关卡2-2

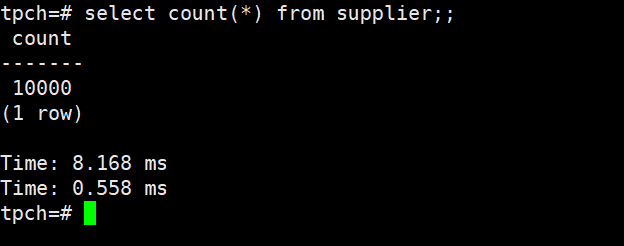
openGauss数据导入及基本操作

openGauss数据导入及基本操作

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;;

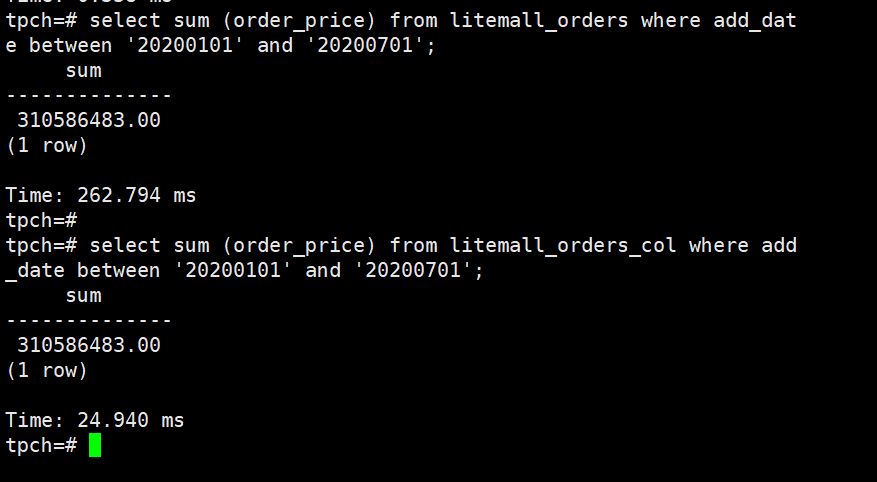


任务二：行存表与列存表执行效率对比

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';

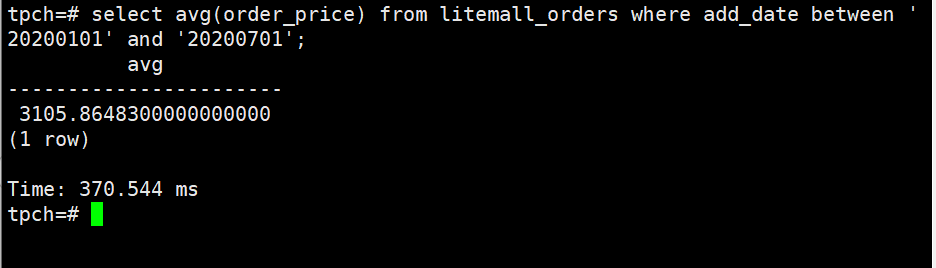
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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';

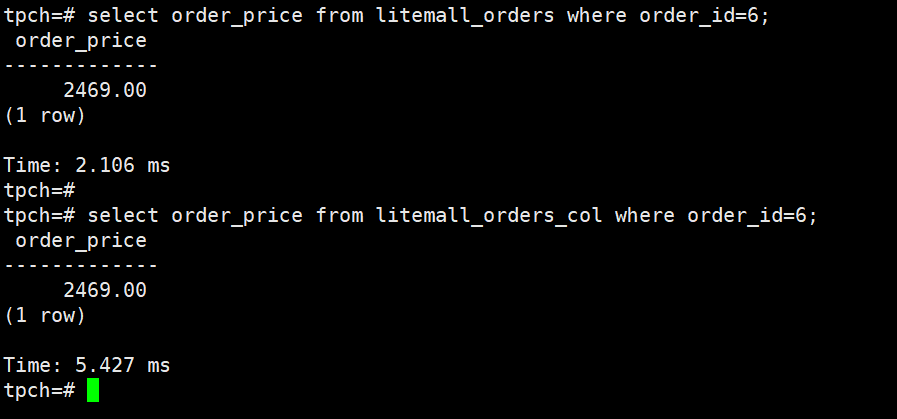
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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;

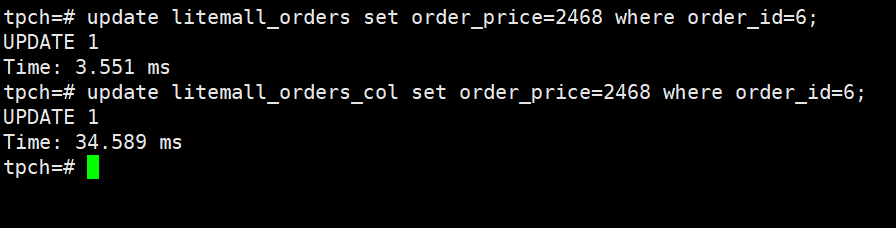
select order\_price from litemall\_orders\_col where order\_id=6;



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。

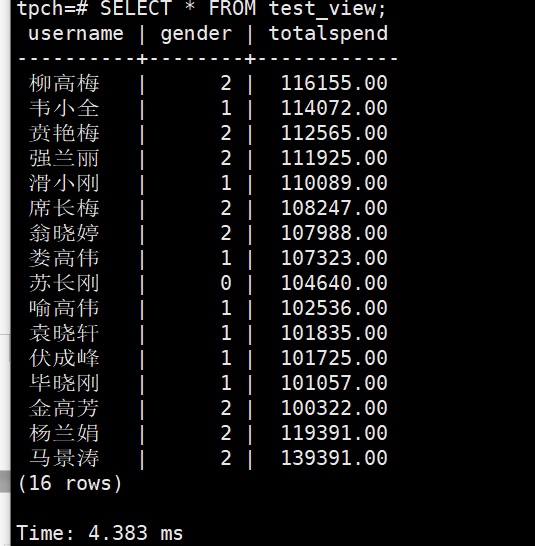
update litemall\_orders set order\_price=2468 where order\_id=6;

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

任务三：物化视图的使用

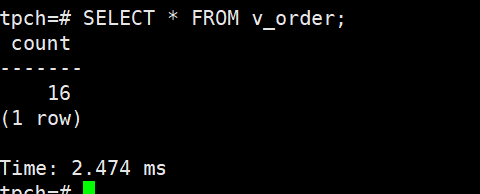
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



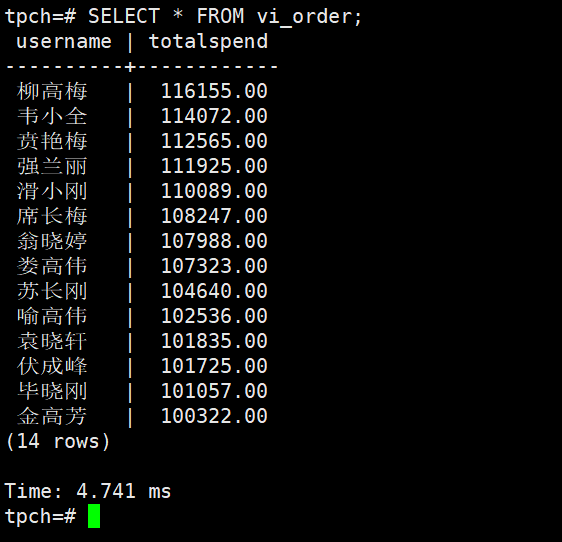
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



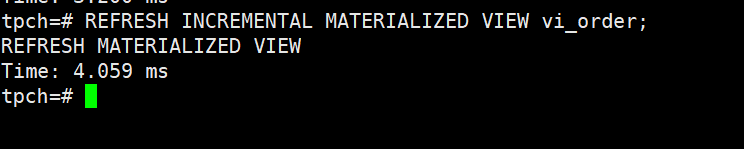
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



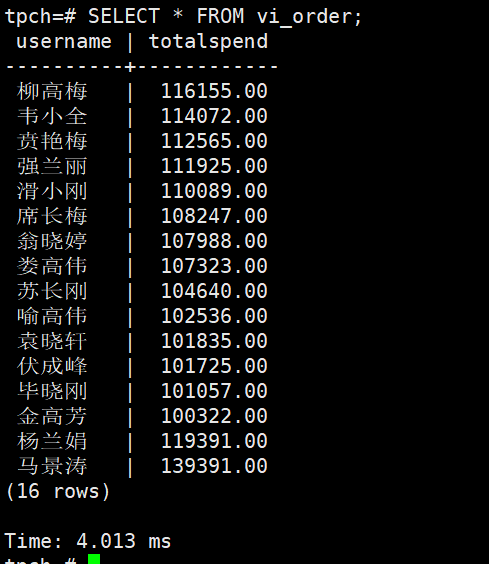
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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;



实践思考题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？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？

1）行存储的写入是一次性完成，消耗的时间比列存储少，并且能够保证数据的完整性，缺点是数据读取过程中会产生冗余数据，如果只有少量数据，此影响可以忽略;数量大可能会影响到数据的处理效率。

2）列存储在写入效率、保证数据完整性上都不如行存储，它的优势是在读取过程，不会产生冗余数据，这对数据完整性要求不高的大数据处理领域，比如互联网，犹为重要。查询过程中，可针对各列的运算并发执行(SMP)，\*\*\*在内存中聚合完整记录集，\*\*\*可能降低查询响应时间;可在数据列中高效查找数据，无需维护索引(任何列都能作为索引)，查询过程中能够尽量减少无关IO，避免全表扫描;因为各列独立存储，且数据类型已知，可以针对该列的数据类型、数据量大小等因素动态选择压缩算法，以提高物理存储利用率;如果某一行的某一列没有数据，那在列存储时，就可以不存储该列的值，这将比行式存储更节省空间。

如果你大部分时间都是关注整张表的内容，而不是单独某几列，并且所关注的内容是不需要通过任何聚集运算的，那么推荐使用行式存储。原因是重构每一行数据（即解压缩过程）对于HANA来说，是一个不小的负担。列式存储的话，比如你比较关注的都是某几列的内容，或者有频繁聚集需要的，通过聚集之后进行数据分析的表。

**行式存储的适用场景：**

　　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更为相关。

**列式存储的适用场景：**

　　一般来说，一个OLAP类型的查询可能需要访问几百万甚至几十亿个数据行，且该查询往往只关心少数几个数据列。例如，查询今年销量最高的前20个商品，这个查询只关心三个数据列：时间（date）、商品（item）以及销售量（sales amount）。商品的其他数据列，例如商品URL、商品描述、商品所属店铺，等等，对这个查询都是没有意义的。而列式数据库只需要读取存储着“时间、商品、销量”的数据列，而行式数据库需要读取所有的数据列。因此，列式数据库大大地提高了OLAP大数据量查询的效率。

　　很多列式数据库还支持列族（column group，Bigtable系统中称为locality group），即将多个经常一起访问的数据列的各个值存放在一起。如果读取的数据列属于相同的列族，列式数据库可以从相同的地方一次性读取多个数据列的值，避免了多个数据列的合并。列族是一种行列混合存储模式，这种模式能够同时满足OLTP和OLAP的查询需求。

　　实操中我们会发现，行式数据库在读取数据的时候，会存在一个固有的“缺陷”。比如，所选择查询的目标即使只涉及少数几项属性，但由于这些目标数据埋藏在各行数据单元中，而行单元往往又特别大，应用程序必须读取每一条完整的行记录，从而使得读取效率大大降低，对此，行式数据库给出的优化方案是加“索引”。

　　在OLTP类型的应用中，通过索引机制或给表分区等手段，可以简化查询操作步骤，并提升查询效率。但针对海量数据背景的OLAP应用(例如分布式数据库、数据仓库等等)，行式存储的数据库就有些“力不从心”了，行式数据库建立索引和物化视图，需要花费大量时间和资源，因此还是得不偿失，无法从根本上解决查询性能和维护成本等问题，也不适用于数据仓库等应用场景，所以后来出现了基于列式存储的数据库。对于数据仓库和分布式数据库来说，大部分情况下它会从各个数据源汇总数据，然后进行分析和反馈，其操作大多是围绕同一列属性的数据进行的，而当查询某属性的数据记录时，列式数据库只需返回与列属性相关的值，在大数据量查询场景中，列式数据库可在内存中高效组装各列的值，最终形成关系记录集，因此可以显著减少IO消耗，并降低查询响应时间，非常适合数据仓库和分布式的应用。

实践思考题2：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。