

Database HW6 : Practice 5

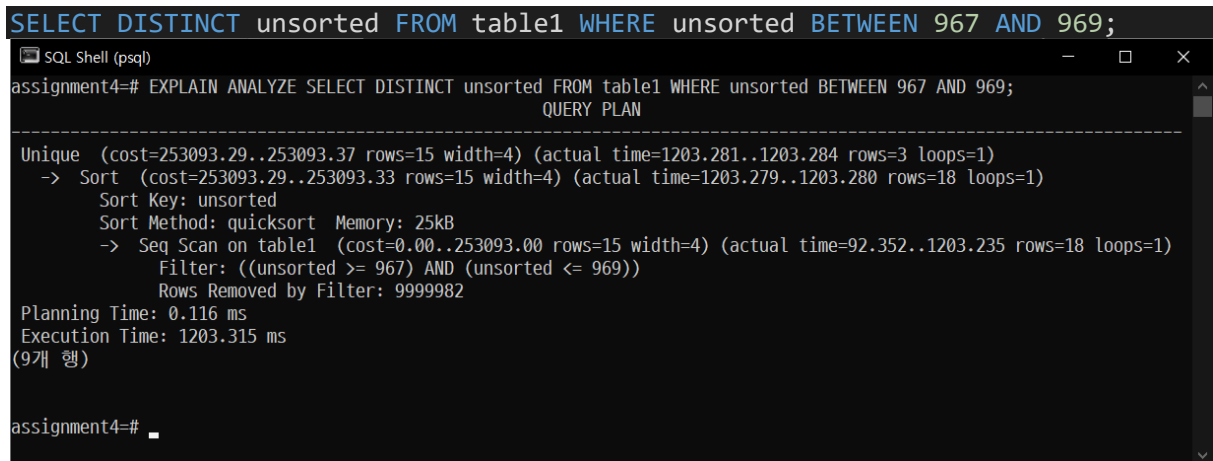
2018320205 신대성

Equivalent SQL Statements

Consider the following query and make corresponding SQL statements

- a. Make an SQL statement using BETWEEN and AND operator

```
SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
```

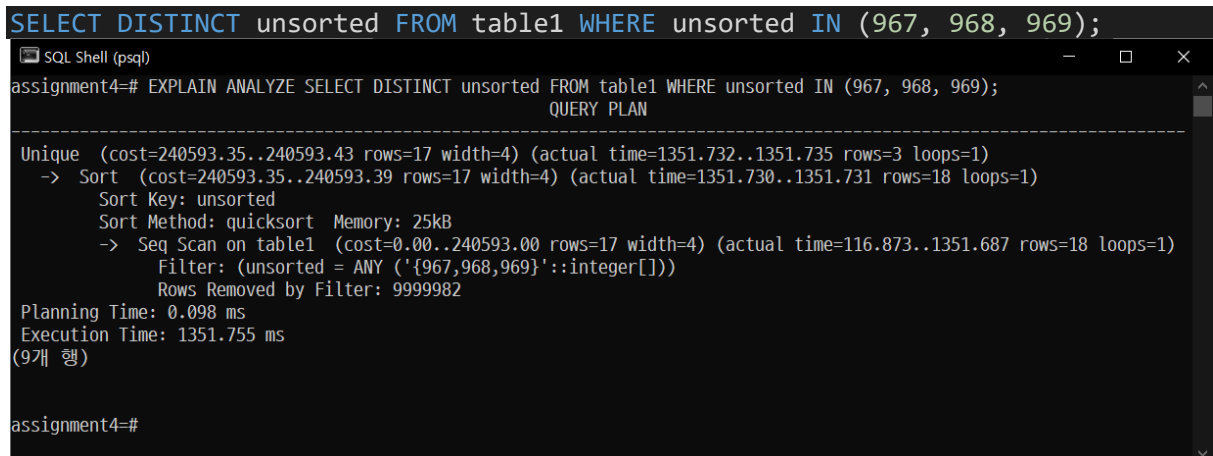


```
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
               QUERY PLAN
-----
Unique  (cost=253093.29..253093.37 rows=15 width=4) (actual time=1203.281..1203.284 rows=3 loops=1)
-> Sort (cost=253093.29..253093.33 rows=15 width=4) (actual time=1203.279..1203.280 rows=18 loops=1)
    Sort Key: unsorted
    Sort Method: quicksort  Memory: 25kB
-> Seq Scan on table1 (cost=0.00..253093.00 rows=15 width=4) (actual time=92.352..1203.235 rows=18 loops=1)
    Filter: ((unsorted >= 967) AND (unsorted <= 969))
    Rows Removed by Filter: 9999982
Planning Time: 0.116 ms
Execution Time: 1203.315 ms
(9개 행)

assignment4=#
```

- b. Make an SQL statement using IN operator

```
SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
```

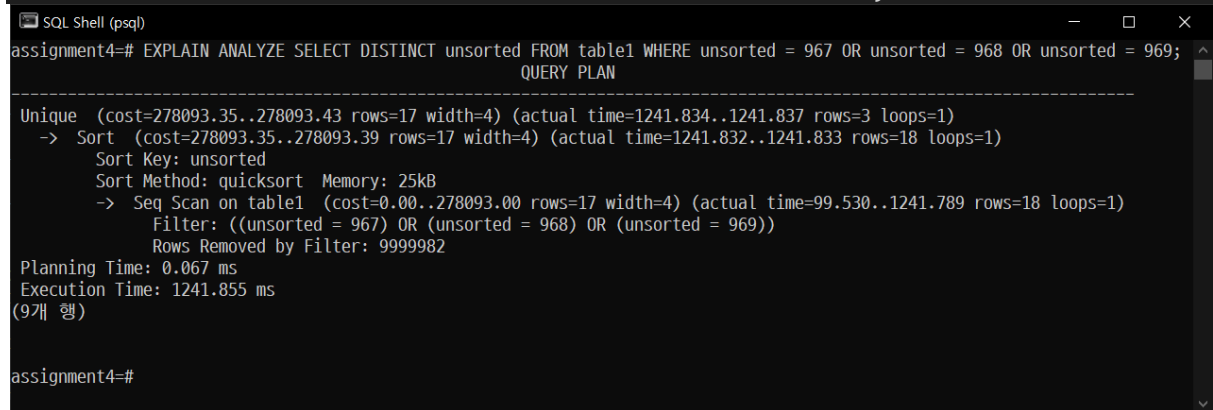


```
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
               QUERY PLAN
-----
Unique  (cost=240593.35..240593.43 rows=17 width=4) (actual time=1351.732..1351.735 rows=3 loops=1)
-> Sort (cost=240593.35..240593.39 rows=17 width=4) (actual time=1351.730..1351.731 rows=18 loops=1)
    Sort Key: unsorted
    Sort Method: quicksort  Memory: 25kB
-> Seq Scan on table1 (cost=0.00..240593.00 rows=17 width=4) (actual time=116.873..1351.687 rows=18 loops=1)
    Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
    Rows Removed by Filter: 9999982
Planning Time: 0.098 ms
Execution Time: 1351.755 ms
(9개 행)

assignment4=#
```

c. Make an SQL statement using = and OR operator

```
SELECT DISTINCT unsorted FROM table1
WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
```

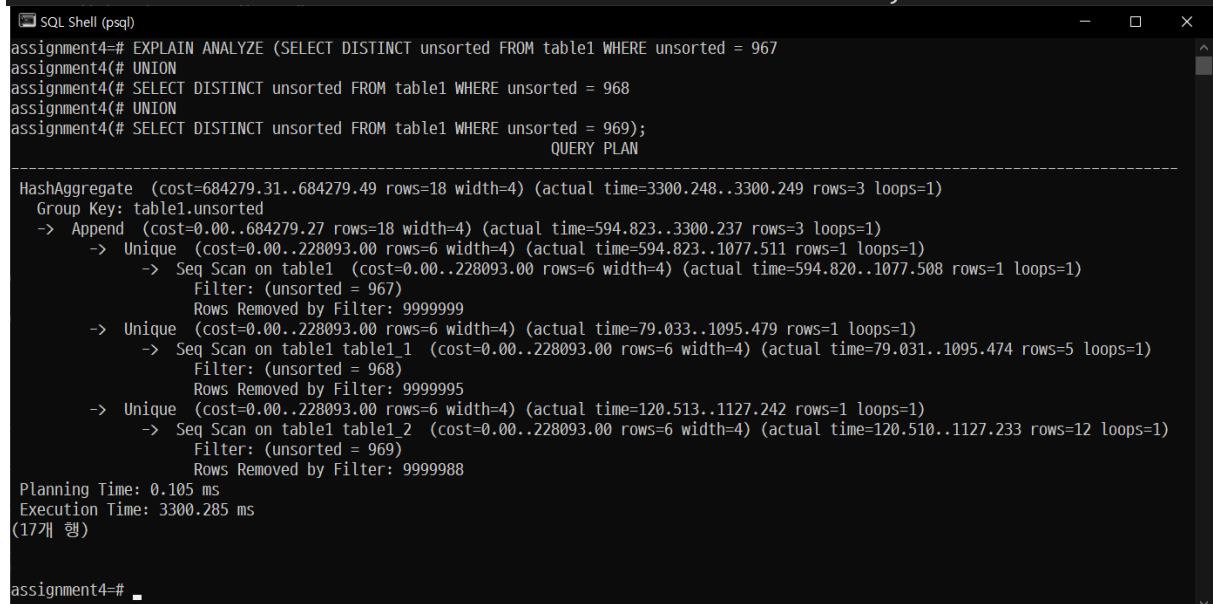


```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
QUERY PLAN
-----
Unique  (cost=278093.35..278093.43 rows=17 width=4) (actual time=1241.834..1241.837 rows=3 loops=1)
  -> Sort (cost=278093.35..278093.39 rows=17 width=4) (actual time=1241.832..1241.833 rows=18 loops=1)
        Sort Key: unsorted
        Sort Method: quicksort  Memory: 25kB
        -> Seq Scan on table1 (cost=0.00..278093.00 rows=17 width=4) (actual time=99.530..1241.789 rows=18 loops=1)
              Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
              Rows Removed by Filter: 9999982
Planning Time: 0.067 ms
Execution Time: 1241.855 ms
(9개 행)

assignment4=#
```

d. Make an SQL statement using UNION operator

```
SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967
UNION
SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 968
UNION
SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 969;
```



```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE (SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 968
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 969);
QUERY PLAN
-----
HashAggregate (cost=684279.31..684279.49 rows=18 width=4) (actual time=3300.248..3300.249 rows=3 loops=1)
  Group Key: table1.unsorted
  -> Append (cost=0.00..684279.27 rows=18 width=4) (actual time=594.823..3300.237 rows=3 loops=1)
        -> Unique (cost=0.00..228093.00 rows=6 width=4) (actual time=594.823..1077.511 rows=1 loops=1)
              -> Seq Scan on table1 (cost=0.00..228093.00 rows=6 width=4) (actual time=594.820..1077.508 rows=1 loops=1)
                    Filter: (unsorted = 967)
                    Rows Removed by Filter: 9999999
        -> Unique (cost=0.00..228093.00 rows=6 width=4) (actual time=79.033..1095.479 rows=1 loops=1)
              -> Seq Scan on table1 table1_1 (cost=0.00..228093.00 rows=6 width=4) (actual time=79.031..1095.474 rows=5 loops=1)
                    Filter: (unsorted = 968)
                    Rows Removed by Filter: 9999995
        -> Unique (cost=0.00..228093.00 rows=6 width=4) (actual time=120.513..1127.242 rows=1 loops=1)
              -> Seq Scan on table1 table1_2 (cost=0.00..228093.00 rows=6 width=4) (actual time=120.510..1127.233 rows=12 loops=1)
                    Filter: (unsorted = 969)
                    Rows Removed by Filter: 9999988
Planning Time: 0.105 ms
Execution Time: 3300.285 ms
(17개 행)

assignment4=#
```

unsorted이기 때문에 sort를 하고 시퀀셜 스캔을 수행하는 것을 확인할 수 있습니다.

a, b, c의 경우는 비슷한 query plan과 수행시간을 보이지만, 마지막의 UNION을 사용하는 query에서는 Seq Scan을 3번 수행하여 더 긴 수행시간을 보입니다.

Create an index on "unsorted" column and repeat the previous exercise

a. btree index

```
CREATE INDEX ON table1 USING btree (unsorted);
```

A. using BETWEEN and AND

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
               QUERY PLAN
-----
Unique  (cost=0.43..64.77 rows=15 width=4) (actual time=0.030..0.244 rows=3 loops=1)
  -> Index Only Scan using table1_unsorted_idx on table1 (cost=0.43..64.73 rows=15 width=4) (actual time=0.029..0.240 rows=18 loops=1)
       Index Cond: ((unsorted >= 967) AND (unsorted <= 969))
       Heap Fetches: 18
Planning Time: 1.422 ms
Execution Time: 0.255 ms
(6개 행)

assignment4=#
```

B. using IN

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
               QUERY PLAN
-----
Unique  (cost=0.43..81.65 rows=17 width=4) (actual time=0.055..0.083 rows=3 loops=1)
  -> Index Only Scan using table1_unsorted_idx on table1 (cost=0.43..81.61 rows=17 width=4) (actual time=0.052..0.074 rows=18 loops=1)
       Index Cond: (unsorted = ANY ('{967,968,969}'::integer[]))
       Heap Fetches: 18
Planning Time: 0.145 ms
Execution Time: 0.111 ms
(6개 행)

assignment4=#
```

C. using = and OR

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
               QUERY PLAN
-----
Unique  (cost=278093.35..278093.43 rows=17 width=4) (actual time=1332.285..1332.289 rows=3 loops=1)
  -> Sort (cost=278093.35..278093.39 rows=17 width=4) (actual time=1332.283..1332.285 rows=18 loops=1)
       Sort Key: unsorted
       Sort Method: quicksort  Memory: 25kB
       -> Seq Scan on table1 (cost=0.00..278093.00 rows=17 width=4) (actual time=114.096..1332.229 rows=18 loops=1)
            Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
            Rows Removed by Filter: 9999982
Planning Time: 0.164 ms
Execution Time: 1332.315 ms
(9개 행)

assignment4=#
```

D. using UNION

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE (SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 968
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 969);
QUERY PLAN
-----
HashAggregate (cost=85.93..86.11 rows=18 width=4) (actual time=0.065..0.066 rows=3 loops=1)
  Group Key: table1.unsorted
  -> Append (cost=0.43..85.89 rows=18 width=4) (actual time=0.021..0.059 rows=3 loops=1)
    -> Unique (cost=0.43..28.54 rows=6 width=4) (actual time=0.021..0.022 rows=1 loops=1)
      -> Index Only Scan using table1_unsorted_idx on table1 (cost=0.43..28.54 rows=6 width=4) (actual time=0.019..0.020 rows=1 loops=1)
        Index Cond: (unsorted = 967)
        Heap Fetches: 1
    -> Unique (cost=0.43..28.54 rows=6 width=4) (actual time=0.010..0.015 rows=1 loops=1)
      -> Index Only Scan using table1_unsorted_idx on table1 table1_1 (cost=0.43..28.54 rows=6 width=4) (actual time=0.009..0.014 rows=5 loops=1)
        Index Cond: (unsorted = 968)
        Heap Fetches: 5
    -> Unique (cost=0.43..28.54 rows=6 width=4) (actual time=0.011..0.022 rows=1 loops=1)
      -> Index Only Scan using table1_unsorted_idx on table1 table1_2 (cost=0.43..28.54 rows=6 width=4) (actual time=0.011..0.021 rows=12 loops=1)
        Index Cond: (unsorted = 969)
        Heap Fetches: 12
Planning Time: 0.228 ms
Execution Time: 0.116 ms
(17개 행)

assignment4=#
```

b. hash index

```
CREATE INDEX ON table1 USING hash (unsorted);
```

A. using BETWEEN and AND

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
QUERY PLAN
-----
Unique (cost=253093.29..253093.37 rows=15 width=4) (actual time=1181.927..1181.932 rows=3 loops=1)
  -> Sort (cost=253093.29..253093.33 rows=15 width=4) (actual time=1181.925..1181.927 rows=18 loops=1)
    Sort Key: unsorted
    Sort Method: quicksort Memory: 25kB
    -> Seq Scan on table1 (cost=0.00..253093.00 rows=15 width=4) (actual time=92.272..1181.879 rows=18 loops=1)
      Filter: ((unsorted >= 967) AND (unsorted <= 969))
      Rows Removed by Filter: 9999982
Planning Time: 2.215 ms
Execution Time: 1181.966 ms
(9개 행)

assignment4=#
```

B. using IN

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
QUERY PLAN
-----
Unique (cost=240593.35..240593.43 rows=17 width=4) (actual time=1373.644..1373.648 rows=3 loops=1)
  -> Sort (cost=240593.35..240593.39 rows=17 width=4) (actual time=1373.642..1373.643 rows=18 loops=1)
    Sort Key: unsorted
    Sort Method: quicksort Memory: 25kB
    -> Seq Scan on table1 (cost=0.00..240593.00 rows=17 width=4) (actual time=106.500..1373.594 rows=18 loops=1)
      Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
      Rows Removed by Filter: 9999982
Planning Time: 0.115 ms
Execution Time: 1373.668 ms
(9개 행)

assignment4=#
```

C. using = and OR

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
QUERY PLAN
-----
Unique  (cost=278093.35..278093.43 rows=17 width=4) (actual time=1313.159..1313.163 rows=3 loops=1)
-> Sort  (cost=278093.35..278093.39 rows=17 width=4) (actual time=1313.158..1313.159 rows=18 loops=1)
    Sort Key: unsorted
    Sort Method: quicksort  Memory: 25kB
-> Seq Scan on table1  (cost=0.00..278093.00 rows=17 width=4) (actual time=101.354..1313.103 rows=18 loops=1)
    Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
    Rows Removed by Filter: 9999982
Planning Time: 0.118 ms
Execution Time: 1313.187 ms
(9개 행)

assignment4=#
```

D. using UNION

```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE (SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 968
assignment4=# UNION
assignment4=# SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 969);
QUERY PLAN
-----
HashAggregate  (cost=84.63..84.81 rows=18 width=4) (actual time=0.523..0.524 rows=3 loops=1)
  Group Key: table1.unsorted
-> Append  (cost=0.00..84.59 rows=18 width=4) (actual time=0.126..0.518 rows=3 loops=1)
-> Unique  (cost=0.00..28.11 rows=6 width=4) (actual time=0.126..0.127 rows=1 loops=1)
-> Index Scan using table1_unsorted_idx on table1  (cost=0.00..28.11 rows=6 width=4) (actual time=0.123..0.124 rows=1 loops=1)
    Index Cond: (unsorted = 967)
-> Unique  (cost=0.00..28.11 rows=6 width=4) (actual time=0.055..0.140 rows=1 loops=1)
-> Index Scan using table1_unsorted_idx on table1 table1_1  (cost=0.00..28.11 rows=6 width=4) (actual time=0.055..0.138 rows=5 loops=1)
    Index Cond: (unsorted = 968)
-> Unique  (cost=0.00..28.11 rows=6 width=4) (actual time=0.026..0.249 rows=1 loops=1)
-> Index Scan using table1_unsorted_idx on table1 table1_2  (cost=0.00..28.11 rows=6 width=4) (actual time=0.025..0.248 rows=12 loops=1)
    Index Cond: (unsorted = 969)
Planning Time: 0.210 ms
Execution Time: 0.581 ms
(14개 행)

assignment4=#
```

인덱스가 없는 경우에는 모든 스캔을 시퀀셜 스캔으로 수행하며, UNION을 사용한 query 외에는 비슷한 query plan과 수행시간을 보여줍니다.

Btree 인덱스를 사용하는 경우는 =과 OR를 사용하는 경우에서만 시퀀셜 스캔을 수행하여 시간이 오래 걸리나, 나머지에서는 인덱스 온리 스캔을 수행하여 수행시간이 상대적으로 짧습니다.

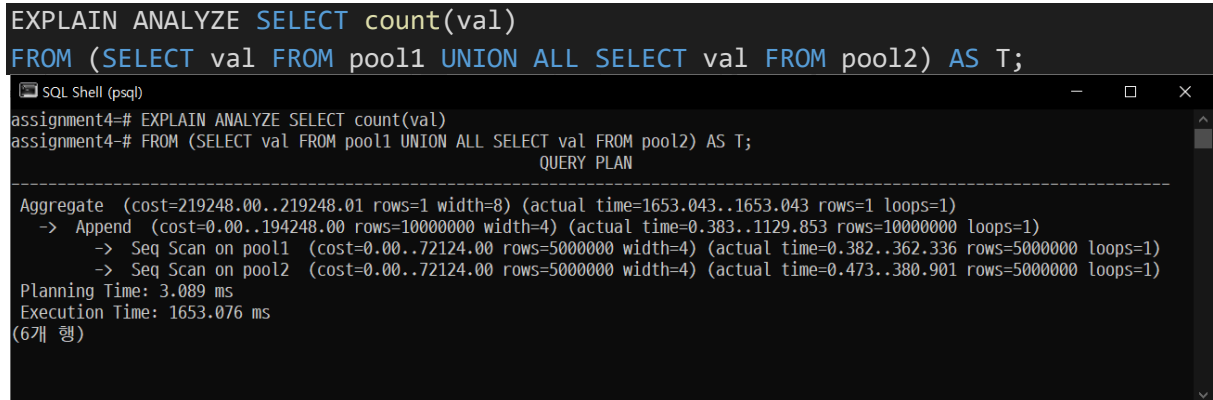
Hash 인덱스를 사용하면 UNION query를 제외한 모든 쿼리에서 시퀀셜 스캔을 사용합니다. 정확하게 한 경우를 결정하는 걸 3번 반복하는 UNION query에서는 인덱스 스캔을 사용해 짧은 수행 시간을 보여줍니다.

Query Plan

Following queries have different syntax but return same result.

- a. Union tables(pool1, pool2), and then perform aggregation with COUNT function

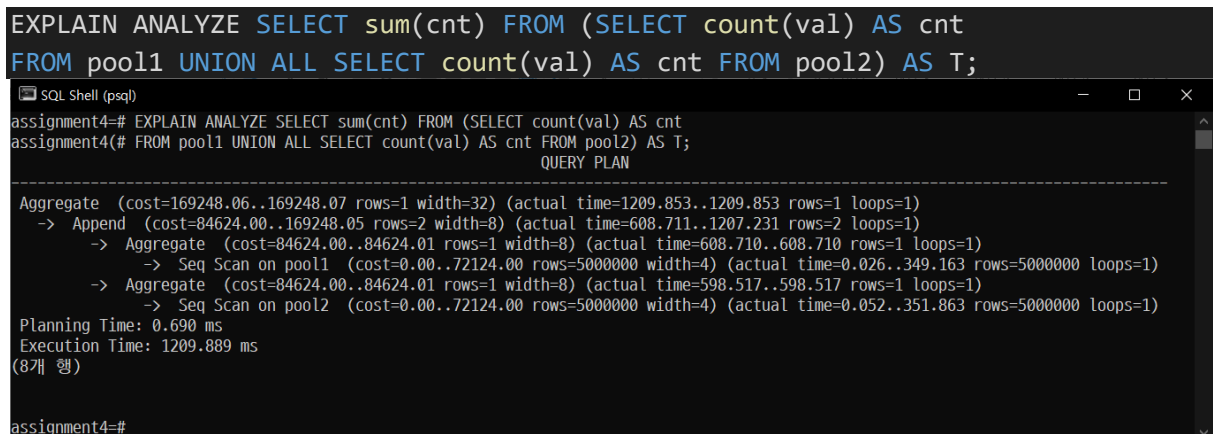
```
EXPLAIN ANALYZE SELECT count(val)
FROM (SELECT val FROM pool1 UNION ALL SELECT val FROM pool2) AS T;
```



```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT count(val)
assignment4=# FROM (SELECT val FROM pool1 UNION ALL SELECT val FROM pool2) AS T;
               QUERY PLAN
-----
Aggregate  (cost=219248.00..219248.01 rows=1 width=8) (actual time=1653.043..1653.043 rows=1 loops=1)
-> Append  (cost=0.00..194248.00 rows=10000000 width=4) (actual time=0.383..1129.853 rows=10000000 loops=1)
-> Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.382..362.336 rows=5000000 loops=1)
-> Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.473..380.901 rows=5000000 loops=1)
Planning Time: 3.089 ms
Execution Time: 1653.076 ms
(6개 행)
```

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

```
EXPLAIN ANALYZE SELECT sum(cnt) FROM (SELECT count(val) AS cnt
FROM pool1 UNION ALL SELECT count(val) AS cnt FROM pool2) AS T;
```



```
SQL Shell (psql)
assignment4=# EXPLAIN ANALYZE SELECT sum(cnt) FROM (SELECT count(val) AS cnt
assignment4=# FROM pool1 UNION ALL SELECT count(val) AS cnt FROM pool2) AS T;
               QUERY PLAN
-----
Aggregate  (cost=169248.06..169248.07 rows=1 width=32) (actual time=1209.853..1209.853 rows=1 loops=1)
-> Append  (cost=84624.00..169248.05 rows=2 width=8) (actual time=608.711..1207.231 rows=2 loops=1)
-> Aggregate (cost=84624.00..84624.01 rows=1 width=8) (actual time=608.710..608.710 rows=1 loops=1)
-> Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.026..349.163 rows=5000000 loops=1)
-> Aggregate (cost=84624.00..84624.01 rows=1 width=8) (actual time=598.517..598.517 rows=1 loops=1)
-> Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.052..351.863 rows=5000000 loops=1)
Planning Time: 0.690 ms
Execution Time: 1209.889 ms
(8개 행)

assignment4=#
```

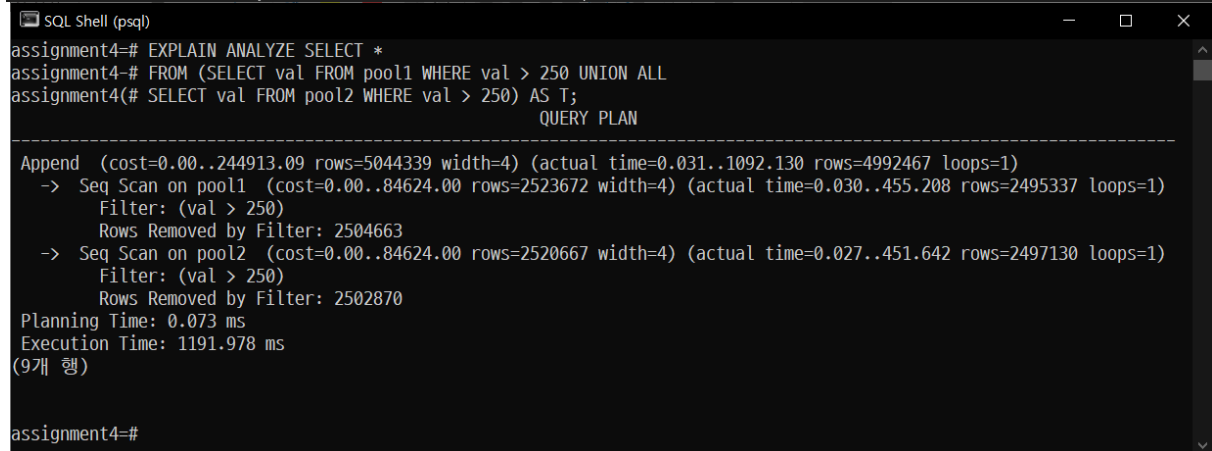
첫번째 쿼리는 50만개의 테이블 두개를 UNION 후 나온 100만개를 카운트하고, 두번째 쿼리는 50만개의 테이블 두개를 카운트하여 나온 2개의 값을 UNION한 뒤 더합니다.

첫번째가 더 연산량이 많기 때문에 더 긴 실행시간을 보여줍니다.

Following queries also return same result but can be written in different ways.

- a. SELECT tuple WHERE value is above 250 on each table and then union them

```
EXPLAIN ANALYZE SELECT *
FROM (SELECT val FROM pool1 WHERE val > 250 UNION ALL
SELECT val FROM pool2 WHERE val > 250) AS T;
```

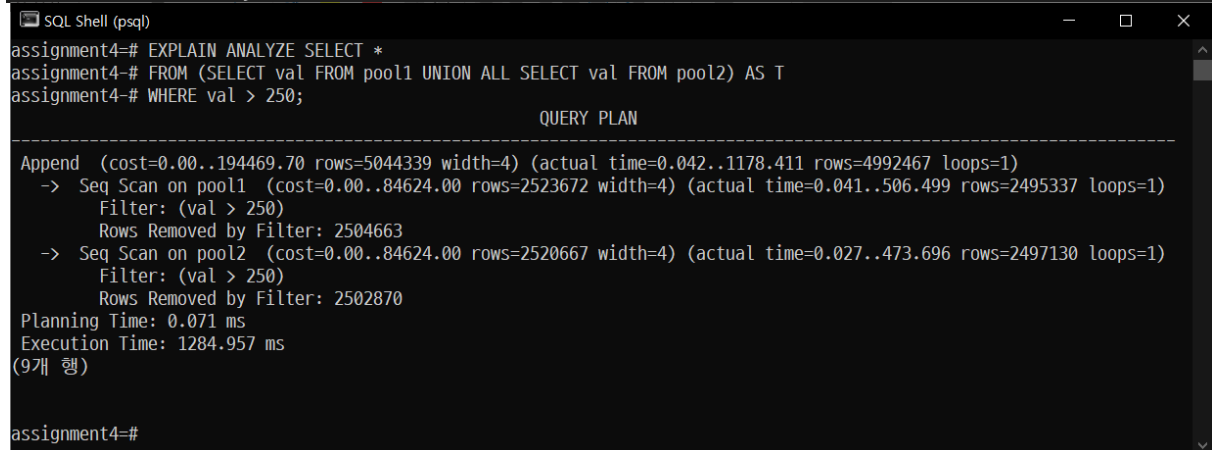


```
assignment4=# EXPLAIN ANALYZE SELECT *
assignment4=# FROM (SELECT val FROM pool1 WHERE val > 250 UNION ALL
assignment4=# SELECT val FROM pool2 WHERE val > 250) AS T;
               QUERY PLAN
-----
 Append  (cost=0.00..244913.09 rows=5044339 width=4) (actual time=0.031..1092.130 rows=4992467 loops=1)
   -> Seq Scan on pool1  (cost=0.00..84624.00 rows=2523672 width=4) (actual time=0.030..455.208 rows=2495337 loops=1)
        Filter: (val > 250)
        Rows Removed by Filter: 2504663
   -> Seq Scan on pool2  (cost=0.00..84624.00 rows=2520667 width=4) (actual time=0.027..451.642 rows=2497130 loops=1)
        Filter: (val > 250)
        Rows Removed by Filter: 2502870
 Planning Time: 0.073 ms
 Execution Time: 1191.978 ms
(9개 행)

assignment4=#
```

- b. Union two tables and SELECT tuples WHERE value is above 250

```
EXPLAIN ANALYZE SELECT *
FROM (SELECT val FROM pool1 UNION ALL SELECT val FROM pool2) AS T
WHERE val > 250;
```



```
assignment4=# EXPLAIN ANALYZE SELECT *
assignment4=# FROM (SELECT val FROM pool1 UNION ALL SELECT val FROM pool2) AS T
assignment4=# WHERE val > 250;
               QUERY PLAN
-----
 Append  (cost=0.00..194469.70 rows=5044339 width=4) (actual time=0.042..1178.411 rows=4992467 loops=1)
   -> Seq Scan on pool1  (cost=0.00..84624.00 rows=2523672 width=4) (actual time=0.041..506.499 rows=2495337 loops=1)
        Filter: (val > 250)
        Rows Removed by Filter: 2504663
   -> Seq Scan on pool2  (cost=0.00..84624.00 rows=2520667 width=4) (actual time=0.027..473.696 rows=2497130 loops=1)
        Filter: (val > 250)
        Rows Removed by Filter: 2502870
 Planning Time: 0.071 ms
 Execution Time: 1284.957 ms
(9개 행)

assignment4=#
```

둘은 같은 query plan을 보여준다.

Why does the user-level optimization important?

일부 쿼리는 query optimization에 의해 가장 효율적인 쿼리 플랜을 찾으나, 같은 결과를 내더라도 효율적인 플랜이 다른 경우도 있고, 테이블에 가지고 있는 인덱스에 따라 동작하기 적합한 쿼리가 있기 때문이다.