2015131406 박가은

1)

```
postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=253093.35..253093.43 rows=17 width=4) (actual time=8168.974..8168.991 rows=3 loops=1)
   ->  Sort  (cost=253093.35..253093.39 rows=17 width=4) (actual time=8168.969..8168.974 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..253093.00 rows=17 width=4) (actual time=518.904..8168.782 rows=18 loops=1)
               Filter: ((unsorted >= 967) AND (unsorted <= 969))
               Rows Removed by Filter: 9999982
 Planning Time: 0.143 ms
 Execution Time: 8169.056 ms
(9개 행)


postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=240593.38..240593.47 rows=18 width=4) (actual time=8753.408..8753.425 rows=3 loops=1)
   ->  Sort  (cost=240593.38..240593.42 rows=18 width=4) (actual time=8753.403..8753.409 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..240593.00 rows=18 width=4) (actual time=575.764..8753.226 rows=18 loops=1)
               Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
               Rows Removed by Filter: 9999982
 Planning Time: 0.269 ms
 Execution Time: 8753.472 ms
(9개 행)

postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=278093.38..278093.47 rows=18 width=4) (actual time=8250.574..8250.589 rows=3 loops=1)
   ->  Sort  (cost=278093.38..278093.42 rows=18 width=4) (actual time=8250.570..8250.574 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..278093.00 rows=18 width=4) (actual time=640.627..8250.415 rows=18 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
               Rows Removed by Filter: 9999982
 Planning Time: 0.121 ms
 Execution Time: 8250.675 ms
(9개 행)


postgres=# EXPLAIN ANALYZE SELECT unsorted FROM table1 WHERE unsorted = 967 UNION SELECT unsorted FROM table1 WHERE unsorted = 968 UNION SELECT unsorted FROM table1 WHERE unsorted = 969;
                                                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=684279.31..684279.49 rows=18 width=4) (actual time=21650.588..21650.590 rows=3 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.00..684279.27 rows=18 width=4) (actual time=3860.854..21650.330 rows=18 loops=1)
         ->  Seq Scan on table1  (cost=0.00..228093.00 rows=6 width=4) (actual time=3860.853..6992.272 rows=1 loops=1)
               Filter: (unsorted = 967)
               Rows Removed by Filter: 9999999
         ->  Seq Scan on table1 table1_1  (cost=0.00..228093.00 rows=6 width=4) (actual time=450.441..7444.905 rows=5 loops=1)
               Filter: (unsorted = 968)
               Rows Removed by Filter: 9999995
         ->  Seq Scan on table1 table1_2  (cost=0.00..228093.00 rows=6 width=4) (actual time=728.815..7213.104 rows=12 loops=1)
               Filter: (unsorted = 969)
               Rows Removed by Filter: 9999988
 Planning Time: 0.198 ms
 Execution Time: 21650.657 ms
(14개 행)
```

모두 동일한 tuple을 result로 가지는 equivalent expressions이지만 실행 시간은 조금씩 차이가 난다. (execution time 기준) 하지만 union을 사용했을 때 시간이 2배 이상 길게 걸린 것을 확인할 수 있다.

2)

```
postgres=# CREATE INDEX unsorted_index ON table1 (unsorted);
CREATE INDEX
postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=0.43..72.82 rows=17 width=4) (actual time=0.072..0.525 rows=3 loops=1)
   ->  Index Only Scan using unsorted_index on table1  (cost=0.43..72.77 rows=17 width=4) (actual time=0.071..0.513 rows=18 loops=1)
         Index Cond: ((unsorted >= 967) AND (unsorted <= 969))
         Heap Fetches: 18
 Planning Time: 2.106 ms
 Execution Time: 0.556 ms
(6개 행)


postgres=#  EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=0.43..85.66 rows=18 width=4) (actual time=0.069..0.099 rows=3 loops=1)
   ->  Index Only Scan using unsorted_index on table1  (cost=0.43..85.62 rows=18 width=4) (actual time=0.066..0.093 rows=18 loops=1)
         Index Cond: (unsorted = ANY ('{967,968,969}'::integer[]))
         Heap Fetches: 18
 Planning Time: 0.183 ms
 Execution Time: 0.123 ms
(6개 행)


postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=278093.38..278093.47 rows=18 width=4) (actual time=4634.767..4634.776 rows=3 loops=1)
   ->  Sort  (cost=278093.38..278093.42 rows=18 width=4) (actual time=4634.764..4634.767 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..278093.00 rows=18 width=4) (actual time=519.074..4634.660 rows=18 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
               Rows Removed by Filter: 9999982
 Planning Time: 0.123 ms
 Execution Time: 4634.808 ms
(9개 행)
```

```
postgres=# EXPLAIN ANALYZE SELECT unsorted FROM table1 WHERE unsorted = 967 UNION SELECT unsorted FROM table1 WHERE unsorted = 968 UNION SELECT unsorted FROM table1 WHERE unsorted = 969;
                                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=85.93..86.11 rows=18 width=4) (actual time=0.153..0.155 rows=3 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.43..85.89 rows=18 width=4) (actual time=0.052..0.129 rows=18 loops=1)
         ->  Index Only Scan using unsorted_index on table1  (cost=0.43..28.54 rows=6 width=4) (actual time=0.051..0.053 rows=1 loops=1)
               Index Cond: (unsorted = 967)
               Heap Fetches: 1
         ->  Index Only Scan using unsorted_index on table1 table1_1  (cost=0.43..28.54 rows=6 width=4) (actual time=0.021..0.032 rows=5 loops=1)
               Index Cond: (unsorted = 968)
               Heap Fetches: 5
         ->  Index Only Scan using unsorted_index on table1 table1_2  (cost=0.43..28.54 rows=6 width=4) (actual time=0.016..0.038 rows=12 loops=1)
               Index Cond: (unsorted = 969)
               Heap Fetches: 12
 Planning Time: 0.293 ms
 Execution Time: 0.238 ms
(14개 행)
```

여기서는 or을 사용했을 때 시간이 훨씬 오래 걸린 것을 확인할 수 있다.

3)

```
postgres=# CREATE INDEX unsorted_index ON table1 USING hash(unsorted);
CREATE INDEX
postgres=# SELECT * FROM pg_indexes WHERE tablename = 'table1';
 schemaname | tablename |    indexname    | tablespace |                                 indexdef
------------+-----------+-----------------+------------+---------------------------------------------------------------------------
 public     | table1    | unsorted_index |            | CREATE INDEX unsorted_index ON public.table1 USING hash (unsorted)
(1개 행)


postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted BETWEEN 967 AND 969;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=253093.35..253093.43 rows=17 width=4) (actual time=4470.977..4470.986 rows=3 loops=1)
   ->  Sort  (cost=253093.35..253093.39 rows=17 width=4) (actual time=4470.974..4470.977 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..253093.00 rows=17 width=4) (actual time=293.096..4470.860 rows=18 loops=1)
               Filter: ((unsorted >= 967) AND (unsorted <= 969))
               Rows Removed by Filter: 9999982
 Planning Time: 1.771 ms
 Execution Time: 4471.018 ms
(9개 행)


postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted IN (967, 968, 969);
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=240593.38..240593.47 rows=18 width=4) (actual time=5099.947..5099.960 rows=3 loops=1)
   ->  Sort  (cost=240593.38..240593.42 rows=18 width=4) (actual time=5099.943..5099.948 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..240593.00 rows=18 width=4) (actual time=510.307..5099.835 rows=18 loops=1)
               Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
               Rows Removed by Filter: 9999982
 Planning Time: 0.110 ms
 Execution Time: 5099.994 ms
(9개 행)
```

```
postgres=# EXPLAIN ANALYZE SELECT DISTINCT unsorted FROM table1 WHERE unsorted = 967 OR unsorted = 968 OR unsorted = 969;
                                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=278093.38..278093.47 rows=18 width=4) (actual time=4784.585..4784.594 rows=3 loops=1)
   ->  Sort  (cost=278093.38..278093.42 rows=18 width=4) (actual time=4784.582..4784.585 rows=18 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..278093.00 rows=18 width=4) (actual time=358.857..4784.475 rows=18 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
               Rows Removed by Filter: 9999982
 Planning Time: 0.132 ms
 Execution Time: 4784.627 ms
(9개 행)


postgres=# EXPLAIN ANALYZE SELECT unsorted FROM table1 WHERE unsorted = 967 UNION SELECT unsorted FROM table1 WHERE unsorted = 968 UNION SELECT unsorted FROM table1 WHERE unsorted = 969;
                                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=84.63..84.81 rows=18 width=4) (actual time=0.108..0.109 rows=3 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.00..84.59 rows=18 width=4) (actual time=0.026..0.095 rows=18 loops=1)
         ->  Index Scan using unsorted_index on table1  (cost=0.00..28.11 rows=6 width=4) (actual time=0.025..0.027 rows=1 loops=1)
               Index Cond: (unsorted = 967)
         ->  Index Scan using unsorted_index on table1 table1_1  (cost=0.00..28.11 rows=6 width=4) (actual time=0.011..0.022 rows=5 loops=1)
               Index Cond: (unsorted = 968)
         ->  Index Scan using unsorted_index on table1 table1_2  (cost=0.00..28.11 rows=6 width=4) (actual time=0.010..0.042 rows=12 loops=1)
               Index Cond: (unsorted = 969)
 Planning Time: 0.226 ms
 Execution Time: 0.173 ms
(11개 행)
```

여기서는 union을 사용했을 때 훨씬 효과적으로 원하는 결과를 얻을 수 있다는 것을 알 수 있다.

4) 인덱스를 사용하지 않았을 경우, 전체적인 수행시간은 8100ms이지만 union을 사용했을 때는 시간이 두 배 이상 걸린다. 인덱스를 사용하지 않았기 때문에 모든 블록을 읽으며 조건에 부합하는 레코드를 추출해야 하기 때문에 그런 것이다. 특히 union같은 경우 모든 union되는 조건의 개수만큼 레코드를 계속 읽어야하기 때문에 다른 expression보다 시간이 배로 걸리는 것을 확인할 수 있다.

b-tree 인덱스를 사용했을 경우, 전체적인 시간은 0.5ms이하로 비슷하지만 OR operation을 썼을 때 4600ms, 상대적으로 오랜 시간이 걸린다. OR에서는 실질적으로 조건에 부합하는 데이터를 찾으며 계속 스캔을 해야하지만 union 같은 경우 인덱스를 사용하여 부분처리를 하기 때문에 더 적은 input을 가지고 있지 때문이다.

Hash 인덱스를 사용했을 경우 전체적인 시간은 4000ms에서 5000ms 정도이지만 union을 사용했을 경우 0.17ms 정도의 아주 짧은 시간만 소요된다.

전체적인 수행 시간은 b+-tree가 제일 짧아 효율적이다. 그리고 hash 인덱스로 union을 사용했을 때도 비슷한 효율을 보여준다. 하지만 다른 operation을 수행할 경우에는 매우 비효율적이다. 인덱스가 없을 경우에는 모든 데이터를 디스크에서 계속 읽어야하기 때문에 hash보다 더 비효율적이라고 할 수 있다.

5)

```
postgres=# EXPLAIN ANALYZE SELECT COUNT(*) FROM (SELECT * FROM pool1 UNION ALL SELECT * FROM pool2) AS pool;
                                                     QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=118164.88..118164.89 rows=1 width=8) (actual time=545.688..545.688 rows=1 loops=1)
   ->  Gather  (cost=118164.66..118164.87 rows=2 width=8) (actual time=545.175..571.538 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate  (cost=117164.66..117164.67 rows=1 width=8) (actual time=488.402..488.402 rows=1 loops=3)              ->  Parallel Append  (cost=0.00..106748.00 rows=4166666
 width=0) (actual time=0.182..376.195 rows=3333333 loops=3)
                     ->  Parallel Seq Scan on pool1  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.344..146.883 rows=1666667 loops=3)
                     ->  Parallel Seq Scan on pool2  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.152..174.012 rows=2500000 loops=2)
 Planning Time: 0.166 ms
 Execution Time: 571.613 ms
(10개 행)

postgres=# EXPLAIN ANALYZE SELECT SUM(count1 + count2) FROM (SELECT COUNT(*)AS count1 FROM pool1) AS p1, (SELECT COUNT(*) AS count2 FROM pool2) AS P2;
                                                     QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=98331.82..98331.83 rows=1 width=32) (actual time=460.062..460.062 rows=1 loops=1)
   ->  Nested Loop  (cost=98331.76..98331.81 rows=1 width=16) (actual time=460.056..460.056 rows=1 loops=1)
         ->  Finalize Aggregate  (cost=49165.88..49165.89 rows=1 width=8) (actual time=245.353..245.353 rows=1 loops=1)
               ->  Gather  (cost=49165.67..49165.88 rows=2 width=8) (actual time=244.881..245.403 rows=3 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     ->  Partial Aggregate  (cost=48165.67..48165.68 rows=1 width=8) (actual time=188.615..188.616 rows=1 loops=3)
                           ->  Parallel Seq Scan on pool1  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.153..134.697 rows=1666667 loops=3)
         ->  Finalize Aggregate  (cost=49165.88..49165.89 rows=1 width=8) (actual time=214.699..214.700 rows=1 loops=1)
               ->  Gather  (cost=49165.67..49165.88 rows=2 width=8) (actual time=214.225..241.373 rows=3 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     ->  Partial Aggregate  (cost=48165.67..48165.68 rows=1 width=8) (actual time=157.015..157.015 rows=1 loops=3)
                           ->  Parallel Seq Scan on pool2  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.206..104.253 rows=1666667 loops=3)
 Planning Time: 0.168 ms
 Execution Time: 486.934 ms
(16개 행)
```

큰 차이는 존재하지 않는다. 테이블 2개를 UNION ALL 한 뒤에 COUNT한 것은 571ms, 테이블 2개의 크기를 각각 COUNT한 뒤에 그 둘의 SUM을 구한 것은 486ms이다. 인풋의 크기가 작을수록 수행 시간이 짧게 걸린다. 즉, 두 테이블을 UNION ALL 하게 되면 input이 커져 연산하는 시간이 오래 걸리고 그에 반해 테이블의 행을 각각 COUNT한 뒤에 둘을 더하는 연산은 상대적으로 input이 작기 때문에 짧은 실행 시간이 소요된다.

6)

```
^Cpostgres=# EXPLAIN ANALYZE SELECT val FROM pool1 WHERE val > 250 UNION ALL SELECT val FROM pool2 WHERE val > 250;
                                                     QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..244293.69 rows=5003046 width=4) (actual time=0.016..998.484 rows=4990595 loops=1)
   ->  Seq Scan on pool1  (cost=0.00..84624.00 rows=2492000 width=4) (actual time=0.014..457.099 rows=2495728 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2504272
   ->  Seq Scan on pool2  (cost=0.00..84624.00 rows=2511046 width=4) (actual time=0.021..373.490 rows=2494867 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2505133
 Planning Time: 0.106 ms
 Execution Time: 1089.029 ms
(9개 행)
```

```
postgres=# EXPLAIN ANALYZE SELECT val FROM (SELECT * FROM pool1 UNION ALL SELECT * FROM pool2) AS p WHERE val > 250;
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..194263.23 rows=5003046 width=4) (actual time=0.028..959.959 rows=4990595 loops=1)
   ->  Seq Scan on pool1  (cost=0.00..84624.00 rows=2492000 width=4) (actual time=0.027..437.685 rows=2495728 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2504272
   ->  Seq Scan on pool2  (cost=0.00..84624.00 rows=2511046 width=4) (actual time=0.016..356.530 rows=2494867 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2505133
 Planning Time: 0.143 ms
 Execution Time: 1048.907 ms
(9개 행)
```

둘 사이에 큰 차이가 없지만 위의 경우가 시간이 조금 더 오래 걸린다. 그 이유는 각각의 테이블에서 250보다 큰 경우를 추출하고 다시 UNION하는 것은 아래의 경우보다 처리하는 input의 크기가 크기 때문이다.

7) 유저 레벨에서의 최적화가 중요한 이유는 데이터 베이스 시스템의 최적화 수준이 곧 데이터 베이스의 성능을 좌지우지하기 때문이다. 디스크와 메모리의 속도는 하드웨어가 아무리 발전을 해도 CPU의 연산 속도를 절대 따라갈 수 없다. 때문에 데이터베이스로 원하는 데이터를 빠르게 넣고, 수정하고 삭제하고, 찾기 위해서는 하드웨어가 아닌 효율적인 알고리즘을 사용해야 하는 것이다. 만약 데이터베이스의 최적화 수준이 낮아 원하는 데이터를 처리하는데 시간이 오래 걸린다면 사람들은 차라리 수작업으로 문서를 찾고, 수정하며 데이터를 처리할 것이다. 데이터베이스의 최적화 수준이 곧 데이터베이스의 성능을 결정하기 때문에 최적화가 중요한 것이다.