2019320097_조이강

1. Create two indexes on “table1”
   a. Indexed attributes are “sorted” and “unsorted”

```
d2019320097=# create index on table1 (sorted)
d2019320097-# ;
CREATE INDEX
d2019320097=# create index on table1 (unsorted)
d2019320097-# ;
CREATE INDEX
```

2. PostgreSQL supports following index-based query execution plans

   a. Make (and execute) three queries each of which uses seq scan, index scan, and index only scan respectively

```
d2019320097=# EXPLAIN ANALYZE SELECT * FROM table1;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..258104.00 rows=10000000 width=176) (actual time=0.012..2111.700 rows=10000000 loops=1)
 Planning Time: 2.000 ms
 Execution Time: 2668.905 ms
(3개 행)
```

```
d2019320097=# EXPLAIN ANALYZE SELECT * FROM table1 where sorted = 100000;
                                                              QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
-------------
 Index Scan using table1_sorted_idx on table1  (cost=0.43..173859.43 rows=50000 width=176) (actual time=0.022..0.024 row
s=5 loops=1)
   Index Cond: (sorted = 100000)
 Planning Time: 1.389 ms
 Execution Time: 0.042 ms
(4개 행)
```

```
d2019320097=# EXPLAIN ANALYZE SELECT sorted FROM table1 where sorted = 100000;
                                                                QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
-----------------
 Index Only Scan using table1_sorted_idx on table1  (cost=0.43..137119.43 rows=50000 width=4) (actual time=0.556..0.558
rows=5 loops=1)
   Index Cond: (sorted = 100000)
   Heap Fetches: 0
 Planning Time: 0.091 ms
 Execution Time: 0.575 ms
(5개 행)
```

   b. Make two queries that are expected to use indices on attributes “sorted” and “unsorted” respectively, then compare their execution times

```
d2019320097=# cluster table1 using table1_sorted_idx;
CLUSTER
d2019320097=# EXPLAIN ANALYZE SELECT * FROM table1 where sorted = 5000000;
                                                        QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
----
 Index Scan using table1_sorted_idx on table1  (cost=0.43..8.82 rows=22 width=53) (actual time=0.046..0.046 rows=0 loops
=1)
   Index Cond: (sorted = 5000000)
 Planning Time: 2.428 ms
 Execution Time: 0.063 ms
(4개 행)
```

```
d2019320097=# EXPLAIN ANALYZE SELECT * FROM table1 where unsorted = 5000000;
                                                        QUERY PLAN

------------------------------------------------------------------------------------------------------------------------
------
 Index Scan using table1_unsorted_idx on table1  (cost=0.43..28.54 rows=6 width=53) (actual time=0.112..0.112 rows=0 loo
ps=1)
   Index Cond: (unsorted = 5000000)
 Planning Time: 0.095 ms
 Execution Time: 0.129 ms
(4개 행)
```

- sorted의 경우 실행 시간 = 0.063ms
- unsorted의 경우 실행 시간 = 0.129ms로, 상대적으로 오래 걸립니다.

c. Execute your queries to Exercise 2.b. after executing each of the following queries respectively, then compare their execution times

```
d2019320097=# cluster table1 using table1_unsorted_idx;
CLUSTER
d2019320097=# EXPLAIN ANALYZE SELECT * FROM table1 where unsorted = 5000000;
                                                        QUERY PLAN

------------------------------------------------------------------------------------------------------------------------
------
 Index Scan using table1_unsorted_idx on table1  (cost=0.43..28.54 rows=6 width=53) (actual time=0.033..0.034 rows=0 loo
ps=1)
   Index Cond: (unsorted = 5000000)
 Planning Time: 1.205 ms
 Execution Time: 0.055 ms
(4개 행)
```

```
d2019320097=# cluster table1 using table1_sorted_idx;
CLUSTER
작업 시간: 33323.094 ms (00:33.323)
d2019320097=# explain analyze select * from table1 where sorted = 5000000;
                                                       QUERY PLAN

------------------------------------------------------------------------------------------------------------------------
----
 Index Scan using table1_sorted_idx on table1  (cost=0.43..8.82 rows=22 width=53) (actual time=0.046..0.046 rows=0 loops
=1)
   Index Cond: (sorted = 5000000)
 Planning Time: 1.713 ms
 Execution Time: 0.069 ms
(4개 행)
```

- cluster를 한 경우 unsorted와 sorted가 각각 0.055ms 와 0.069ms로 비슷한 실행 시간을 가지게 됩니다.

d. Execute and compare the following two queries:
• SELECT sorted, rndm FROM table1 WHERE sorted>1999231 AND rndm=1005;

```
d2019320097=# SELECT sorted, rndm FROM table1 WHERE sorted>1999231 AND rndm=1005;
 sorted | rndm
--------+------
(0개 행)
```

```
d2019320097=# 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..152.78 rows=1 width=8) (actual time=0.949..0.949 rows=0 loops
=1)
   Index Cond: (sorted > 1999231)
   Filter: (rndm = 1005)
   Rows Removed by Filter: 3840
 Planning Time: 0.234 ms
 Execution Time: 0.975 ms
(6개 행)
```

• SELECT sorted, rndm FROM table1 WHERE sorted<1999231 AND rndm=1005;

```
d2019320097=# SELECT sorted, rndm FROM table1 WHERE sorted<1999231 AND rndm=1005;
 sorted  | rndm
---------+------
   22650 | 1005
   41380 | 1005
   64606 | 1005
   67401 | 1005
   78297 | 1005
   82119 | 1005
   89969 | 1005
  104100 | 1005
  119710 | 1005
  149258 | 1005
  154349 | 1005
  183054 | 1005
  235268 | 1005
  240809 | 1005
  265959 | 1005
  292697 | 1005
  297155 | 1005
  302988 | 1005
  319896 | 1005
  332827 | 1005
  348206 | 1005
  362627 | 1005
  390557 | 1005
  399064 | 1005
  406610 | 1005
```

```
d2019320097=# 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=101 width=8) (actual time=19.862..2802.129 rows=114 loops=1)
   Filter: ((sorted < 1999231) AND (rndm = 1005))
   Rows Removed by Filter: 9999886
 Planning Time: 0.201 ms
 Execution Time: 2802.261 ms
(5개 행)
```

• Explain why their query plans are different

- sorted < 199231 이면서 rndm = 1005 라는 조건을 위해서, sorted에서 9999886개 이상의 행이 검색되었고, rndm = 1005라는 조건을 만족하지 못해 삭제되었습니다. 이는 sorted > 199231를 만족하고 rndm = 1005를 만족 못해 삭제된 3840개와 비교했을 때 압도적으로 많은 수치입니다.
  따라서 DBMS에서는 거의 대다수의 행을 검색해야 하는 sorted < 199231에 대해서는 인덱스를 거치지 않는 seq scan을, sorted > 199231에 대해서는 index scan을 실행하려고 할 것입니다.
  따라서 이 둘 사이에 실행 시간과 query plan이 달라지게 됩니다.

3. Setup: Create a synthetic data set that has 5,000,000 rows

```
d2019320097=# create table pool(val integer);
CREATE TABLE
d2019320097=# INSERT INTO pool(val) SELECT * FROM (SELECT
d2019320097(# generate_series(1,5000000)) as T;
INSERT 0 5000000
d2019320097=# SET enable_bitmapscan=false; \timing
SET
작업수행시간  보임
d2019320097=#
```

Consider two cases below. Which case will take a longer time?

- Inserting tuples in a table, and then creating index
- Creating index, and then inserting tuples in a table

- Compare the execution time $t1$ and $t2$

  $t1 = t1.insert + t1.create\_index$

  - Tuple insertion → Index creation

```
d2019320097=# insert into table10 (select * from pool);
INSERT 0 5000000
작업시간: 11211.542 ms (00:11.212)
```

```
d2019320097=# create index on table10(val);
CREATE INDEX
작업시간: 3705.720 ms (00:03.706)
```

  $t2 = t2.create\_index + t2.insert$

  - Index creation → Tuple insertion

```
d2019320097=# create index on table20(val);
CREATE INDEX
작업시간: 11.403 ms
d2019320097=# insert into table20 (select * from pool);
INSERT 0 5000000
작업시간: 22615.823 ms (00:22.616)
d2019320097=#
```

- t1의 경우 삽입에 11211 ms, 인덱스 생성에 3705 ms가 소요되며, t2의 경우 인덱스 생성에 11 ms, 삽입에 22615 ms가 소요된 것을 확인 할 수 있습니다.
  인덱스를 생성하고 삽입을 진행하는 경우, 삽입된 데이터에 맞게 인덱스를 업데이트해야 하므로 t2에서 삽입에 많은 시간을 소모한 것이라고 예상할 수 있습니다.
  삽입 후 인덱스를 생성하는 경우는 각각의 소요 시간은 작지 않으나, 총 작업 시간은 t2에 비해 적음을 알 수 있습니다.