《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

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。

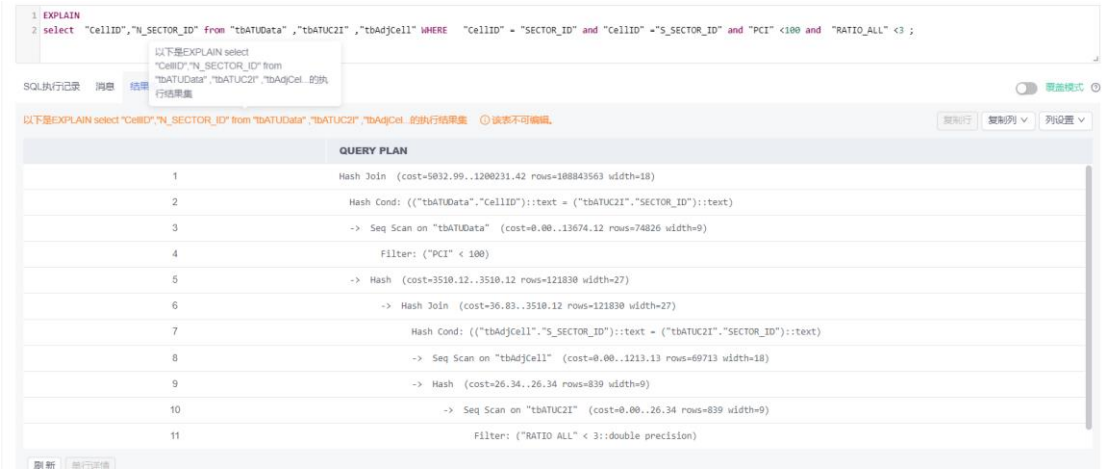
select "CellID","N_SECTOR_ID" from "tbATUData" ,"tbATUC2I" ,"tbAdjCell"
WHERE "PCI" <100 and "CellID" = "SECTOR_ID" and "RATIO_ALL" <3 and "CellID" ="S_SECTOR_ID";

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'

2. 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

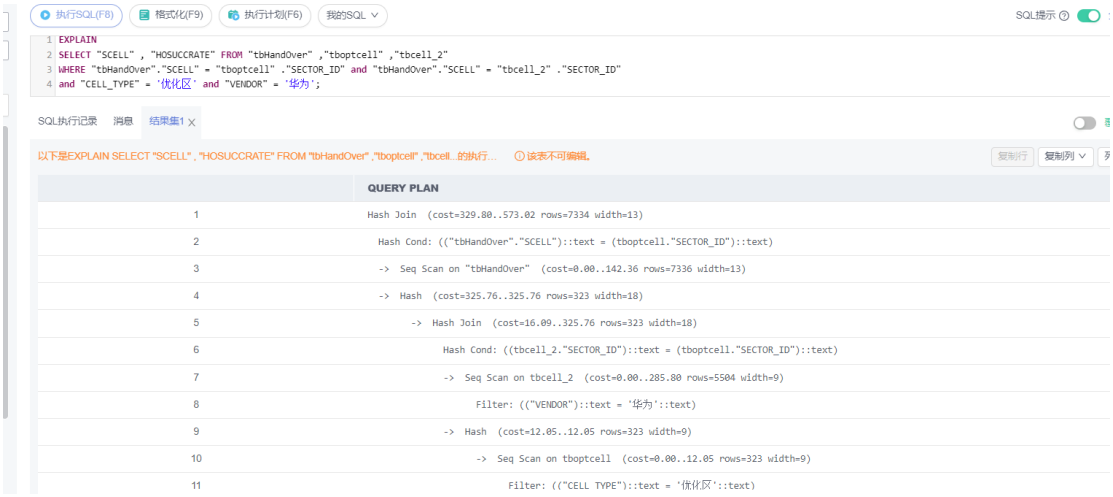
第四层： hash join，将 tbadjceil 与 tbATUC2I 连接在一起，连接条件为

"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

```
1 create view tbcell_view as
2 select * from tbcell_2
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
create view tbcell_view as
select * from tbcell_2
执行成功，耗时：[15ms.]
```

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
```

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
EXPLAIN  ANALYZE
SELECT "SECTOR_ID"
from tbcell_view
执行成功，当前返回：[2]行，耗时：[13ms.]
```

执行结果：

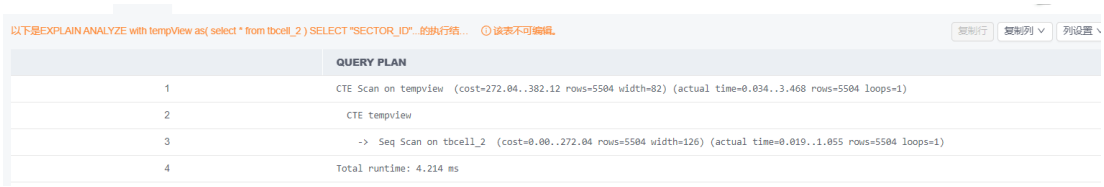
	QUERY PLAN
1	Seq Scan on tbcell_2 (cost=0.00..272.04 rows=5504 width=9) (actual time=0.022..1.290 rows=5504 loops=1)
2	Total runtime: 1.676 ms

(2) 使用 with 临时视图进行查询

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
EXPLAIN  ANALYZE

with tempView as(
  select * from tbcell_2
)

SELECT "SECTOR_ID" from tempView;
执行成功，当前返回：[4]行，耗时：[14ms.]
```



	QUERY PLAN
1	CTE Scan on tempview (cost=272.04..382.12 rows=5504 width=82) (actual time=0.034..3.468 rows=5504 loops=1)
2	CTE tempview
3	-> Seq Scan on tbcell_2 (cost=0.00..272.04 rows=5504 width=126) (actual time=0.019..1.055 rows=5504 loops=1)
4	Total runtime: 4.214 ms

(3) 不使用视图进行查询

```
EXPLAIN  ANALYZE
SELECT "SECTOR_ID" from tbcell_2;
```

SQL执行记录 消息 结果集1

以下是EXPLAIN ANALYZE SELECT "SECTOR_ID" from tbcell_2;的执行结果集 ⓘ该表不可编辑。 复制行 复制列 列设置

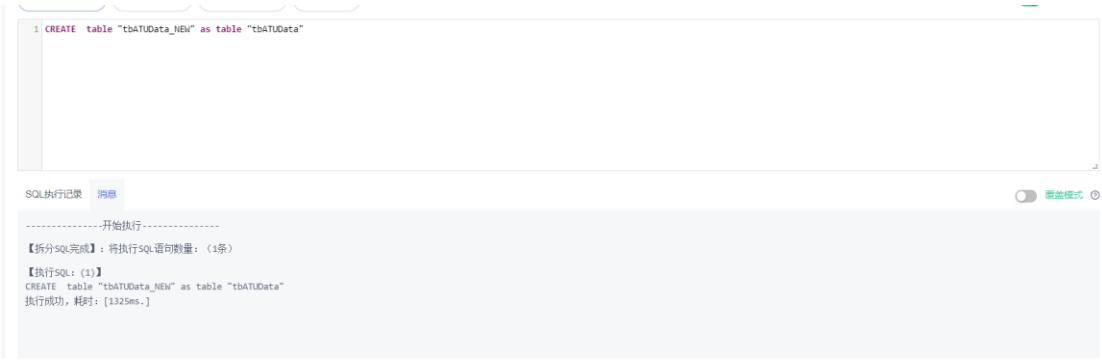
	QUERY PLAN
1	Seq Scan on tbcell_2 (cost=0.00..272.04 rows=5504 width=9) (actual time=0.021..1.292 rows=5504 loops=1)
2	Total runtime: 1.688 ms

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）



2. 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

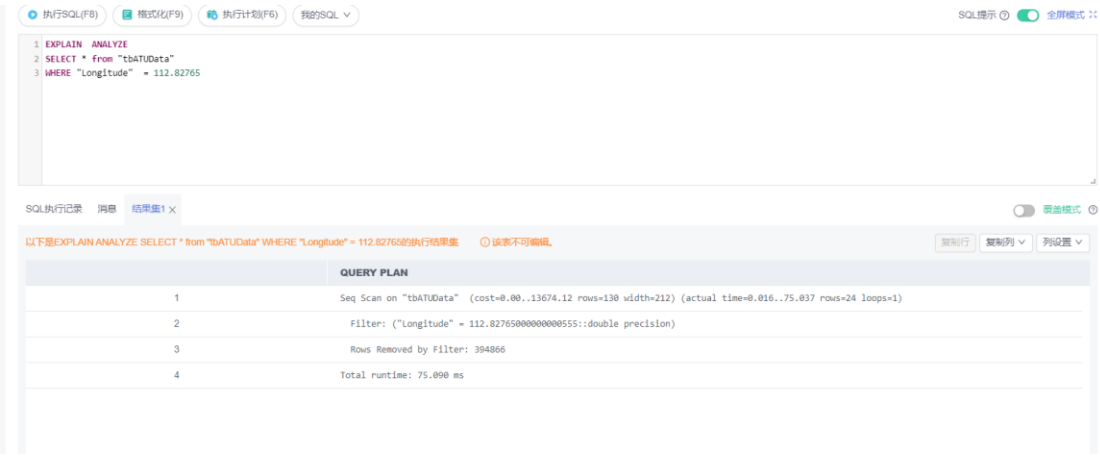
Sql:

EXPLAIN ANALYZE

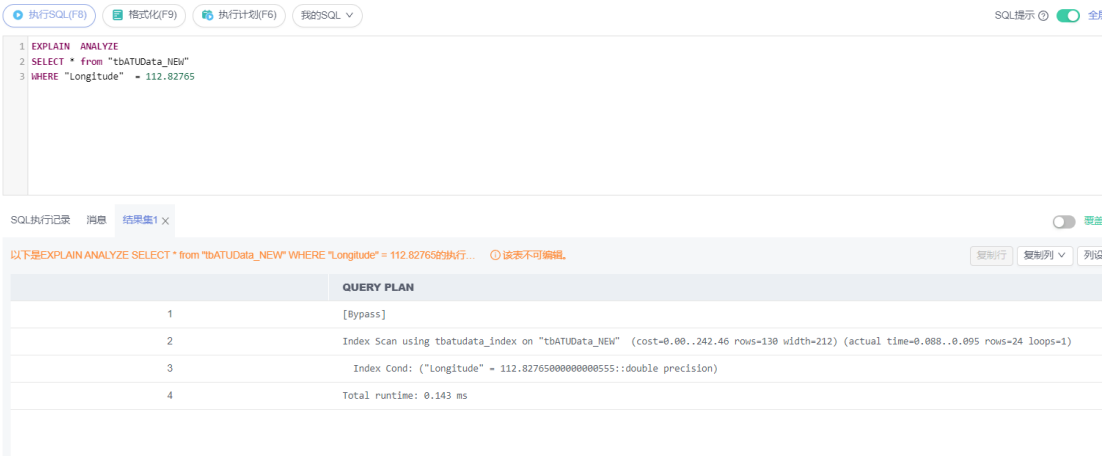
SELECT * from "tbATUData"

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

3. 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

```
EXPLAIN  ANALYZE
SELECT * from "tbATUData_NEW"
WHERE "Latitude"   = 33.75314
```

SQL执行记录 消息 结果集1

以下是EXPLAIN ANALYZE SELECT * from "tbATUData_NEW" WHERE "Latitude" = 33.75314的执行结... ⓘ该表不可编辑。

复制行 复制列 列设置

	QUERY PLAN
1	[Bypass]
2	Index Scan using tbatudata_index on "tbATUData_NEW" (cost=0.00..12869.70 rows=155 width=212) (actual time=24.579..24.579 rows=0 loops=1)
3	Index Cond: ("Latitude" = 33.753140000000001919::double precision)
4	Total runtime: 24.646 ms

不使用索引

```
EXPLAIN  ANALYZE
SELECT * from "tbATUData"
WHERE "Latitude"   = 33.75314
```

SQL执行记录 消息 结果集1

下是EXPLAIN ANALYZE SELECT * from "tbATUData" WHERE "Latitude" = 33.75314的执行结果集 ⓘ该表不可编辑。

复制行 复制列 列设置

	QUERY PLAN
1	Seq Scan on "tbATUData" (cost=0.00..13674.12 rows=154 width=212) (actual time=76.958..76.958 rows=0 loops=1)
2	Filter: ("Latitude" = 33.753140000000001919::double precision)
3	Rows Removed by Filter: 394890
4	Total runtime: 77.007 ms

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

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL SQL提示 全屏模式

```
drop index tbATUData_index
```

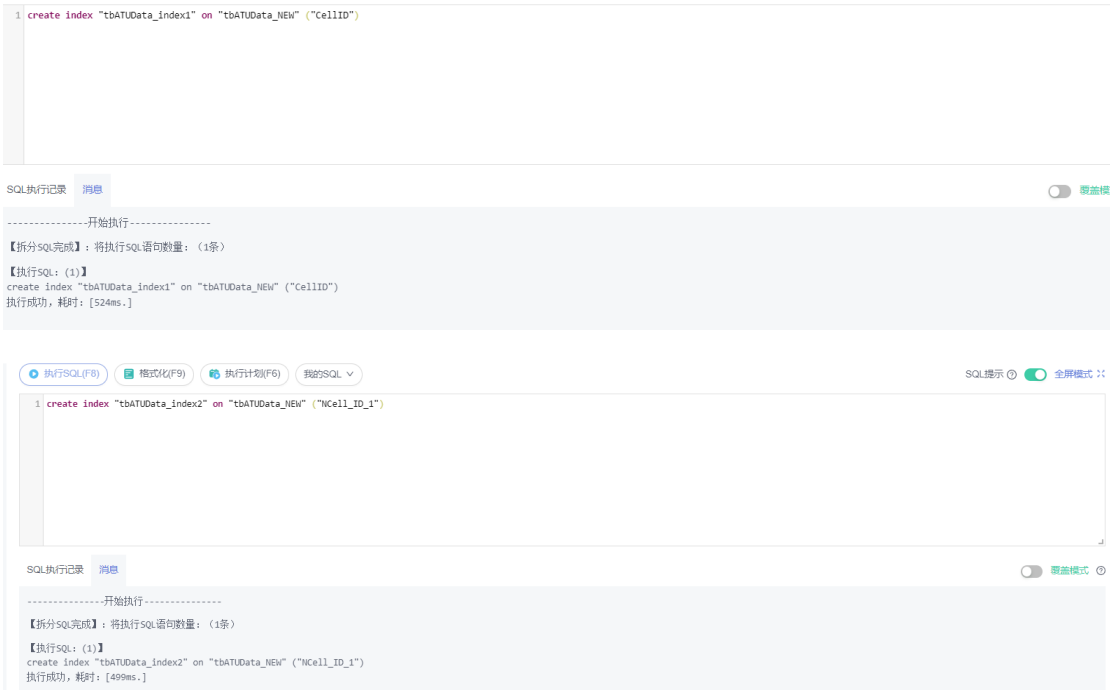
SQL执行记录 消息

----------开始执行----------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
drop index tbATUData_index
执行成功，耗时：[15ms.]

2. 在 CellID 和 NCell_ID_1 上分别创建索引



3. 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";

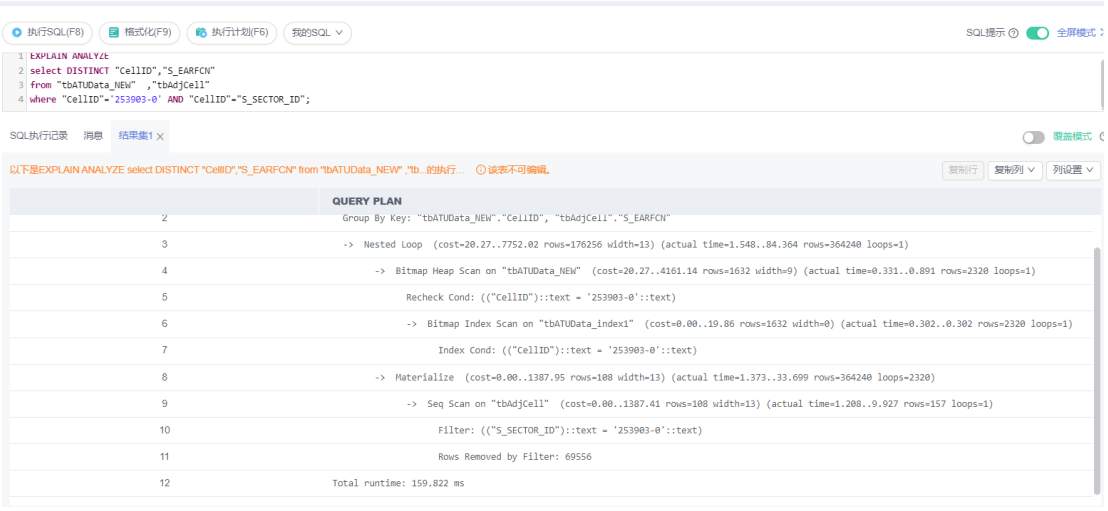
(2) 使用 CellID 索引

EXPLAIN ANALYZE

select DISTINCT "CellID","S_EARFCN"

from "tbATUData_NEW" ,"tbAdjCell"

where "CellID"='253903-0' AND "CellID"="S_SECTOR_ID";

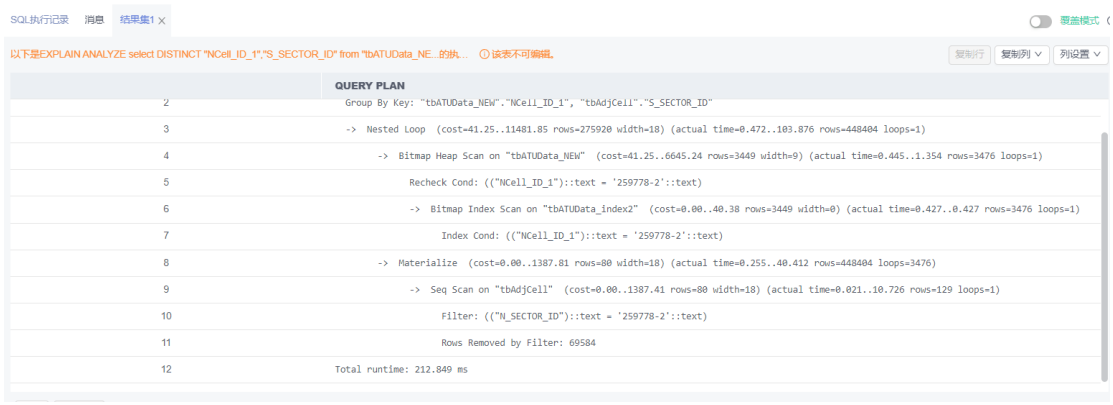


(3) 不使用 NCell_ID_1 索引

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";

(4) 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"

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
CREATE table "tboptcell_new" as table "tboptcell"
执行成功，耗时：[18ms.]
```

2. 在副本上创建索引

```
1 create index "tbOptCell_index" on "tboptcell_new" ("SECTOR_ID")
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
create index "tbOptCell_index" on "tboptcell_new" ("SECTOR_ID")
执行成功，耗时：[15ms.]
```

3. 在 tbOptCell 上查询，观察是否用到索引：

EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell"

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell" where "SECTOR_ID" = '15114-128'
执行成功，当前返回：[2]行，耗时：[6ms.]
```

可见使用的是顺序扫描，故没有使用索引

4. 在新表上查询：

```
EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new"
```

SQL执行记录　消息　结果集1 ×

以下是EXPLAIN SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new"的执行结果集　　ⓘ该表不可编辑。

	QUERY PLAN
1	Seq Scan on tboptcell_new (cost=0.00..10.64 rows=564 width=19)

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

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new" where "SECTOR_ID" = '15114-128'
执行成功，当前返回：[3]行，耗时：[6ms.]
```

```
EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new" where "SECTOR_ID" = '15114-128'
```

SQL执行记录　消息　结果集1 ×　　覆盖模式

以下是EXPLAIN SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new" where "SECTOR_ID"...的...　ⓘ该表不可编辑。　复制行　复制列　列设置

	QUERY PLAN
1	[Bypass]
2	Index Scan using "tbOptCell_index" on tboptcell_new (cost=0.00..2.28 rows=1 width=19)
3	Index Cond: (("SECTOR_ID")::text = '15114-128'::text)

5. 强制使用索引查询 tbOptCell_new

(1) 禁止顺序扫描

```
set enable_seqscan = off
```

SQL执行记录　消息

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
set enable_seqscan = off
执行成功，耗时：[6ms.]
```

(2) 再次在 tbOptCell 上查询

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
EXPLAIN
SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new" where "SECTOR_ID" = '15114-128'
执行成功，当前返回：[3]行，耗时：[7ms.]
```

以下是EXPLAIN SELECT "SECTOR_ID","CELL_TYPE" from "tboptcell_new" where "SECTOR_ID"...的...　ⓘ该表不可编辑。　复制行　复制列　列设置

	QUERY PLAN
1	[Bypass]
2	Index Scan using "tbOptCell_index" on tboptcell_new (cost=0.00..2.28 rows=1 width=19)
3	Index Cond: (("SECTOR_ID")::text = '15114-128'::text)

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

```
drop index "tbATUData_index1";
drop index "tbATUData_index2";
```

SQL执行记录 消息

```
--------------开始执行--------------
【拆分SQL完成】：将执行SQL语句数量：（2条）
【执行SQL：(1)】
drop index "tbATUData_index1";
执行成功，耗时：[12ms.]
【执行SQL：(2)】
drop index "tbATUData_index2";
执行成功，耗时：[12ms.]
```

2. 建立新的索引

```
CREATE index "tbATUData_index1" on "tbATUData_NEW" ("Longitude");
CREATE index "tbATUData_index2" on "tbATUData_NEW"("Latitude");
```

SQL执行记录 消息

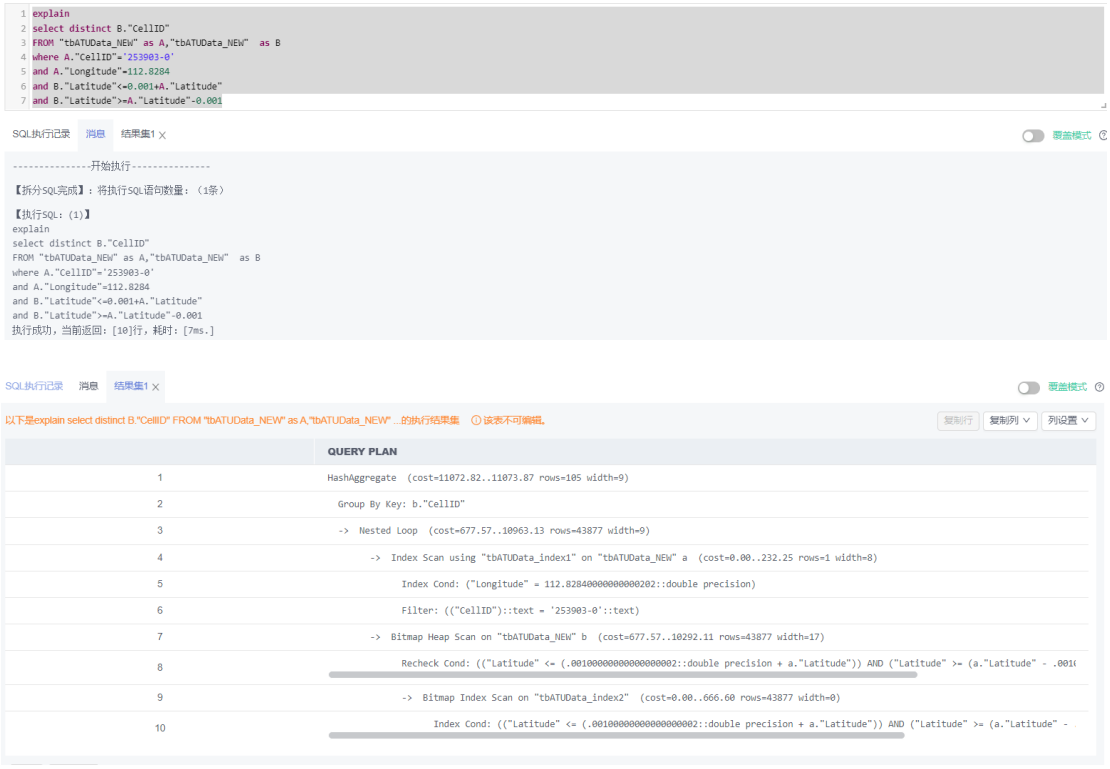
```
--------------开始执行--------------
【拆分SQL完成】：将执行SQL语句数量：（2条）
【执行SQL：(1)】
CREATE index "tbATUData_index1" on "tbATUData_NEW" ("Longitude");
执行成功，耗时：[401ms.]
【执行SQL：(2)】
CREATE index "tbATUData_index2" on "tbATUData_NEW"("Latitude");
执行成功，耗时：[405ms.]
```

3. 使用函数进行查询

```
--------------开始执行--------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
explain
select distinct B."CellID"
FROM "tbATUData_NEW" as A,"tbATUData_NEW"  as B
where A."CellID"='253903-0'
and A."Longitude"=112.8284
and ABS(A."Latitude"-B."Latitude")<=0.001;
执行成功，当前返回：[8]行，耗时：[11ms.]
```

	QUERY PLAN
1	HashAggregate (cost=20157.80..20158.85 rows=105 width=9)
2	Group By Key: b."CellID"
3	-> Nested Loop (cost=0.00..19828.72 rows=131630 width=9)
4	Join Filter: (abs((a."Latitude" - b."Latitude")) <= .0010000000000000002::double precision)
5	-> Index Scan using "tbATUData_index1" on "tbATUData_NEW" a (cost=0.00..232.25 rows=1 width=8)
6	Index Cond: ("Longitude" = 112.82840000000000202::double precision)
7	Filter: (("CellID")::text = '253903-0'::text)
8	-> Seq Scan on "tbATUData_NEW" b (cost=0.00..12685.90 rows=394890 width=17)

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'
);
```

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
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'
);
执行成功，当前返回：[9]行，耗时：[22ms.]
```

以下是explain analyze select A."PCI" from "tbPCIAssignment" as A where A."SECTOR_ID"...的执行结果集　⓪ 该表不可编辑。

复制行　复制列　列设置

	QUERY PLAN
1	Hash Right Semi Join (cost=11.21..27.00 rows=246 width=4) (actual time=1.555..1.763 rows=276 loops=1)
2	Hash Cond: ((b."SECTOR_ID")::text = (a."SECTOR_ID")::text)
3	-> Seq Scan on tboptcell b (cost=0.00..12.05 rows=323 width=9) (actual time=0.013..0.120 rows=323 loops=1)
4	Filter: (("CELL_TYPE")::text = '优化区'::text)
5	Rows Removed by Filter: 241
6	-> Hash (cost=7.76..7.76 rows=276 width=13) (actual time=1.348..1.348 rows=276 loops=1)
7	Buckets: 32768 Batches: 1 Memory Usage: 13kB
8	-> Seq Scan on "tbPCIAssignment" a (cost=0.00..7.76 rows=276 width=13) (actual time=0.770..1.270 rows=276 loops=1)
9	Total runtime: 1.894 ms

刷新　单行详情

2. 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";
```

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
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";
执行成功，当前返回：[9]行，耗时：[9ms.]
```

以下是explain analyze select A."PCI" from "tbPCIAssignment" as A,"tboptcell" as B w...的执行结果集　⓪ 该表不可编辑。

复制行　复制列　列设置

	QUERY PLAN
1	Hash Join (cost=11.21..27.70 rows=323 width=4) (actual time=0.303..0.490 rows=276 loops=1)
2	Hash Cond: ((b."SECTOR_ID")::text = (a."SECTOR_ID")::text)
3	-> Seq Scan on tboptcell b (cost=0.00..12.05 rows=323 width=9) (actual time=0.014..0.129 rows=323 loops=1)
4	Filter: (("CELL_TYPE")::text = '优化区'::text)
5	Rows Removed by Filter: 241
6	-> Hash (cost=7.76..7.76 rows=276 width=13) (actual time=0.125..0.125 rows=276 loops=1)
7	Buckets: 32768 Batches: 1 Memory Usage: 13kB
8	-> Seq Scan on "tbPCIAssignment" a (cost=0.00..7.76 rows=276 width=13) (actual time=0.007..0.074 rows=276 loops=1)
9	Total runtime: 0.590 ms

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

```
drop index "tbATUData_index1";
drop index "tbATUData_index2";
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（2条）
【执行SQL：(1)】
drop index "tbATUData_index1";
执行成功，耗时：[12ms.]
【执行SQL：(2)】
drop index "tbATUData_index2";
执行成功，耗时：[10ms.]
```

2. 在 PCI 上创建索引

```
CREATE index tbATUData_index on "tbATUData_NEW" ("PCI")
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
CREATE index tbATUData_index on "tbATUData_NEW" ("PCI")
执行成功，耗时：[350ms.]
```

3. 查看包含 A or B 类型的 sql 语句的查询计划

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL SQL提示 全屏

```
explain
select *
from "tbATUData_NEW"
where "TAC"=14419 AND ("PCI"=166 OR "RSRP"=-96.81);
```

SQL执行记录 消息 结果集1

以下是explain select * from "tbATUData_NEW" where "TAC"=14419 AND ("PCI"=166 OR "RS...的执行结... 该表不可编辑。 复制行 复制列 列设

	QUERY PLAN
1	Seq Scan on "tbATUData_NEW" (cost=0.00..15647.58 rows=1956 width=212)
2	Filter: ((("TAC" = 14419) AND (("PCI" = 166) OR ("RSRP" = (-96.810000000000023)::double precision)))

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

```
explain
(select *
from "tbATUData_NEW"
where "TAC"=14419 AND "PCI"=166)
UNION
(select *
from "tbATUData_NEW"
where "TAC"=14419 AND "RSRP"=-96.81);
```

	QUERY PLAN
1	HashAggregate (cost=19009.00..19028.04 rows=1904 width=212)
2	Group By Key: bupt2019dbs37."tbATUData_NEW".seq, bupt2019dbs37."tbATUData_NEW"."FileName", bupt2019dbs37."tbATUData_NEW"."Time", bupt2019db:
3	-> Append (cost=18.05..18842.40 rows=1904 width=212)
4	-> Bitmap Heap Scan on "tbATUData_NEW" (cost=18.05..4163.01 rows=1505 width=212)
5	Recheck Cond: ("PCI" = 166)
6	Filter: ("TAC" = 14419)
7	-> Bitmap Index Scan on tbatudata_index (cost=0.00..17.68 rows=1632 width=0)
8	Index Cond: ("PCI" = 166)
9	-> Seq Scan on "tbATUData_NEW" (cost=0.00..14660.35 rows=399 width=212)
10	Filter: (("TAC" = 14419) AND ("RSRP" = (-96.8100000000000023)::double precision))

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。

```
drop index tbatudata_index;
CREATE index tbatudata_index1 on "tbATUData_NEW" ("Longitude");
CREATE index tbatudata_index2 on "tbATUData_NEW" ("Latitude");
```

SQL执行记录 消息

```
--------------开始执行--------------
【拆分SQL完成】：将执行SQL语句数量：（3条）
【执行SQL：(1)】
drop index tbatudata_index;
执行成功，耗时：[11ms.]
【执行SQL：(2)】
CREATE index tbatudata_index1 on "tbATUData_NEW" ("Longitude");
执行成功，耗时：[373ms.]
【执行SQL：(3)】
CREATE index tbatudata_index2 on "tbATUData_NEW" ("Latitude");
执行成功，耗时：[409ms.]
```

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";
```

	QUERY PLAN
1	HashAggregate (cost=14661.35..14685.31 rows=1917 width=48)
2	Group By Key: "Latitude"
3	-> Seq Scan on "tbATUData" (cost=0.00..12686.90 rows=394890 width=16)

3. 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";
```

	QUERY PLAN
1	HashAggregate (cost=14660.35..14684.23 rows=1910 width=48)
2	Group By Key: "Latitude"
3	-> Seq Scan on "tbATUData_NEW" (cost=0.00..12685.90 rows=394890 width=16)

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。

4. 强制使用索引

```
set enable_seqscan = off;
explain
select "Latitude",avg("Longitude") as avg_Longitude
from "tbATUData_NEW"
group by "Latitude";
```

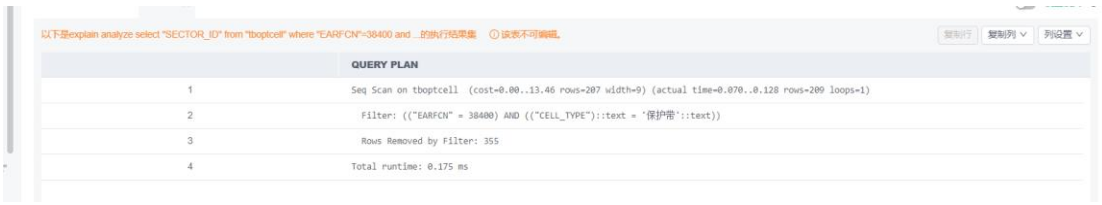
	QUERY PLAN
1	GroupAggregate (cost=0.00..46874.19 rows=1910 width=48)
2	Group By Key: "Latitude"
3	-> Index Scan using tbatudata_index2 on "tbATUData_NEW" (cost=0.00..44875.86 rows=394890 width=16)

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';
```



	QUERY PLAN
1	Seq Scan on tboptcell (cost=0.00..13.46 rows=207 width=9) (actual time=0.070..0.128 rows=209 loops=1)
2	Filter: (("EARFCN" = 38400) AND (("CELL_TYPE")::text = '保护带'::text))
3	Rows Removed by Filter: 355
4	Total runtime: 0.175 ms

2. 有 distinct

```
explain analyze
select DISTINCT    "SECTOR_ID"
from "tboptcell"
where "EARFCN"=38400
and "CELL_TYPE"='保护带';
```

	QUERY PLAN
1	HashAggregate (cost=13.98..16.05 rows=207 width=9) (actual time=0.272..0.303 rows=209 loops=1)
2	Group By Key: "SECTOR_ID"
3	-> Seq Scan on tboptcell (cost=0.00..13.46 rows=207 width=9) (actual time=0.105..0.197 rows=209 loops=1)
4	Filter: (("EARFCN" = 38400) AND (("CELL_TYPE")::text = '保护带'::text))
5	Rows Removed by Filter: 355
6	Total runtime: 0.391 ms

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" ;
```

以下是explain analyze select "S_SECTOR_ID" from "tbAdjCell" union select "S_SECTOR_...的执行结果集 ⓘ该表不可编辑。

	QUERY PLAN
1	HashAggregate (cost=3556.97..4763.18 rows=120621 width=9) (actual time=65.738..66.016 rows=546 loops=1)
2	Group By Key: "tbAdjCell"."S_SECTOR_ID"
3	-> Append (cost=0.00..3255.42 rows=120621 width=9) (actual time=0.712..43.501 rows=120621 loops=1)
4	-> Seq Scan on "tbAdjCell" (cost=0.00..1213.13 rows=69713 width=9) (actual time=0.711..17.957 rows=69713 loops=1)
5	-> Seq Scan on "tbSecAdjCell" (cost=0.00..836.08 rows=50908 width=9) (actual time=0.745..12.451 rows=50908 loops=1)
6	Total runtime: 66.270 ms

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" ;
```

以下是explain analyze select "S_SECTOR_ID" from "tbAdjCell" union ALL select "S_SEC...的执行结果集 ⓘ该表不可编辑。

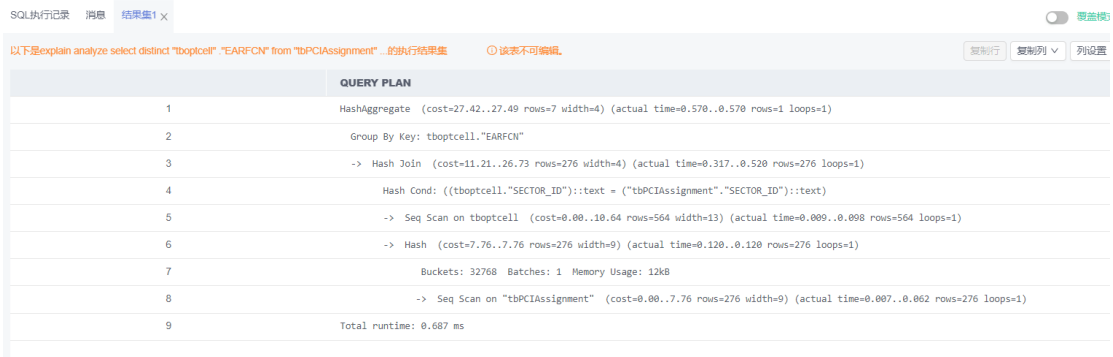
	QUERY PLAN
1	Result (cost=0.00..2049.21 rows=120621 width=9) (actual time=0.011..51.665 rows=120621 loops=1)
2	-> Append (cost=0.00..2049.21 rows=120621 width=9) (actual time=0.010..36.802 rows=120621 loops=1)
3	-> Seq Scan on "tbAdjCell" (cost=0.00..1213.13 rows=69713 width=9) (actual time=0.010..12.464 rows=69713 loops=1)
4	-> Seq Scan on "tbSecAdjCell" (cost=0.00..836.08 rows=50908 width=9) (actual time=0.014..10.321 rows=50908 loops=1)
5	Total runtime: 59.058 ms

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";
```



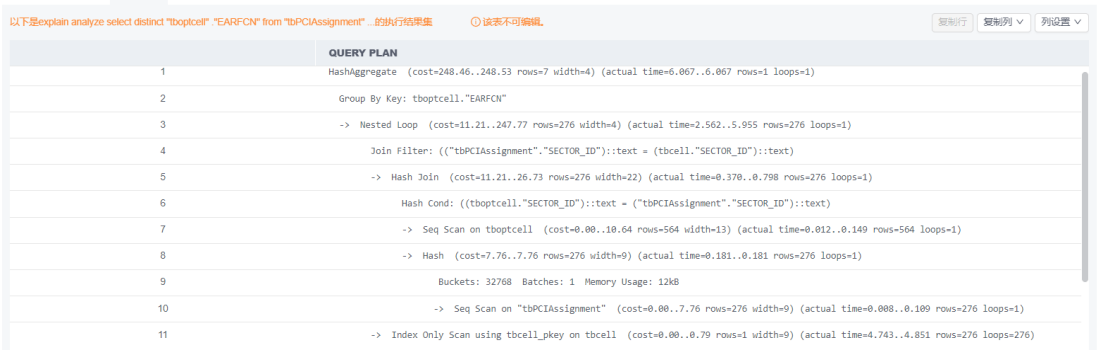
SQL执行记录 消息 结果集1

以下是explain analyze select distinct "tboptcell" ."EARFCN" from "tbPCIAssignment" ...的执行结果集 ⓘ该表不可编辑。

	QUERY PLAN
1	HashAggregate (cost=27.42..27.49 rows=7 width=4) (actual time=0.570..0.570 rows=1 loops=1)
2	Group By Key: tboptcell."EARFCN"
3	-> Hash Join (cost=11.21..26.73 rows=276 width=4) (actual time=0.317..0.520 rows=276 loops=1)
4	Hash Cond: ((tboptcell."SECTOR_ID")::text = ("tbPCIAssignment"."SECTOR_ID")::text)
5	-> Seq Scan on tboptcell (cost=0.00..10.64 rows=564 width=13) (actual time=0.009..0.098 rows=564 loops=1)
6	-> Hash (cost=7.76..7.76 rows=276 width=9) (actual time=0.120..0.120 rows=276 loops=1)
7	Buckets: 32768 Batches: 1 Memory Usage: 12kB
8	-> Seq Scan on "tbPCIAssignment" (cost=0.00..7.76 rows=276 width=9) (actual time=0.007..0.062 rows=276 loops=1)
9	Total runtime: 0.687 ms

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"
```



以下是explain analyze select distinct "tboptcell" ."EARFCN" from "tbPCIAssignment" ...的执行结果集　　① 该表不可编辑。

复制行　复制列　列设置

	QUERY PLAN
1	HashAggregate (cost=248.46..248.53 rows=7 width=4) (actual time=6.067..6.067 rows=1 loops=1)
2	Group By Key: tboptcell."EARFCN"
3	-> Nested Loop (cost=11.21..247.77 rows=276 width=4) (actual time=2.562..5.955 rows=276 loops=1)
4	Join Filter: (("tbPCIAssignment"."SECTOR_ID")::text = (tbcell."SECTOR_ID")::text)
5	-> Hash Join (cost=11.21..26.73 rows=276 width=22) (actual time=0.370..0.798 rows=276 loops=1)
6	Hash Cond: ((tboptcell."SECTOR_ID")::text = ("tbPCIAssignment"."SECTOR_ID")::text)
7	-> Seq Scan on tboptcell (cost=0.00..10.64 rows=564 width=13) (actual time=0.012..0.149 rows=564 loops=1)
8	-> Hash (cost=7.76..7.76 rows=276 width=9) (actual time=0.181..0.181 rows=276 loops=1)
9	Buckets: 32768 Batches: 1 Memory Usage: 12kB
10	-> Seq Scan on "tbPCIAssignment" (cost=0.00..7.76 rows=276 width=9) (actual time=0.008..0.109 rows=276 loops=1)
11	-> Index Only Scan using tbcell_pkey on tbcell (cost=0.00..0.79 rows=1 width=9) (actual time=4.743..4.851 rows=276 loops=276)

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。