# ADC Assignment 7
# Harshit Rakesh Shiroiya

1. Assume the relation schemas P (x), Q(x), R(x, y) and S(x, z). Consider the NOT ALL generalized query where

$\{(p.x, q.x)|P(p) \wedge Q(p) \wedge R(p.x)/\supset S(p.x)\}$ $R(p.x) = \{r.y \mid R(r) \wedge r.x = p.x\}$

$S(q.x) = \{s.z \mid S(s) \wedge s.x = q.x\}$

Sol:

1. QUERY 1

```
                QUERY PLAN
-----------------------------------------------
 Nested Loop Semi Join
   Join Filter: ((p.x = r.x) AND (SubPlan 1))
   ->  Nested Loop
         ->  Seq Scan on p
         ->  Seq Scan on q
   ->  Seq Scan on r
   SubPlan 1
     ->  Seq Scan on s
           Filter: ((x <> q.x) OR (r.y <> z))
(9 rows)
SET

SET
```

2. QUERY 2

```
                QUERY PLAN
-----------------------------------------------
 Hash Semi Join
   Hash Cond: (p.x = r.x)
   Join Filter: (SubPlan 1)
   ->  Nested Loop
         ->  Seq Scan on p
         ->  Seq Scan on q
   ->  Hash
         ->  Seq Scan on r
   SubPlan 1
     ->  Seq Scan on s
           Filter: ((x <> q.x) OR (r.y <> z))
(11 rows)
```

3. QUERY 3

```
                QUERY PLAN
-----------------------------------------------
 Hash Semi Join
   Hash Cond: (p.x = r.x)
   Join Filter: (SubPlan 1)
   ->  Nested Loop
         ->  Seq Scan on p
         ->  Seq Scan on q
   ->  Hash
         ->  Seq Scan on r
   SubPlan 1
     ->  Seq Scan on s
           Filter: ((x <> q.x) OR (r.y <> z))
(11 rows)
```

4. QUERY 4

```
                    QUERY PLAN
------------------------------------------------------------------------
 Hash Semi Join
   Hash Cond: (p.x = r.x)
   Join Filter: ((NOT (hashed SubPlan 1)) OR (NOT (hashed SubPlan 2)))
   ->  Nested Loop
         ->  Seq Scan on p
         ->  Seq Scan on q
   ->  Hash
         ->  Seq Scan on r
   SubPlan 1
     ->  Seq Scan on s
   SubPlan 2
     ->  Seq Scan on s s_1
(12 rows)
```

5. QUERY 5

QUERY PLAN

---

```
Merge Join
  Merge Cond: (p.x = r.x)
  ->  Sort
        Sort Key: p.x
        ->  Seq Scan on p
  ->  Sort
        Sort Key: r.x
        ->  Nested Loop
              Join Filter: ((NOT (hashed SubPlan 1)) OR (NOT (hashed SubPlan 2)))
              ->  Seq Scan on r
              ->  Seq Scan on q
              SubPlan 1
                    ->  Seq Scan on s
                SubPlan 2
                ->  Seq Scan on s s_1
```

6. QUERY 6

```
                    QUERY PLAN
-------------------------------------------------
 Nested Loop
   Join Filter: ((q.x <> s.x) OR (r.y <> s.z))
   ->  Nested Loop
         ->  Merge Join
               Merge Cond: (r.x = p.x)
               ->  Sort
                     Sort Key: r.x
                     ->  Seq Scan on r
               ->  Sort
                     Sort Key: p.x
                     ->  Seq Scan on p
         ->  Seq Scan on s
   ->  Seq Scan on q
(13 rows)
```

7. QUERY 7

```
                    QUERY PLAN
-----------------------------------------------
 Nested Loop
   Join Filter: (q.x <> s.x)
   ->  Merge Join
         Merge Cond: (p.x = r.x)
         ->  Sort
               Sort Key: p.x
               ->  Seq Scan on p
         ->  Sort
               Sort Key: r.x
               ->  Nested Loop
                     Join Filter: (r.y <> s.z)
                     ->  Seq Scan on s
                     ->  Seq Scan on r
   ->  Seq Scan on q
(14 rows)
```

2. Consider query $Q_3$

```
select distinct p.a
from   P p, R r1, R r2, R r3, S s
where  p.a = r1.a and r1.b = r2.a and r2.b = r3.a and r.b = S.b;
```

Intuitively, if we view R as a graph, and P and S as node types (properties), then $Q_3$ determines each P-node in the graph from which there emanates a path of length 3 that ends at a S-node.[3] I.e., a P-node $n_0$ is in the answer if there exists sequence of nodes ($n_0$, $n_1$, $n_2$, $n_3$) such that ($n_0$, $n_1$), ($n_1$, $n_2$), and ($n_2$, $n_3$) are edges in R and $n_3$ is a S-node.

(a) Translate and optimize this query and call it $Q_4$. Then write $Q_4$ as an RA SQL query just as was done for query $Q_2$ in Example 1.

Sol:

```
select distinct p.a
from P as p
inner join R as r1 on p.a = r1.a
inner join R as r2 on r1.b = r2.a
inner join R as r3 on r2.a = r3.a
inner join S as s on r3.b = s.b;
```

(b) Compare queries $Q_3$ and $Q_4$ in a similar way as we did for $Q_1$ and $Q_2$ in Example 1. You should experiment with different sizes for R. Incidentally, these relations do not need to use the same parameters as those shown in the above table for $Q_1$ and $Q_2$ in Example 1.

Sol:

| R | Q3 | Q4 |
|---|---|---|
| $10^4$ | 1.223 ms | 0.247 ms |
| $10^5$ | 0.283 ms | 0.297 ms |
| $10^6$ | 0.162 ms | 0.180 ms |

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

Sol:

Manual optimization has improved query performance many times. However, looking at the maturity of $10^5$, there is no significant difference in the maturity of Q3 and Q4.

3. Consider the Pure SQL $Q_5$ which is an formulation of a variation of the not subset (not only) set semijoin query

where

$$\{p.a \mid P(p) \wedge R(p.a)/\subseteq S\}\ R(p.a) = \{r.b \mid R(r) \wedge r.a = p.a\}.$$

```
select p.a
from Pp
where exists (select 1 from Rr
where r.a = p.a and not exists (select 1 from S s where r.b = s.b));
```

(a) Translate and optimize this query and call it Q6. Then write Q6 as an RA SQL query just as was done for Q2 in Example 1.

Sol:

```
select p.a
 from p p
 natural join (select r.*
 from r
 except
select r.*
from r
natural join s) q;
```

(b) An alternative way to write a query equivalent with $Q_5$ is as the object-relational query

```
with nested R as (select P.a, array_agg(R.b) as bs


 from   P natural join R
 group by (P.a)),
Ss as (select array(select b from S) as bs)
 select a
from   nestedR
where  not (bs <@ (select bs from Ss));
```

Call this query $Q_7$.
Compare queries $Q_5$, $Q_6$, and $Q_7$ in a similar way as we did in Example 1. However, now you should experiment with different sizes for P, R and S as well as consider how P and S interact with R.

Sol:

| P | S | R | $Q_5$ | $Q_6$ | $Q_7$ |
|---|---|---|---|---|---|
| $10^2$ | $10^2$ | $10^2$ | 0.127 ms | 0.645 ms | 1.589 ms |
| $10^2$ | $10^3$ | $10^3$ | 0.257 ms | 0.563 ms | 0.711 ms |
| $10^3$ | $10^3$ | $10^4$ | 2.671 ms | 4.942 ms | 1.453 ms |
| $10^2$ | $10^4$ | $10^2$ | 0.547 ms | 0.873 ms | 0.743 ms |
| $10^3$ | $10^2$ | $10^5$ | 24.711 ms | 69.533 ms | 32.982 ms |
| $10^4$ | $10^4$ | $10^5$ | 18.367 ms | 74.671 ms | 57.858 ms |
| $10^5$ | $10^5$ | $10^5$ | 15.487 ms | 94.643 ms | 157.132 ms |

(c) What conclusions do you draw from the results of these experiments?

Sol:

In the case of these queries, optimization has no effect on performance.

4. Consider the Pure SQL $Q_8$ which is an formulation of a variation of the not superset, (not all) set semi join query where,

$\{p.a \mid |P(p) \wedge R(p.a)/\supseteq S\}\ R(p.a) = \{r.b \mid R(r) \wedge r.a = p.a\}$.

```
select p.a
from Pp
where exists (select 1

from Ss
where not exists (select 1 from R where p.a = r.a and r.b = s.b));
```

(a) Translate and optimize this query and call it Q9. Then write Q9 as an RA SQL query just as was done for Q2 in Example 1.

```
select p.a
from p
except
select r.a
from r
natural join s;
```

(b) An alternative way to write a query equivalent with $Q_8$ is as the object-relational query with nestedR as (select P.a, array_agg(R.b) as bs

```
            from   P natural join R
            group by (P.a)),
      Ss as (select array(select b from S) as bs)
   select a
   from   P
   where  a not in (select a from nestedR) and
not((select bs from Ss) <@ '{}') union

   select a
   from   nestedR
   where  not((select bs from Ss) <@ bs);
```

Call this query $Q_{10}$.
Compare queries $Q_8$, $Q_9$, and $Q_{10}$ in a similar way as we did In Example 1. However, now you should experiment with different sizes for P, R and S as well as consider how P and S interact with R. For example, it could be that

| P | S | R | $Q_8$ | $Q_9$ | $Q_{10}$ |
|---|---|---|---|---|---|
| $10^2$ | $10^2$ | $10^2$ | 0.378 ms | 0.543 ms | 0.873 ms |
| $10^2$ | $10^3$ | $10^3$ | 0.486 ms | 0.629 ms | 0.752 ms |
| $10^3$ | $10^3$ | $10^4$ | 3.892 ms | 5.371 ms | 2.825 ms |
| $10^2$ | $10^4$ | $10^2$ | 0.547 ms | 0.873 ms | 0.743 ms |
| $10^3$ | $10^2$ | $10^5$ | 36.923 ms | 72.862 ms | 41.875 ms |
| $10^4$ | $10^4$ | $10^5$ | 58.367 ms | 67.851 ms | 85.378 ms |
| $10^5$ | $10^5$ | $10^5$ | 45.487 ms | 94.643 ms | 107.132 ms |
| $10^6$ | $10^6$ | $10^6$ | 91568.234 ms | 287.54 ms | 307.56 ms |

(c) What conclusions do you draw from the results of these experiments?

Sol:

Q9 and Q10 which are optimized performs better than Q8 which is unoptimized.

5. Give a brief comparison of your results for Problem 3 and Problem 4. In particular, where the results show significant differences, explain why you think that is the case. And, where the results show similarities, explain why you think that is the case.

Sol:

The differences in query efficiency between optimized and unoptimized queries, are due to the relations employed in the query. When we optimize Q8, for example, the importance of relation S in the query is significantly reduced, resulting in a significant difference in the execution times of Q8 and Q9. For smaller inputs, the performance of unoptimized and optimized searches is comparable.

Collaborated with Satin Sunil Jain and Rahul Gomathi Sankarakrihnan