KOREA UNIVERSITY
DATAX LAB

Chapter 14 – Lab Solution

Indexing 2

Exercise 1 Answer

- Index creation
  - CREATE INDEX btree on table_btree using btree(recordid);
  - CREATE INDEX hash on table_hash using hash(recordid);

```
postgres=# \h CREATE INDEX
명령: CREATE INDEX
설명: 새 인덱스 만들기
문법:
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] 이름 ] ON [ ONLY ] 테이블이름 [ USING 색인방법 ]
    ( { 칼럼이름 | ( 표현식 ) } [ COLLATE collation ] [ 연산자클래스 [ ( opclass_매개변수 = 값 [, ... ] ) ] ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( 칼럼이름 [, ...] ) ]
    [ WITH ( 스토리지_매개변수 [= 값] [, ... ] ) ]
    [ TABLESPACE 테이블스페이스이름 ]
    [ WHERE 범위한정구문 ]
URL: https://www.postgresql.org/docs/14/sql-createindex.html
```

```
postgres=# CREATE INDEX btree on table_btree using btree(recordid);
CREATE INDEX
postgres=# CREATE INDEX hash on table_hash using hash(recordid);
CREATE INDEX
```

Exercise 2.a Answer

- On equal query: hash index faster than b-tree

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table_btree WHERE recordid=10001;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using btree on table_btree  (cost=0.43..162395.43 rows=50000 width=172) (actual time=0.159..0.162 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 9.508 ms
 Execution Time: 0.532 ms
(4개 행)


postgres=# EXPLAIN ANALYZE SELECT * FROM table_hash WHERE recordid=10001;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using hash on table_hash  (cost=0.00..162667.00 rows=50000 width=172) (actual time=0.049..0.049 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 1.094 ms
 Execution Time: 0.059 ms
(4개 행)
```

Exercise 2.b Answer

- On range query: hash index does not work

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table_btree WHERE recordid>250 AND recordid<550;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using btree on table_btree  (cost=0.43..162520.43 rows=50000 width=172) (actual time=0.105..0.352 rows=299 loops=1)
   Index Cond: ((recordid > 250) AND (recordid < 550))
 Planning Time: 0.226 ms
 Execution Time: 0.414 ms
(4개 행)


postgres=# EXPLAIN ANALYZE SELECT * FROM table_hash WHERE recordid>250 AND recordid<550;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table_hash  (cost=0.00..253093.00 rows=50000 width=172) (actual time=0.058..838.074 rows=299 loops=1)
   Filter: ((recordid > 250) AND (recordid < 550))
   Rows Removed by Filter: 9999701
 Planning Time: 1.100 ms
 Execution Time: 838.094 ms
(5개 행)
```

Exercise 3.a Answer

- Update a single record
  - UPDATE table_btree SET recordid=recordid+1 WHERE recordid=9999997;
  - UPDATE table_noindex SET recordid=recordid+1 WHERE recordid=9999997;

```
postgres=# EXPLAIN ANALYZE UPDATE table_btree SET recordid=recordid+1 WHERE recordid=9999997;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.43..8.46 rows=0 width=0) (actual time=1.866..1.867 rows=0 loops=1)
   ->  Index Scan using btree on table_btree  (cost=0.43..8.46 rows=1 width=10) (actual time=0.182..0.185 rows=1 loops=1)
         Index Cond: (recordid = 9999997)
 Planning Time: 3.425 ms
 Execution Time: 6.306 ms
(5개 행)


postgres=# EXPLAIN ANALYZE UPDATE table_noindex SET recordid=recordid+1 WHERE recordid=9999997;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..228092.36 rows=0 width=0) (actual time=721.556..721.557 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..228092.36 rows=1 width=10) (actual time=721.112..721.113 rows=1 loops=1)
         Filter: (recordid = 9999997)
         Rows Removed by Filter: 9999999
 Planning Time: 1.325 ms
 Execution Time: 721.589 ms
(6개 행)
```

Exercise 3.b Answer

- Update 2,000,000 records
  - UPDATE table_btree SET recordid=recordid*2 WHERE recordid>8000000;
  - UPDATE table_noindex SET recordid=recordid*2 WHERE recordid>8000000;

```
postgres=# EXPLAIN ANALYZE UPDATE table_btree SET recordid=recordid*2 WHERE recordid>8000000;
                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.43..82611.84 rows=0 width=0) (actual time=13753.933..13753.934 rows=0 loops=1)
   -> Index Scan using btree on table_btree  (cost=0.43..82611.84 rows=2001220 width=10) (actual time=0.177..960.205 rows=1999999 loops=1)
         Index Cond: (recordid > 8000000)
 Planning Time: 0.258 ms
 Execution Time: 13753.975 ms
(5개 행)


postgres=# EXPLAIN ANALYZE UPDATE table_noindex SET recordid=recordid*2 WHERE recordid>8000000;
                                                                                 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..233083.79 rows=0 width=0) (actual time=11648.289..11648.290 rows=0 loops=1)
   -> Seq Scan on table_noindex  (cost=0.00..233083.79 rows=1996570 width=10) (actual time=576.401..1037.098 rows=1999999 loops=1)
         Filter: (recordid > 8000000)
         Rows Removed by Filter: 8000001
 Planning Time: 0.041 ms
 Execution Time: 11648.304 ms
(6개 행)
```

Exercise 3.c Answer

- Update all records
  - UPDATE table_btree SET recordid=recordid*1.1;
  - UPDATE table_noindex SET recordid=recordid*1.1;

```
postgres=# EXPLAIN ANALYZE UPDATE table_btree SET recordid=recordid*1.1;
                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.00..299309.45 rows=0 width=0) (actual time=90125.784..90125.785 rows=0 loops=1)
   ->  Seq Scan on table_btree  (cost=0.00..299309.45 rows=10034140 width=10) (actual time=0.030..4808.904 rows=10000000 loops=1)
 Planning Time: 0.953 ms
 Execution Time: 90125.816 ms
(4개 행)


postgres=# EXPLAIN ANALYZE UPDATE table_noindex SET recordid=recordid*1.1;
                                                QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..275068.19 rows=0 width=0) (actual time=38234.376..38234.377 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..275068.19 rows=8648925 width=10) (actual time=0.022..4381.879 rows=10000000 loops=1)
 Planning Time: 0.704 ms
 Execution Time: 38234.840 ms
(4개 행)
```

Exercise 4.a Answer

- SET enable_indexscan=true;
  - SELECT * FROM test0 WHERE 1<x AND x<10 AND 1<y AND y<10;
  - SELECT * FROM test1 WHERE p <@ box'((1,1),(10,10))';

```
postgres=# SET enable_indexscan=true;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM test0 WHERE 1<x AND x<10 AND 1<y AND y<10;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on test0  (cost=0.00..26370.00 rows=1267 width=20) (actual time=0.179..85.727 rows=1252 loops=1)
   Filter: (('1'::double precision < x) AND (x < '10'::double precision) AND ('1'::double precision < y) AND (y < '10'::double precision))
   Rows Removed by Filter: 998748
 Planning Time: 0.094 ms
 Execution Time: 85.786 ms
(5개 행)


postgres=# 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..3757.78 rows=1000 width=20) (actual time=0.025..0.724 rows=1230 loops=1)
   Index Cond: (p <@ '(10,10),(1,1)'::box)
 Planning Time: 0.044 ms
 Execution Time: 0.761 ms
(4개 행)
```

Exercise 4.a Answer

- SET enable_indexscan=false;
  - SELECT * FROM test0 WHERE 1<x AND x<10 AND 1<y AND y<10;
  - SELECT * FROM test1 WHERE p <@ box'((1,1),(10,10))';

```
postgres=# SET enable_indexscan=false;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM test0 WHERE 1<x AND x<10 AND 1<y AND y<10;
                                                             QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on test0  (cost=0.00..26370.00 rows=1267 width=20) (actual time=0.186..81.080 rows=1252 loops=1)
   Filter: (('1'::double precision < x) AND (x < '10'::double precision) AND ('1'::double precision < y) AND (y < '10'::double precision))
   Rows Removed by Filter: 998748
 Planning Time: 0.093 ms
 Execution Time: 81.135 ms
(5개 행)


postgres=# 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 time=0.053..69.421 rows=1230 loops=1)
   Filter: (p <@ '(10,10),(1,1)'::box)
   Rows Removed by Filter: 998770
 Planning Time: 0.043 ms
 Execution Time: 69.473 ms
(5개 행)
```

Exercise 4.b Answer

- SET enable_indexscan=true;
  - SELECT * FROM test2 WHERE testbox && box'((0,0),(1,1))' AND testbox && box'((9,9),(10,10))';

```
postgres=# SET enable_indexscan=true;
SET
postgres=# 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 width=36) (actual time=0.147..15.367 rows=1433 loops=1)
   Index Cond: ((testbox && '(1,1),(0,0)'::box) AND (testbox && '(10,10),(9,9)'::box))
 Planning Time: 1.150 ms
 Execution Time: 15.441 ms
(4개 행)
```

Exercise 4.b Answer

- SET enable_indexscan=false;
  - SELECT * FROM test2 WHERE testbox && box'((0,0),(1,1))' AND testbox && box'((9,9),(10,10))';

```
postgres=# SET enable_indexscan=false;
SET
postgres=# 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=0.077..82.698 rows=1433 loops=1)
   Filter: ((testbox && '(1,1),(0,0)'::box) AND (testbox && '(10,10),(9,9)'::box))
   Rows Removed by Filter: 998567
 Planning Time: 0.047 ms
 Execution Time: 82.767 ms
(5개 행)
```

Exercise 4.c Answer

- SET enable_indexscan=true;
  - SELECT * FROM test0 ORDER BY (x*x + y*y) asc limit 10;
  - SELECT * FROM test1 ORDER BY p <-> point'(0,0)' asc limit 10;

```
postgres=# SET enable_indexscan=true;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM test0 ORDER BY (x*x + y*y) asc limit 10;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=45479.64..45479.67 rows=10 width=28) (actual time=190.166..190.167 rows=10 loops=1)
   ->  Sort  (cost=45479.64..47979.64 rows=1000000 width=28) (actual time=190.164..190.164 rows=10 loops=1)
         Sort Key: (((x * x) + (y * y)))
         Sort Method: top-N heapsort  Memory: 26kB
         ->  Seq Scan on test0  (cost=0.00..23870.00 rows=1000000 width=28) (actual time=0.021..114.797 rows=1000000 loops=1)
 Planning Time: 0.088 ms
 Execution Time: 190.183 ms
(7개 행)


postgres=# EXPLAIN ANALYZE SELECT * FROM test1 ORDER BY p <-> point'(0,0)' asc limit 10;
                                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.29..0.98 rows=10 width=28) (actual time=0.220..0.303 rows=10 loops=1)
   ->  Index Scan using test_rtree_idx on test1  (cost=0.29..69732.29 rows=1000000 width=28) (actual time=0.219..0.301 rows=10 loops=1)
         Order By: (p <-> '(0,0)'::point)
 Planning Time: 0.083 ms
 Execution Time: 0.327 ms
(5개 행)
```

Exercise 4.c Answer

- SET enable_indexscan=false;
  - SELECT * FROM test0 ORDER BY (x*x + y*y) asc limit 10;
  - SELECT * FROM test1 ORDER BY p <-> point'(0,0)' asc limit 10;

```
postgres=# SET enable_indexscan=false;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM test0 ORDER BY (x*x + y*y) asc limit 10;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=45479.64..45479.67 rows=10 width=28) (actual time=188.666..188.667 rows=10 loops=1)
   ->  Sort  (cost=45479.64..47979.64 rows=1000000 width=28) (actual time=188.665..188.666 rows=10 loops=1)
         Sort Key: (((x * x) + (y * y)))
         Sort Method: top-N heapsort  Memory: 26kB
         ->  Seq Scan on test0  (cost=0.00..23870.00 rows=1000000 width=28) (actual time=0.023..113.309 rows=1000000 loops=1)
 Planning Time: 0.057 ms
 Execution Time: 188.682 ms
(7개 행)


postgres=# 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=193.870..193.871 rows=10 loops=1)
   ->  Sort  (cost=40479.64..42979.64 rows=1000000 width=28) (actual time=193.869..193.870 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=0.025..118.377 rows=1000000 loops=1)
 Planning Time: 0.051 ms
 Execution Time: 193.885 ms
(7개 행)
```