DataBase Assignment

hw#6 - practice #5

정보대학 컴퓨터학과 2017320108 고재영

Consider the following query and make corresponding SQL statements. Select “unsorted” from table1 where the “unsorted” value is 967 or 968 or 969 (967~969)

(Use the DISTINCT for the question a,b,c)

a. Make an SQL statement using BETWEEN and AND operator

b. Make an SQL statement using IN operator

c. Make an SQL statement using = and OR operator

d. Make an SQL statement using UNION operator

(a) between, and

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted between 967 and 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=16.01..16.02 rows=2 width=4) (actual time=0.039..0.039 rows=0 loops=1)
   ->  Sort  (cost=16.01..16.02 rows=2 width=4) (actual time=0.035..0.035 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..16.00 rows=2 width=4) (actual time=0.017..0.017 rows=0 loops=1)
               Filter: ((unsorted >= 967) AND (unsorted <= 969))
 Planning Time: 0.687 ms
 Execution Time: 0.120 ms
(8개 행)
```

(b) in

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted in (967, 968, 969);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=15.58..15.61 rows=6 width=4) (actual time=0.021..0.021 rows=0 loops=1)
   ->  Sort  (cost=15.58..15.59 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..15.50 rows=6 width=4) (actual time=0.014..0.014 rows=0 loops=1)
               Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
 Planning Time: 0.142 ms
 Execution Time: 0.043 ms
(8개 행)
```

(c) =, or

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted = 967 or unsorted = 968 or unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=17.08..17.11 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
   ->  Sort  (cost=17.08..17.09 rows=6 width=4) (actual time=0.019..0.020 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..17.00 rows=6 width=4) (actual time=0.014..0.014 rows=0 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
 Planning Time: 0.107 ms
 Execution Time: 0.043 ms
(8개 행)
```

(d) union

```
hw6=# explain analyze
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 967
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 968
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=45.11..45.17 rows=6 width=4) (actual time=0.032..0.032 rows=0 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.00..45.09 rows=6 width=4) (actual time=0.031..0.032 rows=0 loops=1)
         ->  Seq Scan on table1  (cost=0.00..15.00 rows=2 width=4) (actual time=0.013..0.013 rows=0 loops=1)
               Filter: (unsorted = 967)
         ->  Seq Scan on table1 table1_1  (cost=0.00..15.00 rows=2 width=4) (actual time=0.009..0.009 rows=0 loops=1)
               Filter: (unsorted = 968)
         ->  Seq Scan on table1 table1_2  (cost=0.00..15.00 rows=2 width=4) (actual time=0.009..0.009 rows=0 loops=1)
               Filter: (unsorted = 969)
 Planning Time: 0.157 ms
 Execution Time: 0.088 ms
(11개 행)
```

Execute your SQL statements and discuss the results of the queries.

수행시간을 보면,
(b), (c) < (d) < (a)의 결과를 보이며, 소요시간 면에서 양적으로 1배, 2배, 3배 차이를 보인다.

(a)의 경우, between을 이용하여 다루고, sorting하는 데에 시간이 많이 걸리기에 다른 방법보다 훨씬 상이하게 긴 시간이 소요된다는 것을 알 수 있다.

(b)의 경우, in이란 절 안에서 한 번에 3가지의 unsorted value를 다루기 때문에 rows = 6을 통해 한 번에 6개의 row를 다루어 상대적으로 앞의 (a)보다 약 3배 이상 빠른 시간을 보인다.

(c)는 (b)와 거의 동일한 수치를 보이는데, 아마 이것또한 or를 이용하여 세 가지 unsorted value를 다뤄 6개의 row를 한 번에 다루기 때문인 것으로 보인다.

마지막으로, (d)의 경우는 query plan을 보더라도 앞선 3가지 경우와는 상이한 과정을 보이는데, (d)의 경우는 set operation union을 통해, 2개 row를 다루는 과정을 개별적으로 3번 반복하기 때문에 b, c보다는 더 긴 소요시간을 보인다.

Create an index on “unsorted” column and repeat the previous exercise

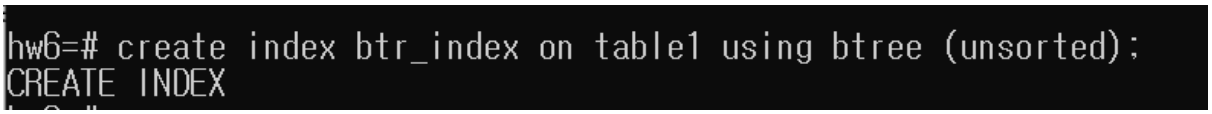
a. Btree index

b. Hash index

▪ Compare each SQL statements’ performances on three cases(no index, Btree index, hash index)

create index btr_index on table1 using btree (unsorted);

create index hsh_index on table1 using hash (unsorted);




```
hw6=# create index btr_index on table1 using btree (unsorted);
CREATE INDEX
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted between 967 and 969;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=0.15..12.19 rows=2 width=4) (actual time=0.009..0.009 rows=0 loops=1)
   ->  Index Only Scan using btr_index on table1  (cost=0.15..12.19 rows=2 width=4) (actual time=0.007..0.007 rows=0 loops=1)
         Index Cond: ((unsorted >= 967) AND (unsorted <= 969))
         Heap Fetches: 0
 Planning Time: 1.694 ms
 Execution Time: 0.062 ms
(6개 행)


hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted in (967, 968, 969);
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=15.58..15.61 rows=6 width=4) (actual time=0.021..0.021 rows=0 loops=1)
   ->  Sort  (cost=15.58..15.59 rows=6 width=4) (actual time=0.021..0.021 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..15.50 rows=6 width=4) (actual time=0.014..0.014 rows=0 loops=1)
               Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
 Planning Time: 0.154 ms
 Execution Time: 0.042 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted = 967 or unsorted = 968 or unsorted = 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=17.08..17.11 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
   ->  Sort  (cost=17.08..17.09 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..17.00 rows=6 width=4) (actual time=0.014..0.014 rows=0 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
 Planning Time: 0.143 ms
 Execution Time: 0.046 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 967
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 968
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 969;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=36.65..36.71 rows=6 width=4) (actual time=0.037..0.037 rows=0 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.15..36.64 rows=6 width=4) (actual time=0.036..0.036 rows=0 loops=1)
         ->  Index Only Scan using btr_index on table1  (cost=0.15..12.18 rows=2 width=4) (actual time=0.009..0.009 rows=0 loops=1)
               Index Cond: (unsorted = 967)
               Heap Fetches: 0
         ->  Index Only Scan using btr_index on table1 table1_1  (cost=0.15..12.18 rows=2 width=4) (actual time=0.016..0.016 rows=0 loops=1)
               Index Cond: (unsorted = 968)
               Heap Fetches: 0
         ->  Index Only Scan using btr_index on table1 table1_2  (cost=0.15..12.18 rows=2 width=4) (actual time=0.009..0.009 rows=0 loops=1)
               Index Cond: (unsorted = 969)
               Heap Fetches: 0
 Planning Time: 0.217 ms
 Execution Time: 0.095 ms
(14개 행)
```

//hash index 삽입

```
hw6=# drop index btr_index;
DROP INDEX
hw6=# create index hash_index on table1 using hash (unsorted);
CREATE INDEX
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted between 967 and 969;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=16.01..16.02 rows=2 width=4) (actual time=0.021..0.021 rows=0 loops=1)
   ->  Sort  (cost=16.01..16.02 rows=2 width=4) (actual time=0.021..0.021 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..16.00 rows=2 width=4) (actual time=0.016..0.016 rows=0 loops=1)
               Filter: ((unsorted >= 967) AND (unsorted <= 969))
 Planning Time: 1.549 ms
 Execution Time: 0.044 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted in (967, 968, 969);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Unique  (cost=15.58..15.61 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
   ->  Sort  (cost=15.58..15.59 rows=6 width=4) (actual time=0.020..0.020 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..15.50 rows=6 width=4) (actual time=0.015..0.015 rows=0 loops=1)
               Filter: (unsorted = ANY ('{967,968,969}'::integer[]))
 Planning Time: 0.119 ms
 Execution Time: 0.041 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# select distinct unsorted
hw6-# from table1
hw6-# where unsorted = 967 or unsorted = 968 or unsorted = 969;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=17.08..17.11 rows=6 width=4) (actual time=0.019..0.019 rows=0 loops=1)
   ->  Sort  (cost=17.08..17.09 rows=6 width=4) (actual time=0.018..0.019 rows=0 loops=1)
         Sort Key: unsorted
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on table1  (cost=0.00..17.00 rows=6 width=4) (actual time=0.013..0.013 rows=0 loops=1)
               Filter: ((unsorted = 967) OR (unsorted = 968) OR (unsorted = 969))
 Planning Time: 0.112 ms
 Execution Time: 0.043 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 967
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 968
hw6-# union
hw6-# select unsorted
hw6-# from table1
hw6-# where unsorted = 969;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=36.21..36.27 rows=6 width=4) (actual time=0.049..0.049 rows=0 loops=1)
   Group Key: table1.unsorted
   ->  Append  (cost=0.00..36.20 rows=6 width=4) (actual time=0.048..0.048 rows=0 loops=1)
         ->  Index Scan using hash_index on table1  (cost=0.00..12.04 rows=2 width=4) (actual time=0.043..0.043 rows=0 loops=1)
               Index Cond: (unsorted = 967)
         ->  Index Scan using hash_index on table1 table1_1  (cost=0.00..12.04 rows=2 width=4) (actual time=0.002..0.002 rows=0 loops=1)
               Index Cond: (unsorted = 968)
         ->  Index Scan using hash_index on table1 table1_2  (cost=0.00..12.04 rows=2 width=4) (actual time=0.002..0.002 rows=0 loops=1)
               Index Cond: (unsorted = 969)
 Planning Time: 0.188 ms
 Execution Time: 0.088 ms
(11개 행)
```

첫 번째 query에 관해서, 실행시간이 no index, btree, hash 순서로 길다. btree의 경우 index only scan을 이용하여 수행한다는 점이 특징이다.

네 번째 query에 관해서, btree는 index only scan을 하는 반면, hash는 index scan을 사용한다.

▪ Create two synthetic data tables that has 5000000 rows with values between 0 and 500

1. CREATE TABLE pool1 (val integer);

2. INSERT INTO pool1(val)

SELECT random()*500

FROM (SELECT generate_series(1,5000000)) AS T;

3. CREATE TABLE pool2 (val integer);

4. INSERT INTO pool2(val)

SELECT random()*500

FROM (SELECT generate_series(1,5000000)) AS T;

```
hw6=# CREATE TABLE pool1 (val integer);
CREATE TABLE
hw6=# INSERT INTO pool1(val)
hw6-# SELECT random()*500
hw6-# FROM (SELECT generate_series(1,5000000)) AS T;
INSERT 0 5000000
hw6=# CREATE TABLE pool2 (val integer);
CREATE TABLE
hw6=# INSERT INTO pool2(val)
hw6-# SELECT random()*500
hw6-# FROM (SELECT generate_series(1,5000000)) AS T;
INSERT 0 5000000
```

▪ Following queries have different syntax but return same result. You have to use UNION ALL operator!(different from Equi-SQL statement problem)

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

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

▪ Write the queries and use EXPLAIN ANALYZE to see how the query execution is actually planned

```
hw6=# explain analyze
hw6-# with uniofpool(value) as
hw6-# (select * from pool1
hw6(# union all
hw6(# select * from pool2)
hw6-# select count(*)
hw6-# from uniofpool;
                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=419248.00..419248.01 rows=1 width=8) (actual time=6971.018..6971.018 rows=1 loops=1)
   CTE uniofpool
     ->  Append  (cost=0.00..194248.00 rows=10000000 width=4) (actual time=0.616..2741.739 rows=10000000 loops=1)
           ->  Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.614..924.571 rows=5000000 loops=1)
           ->  Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.533..842.843 rows=5000000 loops=1)
   ->  CTE Scan on uniofpool  (cost=0.00..200000.00 rows=10000000 width=0) (actual time=0.624..6039.146 rows=10000000 loops=1)
 Planning Time: 8.690 ms
 Execution Time: 7002.447 ms
(8개 행)
```

```
hw6=# explain analyze
hw6-# with cofp1(value) as
hw6-# (select count(*)
hw6(# from pool1),
hw6-# cofp2(value) as
hw6-# (select count(*)
hw6(# from pool2),
hw6-# unionofp(value) as
hw6-# (select *
hw6(# from cofp1
hw6(# union all
hw6(# select *
hw6(# from cofp2)
hw6-# select sum(value)
hw6-# from unionofp;
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=98331.88..98331.89 rows=1 width=32) (actual time=3228.629..3228.629 rows=1 loops=1)
   CTE cofp1
     ->  Finalize Aggregate  (cost=49165.88..49165.89 rows=1 width=8) (actual time=1490.518..1490.518 rows=1 loops=1)
           ->  Gather  (cost=49165.67..49165.88 rows=2 width=8) (actual time=1489.091..1489.672 rows=3 loops=1)
                 Workers Planned: 2
                 Workers Launched: 2
                 ->  Partial Aggregate  (cost=48165.67..48165.68 rows=1 width=8) (actual time=1380.895..1380.895 rows=1 loops=3)
                       ->  Parallel Seq Scan on pool1  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.736..1284.960 rows=1666667 loops=3)
   CTE cofp2
     ->  Finalize Aggregate  (cost=49165.88..49165.89 rows=1 width=8) (actual time=1736.940..1736.940 rows=1 loops=1)
           ->  Gather  (cost=49165.67..49165.88 rows=2 width=8) (actual time=1736.274..1753.987 rows=3 loops=1)
                 Workers Planned: 2
                 Workers Launched: 2
                 ->  Partial Aggregate  (cost=48165.67..48165.68 rows=1 width=8) (actual time=1634.360..1634.360 rows=1 loops=3)
                       ->  Parallel Seq Scan on pool2  (cost=0.00..42957.33 rows=2083333 width=0) (actual time=0.621..1532.356 rows=1666667 loops=3)
   CTE unionofp
     ->  Append  (cost=0.00..0.05 rows=2 width=8) (actual time=1490.521..3227.467 rows=2 loops=1)
           ->  CTE Scan on cofp1  (cost=0.00..0.02 rows=1 width=8) (actual time=1490.520..1490.521 rows=1 loops=1)
           ->  CTE Scan on cofp2  (cost=0.00..0.02 rows=1 width=8) (actual time=1736.942..1736.943 rows=1 loops=1)
   ->  CTE Scan on unionofp  (cost=0.00..0.04 rows=2 width=8) (actual time=1490.525..3227.479 rows=2 loops=1)
 Planning Time: 16.965 ms
 Execution Time: 3249.879 ms
(22개 행)
```

aggregate하는 cost 면에서 전자가 후자에 비해 압도적으로 긴 시간을 가지는데, 전자의 경우에 두 테이블을 유니온 한 후에 count aggregate 작업을 한 반면, 후자의 경우는 각 테이블에 count aggregate을 수행한 이후에 sum으로 다시금 유니온한 결과를 aggregate하는 식이기 때문에 다루는 row 수라든지 비용적인 측면에서 후자가 훨씬 절감한다.

▪ Following queries also return same result but can be written in different ways. You have to use UNION ALL operator!(different from Equi-SQL statement problem)

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

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

▪ Write the queries and use EXPLAIN ANALYZE to see how query execution is actually planned

```
hw6=# explain analyze
hw6-# (select *
hw6(# from pool1
hw6(# where val > 250)
hw6-# union all
hw6-# (select *
hw6(# from pool2
hw6(# where val > 250);
                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..244414.29 rows=5011086 width=4) (actual time=0.043..1289.738 rows=4990169 loops=1)
   ->  Seq Scan on pool1  (cost=0.00..84624.00 rows=2516023 width=4) (actual time=0.042..539.978 rows=2494532 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2505468
   ->  Seq Scan on pool2  (cost=0.00..84624.00 rows=2495063 width=4) (actual time=0.036..531.155 rows=2495637 loops=1)
         Filter: (val > 250)
         Rows Removed by Filter: 2504363
 Planning Time: 6.851 ms
 Execution Time: 1406.672 ms
(9개 행)


hw6=# explain analyze
hw6-# with uniofpool(value) as
hw6-# (select *
hw6(# from pool1
hw6(# union all
hw6(# select *
hw6(# from pool2)
hw6-# select *
hw6-# from uniofpool
hw6-# where value > 250;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 CTE Scan on uniofpool  (cost=194248.00..419248.00 rows=3333333 width=4) (actual time=0.096..3240.807 rows=4990169 loops=1)
   Filter: (value > 250)
   Rows Removed by Filter: 5009831
   CTE uniofpool
     ->  Append  (cost=0.00..194248.00 rows=10000000 width=4) (actual time=0.091..1342.038 rows=10000000 loops=1)
           ->  Seq Scan on pool1  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.090..447.253 rows=5000000 loops=1)
           ->  Seq Scan on pool2  (cost=0.00..72124.00 rows=5000000 width=4) (actual time=0.067..447.718 rows=5000000 loops=1)
 Planning Time: 0.139 ms
 Execution Time: 3398.927 ms
(9개 행)
```

정제하지 않은 상태로 두 테이블을 먼저 유니온 한 후에 where 절의 조건으로 정제를 한 후자의 경우가 당연하게도 전자에 비해 압도적으로 긴 수행시간을 가진다.

▪ Why does the user-level optimization important?

user-level에서의 optimization 과정에 따라, 같은 결과를 도출하는 작업을 수행하더라도, 하드웨어나 메모리 사용량 등 리소스를 절약함과 수행시간을 줄여 더 빠르게 결과를 얻을 수 있기 때문에 몹시 중요하다.