2015131406 박가은

Q1) "Sorted"는 clustering index이고 "unsorted"는 non-clustering index이다.

```
postgres=# CREATE INDEX index_Sorted ON table1(Sorted);
CREATE INDEX
postgres=# SELECT * FROM pg_indexes WHERE tablename = 'table1';
 schemaname | tablename |  indexname   | tablespace |                              indexdef
------------+-----------+--------------+------------+---------------------------------------------------------------------
 public     | table1    | index_sorted |            | CREATE INDEX index_sorted ON public.table1 USING btree (sorted)
(1개 행)
```

Q2)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE sorted < 3;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using sorted on table1  (cost=0.43..8.68 rows=14 width=53) (actual time=0.007..0.011 rows=15 loops=1)
   Index Cond: (sorted < 3)
 Planning Time: 0.174 ms
 Execution Time: 0.032 ms
(4개 행)


postgres=# EXPLAIN ANALYZE SELECT unsorted FROM table1;
                                                      QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..203093.00 rows=10000000 width=4) (actual time=0.032..1139.704 rows=10000000 loops=1)
 Planning Time: 0.091 ms
 Execution Time: 1318.773 ms
(3개 행)
```

```
postgres=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF) SELECT sorted FROM table1 WHERE sorted < 100000;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Only Scan using index_sorted on table1 (actual time=0.592..136.249 rows=500000 loops=1)
   Index Cond: (sorted < 100000)
   Heap Fetches: 500000
   Buffers: shared hit=1 read=6523
 Planning Time: 0.107 ms
 Execution Time: 147.500 ms
(6개 행)
```

Q3)

```
postgres=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE sorted > 100 and sorted < 500 AND rndm = 55;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using index_multiple on table1  (cost=0.43..3384.24 rows=1 width=8) (actual time=1.006..1.006 rows=0 loops=1)
   Index Cond: ((sorted > 100) AND (sorted < 500))
   Filter: (rndm = 55)
   Rows Removed by Filter: 1995
 Planning Time: 1.211 ms
 Execution Time: 1.030 ms
(6개 행)


postgres=# EXPLAIN ANALYZE SELECT unsorted, rndm FROM table1 WHERE unsorted > 100 AND unsorted < 500 AND rndm = 55;
                                                      QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using index_multiple on table1  (cost=0.43..212291.89 rows=1 width=8) (actual time=427.686..427.686 rows=0 loops=1)
   Index Cond: ((unsorted > 100) AND (unsorted < 500))
   Filter: (rndm = 55)
   Rows Removed by Filter: 2021
 Planning Time: 0.139 ms
 Execution Time: 427.713 ms
(6개 행)
```

In case of SELECT sorted, the order of data accords with the order of index so that actual time takes shorter than the case of SELECT unsorted. On the other hand, in case of SELECT unsorted, the order of data does not accord with the order of index file so that actual time takes longer than the other.

Q4)

```
postgres=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE sorted > 1999231 AND rndm = 1005;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using index_multiple on table1  (cost=0.43..6463.79 rows=1 width=8) (actual time=1.504..1.504 rows=0 loops=1)
   Index Cond: (sorted > 1999231)
   Filter: (rndm = 1005)
   Rows Removed by Filter: 3840
 Planning Time: 0.267 ms
 Execution Time: 1.523 ms
(6개 행)


postgres=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE sorted < 1999231 AND rndm = 1005;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..253093.00 rows=100 width=8) (actual time=21.174..1225.465 rows=114 loops=1)
   Filter: ((sorted < 1999231) AND (rndm = 1005))
   Rows Removed by Filter: 9999886
 Planning Time: 0.116 ms
 Execution Time: 1225.499 ms
(5개 행)
```

The limit of number of sorted is 2000000. In case of WHERE sorted > 1999231, there only exists approximately 800(2000000 – 1999231) case that satisfied sorted > 1999231. On the other hand, there exists 1999232(including 0) possibility in case of WHERE sorted < 1999231. So second query has longer execution time.

Q5)

```
postgres=# SELECT * FROM pg_indexes WHERE tablename = 'table_hash';
 schemaname | tablename  | indexname | tablespace |                          indexdef
------------+------------+-----------+------------+-------------------------------------------------------------
 public     | table_hash | hash      |            | CREATE INDEX hash ON public.table_hash USING hash (recordid)
(1개 행)


postgres=# SELECT * FROM pg_indexes WHERE tablename = 'table_btree';
 schemaname |  tablename  | indexname | tablespace |                           indexdef
------------+-------------+-----------+------------+---------------------------------------------------------------
 public     | table_btree | btree     |            | CREATE INDEX btree ON public.table_btree USING btree (recordid)
(1개 행)


postgres=#
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table_btree WHERE recordid = 10001;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Index Scan using btree on table_btree  (cost=0.43..8.45 rows=1 width=49) (actual time=0.036..0.037 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 0.107 ms
 Execution Time: 0.320 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..8.02 rows=1 width=49) (actual time=7.620..7.623 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 1.160 ms
 Execution Time: 7.649 ms
(4개 행)
```

B_tree를 사용하는 table_Btree의 경우, recordid라는 인덱스가 직접적으로 레코드를 가리키기 때문에 실행 시간이 짧게 걸린다. 하지만 hash를 사용하는 table_hash의 경우, hash function을 사용하여 key값을 변환시키고 이를 레코드와 mapping하는 과정이 있기 때문에 실행 시간이 더 오래 걸린다.

Q6)

```
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..17.50 rows=303 width=49) (actual time=0.253..1.440 rows=299 loops=1)   Index Cond: ((recordid > 250) AND (recordid < 550))
 Planning Time: 8.274 ms
 Execution Time: 1.488 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=1 width=49) (actual time=0.116..1156.415 rows=299 loops=1)
   Filter: ((recordid > 250) AND (recordid < 550))
   Rows Removed by Filter: 9999701
 Planning Time: 0.426 ms
 Execution Time: 1156.446 ms
(5개 행)
```

Btree의 경우 analyze를 보면 index scan을 사용한 것을 확인할 수 있다. 레코드를 모두 읽을 필요 없이 상대적으로 크기가 작은 인덱스 파일만으로 조건에 부합하는 레코드를 뽑을 수 있다. Hash의 경우 analyze를 보면 seq scan을 사용하였다. 즉, 모든 레코드를 읽으며 조건에 부합한 레코드를 추출해야 하기 때문에 시간이 상대적으로 오래 걸린다.

Q7)

```
postgres=# EXPLAIN ANALYZE INSERT INTO table_noindex VALUES (1, 1, 'abc');
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Insert on table_noindex  (cost=0.00..0.01 rows=1 width=172) (actual time=0.085..0.085 rows=0 loops=1)
   ->  Result  (cost=0.00..0.01 rows=1 width=172) (actual time=0.001..0.001 rows=1 loops=1)
 Planning Time: 0.042 ms
 Execution Time: 0.149 ms
(4개 행)


postgres=# EXPLAIN ANALYZE INSERT INTO table_btree VALUES (1, 1, 'abc');
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Insert on table_btree  (cost=0.00..0.01 rows=1 width=172) (actual time=0.042..0.042 rows=0 loops=1)
   ->  Result  (cost=0.00..0.01 rows=1 width=172) (actual time=0.001..0.001 rows=1 loops=1)
 Planning Time: 0.041 ms
 Execution Time: 0.066 ms
(4개 행)
```

실행 시간에 약간의 차이가 있다.

Q8)

```
postgres=# UPDATE table_noindex SET recordid = recordid + 1 WHERE recordid < 2000000;
UPDATE 2000002
postgres=# EXPLAIN ANALYZE UPDATE table_noindex SET recordid = recordid - 1 WHERE recordid < 2000000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..253330.47 rows=1925123 width=55) (actual time=7445.937..7445.937 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..253330.47 rows=1925123 width=55) (actual time=1305.735..2029.166 rows=2000001 loops=1)
         Filter: (recordid < 2000000)
         Rows Removed by Filter: 8000001
 Planning Time: 0.593 ms
 Execution Time: 7445.982 ms
(6개 행)


postgres=# EXPLAIN ANALYZE UPDATE table_btree SET recordid = recordid + 1 WHERE recordid < 2000000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.43..83347.05 rows=2019031 width=55) (actual time=10120.600..10120.600 rows=0 loops=1)
   ->  Index Scan using btree on table_btree  (cost=0.43..83347.05 rows=2019031 width=55) (actual time=0.111..735.519 rows=2000002 loops=1)
         Index Cond: (recordid < 2000000)
 Planning Time: 0.263 ms
 Execution Time: 10121.284 ms
(5개 행)
```

같은 연산을 수행했으나 table_Btree에서의 실행 시간이 아주 크게 나왔다. 직접적으로 레코드에 접근하여 값을 변경할 수 있는 table_noindex와는 달리 btree는 인덱스를 한 번 거쳐서 레코드에 접근해야 하기 때문에 시간이 오래 걸린 듯 하다. 또한 업데이트한 recordid를 다시 인덱스 엔트리에도 똑같이 적용해주어야 하기 때문에 실행 시간이 거의 두 배가 차이난다.

Q9)

8번과 같은 이유로 실행 시간이 거의 두 배가 차이나고 있다.

```
postgres=# EXPLAIN ANALYZE UPDATE table_btree SET recordid = recordid - 1;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.00..248712.03 rows=10000002 width=55) (actual time=68928.646..68928.646 rows=0 loops=1)
   ->  Seq Scan on table_btree  (cost=0.00..248712.03 rows=10000002 width=55) (actual time=118.034..3384.045 rows=10000002 loops=1)
 Planning Time: 1.039 ms
 Execution Time: 68928.702 ms
(4개 행)


postgres=# EXPLAIN ANALYZE UPDATE table_noindex SET recordid = recordid + 1;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..268792.67 rows=9957014 width=55) (actual time=38173.287..38173.287 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..268792.67 rows=9957014 width=55) (actual time=123.298..7884.365 rows=10000002 loops=1)
 Planning Time: 3.183 ms
 Execution Time: 38173.367 ms
(4개 행)
```