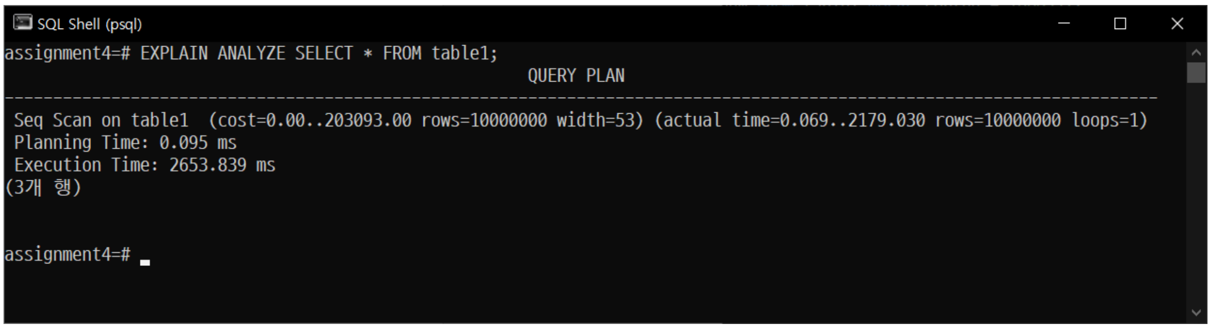
Database HW5 : Practice 4

2018320205 신대성

Index Creation and Scan

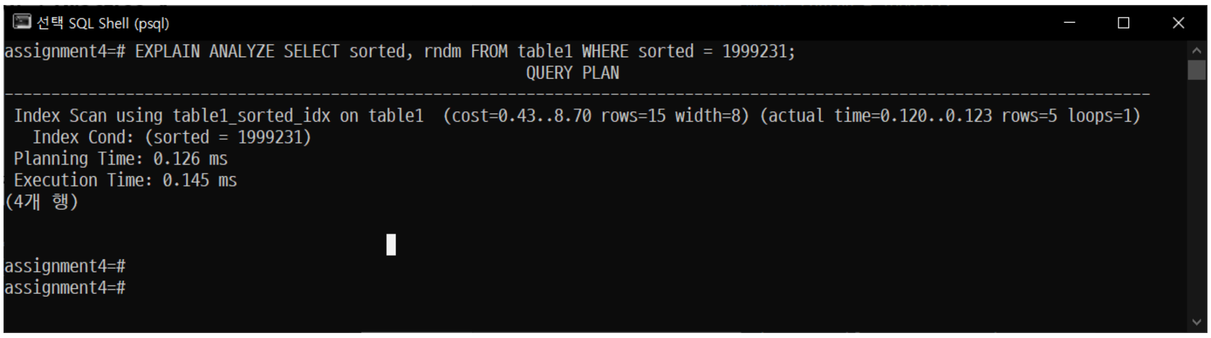
1. Make (and execute) three queries each of which uses seq scan, index scan, and index only scan respectively. (use the 'explain analyze')

- Seq Scan



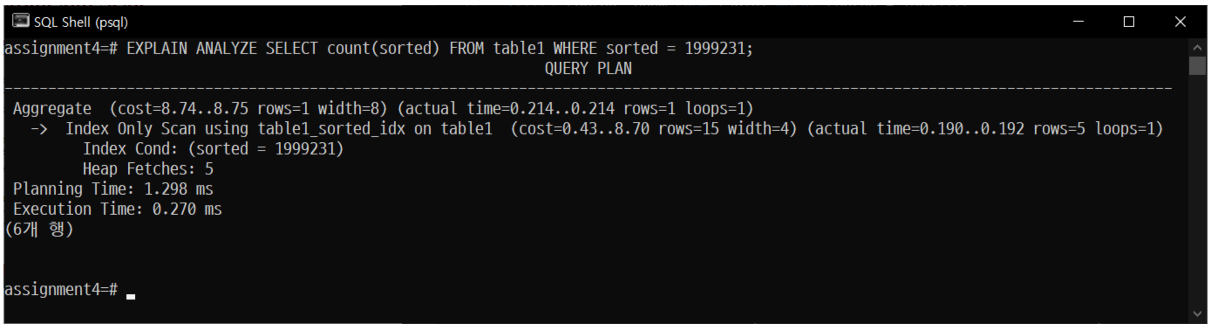
```
assignment4=# EXPLAIN ANALYZE SELECT * FROM table1;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..203093.00 rows=10000000 width=53) (actual time=0.069..2179.030 rows=10000000 loops=1)
 Planning Time: 0.095 ms
 Execution Time: 2653.839 ms
(3개 행)


assignment4=#
```

- Index Scan



```
assignment4=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE sorted = 1999231;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Index Scan using table1_sorted_idx on table1  (cost=0.43..8.70 rows=15 width=8) (actual time=0.120..0.123 rows=5 loops=1)
   Index Cond: (sorted = 1999231)
 Planning Time: 0.126 ms
 Execution Time: 0.145 ms
(4개 행)


assignment4=#
assignment4=#
```

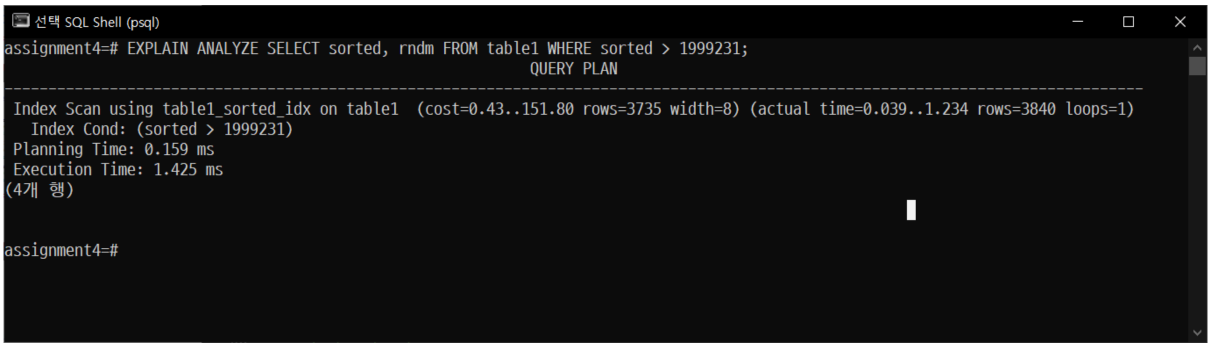
- Index only Scan



```
assignment4=# EXPLAIN ANALYZE SELECT count(sorted) FROM table1 WHERE sorted = 1999231;
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=8.74..8.75 rows=1 width=8) (actual time=0.214..0.214 rows=1 loops=1)
   ->  Index Only Scan using table1_sorted_idx on table1  (cost=0.43..8.70 rows=15 width=4) (actual time=0.190..0.192 rows=5 loops=1)
         Index Cond: (sorted = 1999231)
         Heap Fetches: 5
 Planning Time: 1.298 ms
 Execution Time: 0.270 ms
(6개 행)


assignment4=#
```

2. Make two queries using clustered index and non-clustered index. Compare their execution times.

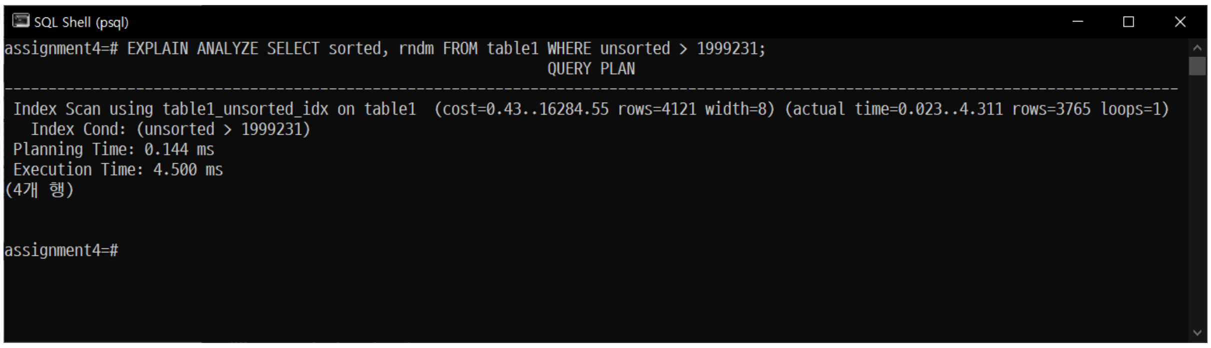
- Clustered Index



선택 SQL Shell (psql)

```
assignment4=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE sorted > 1999231;
                                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------
 Index Scan using table1_sorted_idx on table1  (cost=0.43..151.80 rows=3735 width=8) (actual time=0.039..1.234 rows=3840 loops=1)
   Index Cond: (sorted > 1999231)
 Planning Time: 0.159 ms
 Execution Time: 1.425 ms
(4개 행)


assignment4=#
```

- Non-clustered Index



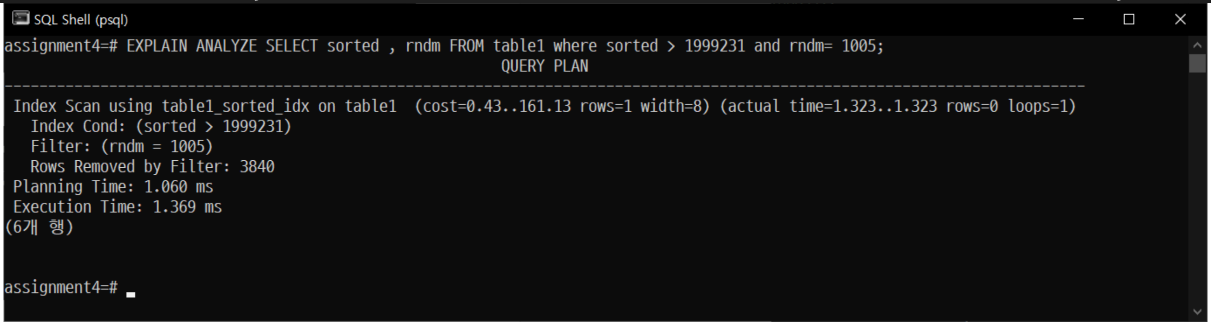
SQL Shell (psql)

```
assignment4=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 WHERE unsorted > 1999231;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using table1_unsorted_idx on table1  (cost=0.43..16284.55 rows=4121 width=8) (actual time=0.023..4.311 rows=3765 loops=1)
   Index Cond: (unsorted > 1999231)
 Planning Time: 0.144 ms
 Execution Time: 4.500 ms
(4개 행)


assignment4=#
```

연속된 key의 row를 가져올 때, clustered index가 non-clustered index보다 빠르다. 이는 데이터의 정렬이 sorted를 기준으로 되어있어 disk access에 시간 차이가 있기 때문이다.

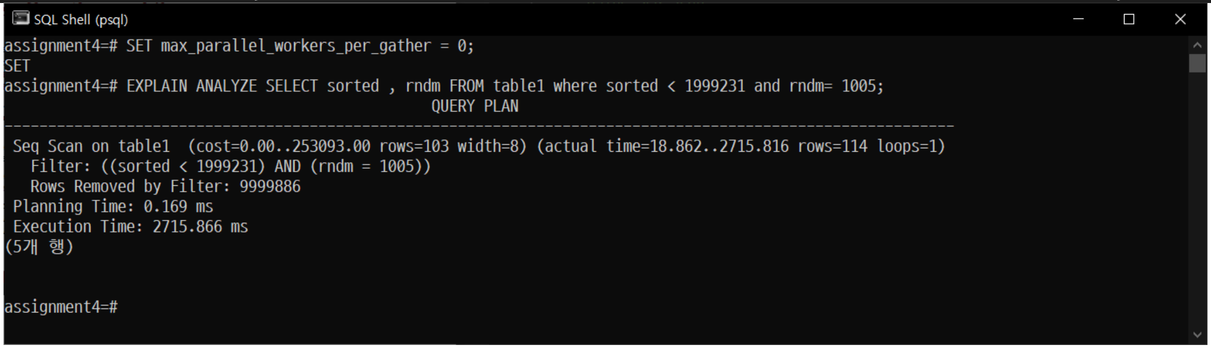
3. Execute and compare the following two queries. Explain why their query plans are different.

```
● SELECT sorted, rndm FROM table1 where sorted > 1999231 and rndm= 1005;
```



```
assignment4=# EXPLAIN ANALYZE SELECT sorted , rndm FROM table1 where sorted > 1999231 and rndm= 1005;
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Index Scan using table1_sorted_idx on table1  (cost=0.43..161.13 rows=1 width=8) (actual time=1.323..1.323 rows=0 loops=1)
   Index Cond: (sorted > 1999231)
   Filter: (rndm = 1005)
   Rows Removed by Filter: 3840
 Planning Time: 1.060 ms
 Execution Time: 1.369 ms
(6개 행)


assignment4=#
```

```
● SELECT sorted, rndm FROM table1 where sorted < 1999231 and rndm= 1005;
```

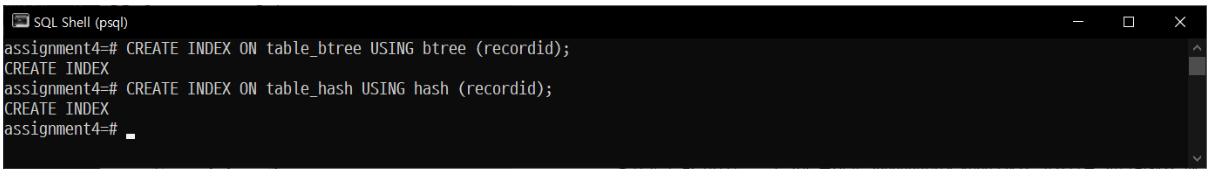


```
assignment4=# SET max_parallel_workers_per_gather = 0;
SET
assignment4=# EXPLAIN ANALYZE SELECT sorted , rndm FROM table1 where sorted < 1999231 and rndm= 1005;
                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..253093.00 rows=103 width=8) (actual time=18.862..2715.816 rows=114 loops=1)
   Filter: ((sorted < 1999231) AND (rndm = 1005))
   Rows Removed by Filter: 9999886
 Planning Time: 0.169 ms
 Execution Time: 2715.866 ms
(5개 행)


assignment4=#
```

후자가 느린 이유는 sorted의 범위가 2000000인데 1999231을 기준으로 필터링했을 때 이보다 큰 row는 얼마 없지만, 이보다 작은 row는 매우 많기 때문에 그 다음 조건인 rndm = 1005를 확인할 때 더 많은 시간이 걸리게 되는 것이다.

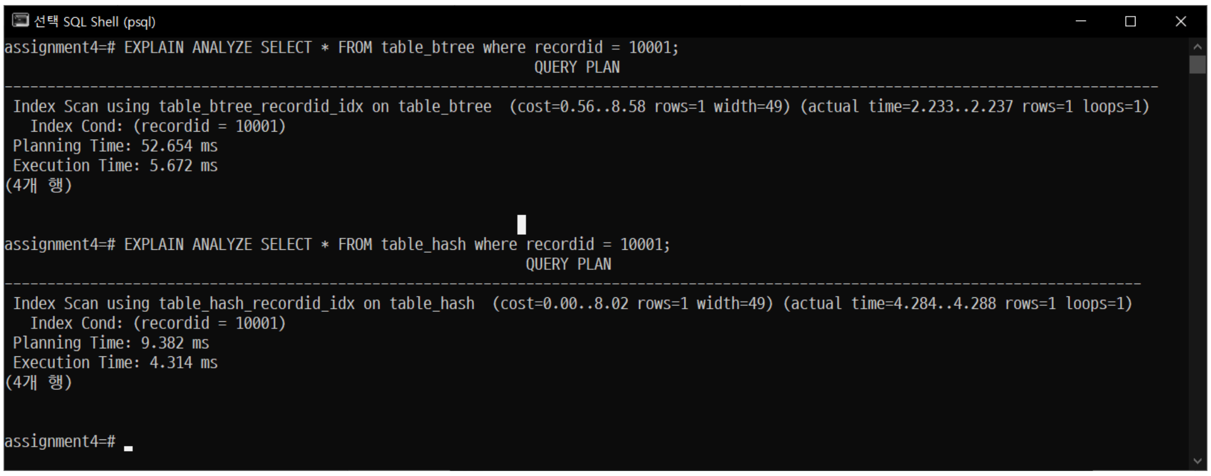
B-tree and Hash

1. Create two indexes



```
assignment4=# CREATE INDEX ON table_btree USING btree (recordid);
CREATE INDEX
assignment4=# CREATE INDEX ON table_hash USING hash (recordid);
CREATE INDEX
assignment4=#
```

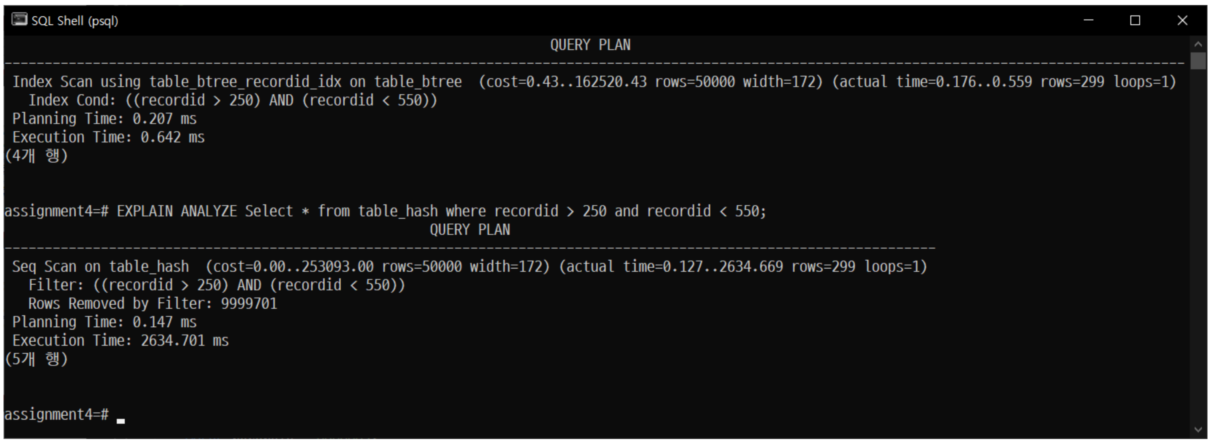
2. Run two queries. And compare the query execution plan and total execution time



```
선택 SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT * FROM table_btree where recordid = 10001;
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using table_btree_recordid_idx on table_btree  (cost=0.56..8.58 rows=1 width=49) (actual time=2.233..2.237 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 52.654 ms
 Execution Time: 5.672 ms
(4개 행)


assignment4=# EXPLAIN ANALYZE SELECT * FROM table_hash where recordid = 10001;
                                                       QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using table_hash_recordid_idx on table_hash  (cost=0.00..8.02 rows=1 width=49) (actual time=4.284..4.288 rows=1 loops=1)
   Index Cond: (recordid = 10001)
 Planning Time: 9.382 ms
 Execution Time: 4.314 ms
(4개 행)


assignment4=#
```

같은 index scan을 수행하지만, hash의 스캔이 조금 더 빠르다.

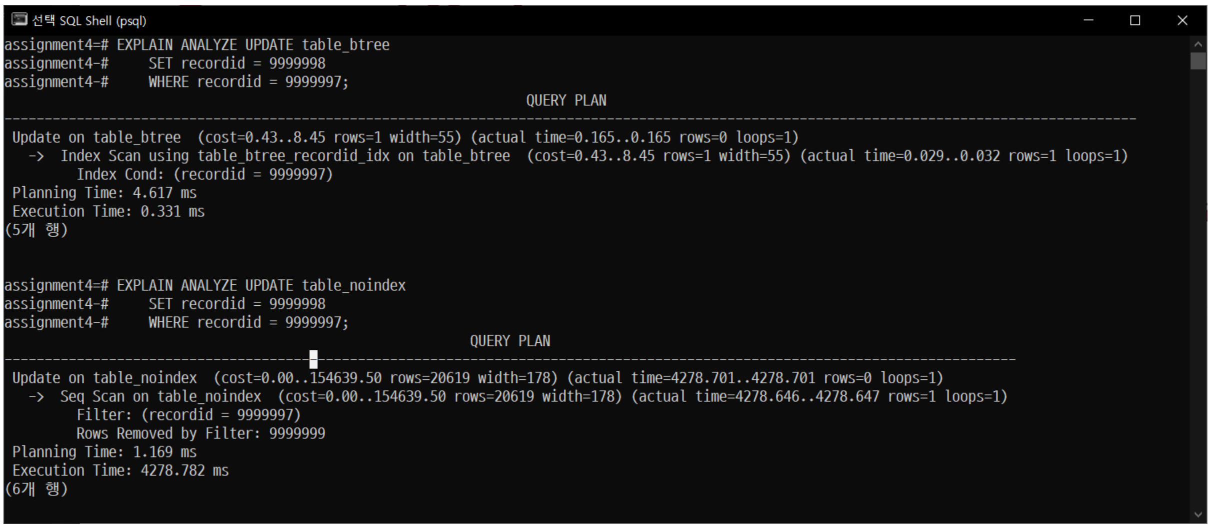
3. Run two queries. And compare the query execution plan and total execution time



```
SQL Shell (psql)
                                                           QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using table_btree_recordid_idx on table_btree  (cost=0.43..162520.43 rows=50000 width=172) (actual time=0.176..0.559 rows=299 loops=1)
   Index Cond: ((recordid > 250) AND (recordid < 550))
 Planning Time: 0.207 ms
 Execution Time: 0.642 ms
(4개 행)


assignment4=# 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=0.127..2634.669 rows=299 loops=1)
   Filter: ((recordid > 250) AND (recordid < 550))
   Rows Removed by Filter: 9999701
 Planning Time: 0.147 ms
 Execution Time: 2634.701 ms
(5개 행)


assignment4=#
```

큰 실행 시간 차이와 두번째 SQL문에서 Index Scan을 사용하지 않는 것을 볼 수 있다. 순차적으로 여러 개의 row를 찾는 데에는 btree가 유효하기 때문이다. psql에서는 자동으로 인식하여 hash를 사용하지 않는 것을 볼 수 있다.

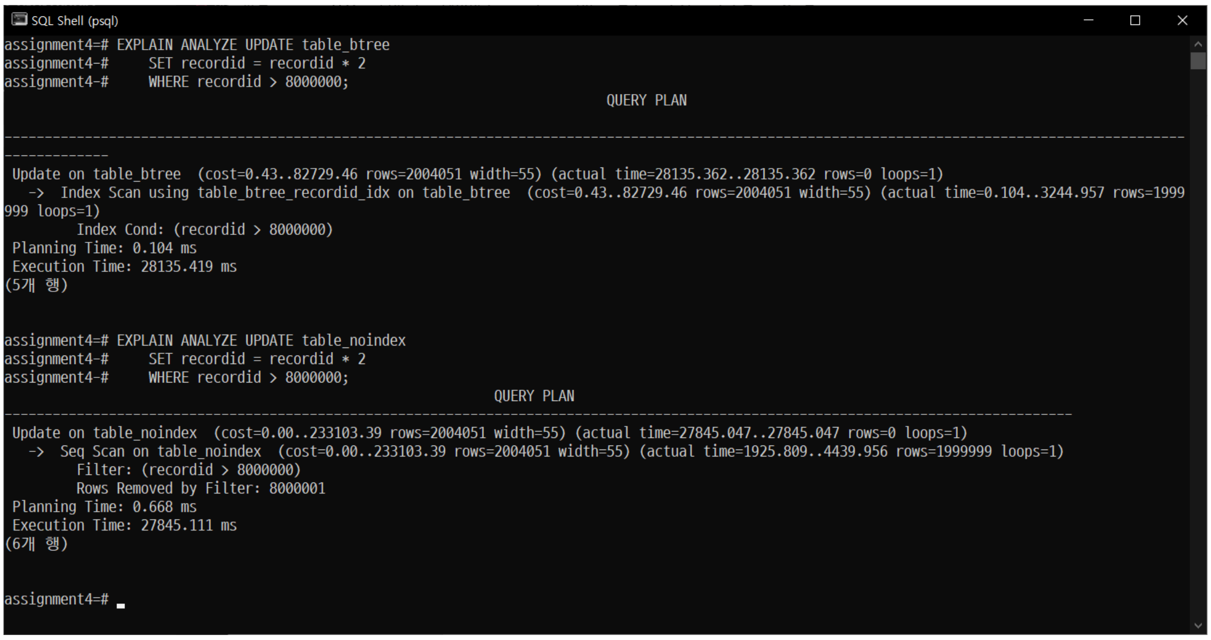
4. Update a single "recordid" field in "table_btree". And update a single "recordid" field in "table_noindex". Then find a difference



```
assignment4=# EXPLAIN ANALYZE UPDATE table_btree
assignment4-#     SET recordid = 9999998
assignment4-#     WHERE recordid = 9999997;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.43..8.45 rows=1 width=55) (actual time=0.165..0.165 rows=0 loops=1)
   ->  Index Scan using table_btree_recordid_idx on table_btree  (cost=0.43..8.45 rows=1 width=55) (actual time=0.029..0.032 rows=1 loops=1)
         Index Cond: (recordid = 9999997)
 Planning Time: 4.617 ms
 Execution Time: 0.331 ms
(5개 행)


assignment4=# EXPLAIN ANALYZE UPDATE table_noindex
assignment4-#     SET recordid = 9999998
assignment4-#     WHERE recordid = 9999997;
                                                           QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..154639.50 rows=20619 width=178) (actual time=4278.701..4278.701 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..154639.50 rows=20619 width=178) (actual time=4278.646..4278.647 rows=1 loops=1)
         Filter: (recordid = 9999997)
         Rows Removed by Filter: 9999999
 Planning Time: 1.169 ms
 Execution Time: 4278.782 ms
(6개 행)
```

한 개의 key로 찾은 뒤에 교체하므로, btree를 이용한 index scan이 더 빠르다.

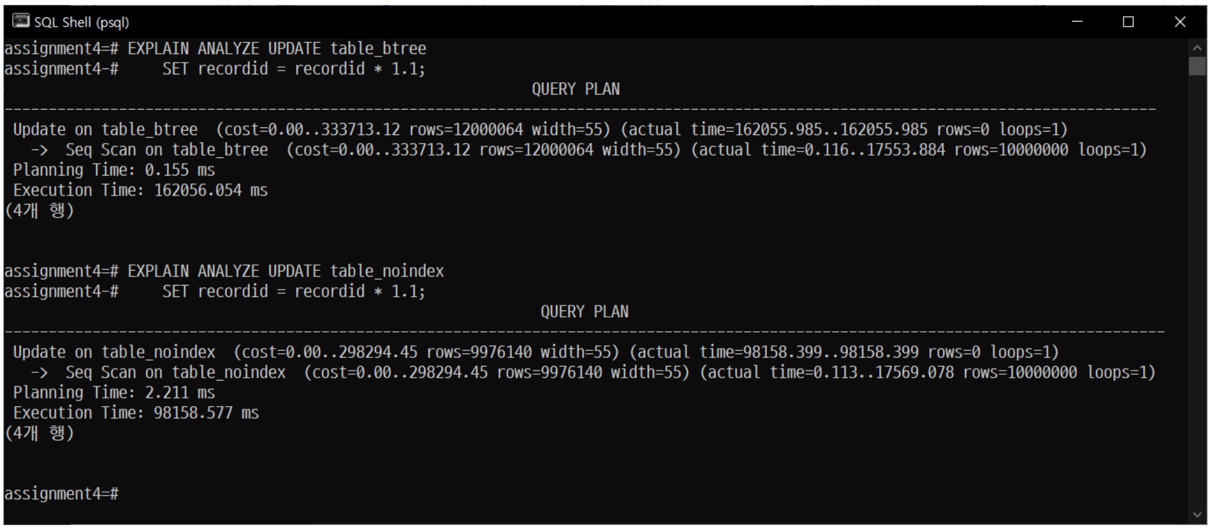
5. Update 2,000,000 "recordid" fields in "table_btree". And update 2,000,000 "recordid" fields in "table_noindex". Then find a difference



```
assignment4=# EXPLAIN ANALYZE UPDATE table_btree
assignment4-#     SET recordid = recordid * 2
assignment4-#     WHERE recordid > 8000000;
                                                                            QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------------
------------
 Update on table_btree  (cost=0.43..82729.46 rows=2004051 width=55) (actual time=28135.362..28135.362 rows=0 loops=1)
   ->  Index Scan using table_btree_recordid_idx on table_btree  (cost=0.43..82729.46 rows=2004051 width=55) (actual time=0.104..3244.957 rows=1999
999 loops=1)
         Index Cond: (recordid > 8000000)
 Planning Time: 0.104 ms
 Execution Time: 28135.419 ms
(5개 행)


assignment4=# EXPLAIN ANALYZE UPDATE table_noindex
assignment4-#     SET recordid = recordid * 2
assignment4-#     WHERE recordid > 8000000;
                                                              QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..233103.39 rows=2004051 width=55) (actual time=27845.047..27845.047 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..233103.39 rows=2004051 width=55) (actual time=1925.809..4439.956 rows=1999999 loops=1)
         Filter: (recordid > 8000000)
         Rows Removed by Filter: 8000001
 Planning Time: 0.668 ms
 Execution Time: 27845.111 ms
(6개 행)


assignment4=#
```

두가지 SQL문의 큰 실행 시간 차이가 없다.

6. Update all "recordid" fields in "table_btree". And update all "recordid" fields in "table_noindex". Then find a difference



```
assignment4=# EXPLAIN ANALYZE UPDATE table_btree
assignment4-#     SET recordid = recordid * 1.1;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Update on table_btree  (cost=0.00..333713.12 rows=12000064 width=55) (actual time=162055.985..162055.985 rows=0 loops=1)
   ->  Seq Scan on table_btree  (cost=0.00..333713.12 rows=12000064 width=55) (actual time=0.116..17553.884 rows=10000000 loops=1)
 Planning Time: 0.155 ms
 Execution Time: 162056.054 ms
(4개 행)


assignment4=# EXPLAIN ANALYZE UPDATE table_noindex
assignment4-#     SET recordid = recordid * 1.1;
                                                             QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 Update on table_noindex  (cost=0.00..298294.45 rows=9976140 width=55) (actual time=98158.399..98158.399 rows=0 loops=1)
   ->  Seq Scan on table_noindex  (cost=0.00..298294.45 rows=9976140 width=55) (actual time=0.113..17569.078 rows=10000000 loops=1)
 Planning Time: 2.211 ms
 Execution Time: 98158.577 ms
(4개 행)


assignment4=#
```

table_noindex의 update가 더 빠르다.