表（Row-based Table）和列存表（Column-based Table）的效率取决于查询涉及的数据的类型以及查询操作本身。  
行存表适合于如下类型 SQL：  
  
对数据单行的增、删、改操作，例如：INSERT、UPDATE 和 DELETE。  
需要使用聚合函数如 SUM、COUNT 和 AVG 的查询，由于聚合函数需要对每行数据进行计算。  
在小规模数据操作时，由于每行数据内容较少，对于单条记录的增、删、改操作，行存表的 I/O 开销会更低。  
  
而对于如下类型 SQL，一般来说对于较大数量的数据使用列存表效率会更高：  
  
查询某个字段的值时，由于列存表将同一列数据进行整合存储在一起，因此从磁盘读取时可以减少 I/O 操作，效率更高。  
需要跨多个表进行联合查询时，复杂 JOIN 操作可能需要读取大量的数据来完成关联，此时列存表会有更好的查询性能。  
针对大型数据集的 OLAP（Online Analytical Processing）查询。这种查询通常需要在大数据集中调查数据的分布情况，而且通常不需要返回完整行数据。

# 关卡三：openGauss物化视图应用

任务一：物化视图的使用

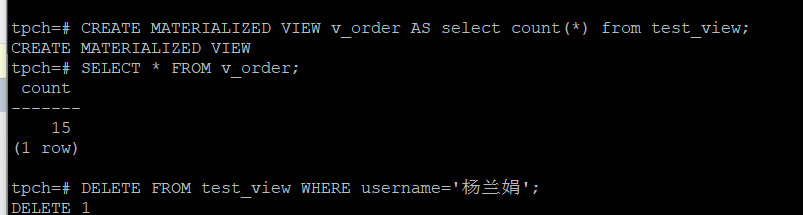
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



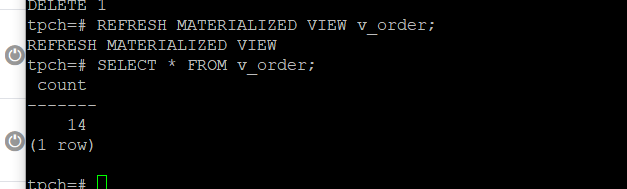
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



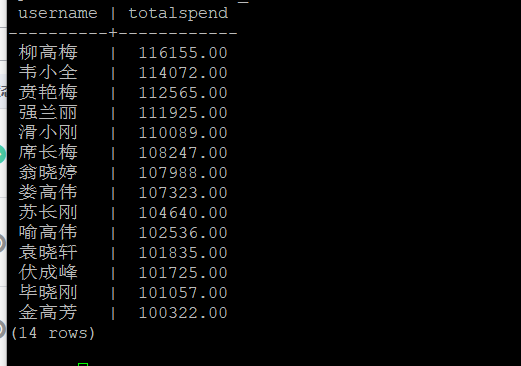
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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。

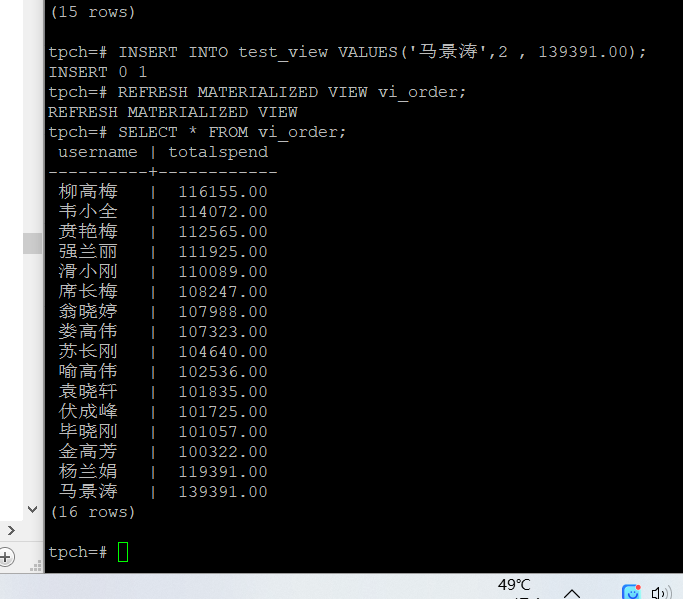
SELECT \* FROM vi\_order;



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？

全量物化视图（Materialized View，MV）和增量物化视图（Incremental Materialized View，IMV）是Oracle数据库中两种不同类型的物化视图。  
全量物化视图是一种基于关联或聚合查询结果的表格形式副本，其中包含查询所涉及的数据，而且这些数据都是预先计算好的。全量物化视图可以起到加快查询速度的作用，但是在数据量大的情况下，这种视图会占用大量的存储空间，同时更新操作也需要较长时间。  
增量物化视图（IMV）是一种只存储查询所需的部分数据，并且在基表的数据发生变化时，只更新变化的数据。这种视图适用于基表数据被频繁修改的情况下，可以减少存储空间并且减少了维护成本。IMV可以通过Oracle数据库的MATERIALIZED VIEW LOG功能来实现增量刷新。  
因此，全量物化视图和增量物化视图的差异在于全量物化视图会存储全部数据而且需要全部更新，而增量物化视图只存储部分数据且只更新变化的数据，节约了存储空间和更新时间。但是增量物化视图对于数据更改的追踪和同步要求更高。

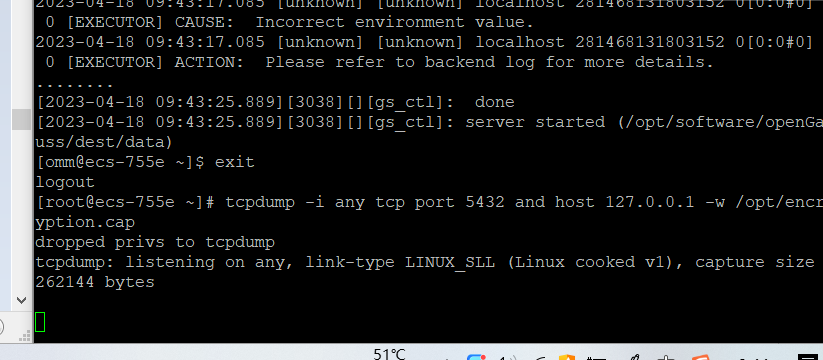
思考题2：物化视图适用那些使用场景？

物化视图适用于以下场景：  
  
频繁的数据查询：物化视图可以预先计算和存储查询结果，加速查询速度，尤其是在大型复杂查询场景下。  
数据量较大且复杂查询：当查询语句需要对多个表格进行聚合、关联等复杂操作时，物化视图可以提高查询的效率。  
低延迟的查询需求：如果需要快速进行查询且对查询结果的实时性要求不高，可以使用定期物化视图刷新，满足查询需求。  
缓存查询结果：如果系统需要对特定查询结果进行缓存，那么物化视图提供了一种有效的缓存机制，可以缩短反复执行同样查询的响应时间。  
数据仓库和报表：物化视图常用于数据仓库和报表系统，可以帮助其快速查询和分析大量数据。  
  
总的来说，物化视图的主要优势是提高数据库的查询效率、加速数据访问以及降低数据库负载，能够在满足查询需求的同时避免冗余计算，提高整个系统的性能。

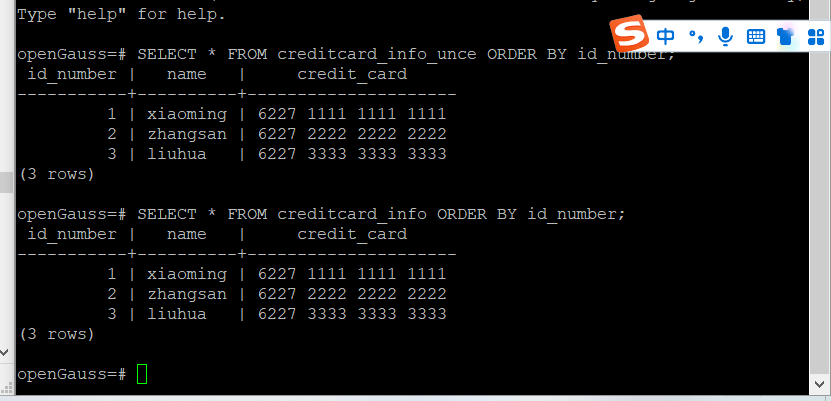
# 关卡四：openGauss密态数据库特性应用

任务一：物化视图的使用

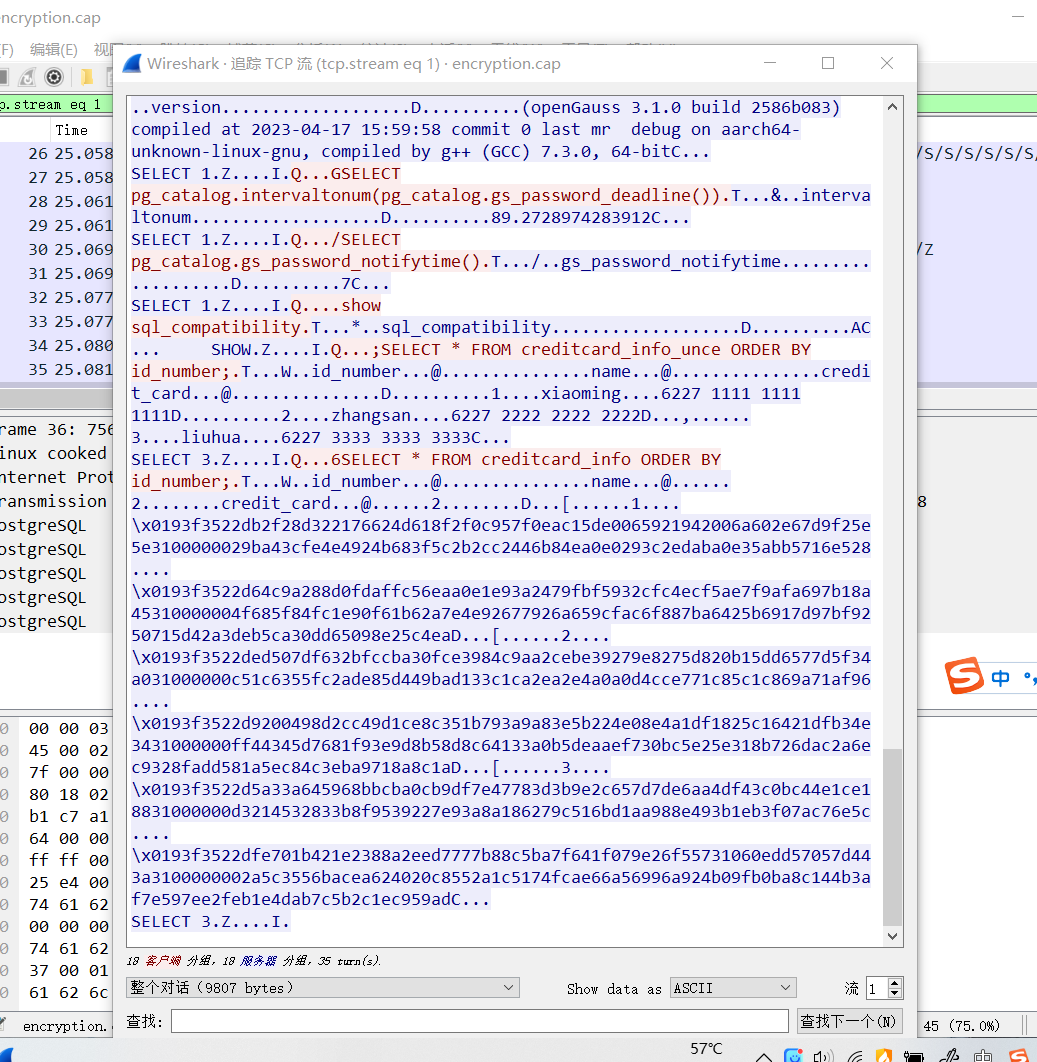
1. 通过tcpdump抓取数据流，此putty窗口暂时保持不动，将执行结果截图：



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？

明文

一般情况下，数据的加解密动作是在客户端完成的，而不是在服务端。使用加密算法（如AES、RSA等）将明文数据加密为密文，然后将加密后的数据发送给服务端。服务端在接收到数据后，将密文传入解密算法进行解密，还原出明文数据进行处理。