**《数据库系统原理》**

**课程实验报告**

徽标

描述已自动生成

**学院：计算机（国家示范性软件）学院**

**班级：** 2021211304

**姓名：** **张梓良 杨晨 苗雨**

**学号：**2021212484 2021212171 2021212492

**前言**

**实验环境说明**

本实验环境为VMware虚拟机openEuler20.03系统上的openGauss1.1.0/openGauss2.0.0数据库和华为云GaussDB(openGauss)数据库，实验数据采用电商数据库的八张表。

**第一章 查询优化实验概述**

##### 1.1 实验目的

在 openGauss 平台下，通过观察 Select/Insert/delete/update 等 SQL 语句的查询执行计划，分析查询执行计划中连接、选择、投影等关系代数操作的实现方式及其执行成本。熟悉了解 openGauss 数据库中查询优化的使用，理解数据库查询优化的基本概念。

掌握利用explain 命令，分析对比形式不同、执行结果等价的不同 SQL 语句的查询执行计划的执行成本和执行时间差异。

熟悉了解视图和with 临时视图的创建，观察视图查询、with 临时视图查询的执行计划。

参照教科书中SQL 语句查询优化相关内容，在多种情况下，对比实现方式不同但查询结果相同的等价 SQL 语句在执行计划和成本方面的差异，加深对查询优化的理解，进行书写优化SQL 语句的初步训练，提高编写高效 SQL 语句的能力。

##### 1.2 实验要求

涉及以下几方面：

1） 复合索引左前缀；

2） 多表连接属性上建立索引；

3） 索引对小表查询的作用；

4） 查询条件中函数对索引的影响；

5） 多表嵌入式SQL查询；

6） where查询条件中复合查询条件OR对索引的影响；

7） 聚集运算中索引设计；

8） Select子句中有无distinct的区别；

9）union、union all的区别；

10）from 子句中多余关系的影响。

##### 1.3 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# **第二章 实验过程**

##### 2.1 openGauss 执行计划的查看与分析

# **2.1.1 实验要求**

在 openGauss 平台下，按照下述实验步骤，编写相应的 SQL 语句，观察 Select/Insert/delete/update 等 SQL语句的查询执行计划，分析查询执行计划中连接、选择、投影等关系代数操作的实现方式，观察分析查询执行计划中关系代数操作的执行成本。

步骤 1. 编写 SQL 语句完成下面两条查询；

查询零部件表中零件零售价小于 920 且在零部件供应表中零件供应商 key 为 5 的零件 key。查询客户表中客户账户余额小于 1000 且客户国家为 ALGERIA 的客户名称和客户电话。

步骤 2. 查看语句的执行计划。

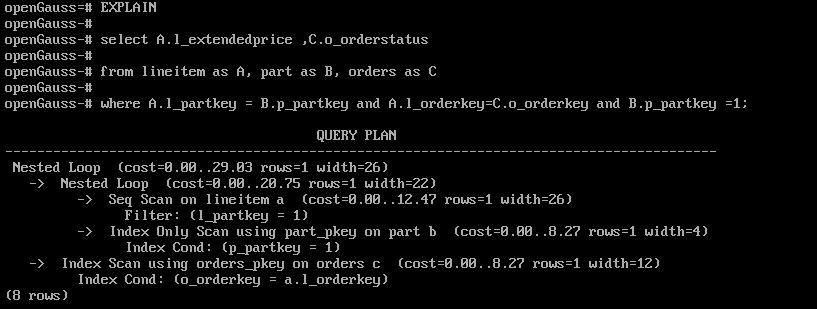
步骤 3. 分析执行计划中各关系代数的实现方式与执行成本。

# **2.1.2 实验步骤**

查询订单明细表中，使用了零件 key 为 1 的订单价格和订单状态。执行如下 SQL 语句：

1. EXPLAIN
2. select A.l\_extendedprice ,C.o\_orderstatus
3. from lineitem as A, part as B, orders as C
4. where A.l\_partkey = B.p\_partkey and A.l\_orderkey=C.o\_orderkey and B.p\_partkey =1;

 查看执行计划



分析:

执行计划需要看每个字段对应的含义，理论上，通过优化开销最大的部分可以优化整个查询。按照执行计划层级解读（纵向）：

第一层：Seq Scan on lineitem a，表扫描算子，在表 a 上执行全表顺序扫描，扫描检查的条件是 l\_partkey=1,这一层的作用是把表 a 符合检查条件的数据从 buffer 或者磁盘上读上来输送给上层节点参与计算。

第二层：Index Only Scan using part\_pkey on part b，表扫描算子，在表 b 上执行主键上的索引扫描（表 b 在建立时创建过主键索引）。扫描检查的条件是 l\_partkey=1，这一层的作用是把表 b 符合检查条件的数据从 buffer 或者磁盘上读上来输送给上层节点参与计算。

第三层：Nested Loop，表连接算子，主要作用是将第一层输出的数据和第二层输出的数据通过嵌套循环的方式连接，并输出结果数据。

第四层：Index Scan using orders\_pkey on orders c，表扫描算子，在表 c 上执行主键上的索引扫描（表 c 在建立时创建过主键索引）。扫描检查的条件是 c.o\_orderkey=a.l\_orderkey，这一层的作用是把表 c 符合检查条件的数据从 buffer 或者磁盘上读上来输送给上层节点参与计算。预

第五层：Nested Loop，表连接算子，主要作用是将第三层输出的数据和第四层输出的数据通过嵌套循环的方式连接，并输出结果数据。

##### 2.2 观察视图查询、with 临时视图查询的执行计划

# **2.2.1 实验要求**

从实验数据库中选取一张表在上面建立视图和 with 临时视图，执行一个查询，观察其执行计划。

1. 选取数据表的一张表（示例中的表除外），分别在上面建立视图和 with 临时视图；

2. 分别在两个视图上进行相同的查询；

3. 查看执行计划并分析。

# **2.2.2实验步骤**

在orders上建立视图

1. CREATE VIEW orders\_view AS
2. SELECT \*
3. FROM orders;

查询并查看执行计划

图形用户界面, 文本, 应用程序

描述已自动生成

通过视图进行查询

1. EXPLAIN ANALYZE
2. SELECT o\_orderkey FROM orders\_view;

图形用户界面, 文本

描述已自动生成

通过with临时视图进行查询

1. EXPLAIN ANALYZE
2. WITH orders\_temview AS( SELECT \*
3. FROM orders
4. )
5. SELECT o\_orderkey FROM orders\_temview;

文本

描述已自动生成

不用视图直接查询

1. EXPLAIN ANALYZE
2. SELECT o\_orderkey FROM orders;

文本

描述已自动生成

分析

3个执行计划对比可发现，针对以上的3个查询，通过建立视图来查询和不用视图直接查询，实际的执行计划是相同的，执行时间也相似。但是通过with临时视图查询，执行计划会有所不同，所花费的执行时间也更长。

原因在于，第一个查询中，提前建立好了orders\_view视图，对orders\_view视图的直接查询，在执行中，转变成了对orders的顺序扫描，而在第三个查询中，对orders的直接查询，也是对orders的顺序扫描，则两个查询中的执行过程是一样的（都只有顺序扫描这一层），执行时间自然也相似。在第二个查询中，执行计划一共有2层，第一层是对orders进行顺序扫描，并将扫描出的数据存在临时创建的通用表表达式orders\_temview中（CTE它是一种临时结果集），第二层再通过对orders\_temview进行扫描来得到查询结果。由于多了第二层，执行时间也更长了。

##### 2.3 复合索引左前缀

# **2.3.1 实验要求**

在做这部分实验时，要求使用 lineitem 表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的 2 条 select 语句实现两个要求（在 lineitem 上创建包含至少三个索引的组合索引）：

（1） 比较有最左前缀索引和无索引的 2 条 select 语句的执行结果和执行计划；

（2） 比较组合索引中其他索引（不包含最左前缀索引）和无索引的 2 条 select 语句的执行计划。分别判断 2 条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因

说明：在使用复合索引时要注意最左前缀的原则，以防索引失效

# **2.3.2 实验步骤**

创建 lineitem 的备份表 lineitem\_New（没有主键，没有索引，没有约束）；

CREATE TABLE lineitem\_new AS TABLE lineitem;



在 lineitem\_new 创建组合索引（至少包含三个索引）。

索引属性建议：可以在 l\_quantity、l\_extendedprice、l\_tax 上设计组合索引。

1. CREATE index lineitem\_index
2. ON lineitem\_new(l\_quantity, l\_tax,l\_extendedprice);

文本

描述已自动生成

比较有最左前缀索引和无索引的 2 条 select 语句的执行结果和执行计划

编写 select 语句，使用最左前缀索引访问 lineitem\_new；编写 select 语句，不使用索引方式，访问 lineitem，执行 2 条 select 语句，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

不使用索引，查询 l\_quantity=24 的全部数据

1. explain analyze select \*
2. from lineitem
3. where l\_quantity=24;

文本

描述已自动生成

使用最左前缀索引查询

1. explain analyze select \*
2. from lineitem\_new where l\_quantity=24;

文本

描述已自动生成

在比较组合索引中其他索引（不包含最左前缀索引）和无索引的 2 条 select 语句的执行计划

编写 select 语句，使用不含最左前缀的索引访问 lineitem\_new；编写 select 语句，不使用索引方式，访问

lineitem，执行 2 条 select 语句，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

不使用索引，查询 l\_tax=0.02 的全部数据

1. explain analyze select \*
2. from lineitem where l\_tax=0.02;

文本

描述已自动生成

使用不含最左前缀的索引查询

1. explain analyze select \*
2. from lineitem\_new where l\_tax=0.02;

文本

描述已自动生成

分析：查询结果一致，但查询时间有很大差距，不使用索引的查询时间>使用不含最左前缀的索引的查询时间>使用最左前缀索引的查询时间。

##### 2.4 多表连接操作，在连接属性上建立索引

# **2.4.1 实验要求**

在做这部分实验时，要求使用 lineitem、part 和 supplier 表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的 2 条 select 语句实现两个要求之一（在连接属性上建立索引）：

（1） 在 lineitem\_new 表的 l\_suppkey 上设计索引，列举在 supplier 中找到此供应商的手机号。

（2） 在 lineitem\_new 表的 l\_partkey 上设计索引，列举在 part 中找到此零件的零件名称。

每个要求中，分别判断 2 条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

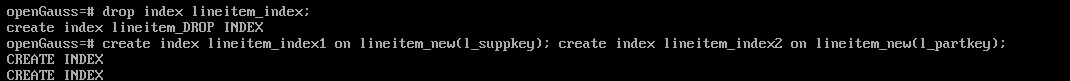
# **2.4.2 实验步骤**

创建 lineitem 的备份表 lineitem\_new；

在 lineitem\_new 创建索引；

删除上个实验的索引，再在 l\_suppkey 和 l\_partkey 上创建 2 个索引

1. drop index lineitem\_index;
2. create index lineitem\_index1 on lineitem\_new(l\_suppkey);
3. create index lineitem\_index2 on lineitem\_new(l\_partkey);



比较有索引和无索引的 2 条两表 select 语句的执行结果和执行计划。

编写 select 语句，使用索引访问 lineitem\_new，part 和 supplier；编写 select 语句，不使用索引方式，访问 lineitem，part 和 supplier，执行 2 条 select 语句，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

不使用 l\_suppkey 索引，l\_suppkey=10 ,列举在 orders 中找到的此订单的订单总价

1. EXPLAIN ANALYZE
2. select DISTINCT l\_suppkey,s\_phone
3. from lineitem, supplier
4. where l\_suppkey=10 AND l\_suppkey=supplier.s\_suppkey;

文本

描述已自动生成

使用 l\_suppkey 索引查询

1. EXPLAIN ANALYZE
2. select DISTINCT l\_suppkey,s\_phone
3. from lineitem\_new, supplier
4. where l\_suppkey=10 AND l\_suppkey=supplier.s\_suppkey;

文本

描述已自动生成

不使用 l\_partkey 索引，l\_partkey= 928,列举在 part 中找到的此邻小区对应的主小区

1. EXPLAIN ANALYZE
2. select DISTINCT l\_partkey,p\_name
3. from lineitem, part
4. where l\_partkey= 928 AND l\_partkey=part.p\_partkey;

文本

描述已自动生成

使用 l\_partkey 索引查询

1. EXPLAIN ANALYZE
2. select DISTINCT l\_partkey,p\_name
3. from lineitem\_new, part
4. where l\_partkey= 928 AND l\_partkey=part.p\_partkey;

文本

描述已自动生成

分析：查询结果一致，但不使用索引的查询时间>使用索引的查询时间，可知，在两表连接上，建立恰当的索引，可以节省查询时间。

##### 2.5 索引对小表查询的作用

# **2.5.1 实验要求**

在做此部分实验时要求使用元组数目少、表占用空间少的 supplier 表，按照下述步骤完成实验内容。在 s\_suppkey 上创建索引，执行一个查询，查看执行计划观察是否用到索引，如果没有则强制使用并与不强制使用做比较。

# **2.5.2 实验步骤**

创建supplier的不带主键索引的备份表supplier\_new

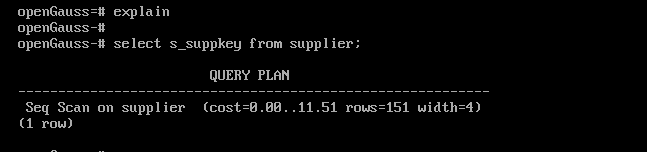
1. CREATE TABLE supplier\_new AS TABLE supplier;



在supplier上新建查询，观察是否用到索引；

1. explain
2. select s\_suppkey from supplier;

并没有用到索引



编写select语句，无索引访问supplier\_new，强制使用索引访问supplier，执行2条select语句，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

无索引查询supplier\_new

1. explain
2. select s\_suppkey from supplier\_new;

文本

描述已自动生成

强制使用索引查询 supplier

先禁止顺序扫描

1. set enable\_seqscan=off;
2. 或 set enable\_seqscan=false;



再查询 supplier

1. explain
2. select s\_suppkey from supplier;

文本

描述已自动生成

分析：强制使用索引开销会增大很多，对于较小的表不建立索引反而查询会更快。

注意：在完成实验后，记得将更改的设置改回去(默认情况下，退出数据库后enable\_seqscan也会自动恢复回on的状态）

1. set enable\_seqscan=on;

文本

描述已自动生成

##### 2.6 查询条件中函数对索引的影响

# **2.6.1 实验要求**

针对同一张表，编写 2 条查询需求完全一样的等价的 select 语句，一条语句在 where 子句查询条件中使用了函数，另外一条语句则没有使用函数。

观察这 2 条等价的 SQL 语句在查询执行计划、索引使用和实现效率方面的差异，掌握如何避免函数导致查询条件索引失效的方法，编写优化的 SQL 语句。

实现以下两个要求之一：

（1） 在订单明细表 lineitem 中查询与价格为 16473.51，折扣为 0.04 的订单数量差在 20 以内订单的订单 key。

（2） 在订单明细表 lineitem 中查询与价格为 79238.70，折扣为 0.02 的订单数量差在 20 以内订单的订单 key。

# **2.6.2 实验步骤**

创建备份表并在备份表上创建索引；

先删除上面实验建立的索引

1. drop index lineitem\_index1;
2. drop index lineitem\_index2;

文本

描述已自动生成

再建立索引

1. create index lineitem\_index1 on lineitem\_new(l\_discount);
2. create index lineitem\_index2 on lineitem\_new(l\_quantity);

文本

描述已自动生成

编写select语句，完成实验要求中的查询，执行2条等价的查询语句，并查看执行计划，比较两个语句的执行结果和执行速度，观察函数对索引的影响。

以要求一为例，

查询条件中使用函数进行查询

1. explain
2. select distinct B.l\_orderkey
3. FROM lineitem\_new as A, lineitem\_new as B
4. where A.l\_extendedprice = 16473.51
5. and A.l\_discount =0.04
6. and ABS(A.l\_quantity-B.l\_quantity)<20;

文本

描述已自动生成

查询条件中不使用函数进行查询

1. explain
2. select distinct B.l\_orderkey
3. FROM lineitem\_new as A, lineitem\_new as B
4. where A.l\_extendedprice = 16473.51
5. and A.l\_discount =0.04
6. and B.l\_quantity >A.l\_quantity -20 and B.l\_quantity <A.l\_quantity +20;

文本

描述已自动生成

分析：第一个执行计划使用了 l\_discount 上的索引，而没有使用 l\_quantity 上的索引，从而查询效率更低，第二个执行计划两个索引都使用了，从而查询效率更高。可知，很多情况下，当索引上存在函数计算时，索引就不起作用了，可转换为等价的没有函数计算的实现方式，来提高查询效率。

##### 2.7 多表嵌入式 SQL 查询

# **2.7.1 实验要求**

对访问相同的两张表且查询需求完全一样的 2 条 select 语句，一条使用嵌套查询，一条使用连接查询，实现两个要求之一：

（1） 根据零部件表和零部件供应表，查询零件供应商 key 为 1002 的供应商供应零件的名称。

（2） 根据订单表和客户表，查询客户 key 为 10 的订单的订单总价。

在每个要求中，分别判断 2 条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

# **2.7.2 实验步骤**

以要求一为例

嵌套查询

1. explain analyze select A.p\_name from part as A
2. where A.p\_partkey IN
3. ( select B.ps\_partkey
4. from partsupp as B
5. where B.ps\_suppkey=1002
6. );

文本

描述已自动生成

连接查询

1. explain analyze select A.p\_name
2. from part as A,partsupp as B
3. where B.ps\_suppkey=1002
4. AND A.p\_partkey =B.ps\_partkey;

文本

描述已自动生成

分析：一般在使用多表查询的时候应该避免使用嵌套查询，但在实际应用中有时会出现两种情况性能相同甚至是嵌套查询性能更优的情况。

##### 2.8 where 查询条件中复合查询条件 OR 对索引的影响

# **2.8.1 实验要求**

在做这部分实验时，要求使用lineitem表,按照下述步骤完成实验内容。在lineitem\_new上创建一个索引，查询条件中为“A OR B”，A为创建的索引条件，B为非索引条件，提交select语句并查看执行计划中是否还能用到索引。再将含有“A OR B”查询条件的语句改为等价的两条select语句的union，对比这两种实现方式的执行效率。

创建lineitem的备份表lineitem\_new，并在lineitem\_new上创建索引；

编写select语句，where查询条件为“A OR B”，A为定义在索引属性上的查询条件，B为定义在非索引属性上的查询条件，执行语句，查看执行计划是否使用了索引

1. Select … From …
2. Where  … and (A OR B);

将含有“A OR B”查询条件的select语句，转换为等价的2条select语句的union

1. Select … From …
2. Where  … and A
3. Union Select … From …
4. Where  … and B;

分析对比这两种查询实现方式的执行效率。

# **2.8.2 实验步骤**

删除上面实验的索引，并创建索引

1. drop index lineitem\_index1;
2. drop index lineitem\_index2;
3. create index lineitem\_index on lineitem\_new(l\_quantity);

文本

描述已自动生成

含有“A OR B”查询条件的 select 语句

1. explain select \*
2. from lineitem\_new
3. where l\_tax=0.06 AND (l\_quantity=36 OR l\_discount=0.09);

文本

描述已自动生成

等价的 2 条 select 语句的union

1. explain (select \*
2. from lineitem\_new
3. where l\_tax=0.06 AND l\_quantity=36) UNION
4. (select \*
5. from lineitem\_new
6. where l\_tax=0.06 AND l\_discount=0.09);

文本

描述已自动生成

分析：含有union的语句执行效率更低，从执行计划来看，第一个语句因为where条件中引入了OR而使得索引不再起作用，但是后一个查询步骤繁多，虽然使用索引，但是却还是进行了全表扫描，效率反而更低。

##### 2.9 聚集运算中的索引设计

# **2.9.1 实验要求**

选定作为聚集运算查询对象的关系表，对group by操作的分组属性，建立聚集/非聚集索引，对聚集运算（如 count、sum、avg）的属性，建立非聚集索引。分析比较等价的有索引聚集运算、无索引聚集运算查询在查询执行计划、执行速度方面的区别。完成以下两个要求之一：

（1） 在订单明细表lineitem，计算有同一供应商订单的平均价格并按供应商分组（注意索引的建立）；

（2） 在订单明细表lineitem，计算有同一供应商订单的平均数量并按供应商分组（注意索引的建立）。

创建备份表，并根据查询要求在备份表上创建索引；

根据查询要求，在原表和备份表上编写并相同的select语句；

查看执行计划，分析比较等价的有索引聚集运算、无索引聚集运算查询在查询执行计划、执行速度方面的区别。

# **2.9.2 实验步骤**

以要求一为例，

删除上面实验创建的索引，并分别在l\_suppkey和l\_extendedprice创建索引；

1. drop index lineitem\_index;
2. create index lineitem\_index1 on lineitem\_new(l\_extendedprice);
3. create index lineitem\_index2 on lineitem\_new(l\_suppkey);

文本

描述已自动生成

查询同一l\_suppkey的平均l\_extendedprice并按l\_suppkey分组；

无索引聚集运算

1. explain
2. select l\_suppkey,avg(l\_extendedprice) as avg\_extendedprice
3. from lineitem
4. group by l\_suppkey;

文本

描述已自动生成

有索引聚集运算

1. explain
2. select l\_suppkey,avg(l\_extendedprice) as avg\_extendedprice
3. from lineitem\_new
4. group by l\_suppkey;

文本

描述已自动生成

强制使用索引后，聚集运算

1. set enable\_seqscan=off; explain
2. select l\_suppkey,avg(l\_extendedprice) as avg\_extendedprice
3. from lineitem\_new
4. group by l\_suppkey;

文本

描述已自动生成

分析：上述例子，有索引的表在查询时没有使用索引（优化器在检测到使用索引查询效率反而更低时，会自动不使用索引），第一个查询和第二个查询的执行计划基本相同，查询效率也基本相同。在有索引的表上强制使用索引后，第三个查询的执行计划采取了索引扫描，查询效率更低，查询速度更慢。注意：在完成实验后，记得将更改的设置改回去(默认情况下，退出数据库后enable\_seqscan也会自动恢复回on的状态）。

1. set enable\_seqscan=on;

文本

描述已自动生成

##### 2.3.10 Select 子句中有无 distinct 的区别

# **2.10.1 实验要求**

在做这部分实验时，按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的 2 条 select 语句，一条使用 distinct，一条不使用 distinct，实现两个要求之一：

（1） 找出所有零部件表，零部件供应表的零件 key。

（2） 找出客户表中的客户 key 和订单表的客户 key 并总和到一起。

在每个要求中，分别判断 2 条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

# **2.10.2 实验步骤**

以要求一为例。

使用union进行查询

1. explain analyze
2. select p\_partkey from part union
3. select ps\_partkey from partsupp;

文本

描述已自动生成

使用union all进行查询

1. explain analyze
2. select p\_partkey from part union all
3. select ps\_partkey from partsupp;

文本

描述已自动生成

分析：两次查询的查询结果不一样，第一次查询结果的数据要远远少于第二次查询结果的数据，第二次查询结果中有大量的重复数据，但是第二次查询效率要高于第一次的查询效率，union all的性能要优于union。

##### 2.11 from 中存在多余的关系表，即查询非最简化

# **2.11.1 实验要求**

在做这部分实验时，按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的 2 条等价 select语句，一条在 from 后面增加一个多余的关系表（但在 where 子句中增加连接条件），另一条 from 后面最简化。

可自行选择查询语句，要求至少两表连接查询。分别判断 2 条语句的执行效果和执行速度，解释执行时间出现差异的原因。

# **2.11.2 实验步骤**

没有多余关系表

1. explain analyze
2. select distinct orders.o\_custkey
3. from lineitem,orders
4. where lineitem.l\_orderkey=orders.o\_orderkey;

文本

描述已自动生成

增加了一个多余的关系表tbCell和对应的连接条件

1. explain analyze
2. select distinct orders.o\_custkey
3. from lineitem,orders,part
4. where lineitem.l\_orderkey=orders.o\_orderkey and lineitem.l\_partkey=part.p\_partkey;

文本

描述已自动生成

分析：两次查询的结果完全一致，但是第一次查询的查询效率远远高于第二次查询，增加多余的表会大大的增加开销。

# **第三章 总结**

本次实验旨在通过在openGauss平台下观察不同SQL语句的查询执行计划，分析查询优化的实现方式和成本，并掌握利用explain命令进行对比分析。除此之外，还涉及了视图和临时视图的创建以及对查询优化的初步训练。

在实验过程中，我们按照教科书中的SQL语句查询优化相关内容，重点探究了以下几个方面：

1. 复合索引左前缀：通过在复合索引中只使用部分字段作为查询条件，可以减少索引的大小，提高查询效率。

2. 多表连接属性上建立索引：在进行多表连接操作时，对连接属性上建立索引可以显著提高查询性能。

3. 索引对小表查询的作用：对于小表的查询，索引的作用可能并不明显，因为扫描整个表的成本相对较低。

4. 查询条件中函数对索引的影响：在查询条件中使用函数可能会导致无法使用索引，因为函数的执行需要对所有记录进行计算。

5. 多表嵌入式SQL查询：在嵌入式SQL查询中，我们需要注意使用适当的连接方式和索引，以提高查询性能。

6. WHERE查询条件中复合查询条件OR对索引的影响：在查询条件中使用复合查询条件OR可能会导致无法使用索引，因为OR操作需要对多个条件进行比较。

7. 聚集运算中索引设计：在进行聚集运算时，合理设计索引可以减少数据的读取和排序操作，提高查询效率。

8. SELECT子句中有无DISTINCT的区别：在SELECT子句中使用DISTINCT会对结果进行去重操作，可能会增加查询的执行成本。

9. UNION和UNION ALL的区别：UNION会对结果进行去重操作，而UNION ALL不会，因此UNION ALL的执行速度通常比UNION更快。

10. FROM子句中多余关系的影响：在FROM子句中包含多余的关系可能会增加查询的执行成本，因此需要注意避免不必要的关系。

通过本次实验，我们对查询优化的基本概念和实践技巧有了更深入的了解。我们学会了使用explain命令分析不同SQL语句的执行计划和成本差异，并进行了优化SQL语句的训练，提高了编写高效SQL语句的能力。

在今后的数据库开发和优化过程中，我们可以根据实验中的经验和知识，针对具体的查询场景进行优化，以提高系统的性能和响应速度。