Exercise 1

- Consider the following query and make corresponding SQL statements, and then show that the results are the same
  - Select “unsorted” from table1 where the “unsorted” value is 967 or 968 or 969 (967~969)

a. Make an SQL statement using “BETWEEN” operator

```
postgres=# SELECT * FROM table1 WHERE unsorted BETWEEN 967 AND 969;
 sorted  | unsorted | rndm  |                   dummy
---------+----------+-------+-------------------------------------------
  152803 |      968 | 66055 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  224629 |      969 | 74033 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  411490 |      969 | 95384 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  613875 |      969 | 59258 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  691366 |      969 | 16666 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  767551 |      969 |  7453 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  787173 |      969 | 97675 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  892181 |      969 | 95073 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  949222 |      968 | 91161 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1096917 |      967 | 50987 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1388668 |      969 |  3507 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1452892 |      969 | 42128 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1493484 |      968 | 84394 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1554813 |      969 | 88993 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1565640 |      969 | 28844 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1699431 |      968 | 93315 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1885239 |      969 | 76266 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1952117 |      968 | 45504 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
(18 rows)
```

b. Make an SQL statement using “IN” operator

```
postgres=# SELECT * FROM table1 WHERE unsorted in (967,968,969);
 sorted  | unsorted | rndm  |                   dummy
---------+----------+-------+-------------------------------------------
  152803 |      968 | 66055 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  224629 |      969 | 74033 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  411490 |      969 | 95384 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  613875 |      969 | 59258 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  691366 |      969 | 16666 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  767551 |      969 |  7453 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  787173 |      969 | 97675 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  892181 |      969 | 95073 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  949222 |      968 | 91161 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1096917 |      967 | 50987 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1388668 |      969 |  3507 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1452892 |      969 | 42128 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1493484 |      968 | 84394 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1554813 |      969 | 88993 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1565640 |      969 | 28844 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1699431 |      968 | 93315 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1885239 |      969 | 76266 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1952117 |      968 | 45504 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
(18 rows)
```

c. Make an SQL statement using “=” and “OR” operator

```
postgres=# SELECT * FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
 sorted  | unsorted | rndm  |                   dummy
---------+----------+-------+-------------------------------------------
  152803 |      968 | 66055 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  224629 |      969 | 74033 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  411490 |      969 | 95384 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  613875 |      969 | 59258 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  691366 |      969 | 16666 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  767551 |      969 |  7453 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  787173 |      969 | 97675 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  892181 |      969 | 95073 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  949222 |      968 | 91161 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1096917 |      967 | 50987 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1388668 |      969 |  3507 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1452892 |      969 | 42128 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1493484 |      968 | 84394 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1554813 |      969 | 88993 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1565640 |      969 | 28844 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1699431 |      968 | 93315 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1885239 |      969 | 76266 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1952117 |      968 | 45504 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
(18 rows)
```

d. Make an SQL statement using “UNION ALL” operator

```
postgres=# SELECT * FROM table1 WHERE unsorted = 967
postgres-# UNION ALL
postgres-# SELECT * FROM table1 WHERE unsorted = 968
postgres-# UNION ALL
postgres-# SELECT * FROM table1 WHERE unsorted = 969;
 sorted  | unsorted | rndm  |                   dummy
---------+----------+-------+-------------------------------------------
 1096917 |      967 | 50987 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  152803 |      968 | 66055 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  949222 |      968 | 91161 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1493484 |      968 | 84394 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1699431 |      968 | 93315 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1952117 |      968 | 45504 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  224629 |      969 | 74033 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  411490 |      969 | 95384 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  613875 |      969 | 59258 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  691366 |      969 | 16666 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  767551 |      969 |  7453 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  787173 |      969 | 97675 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
  892181 |      969 | 95073 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1388668 |      969 |  3507 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1452892 |      969 | 42128 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1554813 |      969 | 88993 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1565640 |      969 | 28844 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
 1885239 |      969 | 76266 | 'abcdefghijklmnopqrstuvwxyzabcdefgh'
(18 rows)
```

Exercise 2

- Execute SQL statements in Exercise 1 under the following conditions, and then compare the execution times

a. No index

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..253093.32 rows=17 width=53) (actual time=100.466..950.465 rows=18 loops=1)
   Filter: ((unsorted >= 967) AND (unsorted <= 969))
   Rows Removed by Filter: 9999982
 Planning Time: 0.145 ms
 Execution Time: 950.497 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted in (967,968,969);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..240593.29 rows=17 width=53) (actual time=113.317..1038.461 rows=18 loops=1)
   Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
   Rows Removed by Filter: 9999982
 Planning Time: 0.088 ms
 Execution Time: 1038.493 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..278093.37 rows=17 width=53) (actual time=104.952..986.828 rows=18 loops=1)
   Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
   Rows Removed by Filter: 9999982
 Planning Time: 0.080 ms
 Execution Time: 986.873 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967
UNION ALL
SELECT * FROM table1 WHERE unsorted = 968
UNION ALL
SELECT * FROM table1 WHERE unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..684280.06 rows=18 width=53) (actual time=476.462..2454.046 rows=18 loops=1)
   ->  Seq Scan on table1  (cost=0.00..228093.26 rows=6 width=53) (actual time=476.461..839.134 rows=1 loops=1)
         Filter: (unsorted = 967)
         Rows Removed by Filter: 9999999
   ->  Seq Scan on table1 table1_1  (cost=0.00..228093.26 rows=6 width=53) (actual time=61.491..804.234 rows=5 loops=1)
         Filter: (unsorted = 968)
         Rows Removed by Filter: 9999995
   ->  Seq Scan on table1 table1_2  (cost=0.00..228093.26 rows=6 width=53) (actual time=90.719..810.651 rows=12 loops=1)
         Filter: (unsorted = 969)
         Rows Removed by Filter: 9999988
 Planning Time: 0.182 ms
 Execution Time: 2454.087 ms
(12 rows)
```

인덱스가 없는 상태에서 SQL문을 실행한 결과는 위와 같다.
BETWEEN 연산자 , IN 연산자 , 그리고 = 와 OR 연산자를 활용해 equi-val selection을 하는 경우
세 경우 모두 Seq Scan이 한 번 이루어지기 때문에 방식의 차이는 있어도 약 950~ 1050 ms 정도로 비슷한 실행시간이 소요된다.
반면 마지막 경우인 UNION ALL의 경우 Seq Scan을 세 번 수행 한 다음 UNION이 이루어지는 것이어서 가장 오랜 수행시간(2454ms)이 소요됐다.

b. B-tree index

```
postgres=# CREATE INDEX idx_unsorted ON table1 USING btree (unsorted);
CREATE INDEX
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using idx_unsorted on table1  (cost=0.43..72.77 rows=17 width=53) (actual time=0.073..0.309 rows=18 loops=1)
   Index Cond: ((unsorted >= 967) AND (unsorted <= 969))
 Planning Time: 0.824 ms
 Execution Time: 1.549 ms
(4 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted in (967,968,969);
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using idx_unsorted on table1  (cost=0.43..81.61 rows=17 width=53) (actual time=0.078..0.120 rows=18 loops=1)
   Index Cond: (unsorted = ANY ('{967,968,969}'::integer[]))
 Planning Time: 0.111 ms
 Execution Time: 0.144 ms
(4 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..278093.00 rows=17 width=53) (actual time=103.508..984.209 rows=18 loops=1)
   Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
   Rows Removed by Filter: 9999982
 Planning Time: 0.106 ms
 Execution Time: 984.242 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967
UNION ALL
SELECT * FROM table1 WHERE unsorted = 968
UNION ALL
SELECT * FROM table1 WHERE unsorted = 969;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.43..85.89 rows=18 width=53) (actual time=0.044..0.119 rows=18 loops=1)
   ->  Index Scan using idx_unsorted on table1  (cost=0.43..28.54 rows=6 width=53) (actual time=0.043..0.045 rows=1 loops=1)
         Index Cond: (unsorted = 967)
   ->  Index Scan using idx_unsorted on table1 table1_1  (cost=0.43..28.54 rows=6 width=53) (actual time=0.020..0.028 rows=5 loops=1)
         Index Cond: (unsorted = 968)
   ->  Index Scan using idx_unsorted on table1 table1_2  (cost=0.43..28.54 rows=6 width=53) (actual time=0.017..0.034 rows=12 loops=1)
         Index Cond: (unsorted = 969)
 Planning Time: 0.289 ms
 Execution Time: 0.204 ms
(9 rows)
```

세 번째 SQL문을 제외하고는 모두 btree 인덱스를 활용한 Index Scan을 수행했다. 수행시간은 Seq Scan에 비해 훨씬 줄어들었다.
세 번째 SQL문의 경우, Index Scan을 3번 수행하는 것도 가능하나, postgresSQL이 Seq Scan 한 번 수행하는 것보다 더 비효율적이라 판단해 Seq Scan이 실행됐다. 마지막 SQL의 경우는 쿼리문 자체가 각각의 값에 대해 scan을 먼저 수행하고 그 결과를 합치도록 하고 있기 때문에 Seq Scan을 한 번 수행하는 대신 Index Scan 세 번을 수행했다.

c. Hash index

```
postgres=# CREATE INDEX idx_hash ON table1 USING hash (unsorted);
CREATE INDEX
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..253093.00 rows=17 width=53) (actual time=100.143..953.198 rows=18 loops=1)
   Filter: ((unsorted >= 967) AND (unsorted <= 969))
   Rows Removed by Filter: 9999982
 Planning Time: 4.733 ms
 Execution Time: 953.234 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted in (967,968,969);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..240593.00 rows=17 width=53) (actual time=102.653..1038.624 rows=18 loops=1)
   Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
   Rows Removed by Filter: 9999982
 Planning Time: 0.105 ms
 Execution Time: 1038.654 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on table1  (cost=0.00..278093.00 rows=17 width=53) (actual time=113.869..1035.000 rows=18 loops=1)
   Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
   Rows Removed by Filter: 9999982
 Planning Time: 0.138 ms
 Execution Time: 1035.043 ms
(5 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM table1 WHERE unsorted = 967
UNION ALL
SELECT * FROM table1 WHERE unsorted = 968
UNION ALL
SELECT * FROM table1 WHERE unsorted = 969;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..84.59 rows=18 width=53) (actual time=1.218..2.118 rows=18 loops=1)
   ->  Index Scan using idx_hash on table1  (cost=0.00..28.11 rows=6 width=53) (actual time=1.216..1.219 rows=1 loops=1)
         Index Cond: (unsorted = 967)
   ->  Index Scan using idx_hash on table1 table1_1  (cost=0.00..28.11 rows=6 width=53) (actual time=0.745..0.778 rows=5 loops=1)
         Index Cond: (unsorted = 968)
   ->  Index Scan using idx_hash on table1 table1_2  (cost=0.00..28.11 rows=6 width=53) (actual time=0.030..0.109 rows=12 loops=1)
         Index Cond: (unsorted = 969)
 Planning Time: 0.186 ms
 Execution Time: 2.178 ms
(9 rows)
```

Hash Index가 사용된 SQL 문은 UNION ALL을 사용한 마지막 쿼리문 뿐이다.
기본적으로 DBMS가 Index Scan 3번을 수행하는 것 보다 Seq Scan 1번을 수행하는 것을 더 선호한다고 볼 수 있다.
마지막 쿼리문의 경우 반드시 Scan을 3번 수행하도록 쿼리문 자체가 제약을 걸고 있기 때문에 Seq Scan 3번 보다는 더 빠른 Index Scan이 사용됐다.
또 하나 관찰할 수 있는 것은 Hash Index가 Btree Index 보다 실행시간이 약간 더 오래 걸린다는 점에서 검색 속도가 상대적으로 느리다는 것도 알 수 있다.

Exercise 3

- The following queries have different syntax but return the same results
- Write the queries and use 'EXPLAIN ANALYZE' statement to see how the query execution is planned

a. UNION ALL tables, and then perform aggregation with COUNT function

```
postgres=# EXPLAIN ANALYZE SELECT count(val)
FROM (SELECT val FROM pool1 UNION ALL SELECT val FROM pool2) as T
GROUP BY val;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=244248.00..244250.00 rows=200 width=12) (actual time=3756.064..3756.140 rows=501 loops=1)
   Group Key: pool1.val
   Batches: 1  Memory Usage: 121kB
   ->  Append  (cost=0.00..194248.00 rows=10000000 width=4) (actual time=0.040..2053.736 rows=10000000 loops=1)
         ->  Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.039..469.178 rows=5000000 loops=1)
         ->  Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.029..446.826 rows=5000000 loops=1)
 Planning Time: 0.134 ms
 Execution Time: 3756.255 ms
(8 rows)
```

b. Perform aggregation with COUNT function on each table, and then aggregate them again with SUM function on the UNION ALL of the aggregated results

```
postgres=# EXPLAIN ANALYZE SELECT val, SUM(cnt)
FROM (SELECT val, count(val) as cnt FROM pool1 GROUP BY val UNION ALL SELECT val, count(val) as cnt FROM pool2 GROUP BY val) as T
GROUP BY val;
                                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=194322.99..194333.01 rows=200 width=36) (actual time=2514.805..2515.141 rows=501 loops=1)
   Group Key: pool1.val
   ->  Sort  (cost=194322.99..194325.50 rows=1002 width=12) (actual time=2514.796..2514.870 rows=1002 loops=1)
         Sort Key: pool1.val
         Sort Method: quicksort  Memory: 71kB
         ->  Append  (cost=97124.00..194273.05 rows=1002 width=12) (actual time=1294.900..2514.504 rows=1002 loops=1)
               ->  HashAggregate  (cost=97124.00..97129.01 rows=501 width=12) (actual time=1294.899..1294.976 rows=501 loops=1)
                     Group Key: pool1.val
                     Batches: 1  Memory Usage: 105kB
                     ->  Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.016..466.995 rows=5000000 loops=1)
               ->  HashAggregate  (cost=97124.00..97129.01 rows=501 width=12) (actual time=1219.346..1219.414 rows=501 loops=1)
                     Group Key: pool2.val
                     Batches: 1  Memory Usage: 105kB
                     ->  Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.027..417.850 rows=5000000 loops=1)
 Planning Time: 0.251 ms
 Execution Time: 2515.259 ms
(16 rows)
```

A의 경우 poo1, poo2 전체 테이블을 합친 다음 Hash Aggregate가 한 번 이루어졌다.
B는 각각에 대해 Hash Aggreagte로 count가 이루어진 다음, 그 결과를 합쳤다. 그리고 Group Aggregate로 count 값에 대한 sum이 이루어졌다.
절차상 B가 더 복잡해 보이지만 실행시간은 오히려 B가 더 작다. 그 이유는 테이블 각각에 대해 Aggregation이 먼저 이루어져서 뒤에 수행되는 UNION과 SUM 과정에서 탐색되는 테이블의 크기가 줄어들었기 때문이라 생각한다.

Exercise 4

- Compare execution plans of the following queries
- Write the queries and use 'EXPLAIN ANALYZE' statement to see how the query execution is planned

a. SELECT tuple WHERE value is above 250 on each table, and then UNION them

```
postgres=# EXPLAIN ANALYZE SELECT *
FROM ( SELECT val FROM pool1 WHERE val > 250 UNION ALL SELECT val FROM pool2 WHERE val>250) as T;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..243961.16 rows=4980877 width=4) (actual time=0.030..1535.532 rows=4989095 loops=1)
   ->  Seq Scan on pool1  (cost=0.00..84624.00 rows=2477714 width=4) (actual time=0.029..501.383 rows=2495549 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2504451
   ->  Seq Scan on pool2  (cost=0.00..84624.00 rows=2503163 width=4) (actual time=0.028..471.045 rows=2493546 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2506454
 Planning Time: 0.188 ms
 Execution Time: 1839.993 ms
(9 rows)
```

b. UNION two tables, and then SELECT tuples WHERE value is above 250

```
postgres=# EXPLAIN ANALYZE SELECT *
FROM ( SELECT * FROM pool1 UNION ALL SELECT * FROM pool2 ) as T
WHERE val > 250;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..194152.39 rows=4980877 width=4) (actual time=0.018..1589.345 rows=4989095 loops=1)
   ->  Seq Scan on pool1  (cost=0.00..84624.00 rows=2477714 width=4) (actual time=0.017..501.209 rows=2495549 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2504451
   ->  Seq Scan on pool2  (cost=0.00..84624.00 rows=2503163 width=4) (actual time=0.034..503.785 rows=2493546 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2506454
 Planning Time: 0.226 ms
 Execution Time: 1905.958 ms
(9 rows)
```

A와 B는 서로 다른 SQL 쿼리문이지만 결과적으로는 둘다 A가 수행됐다.
먼저 각 테이블에서 250 보다 큰 값을 필터링 한 다음 UNION을 하는 것이, 전체 테이블을 UNION 하고 250보다 큰 값을 필터링하는 것보다, 훨씬 효율적이라는 것을 알 수 있다.