# 2019320097_조이강

1. Create indexes on attribute “recordid” in “table_btree” and “table_hash”
   - Create b-tree in “table_btree.recordid”

```
d2019320097=# create index idx_btree on table_btree using btree (recordid);
CREATE INDEX
```

   - Create hash index in “table_hash.recordid”

```
d2019320097=# create index idt_hash on table_hash using hash (recordid);
CREATE INDEX
```

2. Run two queries and compare the query execution plan and execution time
   - SELECT * FROM table_btree WHERE recordid=10001;

```
d2019320097=# explain analyze select * from table_btree where recordid = 10001;
                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Index Scan using idx_btree on table_btree  (cost=0.43..8.45 rows=1 width=49) (actual time=0.099..0.101 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 7.649 ms
 Execution Time: 0.139 ms
(4개 행)
```

   - SELECT * FROM table_hash WHERE recordid=10001;

```
d2019320097=# explain analyze select * from table_hash where recordid = 10001;
                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Index Scan using idt_hash on table_hash  (cost=0.00..8.02 rows=1 width=49) (actual time=0.090..0.092 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 12.380 ms
 Execution Time: 0.118 ms
(4개 행)
```

Hash index가 실행시간이 약간 빠릅니다

Run two queries and compare the query execution plan and execution time

- SELECT * FROM table_btree WHERE recordid>250 AND recordid<550;

```
d2019320097=# explain analyze select * from table_btree where recordid > 250 and recordid < 550;
                                                        QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------
-----
 Index Scan using idx_btree on table_btree  (cost=0.43..16.00 rows=278 width=49) (actual time=0.044..0.133 rows=299 loop
s=1)
   Index Cond: ((recordid > 250) AND (recordid < 550))
 Planning Time: 1.927 ms
 Execution Time: 0.181 ms
(4개 행)
```

- SELECT * FROM table_hash WHERE recordid>250 AND recordid<550;

```
d2019320097=# explain analyze select * from table_hash where recordid > 250 and recordid < 550;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on table_hash  (cost=0.00..253128.00 rows=1 width=49) (actual time=0.121..2638.992 rows=299 loops=1)
   Filter: ((recordid > 250) AND (recordid < 550))
   Rows Removed by Filter: 9999701
 Planning Time: 0.255 ms
 Execution Time: 2649.441 ms
(5개 행)
```

bTree가 range 쿼리에서 더 짧은 실행시간을 보입니다.

3. Update a single “recordid” field in “table_btree”. And update a single “recordid” field in “table_noindex”. Then find a difference
   - Update “recordid” from 9,999,997 to 9,999,998

```
d2019320097=# explain analyze update table_btree set recordid = 9999998 where recordid = 9999997;
                                                          QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------
------
 Update on table_btree  (cost=0.43..8.45 rows=0 width=0) (actual time=0.225..0.226 rows=0 loops=1)
   ->  Index Scan using idx_btree on table_btree  (cost=0.43..8.45 rows=1 width=10) (actual time=0.112..0.114 rows=1 loo
ps=1)
         Index Cond: (recordid = 9999997)
 Planning Time: 12.365 ms
 Execution Time: 3.266 ms
(5개 행)
```

```
d2019320097=# explain analyze update table_noindex set recordid = 9999998 where recordid = 9999997;
                                                        QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------
-
 Update on table_noindex  (cost=0.00..228137.35 rows=0 width=0) (actual time=3009.099..3009.100 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..228137.35 rows=1 width=10) (actual time=3006.570..3007.435 rows=1 loops=1)
         Filter: (recordid = 9999997)
         Rows Removed by Filter: 9999999
 Planning Time: 1.704 ms
 Execution Time: 3009.175 ms
(6개 행)
```

btree 인덱스가 있는 경우, 1000배 가까이 빠른 속도로 단일 필드를 수정할 수 있습니다.

Update 2,000,000 “recordid” fields in “table_btree”. And update 2,000,000 “recordid” fields in “table_noindex”. Then find a difference

- Increase “recordid” fields by 100% whose value is greater than 8,000,000

```
d2019320097=# explain analyze update table_btree set recordid = recordid * 2 where recordid > 8000000;
                                                                QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------
--------------------------
 Update on table_btree  (cost=0.43..83502.17 rows=0 width=0) (actual time=22823.585..22823.587 rows=0 loops=1)
   ->  Index Scan using idx_btree on table_btree  (cost=0.43..83502.17 rows=2022587 width=10) (actual time=0.119..3119.6
13 rows=1999999 loops=1)
         Index Cond: (recordid > 8000000)
 Planning Time: 0.216 ms
 Execution Time: 22823.650 ms
(5개 행)
```

```
d2019320097=# explain analyze update table_noindex set recordid = recordid * 2 where recordid > 8000000;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
-------------
 Update on table_noindex  (cost=0.00..233109.85 rows=0 width=0) (actual time=15750.279..15750.281 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..233109.85 rows=1989001 width=10) (actual time=1814.041..3283.101 rows=1999
999 loops=1)
         Filter: (recordid > 8000000)
         Rows Removed by Filter: 8000001
 Planning Time: 0.112 ms
 Execution Time: 15750.322 ms
(6개 행)
```

다수의 레코드를 수정해야하는 경우, 인덱스도 그에 맞게 수정해야하기 때문에 인덱스가 있는 경우, 실행 시간이 오히려 길어집니다.

Update all “recordid” fields in “table_btree”. And update all “recordid” fields in “table_noindex”. Then find a difference

- Increase all “recordid” fields by 10%

```
d2019320097=# explain analyze update table_btree set recordid = recordid * 1.1;
                                                            QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
----------
 Update on table_btree  (cost=0.00..298294.45 rows=0 width=0) (actual time=179989.904..179989.905 rows=0 loops=1)
   ->  Seq Scan on table_btree  (cost=0.00..298294.45 rows=9976140 width=10) (actual time=0.049..61957.536 rows=10000000
 loops=1)
 Planning Time: 2.266 ms
 Execution Time: 179989.972 ms
(4개 행)
```

```
d2019320097=# explain analyze update table_noindex set recordid = recordid * 1.1;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
-------------
 Update on table_noindex  (cost=0.00..298726.93 rows=0 width=0) (actual time=54604.770..54604.773 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..298726.93 rows=10000853 width=10) (actual time=0.033..12388.764 rows=10000
000 loops=1)
 Planning Time: 0.880 ms
 Execution Time: 54604.833 ms
(4개 행)
```

마찬가지로, 모든 레코드를 수정하면서 그에 맞게 인덱스를 수정해야 하므로 인덱스가 있는 경우 실행 시간이 길어집니다.

4. Find all points within a rectangle ((1,1), (10,10)) on the tables “test0” and “test1”
   - Compare an index scan and seq scan
     - SET enable_indexscan=true;

```
d2019320097=# explain analyze select * from test0 where x between 1 and
10 and y between 1 and 10;
                                                          QUERY PLAN

------------------------------------------------------------------------------
-----------------------------------------------------
 Index Scan using test_idx_y on test0  (cost=0.42..26427.01 rows=1323 wi
dth=20) (actual time=0.041..21.397 rows=1270 loops=1)
   Index Cond: ((y >= '1'::double precision) AND (y <= '10'::double prec
ision))
   Filter: ((x >= '1'::double precision) AND (x <= '10'::double precisio
n))
   Rows Removed by Filter: 24116
 Planning Time: 0.167 ms
 Execution Time: 21.489 ms
(6개 행)
```

```
d2019320097=# explain analyze select * from test1 where p <@ box'((1,1),
 (10, 10))';
                                                           QUERY PLAN

------------------------------------------------------------------------------
-------------------------------------------------------
 Index Scan using test_rtree_idx on test1  (cost=0.29..3765.78 rows=1000
 width=20) (actual time=0.198..14.965 rows=1219 loops=1)
   Index Cond: (p <@ '(10,10),(1,1)'::box)
 Planning Time: 0.305 ms
 Execution Time: 15.089 ms
(4개 행)
```

- SET enable_indexscan=false;

```
d2019320097=# set enable_indexscan = false;
SET
d2019320097=# explain analyze select * from test0 where x between 1 and
10 and y between 1 and 10;
                                                                QUERY
PLAN
------------------------------------------------------------------------------
-------------------------------------------------------------------------
 Seq Scan on test0  (cost=0.00..26370.00 rows=1323 width=20) (actual tim
e=0.020..87.177 rows=1270 loops=1)
   Filter: ((x >= '1'::double precision) AND (x <= '10'::double precisio
n) AND (y >= '1'::double precision) AND (y <= '10'::double precision))
   Rows Removed by Filter: 998730
 Planning Time: 0.115 ms
 Execution Time: 87.251 ms
(5개 행)
```

```
d2019320097=# explain analyze select * from test1 where p <@ box'((1,1),
 (10, 10))';
                                                    QUERY PLAN

------------------------------------------------------------------------------
-------------------------------------
 Seq Scan on test1  (cost=0.00..18870.00 rows=1000 width=20) (actual tim
e=0.016..102.683 rows=1219 loops=1)
   Filter: (p <@ '(10,10),(1,1)'::box)
   Rows Removed by Filter: 998781
 Planning Time: 0.079 ms
 Execution Time: 102.782 ms
(5개 행)
```

인덱스 스캔 방식이 순차적 스캔 방식에 비해 빠름을 볼 수 있습니다.

Find all boxes overlapped with rectangles ((0,0), (1,1)) and ((9,9), (10,10)) at the same time on the table “test2”

- Compare an index scan and seq scan
  - SET enable_indexscan=true;

```
d2019320097=# set enable_indexscan = true;
SET
d2019320097=# explain analyze select * from test2 where testbox && box '
((0,0), (1,1))' and testbox && box '((9,9), (10,10))';
                                                          QUERY PLAN

------------------------------------------------------------------------------
-----------------------------------------------------
 Index Scan using test_box_idx on test2  (cost=0.41..104.91 rows=25 widt
h=36) (actual time=0.085..3.720 rows=1361 loops=1)
   Index Cond: ((testbox && '(1,1),(0,0)'::box) AND (testbox && '(10,10)
,(9,9)'::box))
 Planning Time: 0.076 ms
 Execution Time: 3.772 ms
(4개 행)
```

  - SET enable_indexscan=false;

```
d2019320097=# set enable_indexscan = false;
SET
d2019320097=# explain analyze select * from test2 where testbox && box '
((0,0), (1,1))' and testbox && box '((9,9), (10,10))';
                                                    QUERY PLAN

------------------------------------------------------------------------------
---------------------------------
 Seq Scan on test2  (cost=0.00..23334.00 rows=25 width=36) (actual time=
0.216..140.631 rows=1361 loops=1)
   Filter: ((testbox && '(1,1),(0,0)'::box) AND (testbox && '(10,10),(9,
9)'::box))
   Rows Removed by Filter: 998639
 Planning Time: 0.088 ms
 Execution Time: 140.735 ms
(5개 행)
```

마찬가지로 인덱스 스캔이 훨씬 빠른 실행 시간을 가집니다.

Find 10 nearest points to (0,0) on the tables “test0” and “test1”

- Compare an index scan and seq scan
  - SET enable_indexscan=true;

```
d2019320097=# set enable_indexscan = true;
SET
d2019320097=# explain analyze select * from test0 order by point(x, y) <
-> point (0, 0) asc limit 10;
                                                          QUERY PLAN

------------------------------------------------------------------------------
-----------------------------------------------------
 Limit  (cost=42979.64..42979.67 rows=10 width=28) (actual time=374.245.
.374.249 rows=10 loops=1)
   ->  Sort  (cost=42979.64..45479.64 rows=1000000 width=28) (actual tim
e=374.241..374.243 rows=10 loops=1)
         Sort Key: ((point(x, y) <-> '(0,0)'::point))
         Sort Method: top-N heapsort  Memory: 26kB
         ->  Seq Scan on test0  (cost=0.00..21370.00 rows=1000000 width=
28) (actual time=0.017..203.045 rows=1000000 loops=1)
 Planning Time: 0.191 ms
 Execution Time: 374.289 ms
(7개 행)
```

```
d2019320097=# explain analyze select * from test1 order by p <-> point (
0, 0) asc limit 10;
                                                                QUERY PLA
N
------------------------------------------------------------------------------
-----------------------------------------------------------------
 Limit  (cost=0.29..1.07 rows=10 width=28) (actual time=0.308..0.365 row
s=10 loops=1)
   ->  Index Scan using test_rtree_idx on test1  (cost=0.29..78132.29 ro
ws=1000000 width=28) (actual time=0.306..0.363 rows=10 loops=1)
         Order By: (p <-> '(0,0)'::point)
 Planning Time: 0.137 ms
 Execution Time: 0.388 ms
(5개 행)
```

  - SET enable_indexscan=false;

```
d2019320097=# set enable_indexscan = false;
SET
d2019320097=# explain analyze select * from test0 order by point(x, y) <
-> point (0, 0) asc limit 10;
                                                              QUERY PLAN

--------------------------------------------------------------------------
------------------------------------------------------
 Limit  (cost=42979.64..42979.67 rows=10 width=28) (actual time=394.706.
.394.709 rows=10 loops=1)
   ->  Sort  (cost=42979.64..45479.64 rows=1000000 width=28) (actual tim
e=394.704..394.706 rows=10 loops=1)
         Sort Key: ((point(x, y) <-> '(0,0)'::point))
         Sort Method: top-N heapsort  Memory: 26kB
         ->  Seq Scan on test0  (cost=0.00..21370.00 rows=1000000 width=
28) (actual time=0.015..273.895 rows=1000000 loops=1)
 Planning Time: 0.099 ms
 Execution Time: 394.735 ms
(7개 행)
```

```
d2019320097=# explain analyze select * from test1 order by p <-> point (
0, 0) asc limit 10;
                                                              QUERY PLAN

--------------------------------------------------------------------------
------------------------------------------------------
 Limit  (cost=40479.64..40479.67 rows=10 width=28) (actual time=409.719.
.409.723 rows=10 loops=1)
   ->  Sort  (cost=40479.64..42979.64 rows=1000000 width=28) (actual tim
e=409.716..409.718 rows=10 loops=1)
         Sort Key: ((p <-> '(0,0)'::point))
         Sort Method: top-N heapsort  Memory: 26kB
         ->  Seq Scan on test1  (cost=0.00..18870.00 rows=1000000 width=
28) (actual time=0.013..275.216 rows=1000000 loops=1)
 Planning Time: 0.071 ms
 Execution Time: 409.745 ms
(7개 행)
```

test0 의 경우, x와 y 모두에 대한 인덱스가 없어 순차 스캔을 실행시킬 수 없으나, test1에서 볼 수 있듯 x, y 모두에 대한 인덱스가 존재할 경우 인덱스 스캔이 훨씬 빠름을 알 수 있습니다.

공간 인덱스의 경우에도 특정 조건을 만족하는 점이나 거리를 검색할 때 인덱스를 사용하는 것이 훨씬 빠름을 알 수 있습니다.