ASSIGNMENT 7 – EXPERIMENTS PDF

Submitted by:
Team 18
Team Members:

1. **Aditi Soni**
2. **Sravya Arutla**
3. **Kiran Gurunath Naik**
4. **Aqsa Bhimdiwala**

PROBLEM 1:

Consider the relation schema R(a int, b int) and the 'NO' generalized quantifier query: {(r1.a, r2.a) | R(r1) ∧ R(r2) ∧ R(r1.a) ∩ R(r2.a) = ∅} where R(r1.a) = {r.b | R(r) ∧ r.a = r1.a} R(r2.a) = {r.b | R(r) ∧ r.a = r2.a}. Consider Lecture 19 'Query Processing and Query Plans' and in particular the analysis, using query plans, for the 'SOME' generalized quantifier. In analogy with that analysis, do an analysis for the 'NO' generalized quantifier.1

SOLUTION:

RA Query:

```
with rY as (select r1.a, array_agg(r2.b) as y from R r1 natural join R r2 group by(r1.a)),
    sZ as (select r3.a, array_agg(r4.b) as y from R r3 natural join R r4 group by(r3.a))
select z1.* from (
select rY.a, sZ.a from rY natural join sZ
except
select rY.a, sZ.a from rY join sZ on (rY.y &&  sZ.y)
)z1;
```

Hash Join and merge join =off

R	RA query (ms)
100	7.159
1000	188.97
10000	18682.68

Hash join and merge join =ON

R	RA query (ms)
100	5.214

1000	74.52
10000	4712.68

Analysis: When the hash join and merge join is OFF, then the table with 100000 data takes more time as there will be sequential loop runs. We can check this in the query plans given below. The time taken to run without hash and merge join is more than the one without.

Query plan for the RA query when the merge join and hash join is been set off

"Subquery Scan on z1 (cost=358.74..359.79 rows=1 width=8) (actual time=10.895..10.899 rows=0 loops=1)"
" CTE ry"
"   -> GroupAggregate (cost=179.29..179.37 rows=4 width=36) (actual time=3.817..3.916 rows=63 loops=1)"
"   Group Key: r1.a"
"   -> Sort (cost=179.29..179.30 rows=4 width=8) (actual time=3.804..3.817 rows=100 loops=1)"
"   Sort Key: r1.a"
"   Sort Method: quicksort Memory: 29kB"
"   -> Nested Loop (cost=0.00..179.25 rows=4 width=8) (actual time=0.048..3.134 rows=100 loops=1)"
"   Join Filter: ((r1.a = r2.a) AND (r1.b = r2.b))"
"   Rows Removed by Join Filter: 9900"
"   -> Seq Scan on r r1 (cost=0.00..2.00 rows=100 width=8) (actual time=0.021..0.037 rows=100 loops=1)"
"   -> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.012 rows=100 loops=100)"
"   -> Seq Scan on r r2 (cost=0.00..2.00 rows=100 width=8) (actual time=0.009..0.037 rows=100 loops=1)"
" CTE sz"
"   -> GroupAggregate (cost=179.29..179.37 rows=4 width=36) (actual time=3.102..3.205 rows=63 loops=1)"
"   Group Key: r3.a"
"   -> Sort (cost=179.29..179.30 rows=4 width=8) (actual time=3.097..3.110 rows=100 loops=1)"
"   Sort Key: r3.a"
"   Sort Method: quicksort Memory: 29kB"
"   -> Nested Loop (cost=0.00..179.25 rows=4 width=8) (actual time=0.028..3.036 rows=100 loops=1)"
"   Join Filter: ((r3.a = r4.a) AND (r3.b = r4.b))"

"   Rows Removed by Join Filter: 9900"
"   -> Seq Scan on r r3 (cost=0.00..2.00 rows=100 width=8) (actual time=0.009..0.025 rows=100 loops=1)"
"   -> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.012 rows=100 loops=100)"
"   -> Seq Scan on r r4 (cost=0.00..2.00 rows=100 width=8) (actual time=0.007..0.033 rows=100 loops=1)"
" -> HashSetOp Except (cost=0.00..1.04 rows=1 width=12) (actual time=10.895..10.896 rows=0 loops=1)"
"   -> Append (cost=0.00..1.03 rows=2 width=12) (actual time=6.930..10.820 rows=232 loops=1)"
"   -> Subquery Scan on ""*SELECT* 1"" (cost=0.00..0.53 rows=1 width=12) (actual time=6.929..8.713 rows=63 loops=1)"
"   -> Nested Loop (cost=0.00..0.52 rows=1 width=8) (actual time=6.928..8.698 rows=63 loops=1)"
"   Join Filter: ((ry.a = sz.a) AND (ry.y = sz.y))"
"   Rows Removed by Join Filter: 3906"
"   -> CTE Scan on ry (cost=0.00..0.08 rows=4 width=36) (actual time=3.819..3.953 rows=63 loops=1)"
"   -> CTE Scan on sz (cost=0.00..0.08 rows=4 width=36) (actual time=0.049..0.063 rows=63 loops=63)"
"   -> Subquery Scan on ""*SELECT* 2"" (cost=0.00..0.49 rows=1 width=12) (actual time=0.004..2.074 rows=169 loops=1)"
"   -> Nested Loop (cost=0.00..0.48 rows=1 width=8) (actual time=0.003..2.045 rows=169 loops=1)"
"   Join Filter: (ry_1.y && sz_1.y)"
"   Rows Removed by Join Filter: 3800"
"   -> CTE Scan on ry ry_1 (cost=0.00..0.08 rows=4 width=36) (actual time=0.000..0.010 rows=63 loops=1)"
"   -> CTE Scan on sz sz_1 (cost=0.00..0.08 rows=4 width=36) (actual time=0.000..0.009 rows=63 loops=63)"

Query Plan for the RA query when hash join and merge join is set ON

"Subquery Scan on z1 (cost=14.46..15.11 rows=1 width=8) (actual time=4.438..4.445 rows=0 loops=1)"
" CTE ry"
"   -> GroupAggregate (cost=7.08..7.16 rows=4 width=36) (actual time=0.483..0.878 rows=63 loops=1)"
"   Group Key: r1.a"
"   -> Sort (cost=7.08..7.09 rows=4 width=8) (actual time=0.476..0.484 rows=100 loops=1)"
"   Sort Key: r1.a"
"   Sort Method: quicksort Memory: 29kB"

```
"            ->  Hash Join  (cost=3.50..7.04 rows=4 width=8) (actual time=0.414..0.452
rows=100 loops=1)"
"                  Hash Cond: ((r1.a = r2.a) AND (r1.b = r2.b))"
"                  ->  Seq Scan on r r1  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.017..0.025 rows=100 loops=1)"
"                  ->  Hash  (cost=2.00..2.00 rows=100 width=8) (actual time=0.388..0.388
rows=100 loops=1)"
"                        Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"                        ->  Seq Scan on r r2  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.005..0.018 rows=100 loops=1)"
"  CTE sz"
"    ->  GroupAggregate  (cost=7.08..7.16 rows=4 width=36) (actual time=0.256..0.309
rows=63 loops=1)"
"          Group Key: r3.a"
"          ->  Sort  (cost=7.08..7.09 rows=4 width=8) (actual time=0.247..0.254 rows=100
loops=1)"
"                Sort Key: r3.a"
"                Sort Method: quicksort  Memory: 29kB"
"                ->  Hash Join  (cost=3.50..7.04 rows=4 width=8) (actual time=0.195..0.234
rows=100 loops=1)"
"                      Hash Cond: ((r3.a = r4.a) AND (r3.b = r4.b))"
"                      ->  Seq Scan on r r3  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.143..0.151 rows=100 loops=1)"
"                      ->  Hash  (cost=2.00..2.00 rows=100 width=8) (actual time=0.039..0.039
rows=100 loops=1)"
"                            Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"                            ->  Seq Scan on r r4  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.004..0.017 rows=100 loops=1)"
"  ->  HashSetOp Except  (cost=0.14..0.78 rows=1 width=12) (actual time=4.438..4.440
rows=0 loops=1)"
"        ->  Append  (cost=0.14..0.77 rows=2 width=12) (actual time=1.229..4.270 rows=232
loops=1)"
"              ->  Subquery Scan on ""*SELECT* 1""  (cost=0.14..0.27 rows=1 width=12) (actual
time=1.229..1.680 rows=63 loops=1)"
"                    ->  Hash Join  (cost=0.14..0.26 rows=1 width=8) (actual time=1.228..1.672
rows=63 loops=1)"
"                          Hash Cond: ((ry.a = sz.a) AND (ry.y = sz.y))"
"                          ->  CTE Scan on ry  (cost=0.00..0.08 rows=4 width=36) (actual
time=0.485..0.895 rows=63 loops=1)"
"                          ->  Hash  (cost=0.08..0.08 rows=4 width=36) (actual time=0.736..0.737
rows=63 loops=1)"
"                                Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"                                ->  CTE Scan on sz  (cost=0.00..0.08 rows=4 width=36) (actual
time=0.257..0.326 rows=63 loops=1)"
```

```
"          ->  Subquery Scan on ""*SELECT* 2""  (cost=0.00..0.49 rows=1 width=12) (actual
time=0.004..2.571 rows=169 loops=1)"
"                ->  Nested Loop  (cost=0.00..0.48 rows=1 width=8) (actual time=0.004..2.551
rows=169 loops=1)"
"                      Join Filter: (ry_1.y && sz_1.y)"
"                      Rows Removed by Join Filter: 3800"
"                      ->  CTE Scan on ry ry_1  (cost=0.00..0.08 rows=4 width=36) (actual
time=0.000..0.007 rows=63 loops=1)"
"                      ->  CTE Scan on sz sz_1  (cost=0.00..0.08 rows=4 width=36) (actual
time=0.000..0.016 rows=63 loops=63)"
```

PROBLEM 2:

Consider the relation schema R(a, b) and the 'NOT ONLY' generalized quan tifier query: {(r1.a, r2.a) | R(r1) ∧ R(r2) ∧ R(r1.a) 6⊆ R(r2.a)} where R(r1.a) = {r.b | R(r) ∧ r.a = r1.a} R(r2.a) = {r.b | R(r) ∧ r.a = r2.a}. Consider Lecture 19 'Query Processing and Query Plans' and in particular the analysis, using query plans, for the 'SOME' generalized quantifier. In analogy with that analysis, do an analysis for the 'NOT ONLY' generalized quantifier.

SOLUTION:

```
with R3 as (select r1.a, array_agg(r2.b) as bs from R r1 natural join R r2 group by (r1.a)),
    R4 as (select r1.a, array_agg(r2.b) as bs from R r1 natural join R r2 group by (r1.a))
select r5.a , r6.a
from R3 r5, R4 r6
where not (r5.bs <@ (select bs from R4 where a = r5.a) );
```

Query Plan for HASH and MERGE =OFF

```
"Nested Loop  (cost=178919.22..179498.10 rows=16256 width=8) (actual time=5.748..5.753
rows=0 loops=1)"
"  CTE r4"
"    ->  GroupAggregate  (cost=89458.33..89460.89 rows=128 width=36) (actual
time=3.006..3.112 rows=60 loops=1)"
"          Group Key: r1_1.a"
"          ->  Sort  (cost=89458.33..89458.65 rows=128 width=8) (actual time=2.989..3.003
rows=100 loops=1)"
"                Sort Key: r1_1.a"
"                Sort Method: quicksort  Memory: 29kB"
"                ->  Nested Loop  (cost=0.00..89453.85 rows=128 width=8) (actual
time=0.052..2.953 rows=100 loops=1)"
"                      Join Filter: ((r1_1.a = r2_1.a) AND (r1_1.b = r2_1.b))"
"                      Rows Removed by Join Filter: 9900"
"                      ->  Seq Scan on r r1_1  (cost=0.00..32.60 rows=2260 width=8) (actual
time=0.033..0.050 rows=100 loops=1)"
```

```
"                  ->  Materialize  (cost=0.00..43.90 rows=2260 width=8) (actual
time=0.000..0.011 rows=100 loops=100)"
"                        ->  Seq Scan on r r2_1  (cost=0.00..32.60 rows=2260 width=8) (actual
time=0.008..0.027 rows=100 loops=1)"
"  ->  CTE Scan on r4 r6  (cost=0.00..2.56 rows=128 width=4) (actual time=3.008..3.021
rows=60 loops=1)"
"  ->  Materialize  (cost=89458.33..89831.77 rows=127 width=4) (actual time=0.045..0.045
rows=0 loops=60)"
"        ->  Subquery Scan on r5  (cost=89458.33..89831.13 rows=127 width=4) (actual
time=2.676..2.678 rows=0 loops=1)"
"              Filter: (NOT (r5.bs <@ (SubPlan 2)))"
"              Rows Removed by Filter: 60"
"              ->  GroupAggregate  (cost=89458.33..89460.89 rows=128 width=36) (actual
time=1.590..1.699 rows=60 loops=1)"
"                    Group Key: r1.a"
"                    ->  Sort  (cost=89458.33..89458.65 rows=128 width=8) (actual
time=1.580..1.594 rows=100 loops=1)"
"                          Sort Key: r1.a"
"                          Sort Method: quicksort  Memory: 29kB"
"                          ->  Nested Loop  (cost=0.00..89453.85 rows=128 width=8) (actual
time=0.021..1.552 rows=100 loops=1)"
"                                Join Filter: ((r1.a = r2.a) AND (r1.b = r2.b))"
"                                Rows Removed by Join Filter: 9900"
"                                ->  Seq Scan on r r1  (cost=0.00..32.60 rows=2260 width=8) (actual
time=0.011..0.020 rows=100 loops=1)"
"                                ->  Materialize  (cost=0.00..43.90 rows=2260 width=8) (actual
time=0.000..0.006 rows=100 loops=100)"
"                                      ->  Seq Scan on r r2  (cost=0.00..32.60 rows=2260 width=8) (actual
time=0.004..0.016 rows=100 loops=1)"
"              SubPlan 2"
"                ->  CTE Scan on r4  (cost=0.00..2.88 rows=1 width=32) (actual time=0.007..0.015
rows=1 loops=60)"
"                      Filter: (a = r5.a)"
"                      Rows Removed by Filter: 59"
"Planning Time: 0.671 ms"
"Execution Time: 70.725 ms"
```

Query Plan for Hash and merge=ON

```
"Nested Loop  (cost=13.74..14.52 rows=16 width=8) (actual time=1.282..1.287 rows=0
loops=1)"
"  CTE r4"
"    ->  GroupAggregate  (cost=6.83..6.91 rows=4 width=36) (actual time=0.140..0.225
rows=60 loops=1)"
```

```
"        Group Key: r1_1.a"
"        ->  Sort  (cost=6.83..6.84 rows=4 width=8) (actual time=0.132..0.145 rows=100
loops=1)"
"              Sort Key: r1_1.a"
"              Sort Method: quicksort  Memory: 29kB"
"              ->  Hash Join  (cost=3.50..6.79 rows=4 width=8) (actual time=0.066..0.104
rows=100 loops=1)"
"                    Hash Cond: ((r1_1.a = r2_1.a) AND (r1_1.b = r2_1.b))"
"                    ->  Seq Scan on r r1_1  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.015..0.023 rows=100 loops=1)"
"                    ->  Hash  (cost=2.00..2.00 rows=100 width=8) (actual time=0.039..0.040
rows=100 loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"                          ->  Seq Scan on r r2_1  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.004..0.018 rows=100 loops=1)"
"  ->  CTE Scan on r4 r6  (cost=0.00..0.08 rows=4 width=4) (actual time=0.141..0.153 rows=60
loops=1)"
"  ->  Materialize  (cost=6.83..7.34 rows=4 width=4) (actual time=0.019..0.019 rows=0
loops=60)"
"        ->  Subquery Scan on r5  (cost=6.83..7.32 rows=4 width=4) (actual time=1.105..1.106
rows=0 loops=1)"
"              Filter: (NOT (r5.bs <@ (SubPlan 2)))"
"              Rows Removed by Filter: 60"
"              ->  GroupAggregate  (cost=6.83..6.91 rows=4 width=36) (actual time=0.098..0.198
rows=60 loops=1)"
"                    Group Key: r1.a"
"                    ->  Sort  (cost=6.83..6.84 rows=4 width=8) (actual time=0.096..0.109 rows=100
loops=1)"
"                          Sort Key: r1.a"
"                          Sort Method: quicksort  Memory: 29kB"
"                          ->  Hash Join  (cost=3.50..6.79 rows=4 width=8) (actual time=0.042..0.080
rows=100 loops=1)"
"                                Hash Cond: ((r1.a = r2.a) AND (r1.b = r2.b))"
"                                ->  Seq Scan on r r1  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.003..0.011 rows=100 loops=1)"
"                                ->  Hash  (cost=2.00..2.00 rows=100 width=8) (actual time=0.032..0.032
rows=100 loops=1)"
"                                      Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"                                      ->  Seq Scan on r r2  (cost=0.00..2.00 rows=100 width=8) (actual
time=0.004..0.015 rows=100 loops=1)"
"              SubPlan 2"
"                ->  CTE Scan on r4  (cost=0.00..0.09 rows=1 width=32) (actual time=0.006..0.014
rows=1 loops=60)"
"                    Filter: (a = r5.a)"
```

" Rows Removed by Filter: 59"
"Planning Time: 0.561 ms"
"Execution Time: 1.416 ms"

Analysis: When the hash join and merge join is OFF, then the table with 100000 data takes more time as there will be sequential loop runs. We can check this in the query plans given below. The time taken to run without hash and merge join is more than the one without.

Given that the Problem 1 and Problem2 gives the empty subset, there is the difference in the execution timings

NOTE: All the results from 3-6 have been taken with

```
set enable_hashjoin = on;
set enable_mergejoin = off;
```

PROBLEM 3:

a) Translate and optimize Q3 query and call it Q4. Express Q4 as an RA SQL query just as was done for query Q2 in Example 1. Compare queries Q3 and Q4 in a similar way as we did for Q1 and Q2 in Example 1.

SOLUTION:

ORIGINAL QUERY: Q3

```
select distinct r1.a
from R r1,
    R r2,
    R r3,
    R r4
where r1.b = r2.a
    and r2.b = r3.a
    and r3.b = r4.a;
```

OPTIMISED QUERY: Q4

```
select distinct r1.a
from R r1
    natural join (
        select r2.a as b
        from R r2
            natural join (
                select r3.a as b
                from R r3
                    natural join (
```

```
                    select r4.a as b
                    from R r4
                ) r4
            ) r3
    ) r2
```

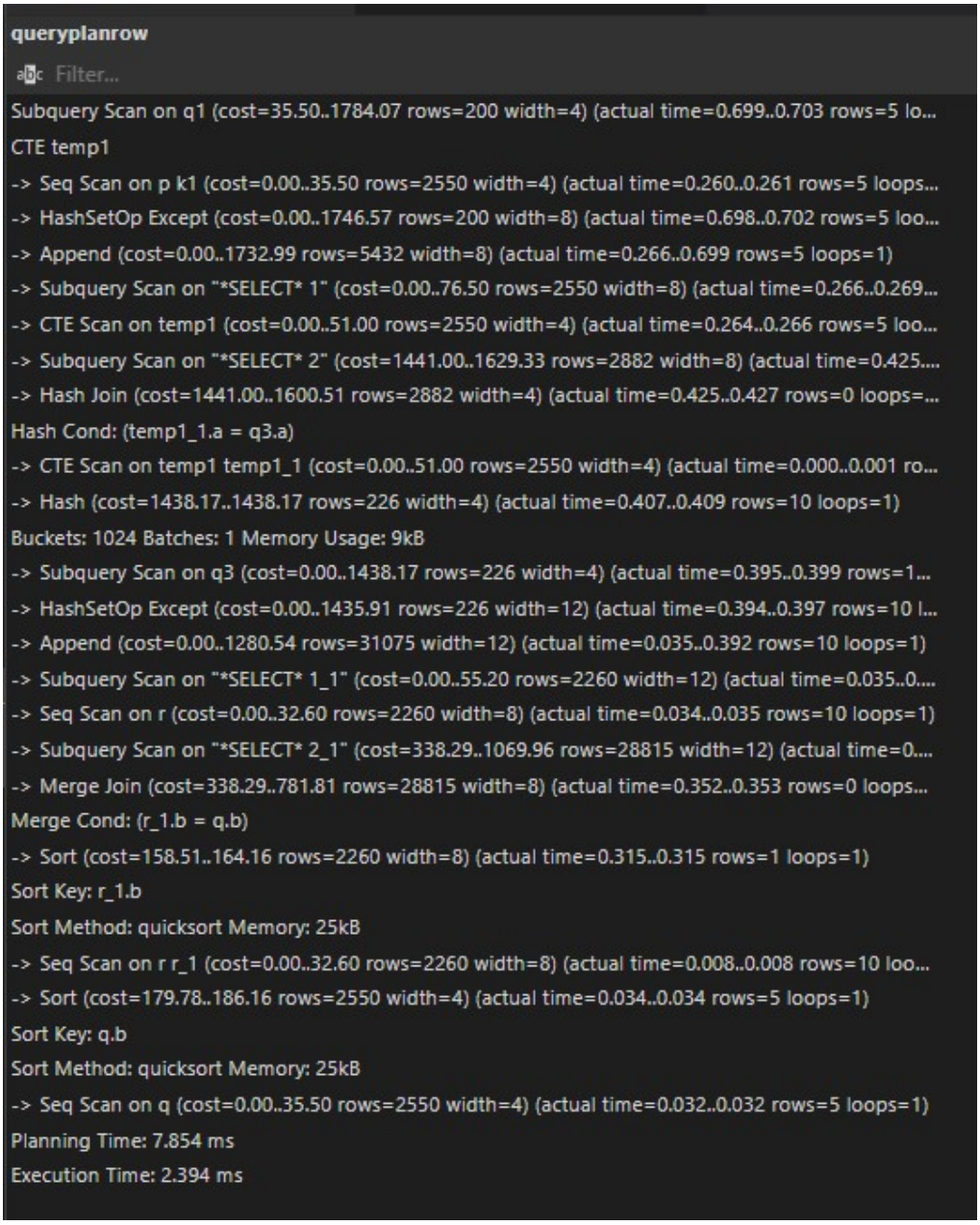
Query Plans for Original Query: Q3

queryplanrow

Filter...

```
HashAggregate (cost=62355.87..62357.87 rows=200 width=4) (actual time=0.153..0.155 rows=0 lo...
Group Key: r1.a
Batches: 1 Memory Usage: 40kB
-> Merge Join (cost=5161.62..54203.51 rows=3260943 width=4) (actual time=0.151..0.153 rows=0 l...
Merge Cond: (r2.b = r3.a)
-> Sort (cost=2580.81..2644.65 rows=25538 width=8) (actual time=0.150..0.151 rows=0 loops=1)
Sort Key: r2.b
Sort Method: quicksort Memory: 25kB
-> Merge Join (cost=317.01..711.38 rows=25538 width=8) (actual time=0.062..0.063 rows=0 loops...
Merge Cond: (r1.b = r2.a)
-> Sort (cost=158.51..164.16 rows=2260 width=8) (actual time=0.044..0.044 rows=8 loops=1)
Sort Key: r1.b
Sort Method: quicksort Memory: 25kB
-> Seq Scan on r r1 (cost=0.00..32.60 rows=2260 width=8) (actual time=0.029..0.030 rows=10 loop...
-> Sort (cost=158.51..164.16 rows=2260 width=8) (actual time=0.013..0.014 rows=10 loops=1)
Sort Key: r2.a
Sort Method: quicksort Memory: 25kB
-> Seq Scan on r r2 (cost=0.00..32.60 rows=2260 width=8) (actual time=0.006..0.006 rows=10 loop...
-> Sort (cost=2580.81..2644.65 rows=25538 width=4) (never executed)
Sort Key: r3.a
-> Merge Join (cost=317.01..711.38 rows=25538 width=4) (never executed)
Merge Cond: (r3.b = r4.a)
-> Sort (cost=158.51..164.16 rows=2260 width=8) (never executed)
Sort Key: r3.b
-> Seq Scan on r r3 (cost=0.00..32.60 rows=2260 width=8) (never executed)
-> Sort (cost=158.51..164.16 rows=2260 width=4) (never executed)
Sort Key: r4.a
-> Seq Scan on r r4 (cost=0.00..32.60 rows=2260 width=4) (never executed)
Planning Time: 6.057 ms
Execution Time: 1.245 ms
```

Query Plans for Optimised Query: Q4



Case 1: With nP = 100 and nQ = 100 and Varying R rows.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
100	100	10	0.362	0.054
100	100	100	0.042	0.047
100	100	1000	0.046	0.034
100	100	10000	0.040	0.035
100	100	100000	0.037	0.056
100	100	1000000	0.054	0.044

(b) What conclusions do you draw from the results of these experiments regarding the effectiveness of query optimization in PostgreSQL and/or by hand?

SOLUTION:

In the above case, as the R value increases, the execution time for the optimised SQL is less than the Original Query. Hence Optimised Query has the better performance than the pure SQL.

Q3 Complexity: $O|R|^4$
Q4 Complexity: $O(|R|+|R|+|R|+|R|)$

PROBLEM 4:

a) Translate and optimize Q3 query and call it Q4. Express Q4 as an RA SQL query just as was done for query Q2 in Example 1. Compare queries Q3 and Q4 in a similar way as we did for Q1 and Q2 in Example 1.

SOLUTION:

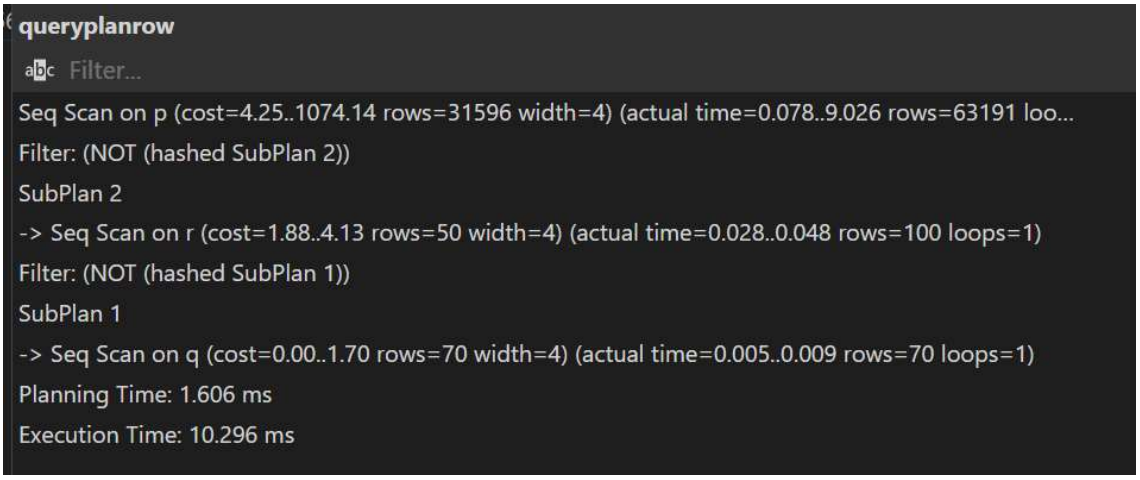
ORIGINAL QUERY: Q4

```
select p.a
from P p
where p.a not in (select r.a
from R r
where r.b not in (select q.b
from Q q));
```

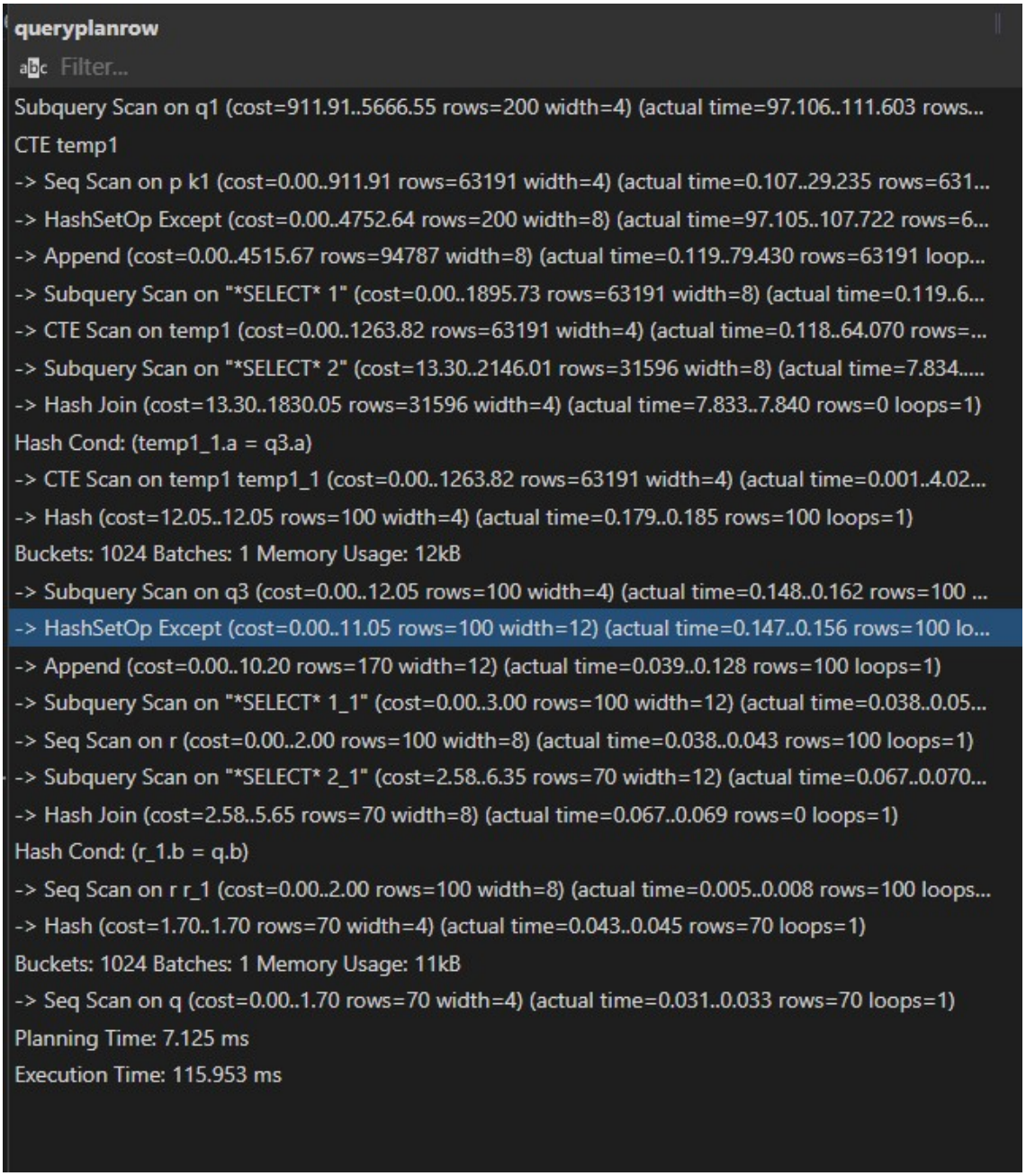
OPTIMISED QUERY: Q5

```
with temp1 as (select k1.* from P k1),
     temp2 as (select q3.a from (
                         select r.*
                         from R r
                         except
                         select r.*
                         from R r NATURAL join Q q ) q3)
select q1.a from (
select * from temp1
except
select * from
temp1 NATURAL join (select * from temp2 ) k ) q1
```

QUERY PLANS for Original Query: Q4



Query Plans for Optimised Query: Q5



The above queries have been experimented with different P, Q and R rows in the tables respectively to test the execution times.

Case 1: Keeping P (nP = 100) and Q (nQ = 100) in the same range and Varying the R rows as shown below.

np	nq	nr	e1	e2
100	100	10	0.050	0.126
100	100	100	0.037	0.085
100	100	1000	0.087	0.368
100	100	10000	0.782	3.852
100	100	100000	5.526	47.194

Case 2: Keeping R (nR = 100) and Q (nQ = 100) with the same rows and Varying the P rows as shown below.

local-host: select * from Ca... | local-host: select * from Ca... | local-host: select * from Ca...

np	nq	nr	e1	e2
10	100	100	0.156	0.225
100	100	100	0.131	0.201
1000	100	100	0.223	0.596
10000	100	100	0.828	4.170
100000	100	100	8.799	58.717
1000000	100	100	149.757	1249.463
10000000	100	100	1207.325	22539.764

Case 3:

Keeping R (nP = 100) with the same rows and Varying the P and R rows as shown in the Figure below.

np	nq	nr	e1	e2
10	10	100	23.497	185.507
100	100	100	32.586	172.869
1000	1000	100	34.782	215.654
10000	10000	100	38.755	251.087
100000	100000	100	51.943	247.825
1000000	1000000	100	12620.112	658.962

(b) What conclusions do you draw from the results of these experiments regarding the effectiveness of query optimization in PostgreSQL and/or by hand?

SOLUTION:

- In case 1, the P and Q rows are kept constant whereas R values have been increased to observe the execution times for the Original Query and Optimised query. It is seen that for the fewer rows of the R data, the execution times of both queries are same or slightly varied, but when the R has the huge values, the execution time of the RA SQL is more than the Pure SQL.
- In Case 2, The R and Q tables have been kept constant by increasing the rows for the P table. It is seen from that, similar results as in case 1. With the huge data, the execution times for the RA SQL is more than the Pure SQL. The performance of the Pure SQL is better with the huge data.
- In Case 3, R size is kept constant and the rows of P and Q are varied. It is seen that, the execution time for Optimised SQL was higher in the initial cases and when both P and Q are very large, Execution time of the optimised SQL is lesser compared to the original SQL.

Q3 Complexity: O (|R| +|P| +|Q|)

Q4 Complexity: O(|R|*|P*|Q|)

PROBLEM 5:

(a) Translate and optimize query Q5 and call it Q6. Express Q6 as an RA SQL query just as was done for Q2 in Example 1. Compare queries Q5 and Q6 in a similar way as we did in Example 1. However, now you should experiment with different sizes or properties for P, R and Q. You might consider how P and Q interact with R.

SOLUTION:

ORIGINAL QUERY: Q5

```
select p.a
from P p
where true = all (
        select p.a != r.a
            or true = some (
                select r.b = q.b
                from Q q
            )
        from R r
    )
order by 1;
```

OPTIMISED QUERY: Q6

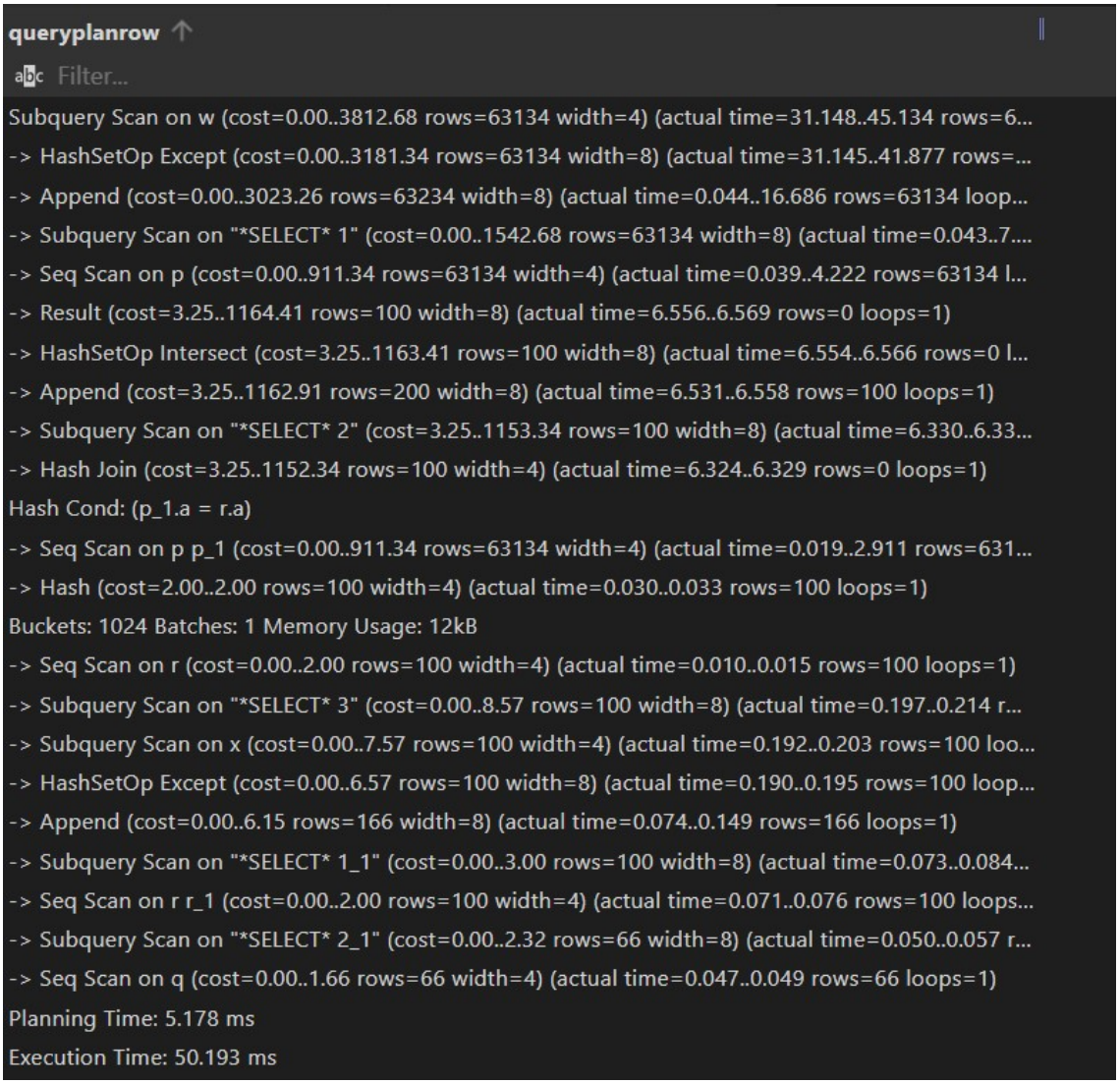
```
select w.a
from (
        select p.a
```

```
        from P p
        except (
                select r.a
                from R r
                    natural join P p
                intersect
                (
                    select x.a
                    from (
                            select r.a
                            from R r
                            except
                            select q.b
                            from Q q
                        ) x
                )
            )
    ) w;
```

QUERY PLANS for Original Query: Q5

```
queryplanrow
abc Filter...
Sort (cost=3223104.36..3223183.28 rows=31567 width=4) (actual time=1536.887..1542.196 rows=6...
Sort Key: p.a
Sort Method: external merge Disk: 872kB
-> Seq Scan on p (cost=0.00..3220745.34 rows=31567 width=4) (actual time=0.213..1508.066 rows...
Filter: (SubPlan 2)
SubPlan 2
-> Seq Scan on r (cost=0.00..101.75 rows=100 width=1) (actual time=0.002..0.014 rows=100 loops...
SubPlan 1
-> Seq Scan on q (cost=0.00..1.83 rows=66 width=1) (never executed)
Planning Time: 2.205 ms
Execution Time: 1550.119 ms
```

Query Plans for Optimised Query: Q6



The above queries have been experimented with different P, Q and R rows in the tables respectively to test the execution times.

Case 1: Keeping P (nP = 100) and Q (nQ = 100) in the same range and Varying the R rows as shown below.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
100	100	10	0.666	0.336
100	100	100	0.810	0.176
100	100	1000	11.844	1.246
100	100	10000	114.899	6.821
100	100	100000	1159.419	40.099

Case 2: Keeping R (nR = 100) and Q (nQ = 100) with the same rows and Varying the P rows as shown below.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
10	100	100	0.526	0.606
100	100	100	0.958	0.189
1000	100	100	13.220	0.614
10000	100	100	123.600	4.607
100000	100	100	1156.888	85.310

Case 3:

Keeping R (nR = 100) with the same rows and Varying the P and R rows as shown below.

select * from Case3(5,5,2,3,'select p.a fro... -- Query Plans -- Original Query -- Query ...

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
10	10	100	201.509	13.730
100	100	100	251.044	16.662
1000	1000	100	291.962	20.496
10000	10000	100	285.255	22.147
100000	100000	100	298.894	32.436

(b) What conclusions do you draw from the results of these experiments regarding the effectiveness of query optimization in PostgreSQL and/or by hand?

SOLUTION:

From the part (a), we see that in all the 3 cases the execution time for running the queries both original and optimised resulted in the lesser amount of time for the optimised query than the original query execution time itself. As the amount of data increases, be it R alone (Case1), P alone (Case2) or both P and Q(Case3), the RA SQL is performing better. This clearly states that the optimization of the query to RA SQL has reduced the complexity of the query.

PROBLEM 6:

(a) Translate and optimize query Q7 and call it Q8. Express Q8 as an RA SQL query just as was done for Q2 in Example 1. Compare queries Q7 and Q8 in a similar way as we did In Example 1. However, now you should experiment with different sizes for P, R and Q. You might consider how P and Q interact with R.

SOLUTION:

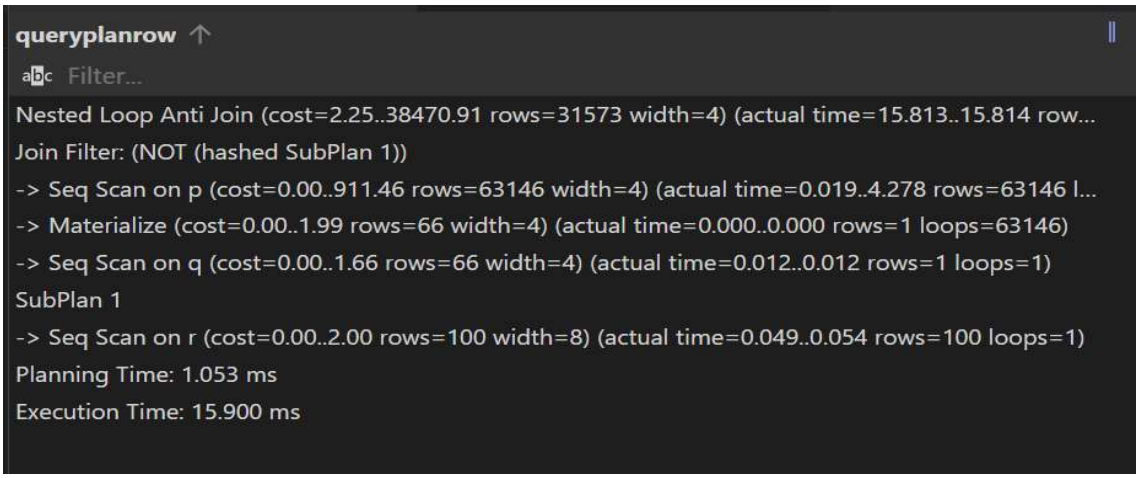
ORIGINAL QUERY: Q7

```
select p.a
from P p
where not exists (
        select 1
        from Q q
        where (p.a, q.b) not in (
                select r.a,
                    r.b
                from R r
            )
    );
```

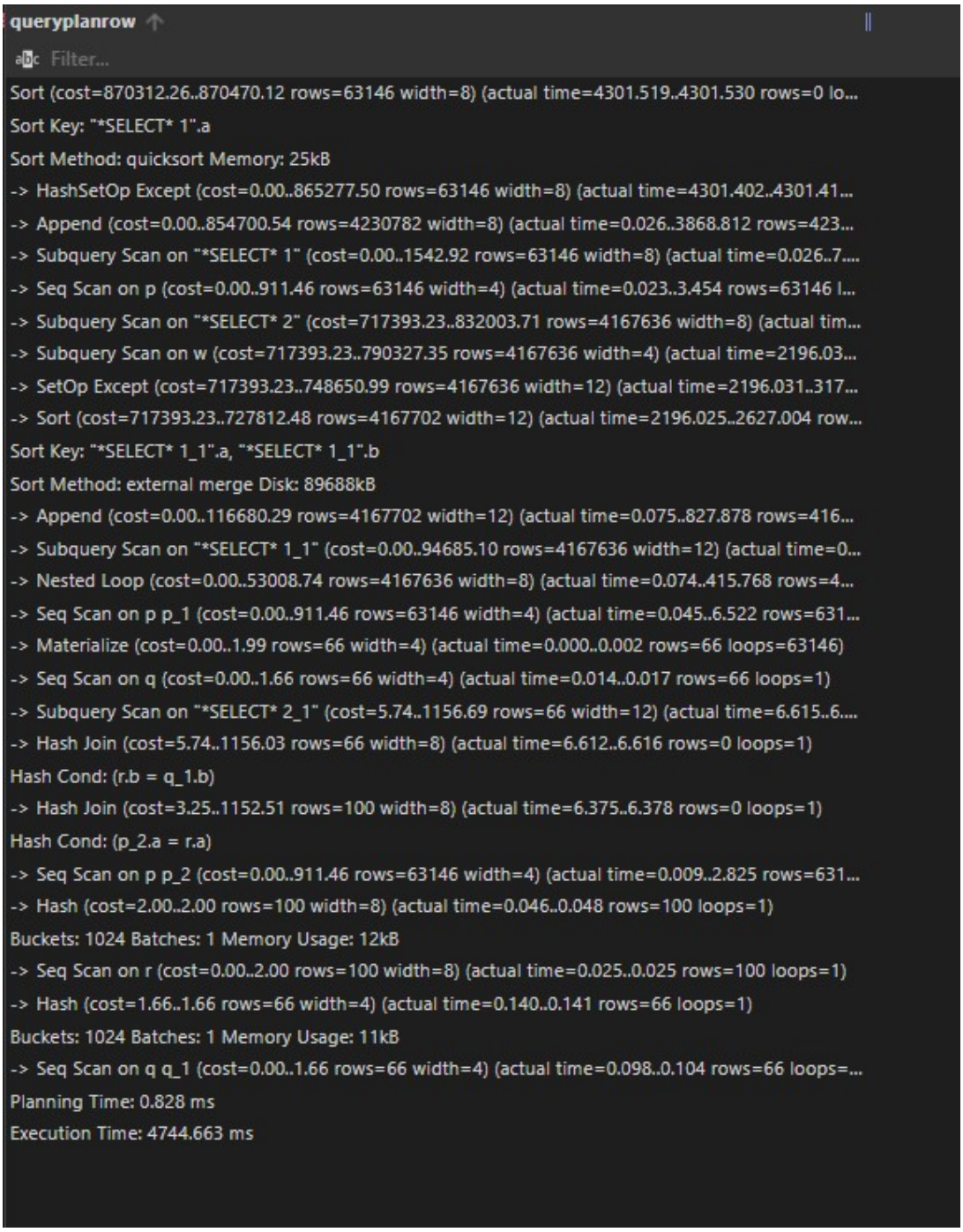
OPTIMISED QUERY: Q8

```
select p.a
from P p
EXCEPT
select w.a
from (
        select p.a,
            q.b
        from P p
            NATURAL JOIN Q q
        except
        select p.a,
            q.b
        from P p
            NATURAL JOIN R r
            JOIN Q q ON (
                p.a = r.a
                and q.b = r.b
            )
    ) w
ORDER BY 1;
```

Query Plans for Original Query: Q7



Query Plans for Optimised Query: Q8



The above queries have been experimented with different P, Q and R rows in the tables respectively to test the execution times.

Case 1: Keeping P (nP = 100) and Q (nQ = 100) in the same range and Varying the R rows as shown below.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
100	100	10	0.062	0.940
100	100	100	0.057	2.764
100	100	1000	0.163	30.002
100	100	10000	1.316	369.385
100	100	100000	14.108	5912.557

Case 2: Keeping R (nR = 100) and Q (nQ = 100) with the same rows and Varying the P rows as shown below.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
10	100	100	0.117	0.910
100	100	100	0.158	3.672
1000	100	100	0.196	38.045
10000	100	100	2.024	433.184

Case 3:

Keeping R (nR = 100) with the same rows and Varying the P and R rows as shown below.

np	nq	nr	e1	e2
Filter...	Filter...	Filter...	Filter...	Filter...
10	10	100	0.594	13.069
100	100	100	0.639	107.304
1000	1000	100	0.512	1394.747
10000	10000	100	1.023	48857.641

(b) What conclusions do you draw from the results of these experiments regarding the effectiveness of query optimization in PostgreSQL and/or by hand?

SOLUTION:

From the part (a), we see that in all the cases the execution time for running the queries both original and optimised resulted in the slightly or drastically taking more amount of time than the original query execution time itself. In cases 1 and 2, when only, P or Q rows altered the time taken by the RA SQL also slightly increased when compared to Pure SQL, in case 3 as the amount of data rows increases for both P and Q tables, the performance of the RA SQL becomes worst. In this case we could confirm that the RA SQL is performing worse than the Pure SQL.

PROBLEM 7:

Give a brief comparison of your results for Problem 4 and Problem 6. In particular, explain why you notice differences or similarities when you make this comparison. Considering the query plans for the queries in volved may aid in this comparison.

SOLUTION:

The Queries from Question 4 be Q5(Pure SQL) and Q6(Optimised SQL) and queries from Question 6 be Q7(Pure SQL) and Q8(Optimised SQL).

Analysis in Case1:

Where nP and nQ = 100, and the R rows being varied.

In Question 4, it is seen that, when the R rows alone varies and as it increases, the Q6 is taking longer time to execute than the Q5. This is the similar case with Q7 and Q8. But in Question 6 there is a drastic change in the execution time from Q7 to Q8 when R values are huge.

Analysis in Case2:

When nQ = 100 and nR = 100;

In both Question 4 and 6, the optimised queries namely Q6 and Q8 are taking more time to execute than the Pure SQL queries. This both are behaving similar to each other when the nP values are kept on increasing.

Analysis in Case3:

When nR = 100;

In Question 4, as the P and Q rows are increased, the execution time for Q6 is drastically lesser than the execution time for Q5. This is not the case with the Question 6.

In Question 6, When P and Q rows are increasing, the performace of Q8 is becoming more worse than the Q7.

Conclusion:

Both the Questions 4 and 6 function almost alike for the cases 1 and 2, where as their functionality opposes in the case3.