《查询优化》  
实验报告



|  |  |
| --- | --- |
| **学院：** | **计算机学院（国家示范性软件学院）** |
| **班级：** | **2019211308 2019211308 2019211308** |
| **姓名：** | **顾天阳 曾世茂 庞仕泽** |
| **学号：** | **2019211539 2019211532 2019211509** |

目录

[2.1 执行计划的查看与分析 3](#_Toc14080)

[2.2 观察视图查询，with临时视图查询的执行计划 5](#_Toc22404)

[2.3 优化sql语句 7](#_Toc18287)

[2.3.1 复合索引做前缀 7](#_Toc23414)

[2.3.2 多表连接操作，在连接属性上建立索引 9](#_Toc11132)

[2.3.3 索引对小表查询的搜索 12](#_Toc12491)

[2.3.4 查询条件中函数对索引的影响 14](#_Toc13229)

[2.3.5 多表嵌入式SQL查询 15](#_Toc22984)

[2.3.6 where 查询条件中复合查询条件 OR 对索引的影响 16](#_Toc26023)

[2.3.7 聚集索引中的索引设计 18](#_Toc26664)

[2.3.8 select子句中有无distinct的区别 19](#_Toc19512)

[2.3.9 union 和 union all的区别 20](#_Toc893)

[2.3.10 from 中存在多余的关系表，即查询非最简化 20](#_Toc191)

[遇到的问题及解决 21](#_Toc30579)

[实验总结 21](#_Toc220)

# 2.1 执行计划的查看与分析

**1. 编写sql语句**

查询 tbATUData 中 PCI 小于 100 且在 tbATUC2I 中干扰强度小于 3 的主小区的切换目标小区。

select "CellID","N\_SECTOR\_ID" from "tbATUData" ,"tbATUC2I" ,"tbAdjCell"

WHERE "PCI" <100 and "CellID" = "SECTOR\_ID" and "RATIO\_ALL" <3 and "CellID" ="S\_SECTOR\_ID";

查询 tbHandOver 中小区类型为“优化区”且设备厂家为华为的切换源小区和切换成功率。

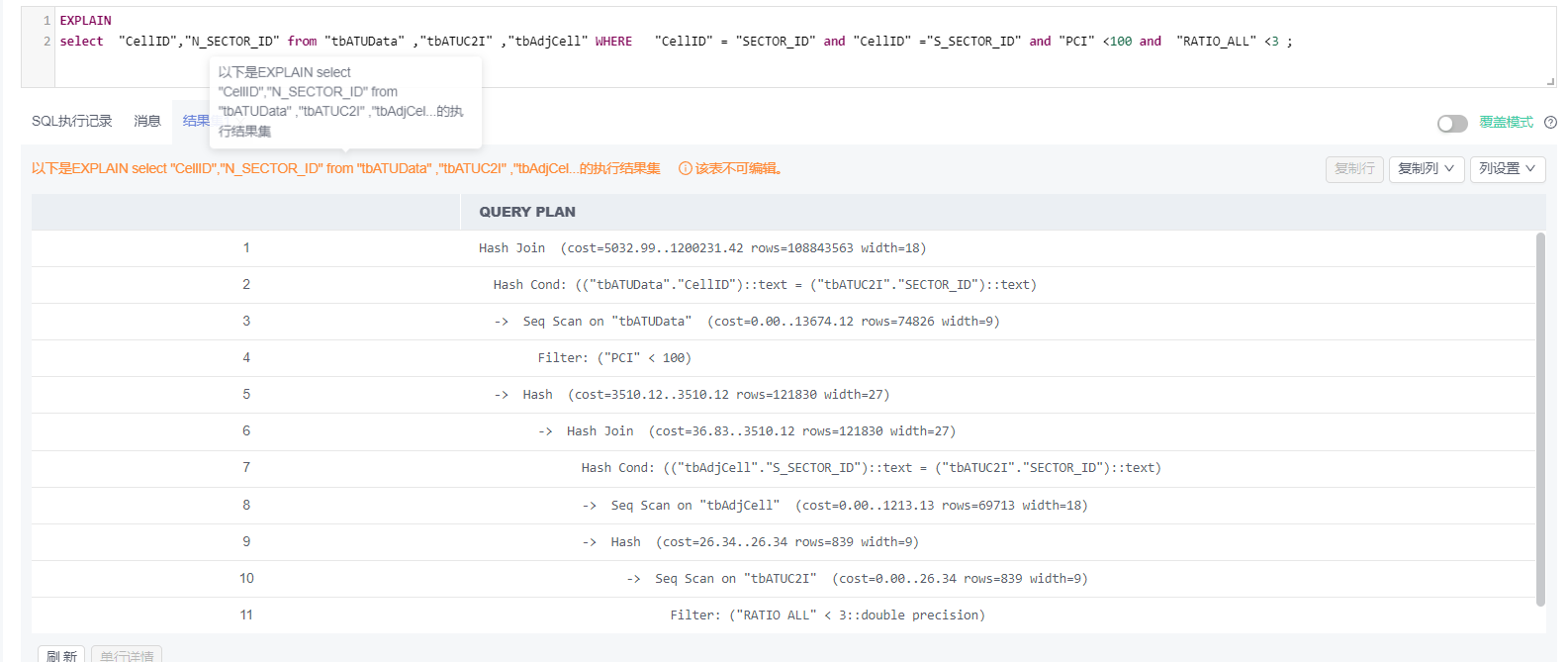
SELECT "SCELL" , "HOSUCCRATE" FROM "tbHandOver" ,"tboptcell" ,"tbcell\_2"

WHERE "tbHandOver"."SCELL" = "tboptcell" ."SECTOR\_ID" and "tbHandOver"."SCELL" = "tbcell\_2" ."SECTOR\_ID"

and "CELL\_TYPE" = '优化区' and "VENDOR" = '华为'

1. **查看查询计划**

查询1：



解释：

第一层：seq scan on tbATUData , 扫描检查的条件为PCI<100 , 将符合条件的数据从buffer或磁盘中读出给上层运算，预计执行时间为0-13674ms

第二层： seq scan on tbATUC2I ， 扫描检查的条件为RATIO\_ALL<3，将符合条件的数据从buffer或磁盘中读出给上层运算，预计执行时间为0-26ms，预计输出行数为839行

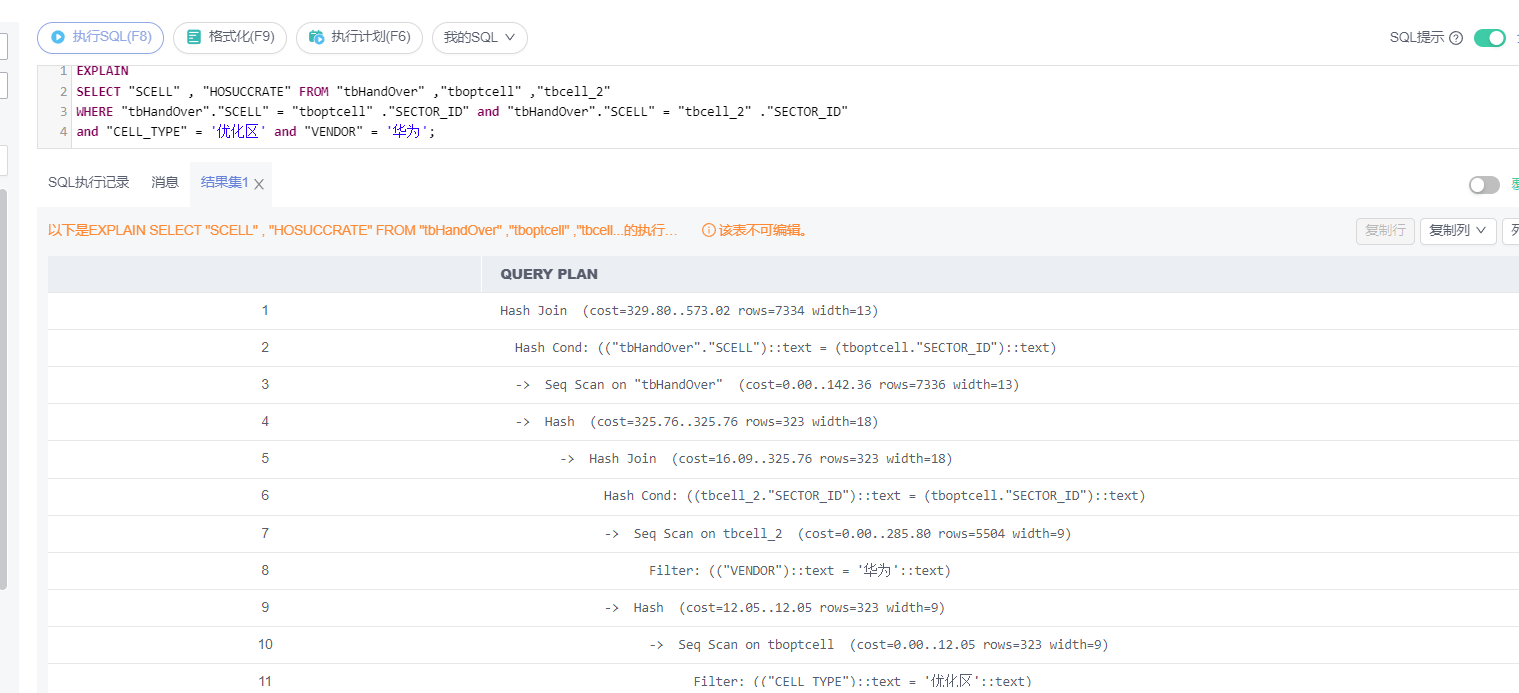
第三层：hash，将第二层的结果计算hash值，为后续hash join做准备

第四层： hash join，将tbadjceil 与 tbATUC2I连接在一起，连接条件为"tbAdjCell"."S\_SECTOR\_ID" = "tbATUC2I"."SECTOR\_ID" ， ，预计执行时间为36-3510ms

第五层：hash ， 将第四层的结果计算hash值，为后续hash join做准备

第六层： hash join ， 将第五层与第一层的结果连接，连接条件为("tbATUData"."CellID") = ("tbATUC2I"."SECTOR\_ID")，预计时间为5032.99-1200231ms预计输出108843563行

查询2：



解释

第一层：seq scan on tboptcell ， 扫描条件四cell type = “优化区”。将符合条件的数据从磁盘或buffer中读出

第二层：hash，将第一层结果求hash ， 为后续连接做准备

第三层 seq scan on tbcell ，扫描条件为vendor为“华为”，将符合条件的数据从磁盘或buffer中读出

第四层：hash join ，将第二三层的结果连接在一起，连接条件为 (tbcell\_2."SECTOR\_ID")= (tboptcell."SECTOR\_ID")

第五层：hash ， 将第四层的结果求hash，为后续连接做准备

第六层 ： seq scan on tbhandover ， 将所有的数据从磁盘或buffer中读出

第七层： hash join，将第五层与第六层的结果进行连接输出，连接条件为("tbHandOver"."SCELL") = (tboptcell."SECTOR\_ID")，总共花费时间预计为329-573ms，输出7334行

# 2.2 观察视图查询，with临时视图查询的执行计划

1. 在tbcel\_2l上建立一个视图



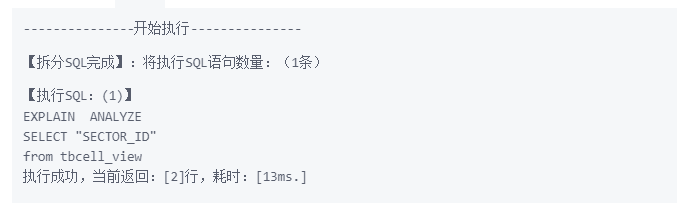
2. 查询并查看执行计划

* 1. 使用视图进行查询

EXPLAIN ANALYZE

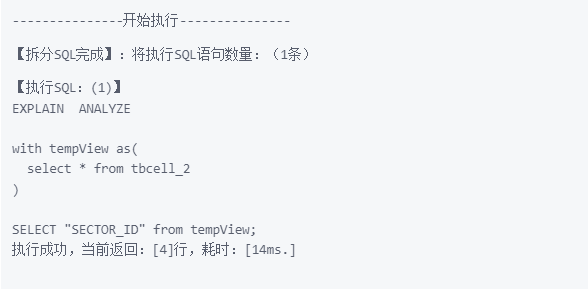
SELECT "SECTOR\_ID"

from tbcell\_view



执行结果：



* 1. 使用with临时视图进行查询  
     



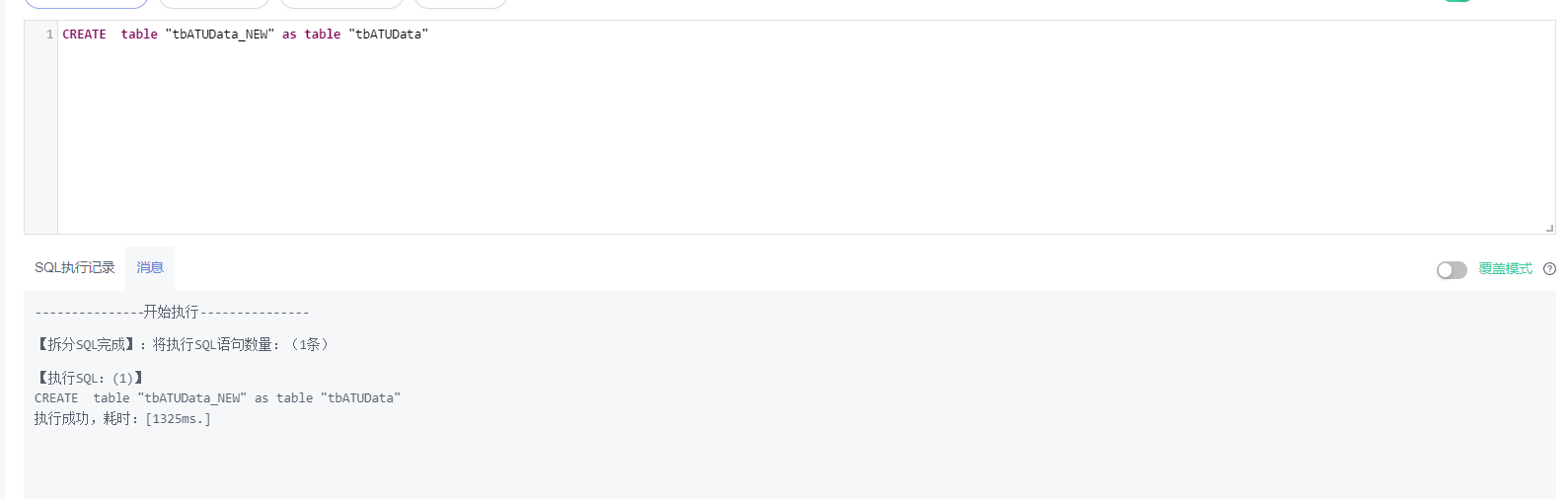
* 1. 不使用视图进行查询  
     

分析：通过上述实验可知，建立与不建立视图在查询时间上影响不大，但使用with进行查询时，会将查询分成两层，第一层首先完成了with内的查询，第二层再从第一层结果中进行查询。所以时间稍微较长一些。建立视图后，可以直接从视图中进行查询，所以速度略快一些。所以当经常需要进行相关查询时可以建立视图，而少用到的视图可以使用with建立临时视图

# 2.3 优化sql语句

**2.3.1 复合索引做前缀**

1. 创建tbATUData的备份表（去除索引以及其他约束）



1. 在tbATUData\_NEW创建组合索引

CREATE index tbATUData\_index on "tbATUData\_NEW"("Longitude","Latitude","CellID")

 3. 最左前缀访问atudate

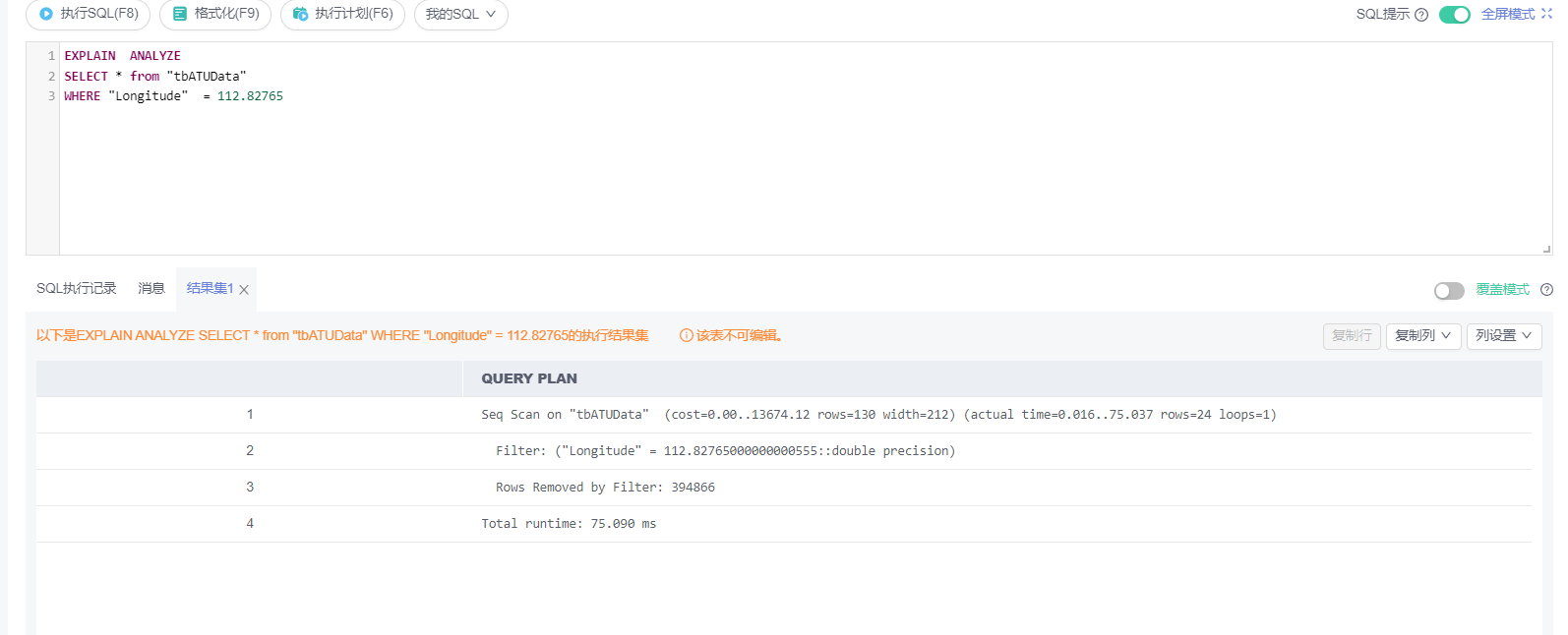
Sql:

EXPLAIN ANALYZE

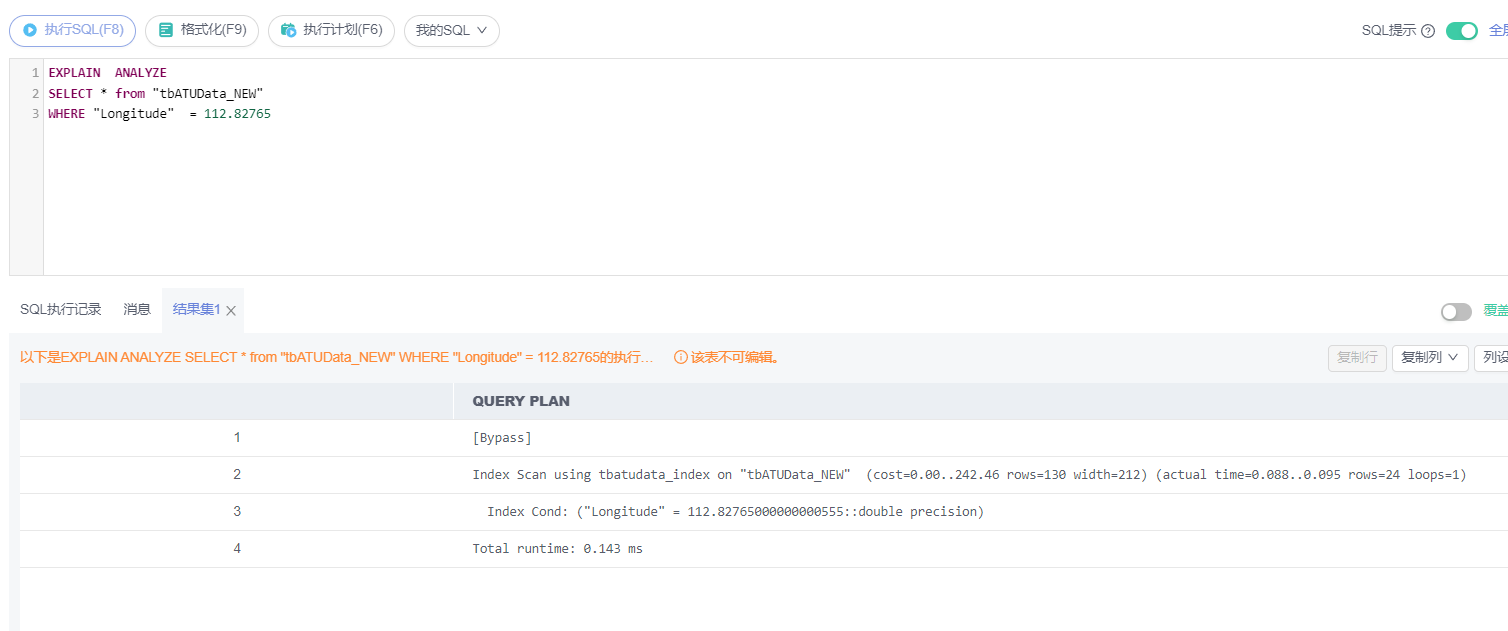
SELECT \* from "tbATUData"

WHERE "Longitude" = 112.82765

结果



使用索引



结论：符合最左前缀规则的查询可以利用索引大大提高搜索速度

1. 组合索引中除最左前缀索引外的其他索引与无索引的区别

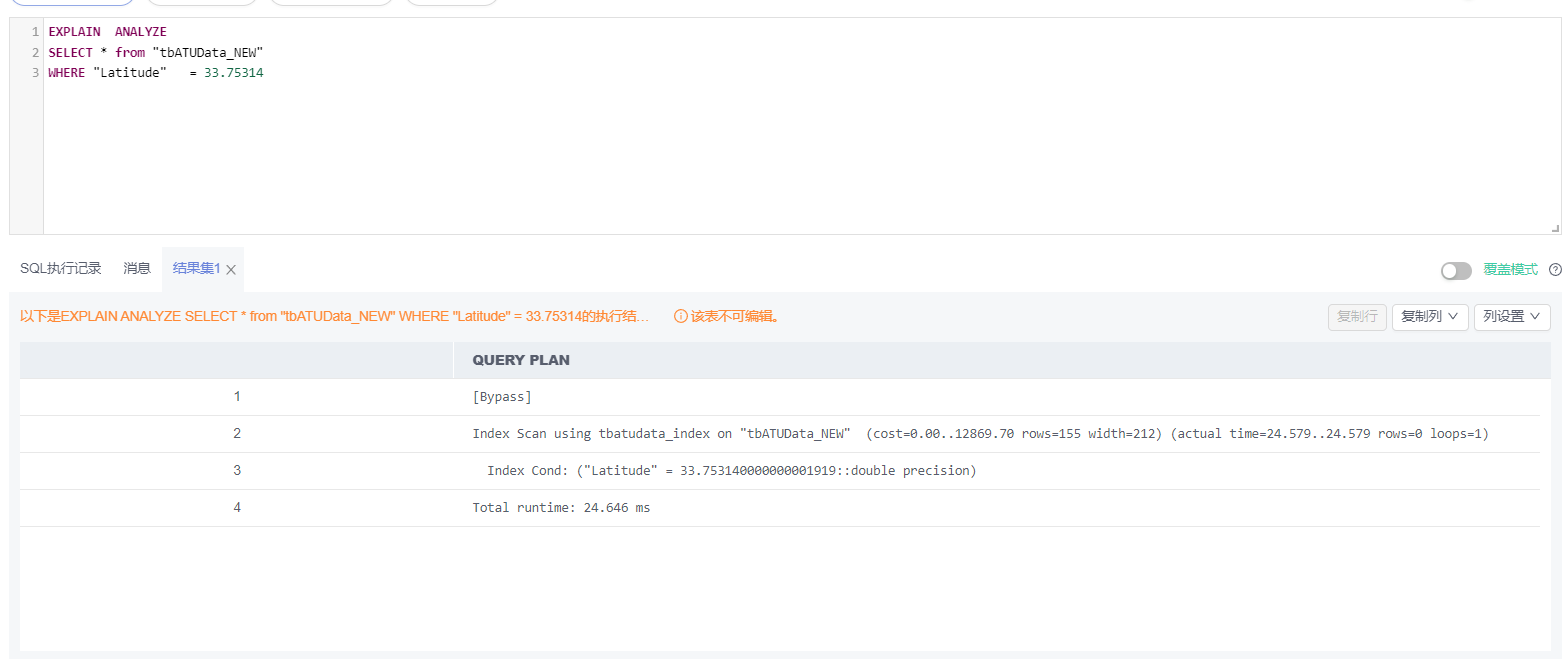
Sql：

EXPLAIN ANALYZE

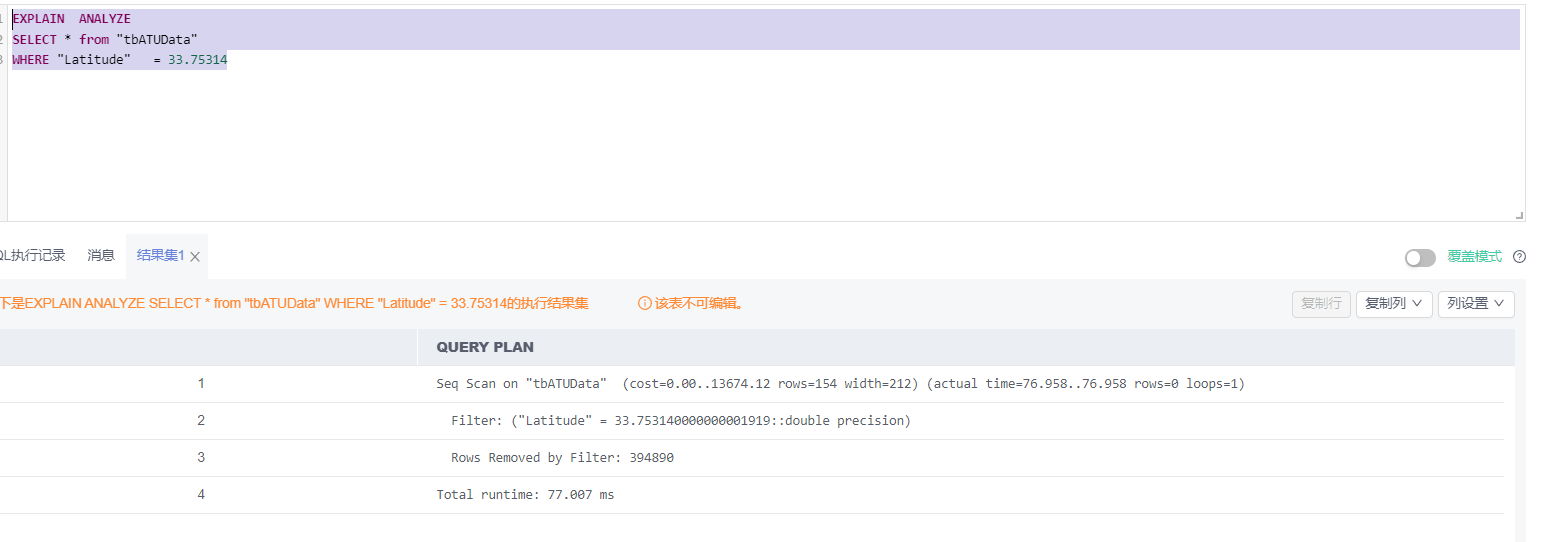
SELECT \* from "tbATUData"

WHERE "Latitude" = 33.75314

不满足最左前缀规则的查询



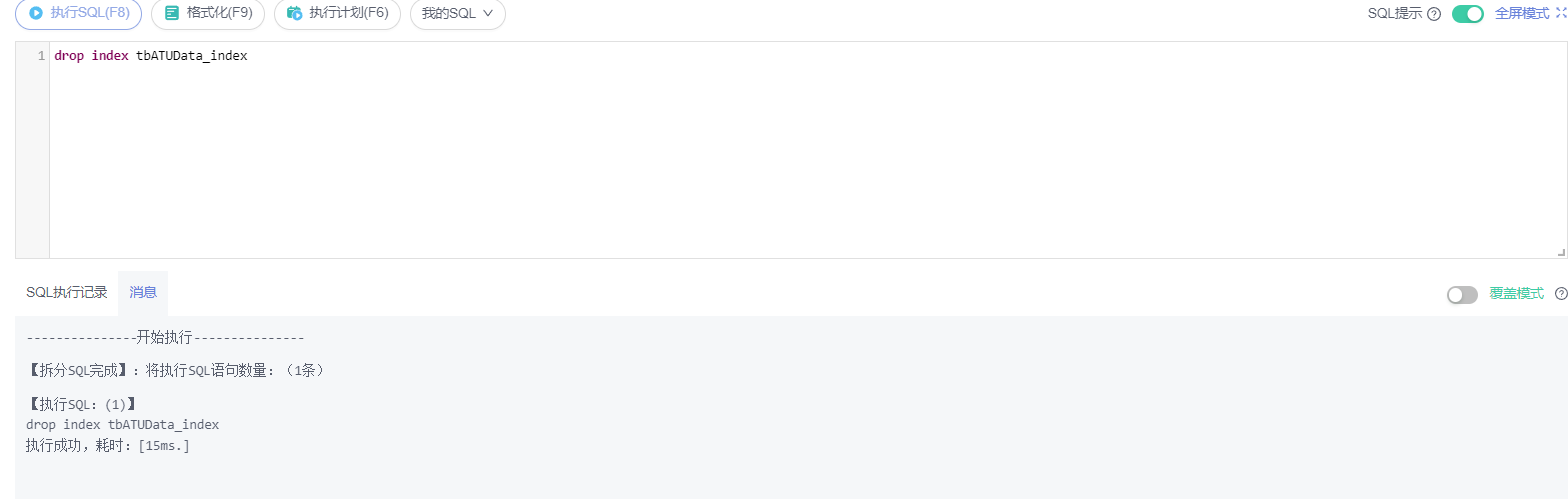
不使用索引



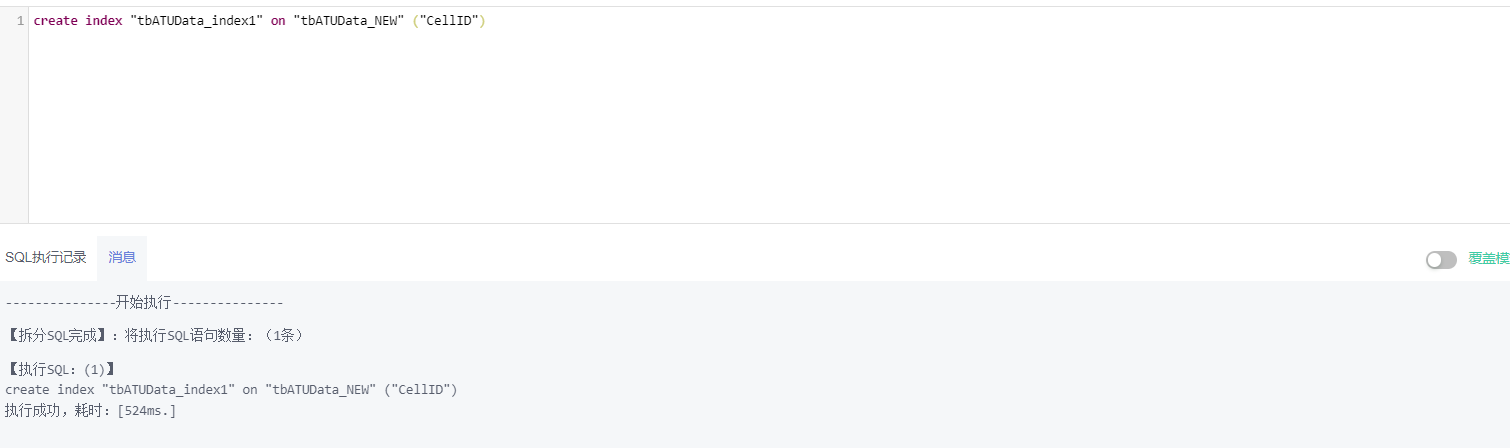
分析：尽管不满足最左前缀，但该索引仍然能加快搜索速度，但提高程度没有满足最左前缀的搜索高。故搜索速度由快到慢分别为 ： 符合最左前缀规则的搜索，不符合最左前缀规则的搜索，不使用索引的搜索

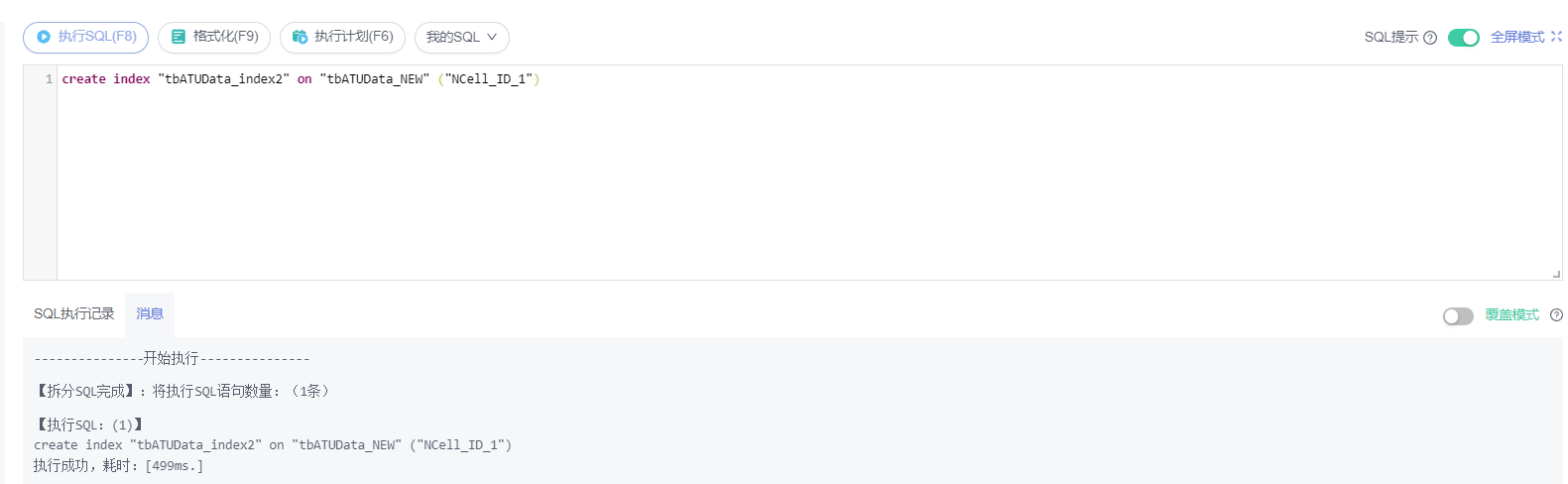
**2.3.2 多表连接操作，在连接属性上建立索引**

1. 删除上个实验的索引



1. 在 CellID 和 NCell\_ID\_1 上分别创建索引





1. 比较有索引与无索引的两条语句的执行情况
   1. 不使用CellID索引  
       sql：

EXPLAIN ANALYZE

select DISTINCT "CellID","S\_EARFCN"

from "tbATUData" ,"tbAdjCell"

where "CellID"='253903-0' AND "CellID"="S\_SECTOR\_ID";

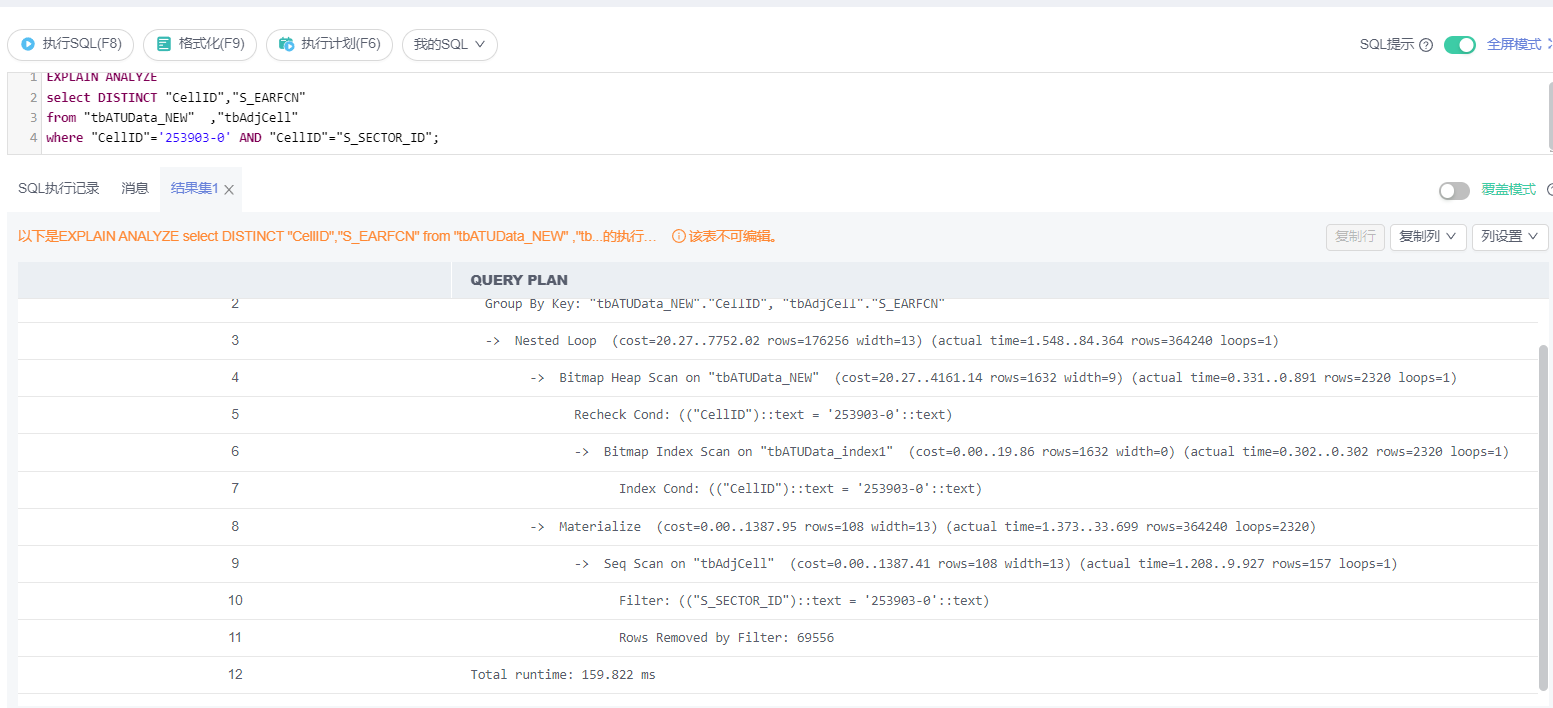


* 1. 使用CellID索引

EXPLAIN ANALYZE

select DISTINCT "CellID","S\_EARFCN"

from "tbATUData\_NEW" ,"tbAdjCell"

where "CellID"='253903-0' AND "CellID"="S\_SECTOR\_ID";  


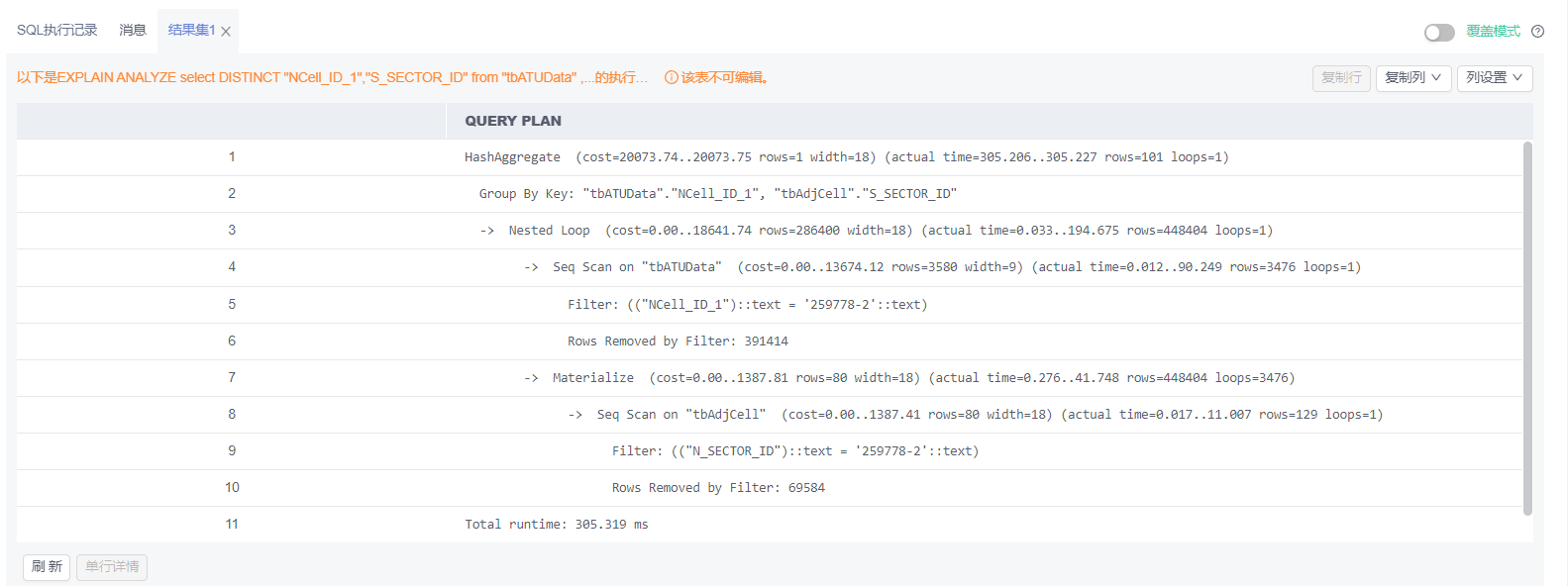
* 1. 不使用NCell\_ID\_1索引

EXPLAIN ANALYZE

select DISTINCT "NCell\_ID\_1","S\_SECTOR\_ID"

from "tbATUData" ,"tbAdjCell"

where "NCell\_ID\_1"= '259778-2' AND "NCell\_ID\_1"="N\_SECTOR\_ID";



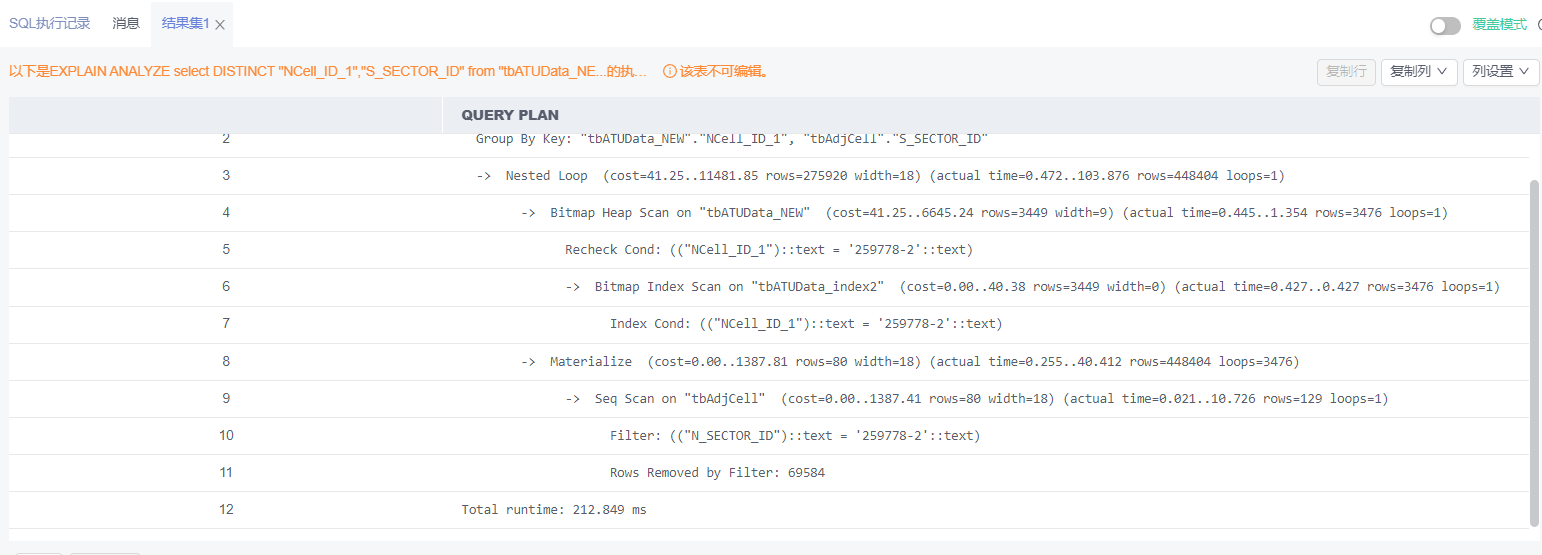
* 1. 使用NCell\_ID\_1索引

EXPLAIN ANALYZE

select DISTINCT "NCell\_ID\_1","S\_SECTOR\_ID"

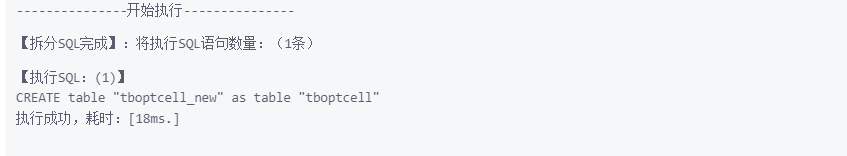
from "tbATUData\_NEW" ,"tbAdjCell"

where "NCell\_ID\_1"= '259778-2' AND "NCell\_ID\_1"="N\_SECTOR\_ID";



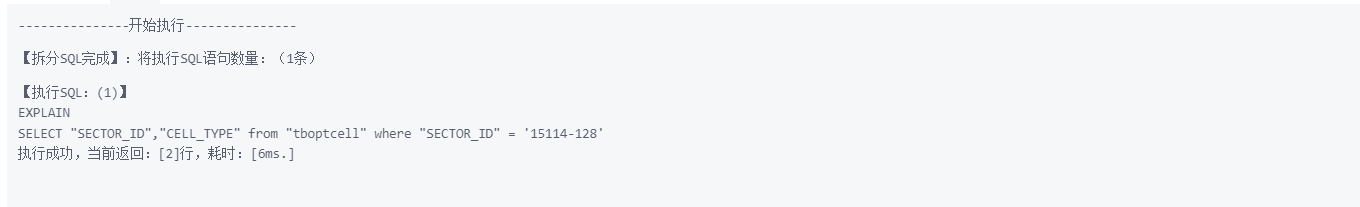
结果分析：尽管没有对两个键建立组合索引，但是有索引的搜索速度仍然快于没有索引的搜索

**2.3.3 索引对小表查询的搜索**

1. 对tbOptCell建立一个副本  
    CREATE table "tboptcell\_new" as table "tboptcell"   
   
2. 在副本上创建索引  
   
3. 在tbOptCell上查询，观察是否用到索引：

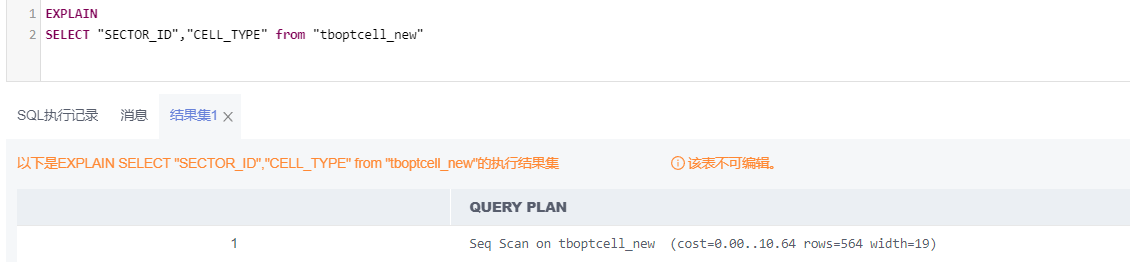
EXPLAIN

SELECT "SECTOR\_ID","CELL\_TYPE" from "tboptcell"



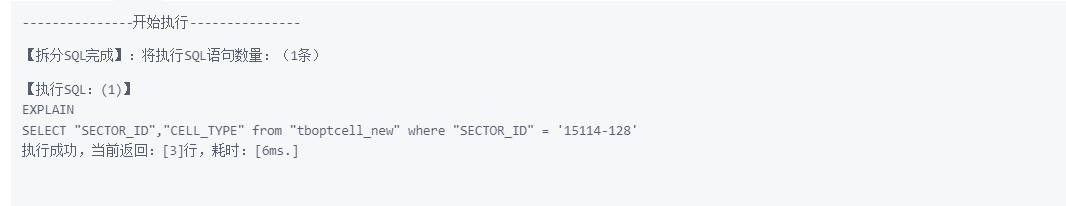


可见使用的是顺序扫描，故没有使用索引

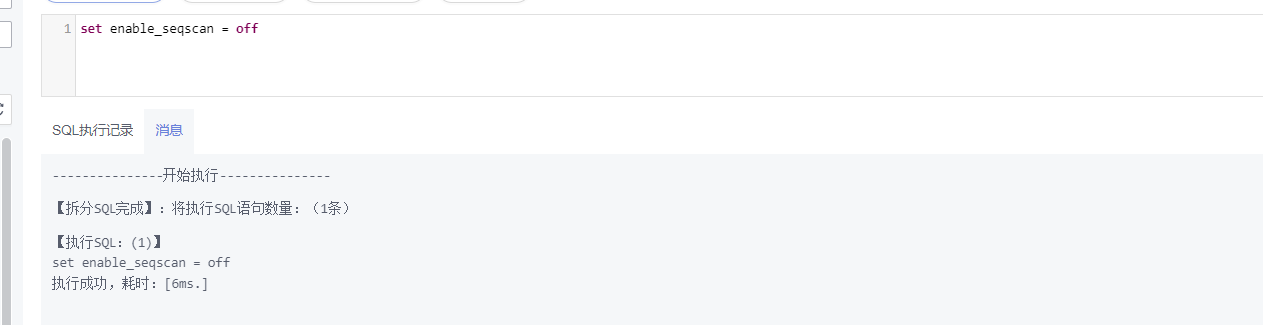
1. 在新表上查询：  
    

可见仍为顺序扫描，没有用到索引

若使用了where条件：则将使用索引





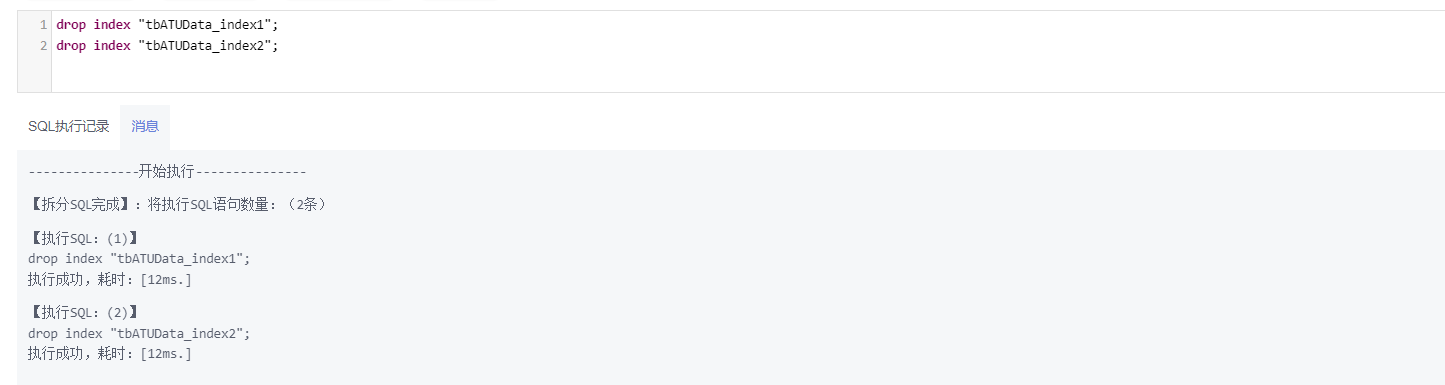
1. 强制使用索引查询tbOptCell\_new
2. 禁止顺序扫描  
   
3. 再次在tbOptCell上查询



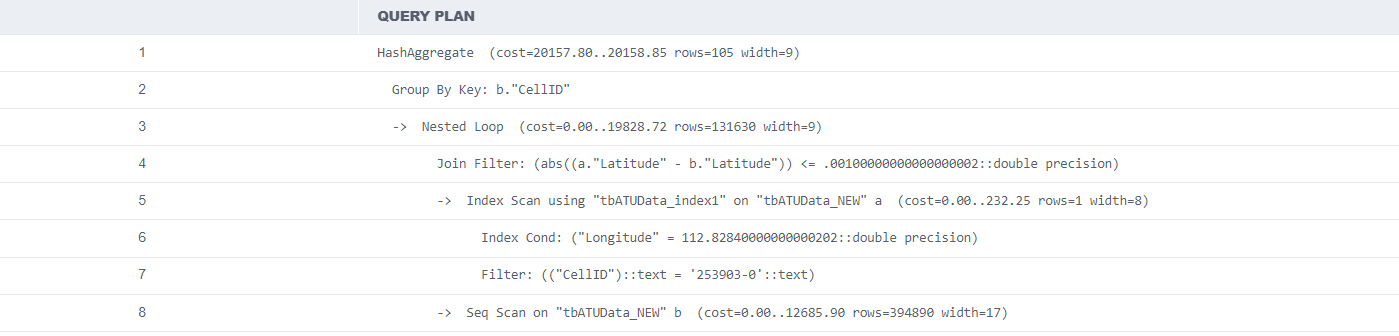


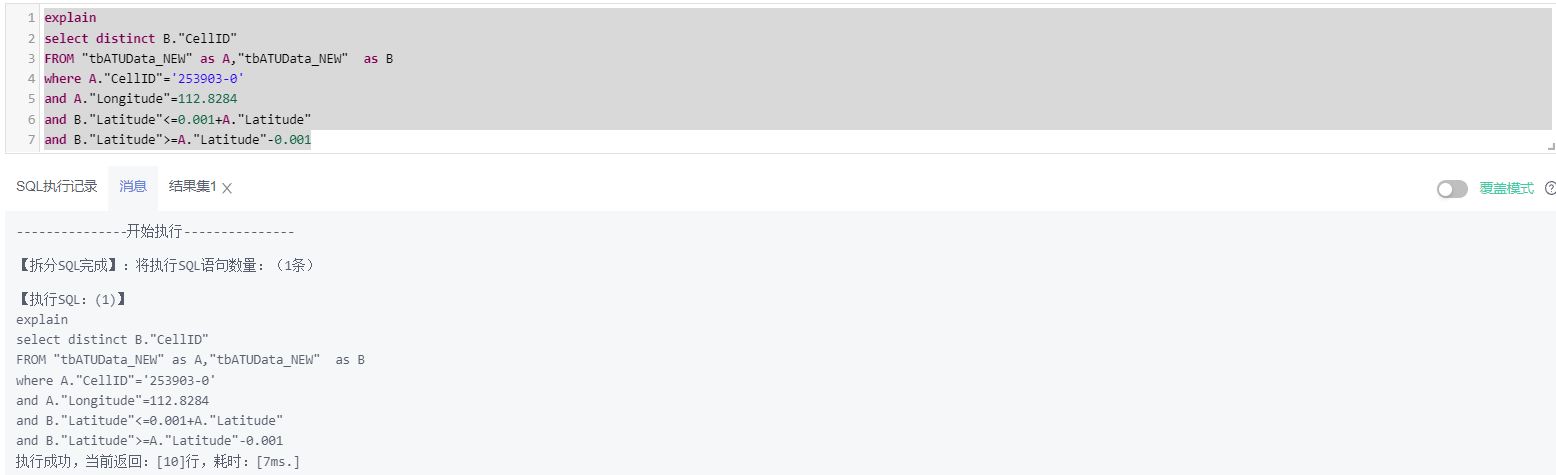
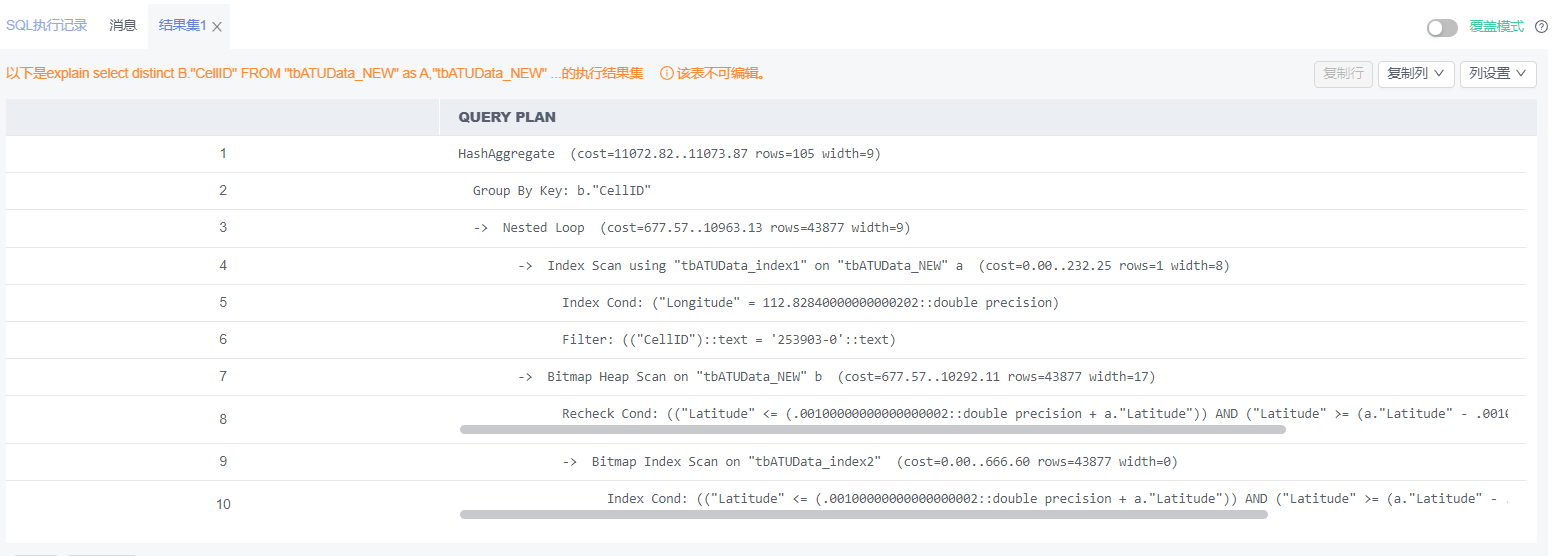
结果：在小表上强制使用索引反而会导致查询速度减慢。如果没有where语句，不管开不开顺序扫描，数据库都将默认顺序扫描

**2.3.4 查询条件中函数对索引的影响**

1. 删除原有的索引  


2. 建立新的索引  


3. 使用函数进行查询  
  


4. 不使用函数进行查询  
  


结果分析：第一个查询只在A.longitude条件中使用到了longitude索引，而没有使用到latitude索引。而第二个查询两个索引都使用到了，故第二个索引的速度会更快。而两个查询实际上是等价的，故当存在函数查询的时候，可以转换成等价的不存在函数的查询，以便使用索引以加快查询速度。

**2.3.5 多表嵌入式SQL查询**

1. 使用嵌套方式进行查询

explain analyze

select A."PCI"

from "tbPCIAssignment" as A

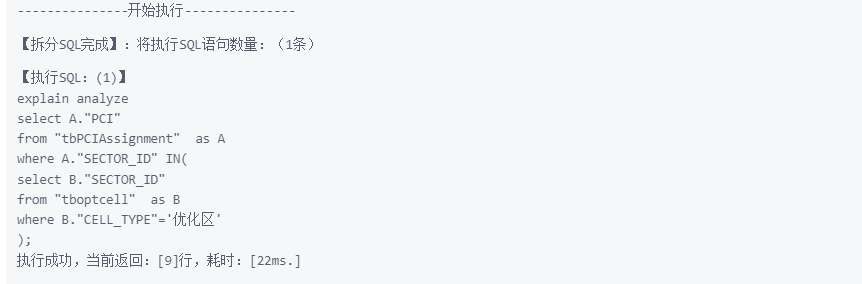
where A."SECTOR\_ID" IN(

select B."SECTOR\_ID"

from "tboptcell" as B

where B."CELL\_TYPE"='优化区'

);





1. 通过连接的方式进行查询

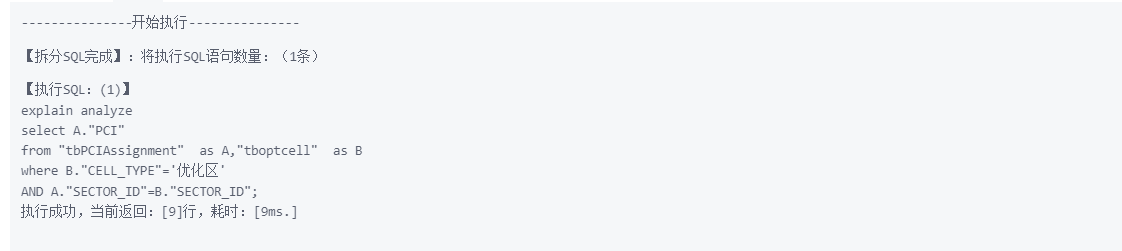
explain analyze

select A."PCI"

from "tbPCIAssignment" as A,"tboptcell" as B

where B."CELL\_TYPE"='优化区'

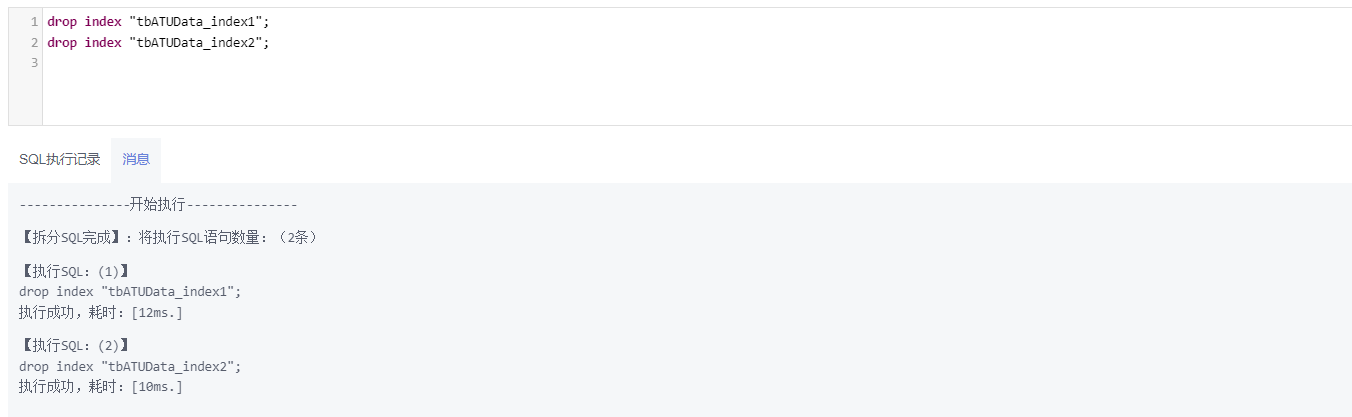
AND A."SECTOR\_ID"=B."SECTOR\_ID";





实验结论：无论是从理论上还是从执行计划上看，两者都是等价的，与书上说的嵌套内查询会被调用多次不同，这应该是数据库进行查询优化后的结果，从执行计划上来看，两者均为分别经过筛选出优化区，并计算hash值后再连接在一起。

**2.3.6 where 查询条件中复合查询条件 OR 对索引的影响**

1. 首先删除“tbATUData\_new”上原有的索引  
 

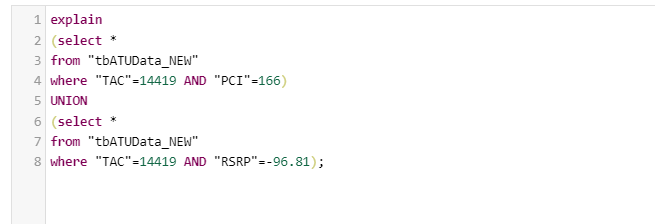
2. 在PCI上创建索引  


3. 查看包含A or B类型的sql语句的查询计划



可见该查询的查询计划很简单，直接顺序扫描所有的数据并选出符合要求的数据，没有使用到索引

4. 查看等价的Unionsql语句的查询计划





结果分析： 可见虽然这个查询使用到了索引，但由于需要分开扫描多次数据，导致在第一个select语句的预计时间就已经远超过了顺序扫描的时间，而最终union操作需要的时间代价更多，所以在查询中应该避免union操作

**2.3.7 聚集索引中的索引设计**

1. 删除原有的索引，在longitude以及latitude上分别重新建立索引。



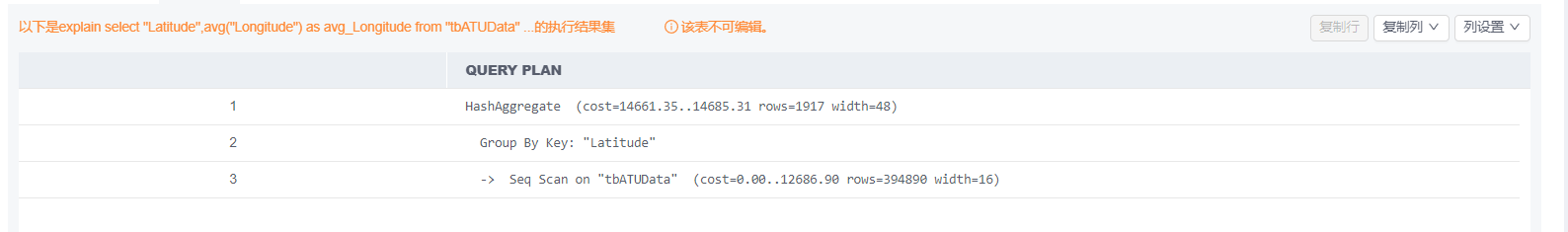
2. 查询无聚集索引运算的执行计划

explain

select "Latitude",avg("Longitude") as avg\_Longitude

from "tbATUData"

group by "Latitude";



1. 查询有聚集索引的执行计划

explain

select "Latitude",avg("Longitude") as avg\_Longitude

from "tbATUData\_NEW"

group by "Latitude";



由结果看出，虽然建立了索引，但数据库仍采用了顺序查询，故两者的预计时间基本相同。

1. 强制使用索引  
    set enable\_seqscan = off;

explain

select "Latitude",avg("Longitude") as avg\_Longitude

from "tbATUData\_NEW"

group by "Latitude";



由执行计划可见，强制开启索引后，顺序扫描变成了使用索引扫描，导致了执行效率更加低下。

**2.3.8 select子句中有无distinct的区别**

1. 无distinct

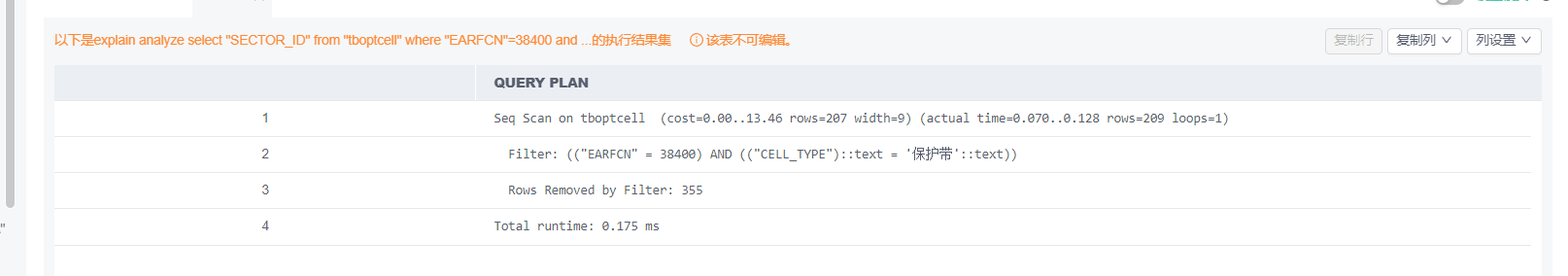
explain analyze

select "SECTOR\_ID"

from "tboptcell"

where "EARFCN"=38400

and "CELL\_TYPE"='保护带';



1. 有distinct

explain analyze

select DISTINCT "SECTOR\_ID"

from "tboptcell"

where "EARFCN"=38400

and "CELL\_TYPE"='保护带';



有两次执行计划的结果可以看出，如果不添加distinct，则数据库直接顺序扫描一遍数据就执行完毕了。而添加distinct后，数据库在扫描完成后还需执行group by语句，导致运行时间增加。但由于SECTOR\_ID本身就是主键，没有重复的值，所以不应该添加distinct徒增运行时间

**2.3.9 union 和 union all的区别**

1. 使用union进行查询

explain analyze

select "S\_SECTOR\_ID" from "tbAdjCell"

union

select "S\_SECTOR\_ID" from "tbSecAdjCell" ;



基本执行顺序为分别顺序扫描，然后通过group by 去除重复的数据

2. 使用 union all进行查询

explain analyze

select "S\_SECTOR\_ID" from "tbAdjCell"

union ALL

select "S\_SECTOR\_ID" from "tbSecAdjCell" ;



由执行计划可知，union all不会执行group by 操作，所以不会完成去重，当确定数据没有重复的或没有数据唯一的要求的画，可以使用union all提高运行速度。

**2.3.10 from 中存在多余的关系表，即查询非最简化**

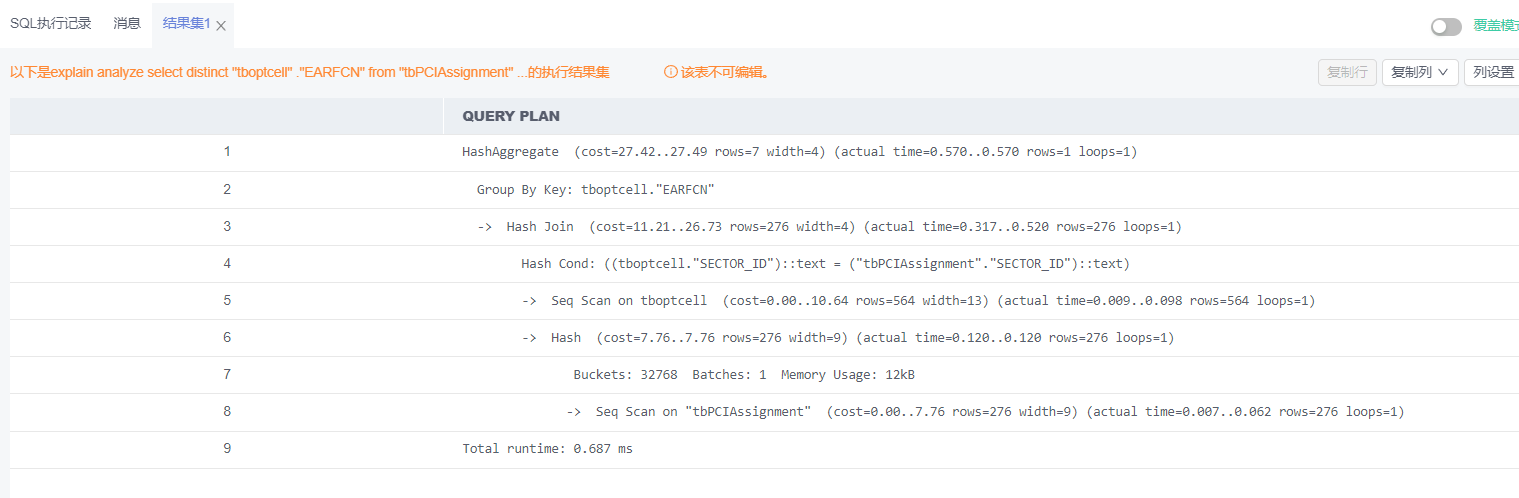
1. 查询语句最简化

explain analyze

select distinct "tboptcell" ."EARFCN"

from "tbPCIAssignment" ,"tboptcell"

where "tbPCIAssignment" ."SECTOR\_ID"="tboptcell" ."SECTOR\_ID";



2. 查询语句存在多余的关系表

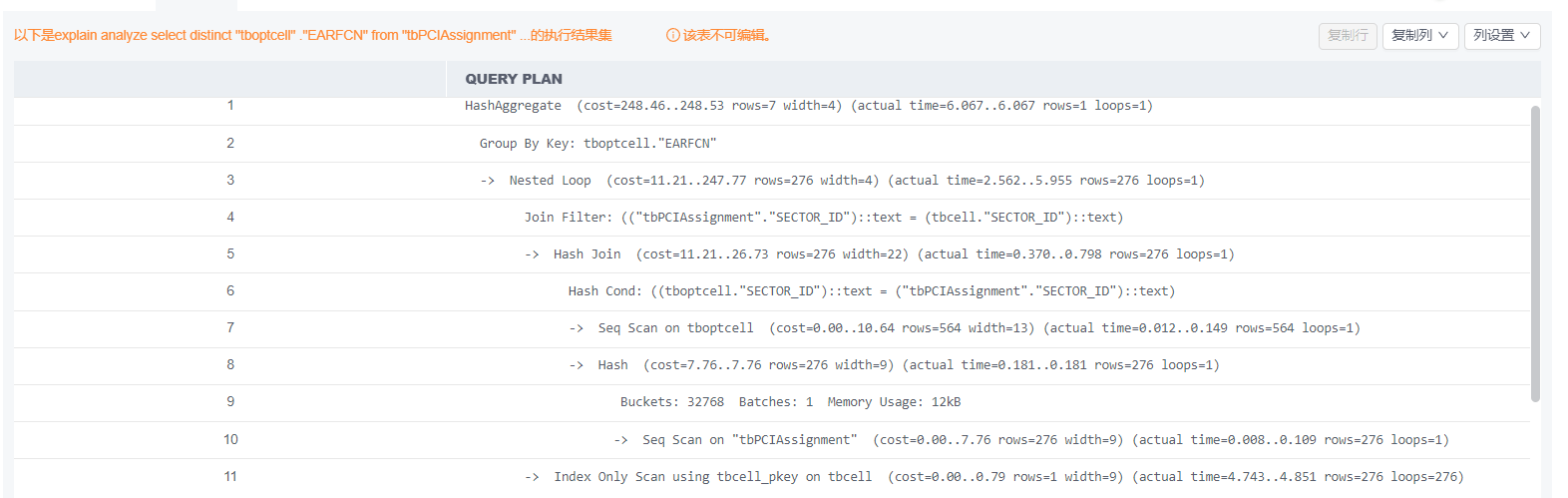
explain analyze

select distinct "tboptcell" ."EARFCN"

from "tbPCIAssignment" ,"tboptcell","tbcell"

where "tbPCIAssignment" ."SECTOR\_ID"="tboptcell" ."SECTOR\_ID"

and "tbcell"."SECTOR\_ID" = "tboptcell"."SECTOR\_ID"



易知两者的查询结果应该是一样的。因为tbcell是多余的表，最终查询中既没有出现tbcell中的字段，也没利用到tbcell 中的信息，但由于tbcell多执行了一次连接操作，所以导致最终运行时间大幅度增加。

# 遇到的问题及解决

问题一：在2.3.3中，禁止顺序扫描后，输入sql语句

Select “SECTOR\_ID”,”CELL\_TYPE”from “tboptcell\_new”

执行计划仍显示为顺序扫描

问题解决：若where子句中没有查询条件的话，数据库默认为顺序查询。故想要体现出小表即使有索引可能也不用应该将sql 改为

Select “SECTOR\_ID”,”CELL\_TYPE”from “tboptcell\_new” where “SECTOR\_ID” = ‘xxx’

# 实验总结

这一次实验的实验大部分都比较简单，均为简单的基本优化以及编写sql语句时的基本注意事项，如distinct会导致数据排序引起查询速度变慢都是上课介绍过的内容。但第一次接触到了explain语句，通过这个语句可以将sql语句转换成执行计划，能让我更加深刻地了解sql语句是怎么执行的、怎么程式化地，也让我更好地明白哪里才是sql语句中最耗时间地地方，哪里是应该需要优化地地方。通过这一次的实验，我进一步加深熟悉了explain的用法，对我的帮助很大。