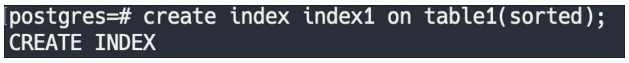
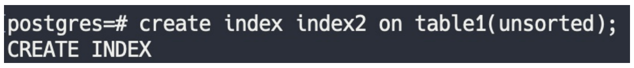
[Exercise 1]

- Create two indexes on “table1”

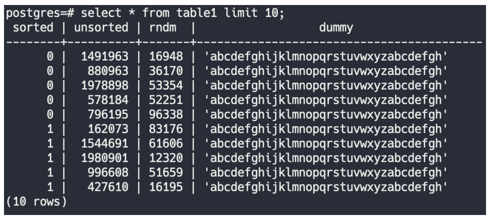


```
postgres=# create index index1 on table1(sorted);
CREATE INDEX
```



```
postgres=# create index index2 on table1(unsorted);
CREATE INDEX
```

- Which is clustered index or non-clustered index?



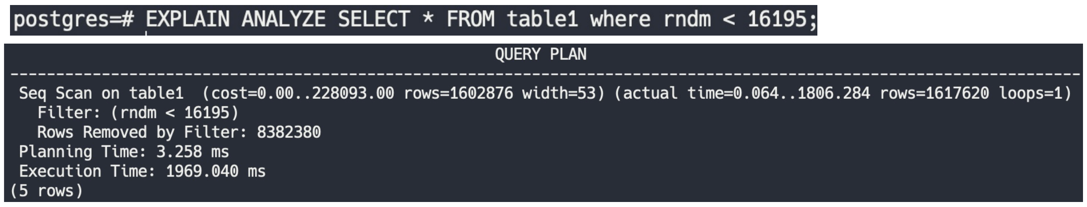
```
postgres=# select * from table1 limit 10;
 sorted | unsorted | rndm  |                  dummy
--------+----------+-------+------------------------------------------
      0 |  1491963 | 16948 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      0 |   880963 | 36170 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      0 |  1978898 | 53354 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      0 |   578184 | 52251 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      0 |   796195 | 96338 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      1 |   162073 | 83176 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      1 |  1544691 | 61606 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      1 |  1980901 | 12320 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      1 |   996608 | 51659 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
      1 |   427610 | 16195 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
(10 rows)
```

The search key of ‘index1’ specifies the sequential order of the file while that of ‘index2’ dose not.
So we can say index1 is clustered index and index2 is non-clustered index.

[Exercise 2]

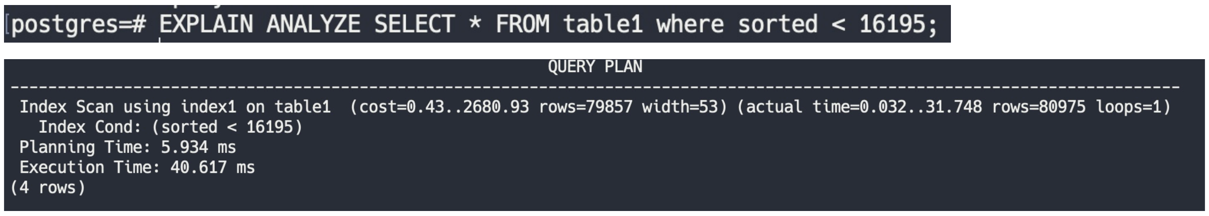
A. Make (and execute) three queries each of which uses seq scan, index scan, and index only scan respectively.

(1) Seq Scan



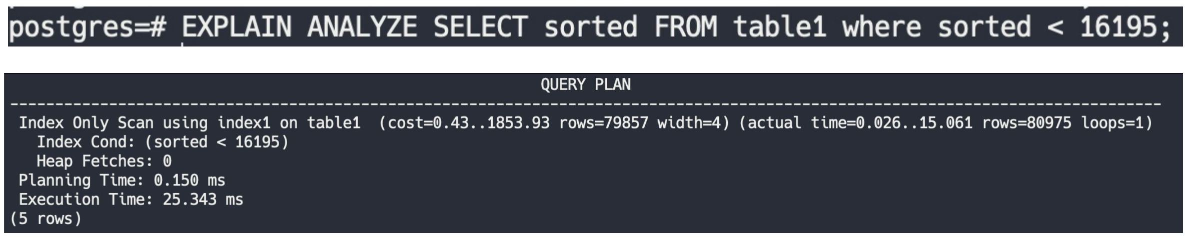
```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 where rndm < 16195;
                                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..228093.00 rows=1602876 width=53) (actual time=0.064..1806.284 rows=1617620 loops=1)
   Filter: (rndm < 16195)
   Rows Removed by Filter: 8382380
 Planning Time: 3.258 ms
 Execution Time: 1969.040 ms
(5 rows)
```

(2) Index Scan



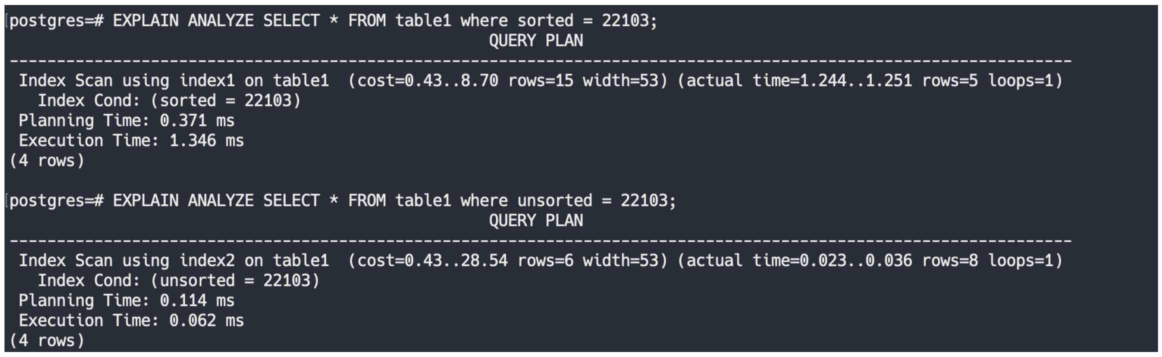
```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 where sorted < 16195;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Index Scan using index1 on table1  (cost=0.43..2680.93 rows=79857 width=53) (actual time=0.032..31.748 rows=80975 loops=1)
   Index Cond: (sorted < 16195)
 Planning Time: 5.934 ms
 Execution Time: 40.617 ms
(4 rows)
```

(3) Index Only Scan



```
postgres=# EXPLAIN ANALYZE SELECT sorted FROM table1 where sorted < 16195;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using index1 on table1  (cost=0.43..1853.93 rows=79857 width=4) (actual time=0.026..15.061 rows=80975 loops=1)
   Index Cond: (sorted < 16195)
   Heap Fetches: 0
 Planning Time: 0.150 ms
 Execution Time: 25.343 ms
(5 rows)
```

B. Make two queries using clustered index and non-clustered index, then compare their execution times.



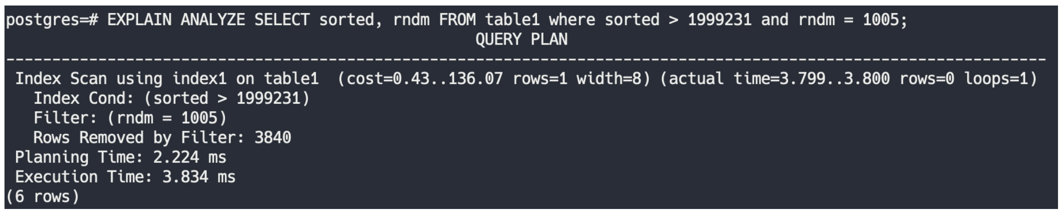
```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 where sorted = 22103;
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Index Scan using index1 on table1  (cost=0.43..8.70 rows=15 width=53) (actual time=1.244..1.251 rows=5 loops=1)
   Index Cond: (sorted = 22103)
 Planning Time: 0.371 ms
 Execution Time: 1.346 ms
(4 rows)

postgres=# EXPLAIN ANALYZE SELECT * FROM table1 where unsorted = 22103;
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Index Scan using index2 on table1  (cost=0.43..28.54 rows=6 width=53) (actual time=0.023..0.036 rows=8 loops=1)
   Index Cond: (unsorted = 22103)
 Planning Time: 0.114 ms
 Execution Time: 0.062 ms
(4 rows)
```

Planning Time 과 Execution Time 모두 non-clustered index(‘index2’)가 clustered index(‘index1’) 보다 더 빠르다.

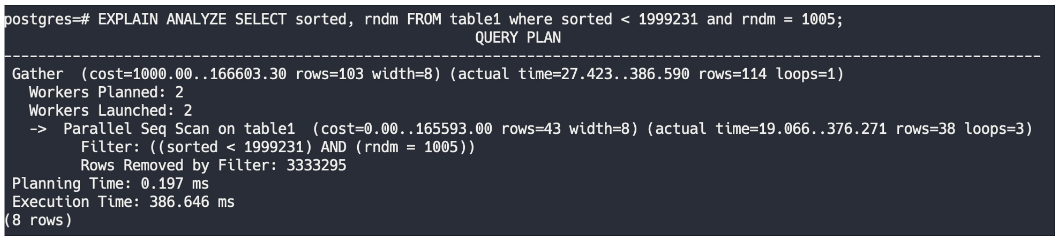
C. Execute and compare the following two queries:

○ SELECT sorted, rndm FROM table1 WHERE sorted>1999231 AND rndm=1005;



```
postgres=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 where sorted > 1999231 and rndm = 1005;
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Index Scan using index1 on table1  (cost=0.43..136.07 rows=1 width=8) (actual time=3.799..3.800 rows=0 loops=1)
   Index Cond: (sorted > 1999231)
   Filter: (rndm = 1005)
   Rows Removed by Filter: 3840
 Planning Time: 2.224 ms
 Execution Time: 3.834 ms
(6 rows)
```

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



```
postgres=# EXPLAIN ANALYZE SELECT sorted, rndm FROM table1 where sorted < 1999231 and rndm = 1005;
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..166603.30 rows=103 width=8) (actual time=27.423..386.590 rows=114 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on table1  (cost=0.00..165593.00 rows=43 width=8) (actual time=19.066..376.271 rows=38 loops=3)
         Filter: ((sorted < 1999231) AND (rndm = 1005))
         Rows Removed by Filter: 3333295
 Planning Time: 0.197 ms
 Execution Time: 386.646 ms
(8 rows)
```

• Explain why their query plans are different

: sorted column은 index1의 attribute이고, index1은 clustered index이다.
따라서 데이터가 이미 sorted 값을 따라 오름차순으로 정렬돼 있기 때문에 첫번째 쿼리의 경우,
sorted > 1999231인 인덱스 중 가장 작은 인덱스 값을 한 번만 찾은 다음 순차적으로 데이터를 읽어나가며 rndm = 1005인 record를 찾으면 된다. 하지만 두번째 쿼리는 199231보다 작은 인덱스 값을 모두 찾은 다음 각각의 데이터를 읽어야하기 때문에 그 과정에서 더 많은 시간이 소요된다. 이에 따라 postgresql도 그와 같은 index scan 방식 대신 sequential scan을 통해서 sorted = 0 부터 순차적으로 데이터를 읽어 나가는 방식을 택했다.

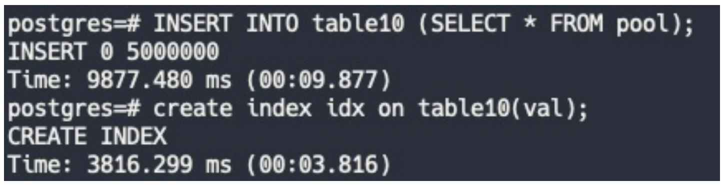
[Exercise 3]

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

    1. Inserting tuples in a table, and then creating index
    2. Creating index, and then inserting tuples in a table

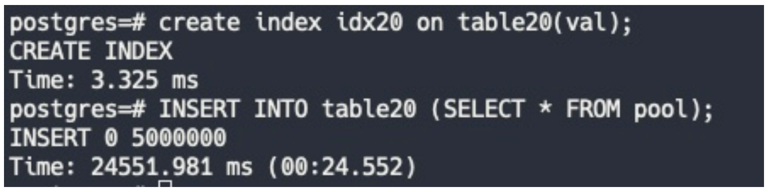
- Compare the execution time t1 and t2

(1) t1 : tuple insertion and index creation



```
postgres=# INSERT INTO table10 (SELECT * FROM pool);
INSERT 0 5000000
Time: 9877.480 ms (00:09.877)
postgres=# create index idx on table10(val);
CREATE INDEX
Time: 3816.299 ms (00:03.816)
```

(2) t2 : index creation and tuple insertion



```
postgres=# create index idx20 on table20(val);
CREATE INDEX
Time: 3.325 ms
postgres=# INSERT INTO table20 (SELECT * FROM pool);
INSERT 0 5000000
Time: 24551.981 ms (00:24.552)
```

T1 = 9.877 + 3.816 = 13.693
T2 = 0.003 + 24.552 = 24.555

실습 결과 T1 < T2 이다.
즉 (1) 레코드 삽입 이후 인덱스를 생성하는 것이 (2) 그 반대의 경우보다 더 빠르다는 것을 확인할 수 있다.