***Problem 1:***

For problem 1, I have discussed the solutions with Srinivas since I was not able to understand some of the details.

The TRC can be converted into the following query to compute NOT ALL .

Working memory = 1GB, All sorting enabled.

QUERY PLAN

-------------------------------------------

Hash Semi Join

Hash Cond: (p.x = r.x)

-> Nested Loop

Join Filter: (NOT (SubPlan 1))

-> Seq Scan on p

-> Materialize

-> Seq Scan on q SubPlan 1

-> Seq Scan on s

Filter: (x = q.x)

-> Hash

-> Seq Scan on r

(12 rows)

Here the postgres uses hash table to speed up the operation and only materialize the Q in memory since it’s smallest of all. Thus resulting in performance boost.

Here we will be disabling the hash join and merge join, since the both merge join and hash join are disallowed the postgres has to materialize both the tables in memory and used nested loop join to execute the query.

QUERY PLAN

-------------------------------------------

Nested Loop

Join Filter: (NOT (SubPlan 1))

-> Nested Loop

Join Filter: (p.x = r.x)

-> HashAggregate

Group Key: r.x

-> Seq Scan on r

-> Materialize

-> Seq Scan on p

-> Materialize

-> Seq Scan on q SubPlan 1

-> Seq Scan on s

Filter: (x = q.x)

(14 rows)

***Problem 2:***

* A) optimized Q4

select distinct p.a from P p join R r1 on (p.a = r1.a) join R r2 on(r1.b = r2.a) join R r3 on (r2.b = r3.a) join S s on(r3.b = S.b);

* B) Compare queries Q3 and Q4 in a similar way as we did for Q1 and Q2 in Example 1.

|  |  |
| --- | --- |
| Q3 | Q4 |
|  |  |

**Execution plan for query 3:**

A screenshot of a computer

Description automatically generated with low confidence

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Aggregate (cost=26.78..27.45 rows=67 width=4) | 67 |
|  | 2. | Hash Inner Join (cost=15.54..26.05 rows=290 width=4)   **Hash Cond**: (r2.b = r3.a) | 290 |
|  | 3. | Hash Inner Join (cost=6.5..12.88 rows=166 width=8)   **Hash Cond**: (r1.b = r2.a) | 166 |
|  | 4. | Hash Inner Join (cost=3.25..7.17 rows=129 width=8)   **Hash Cond**: (p.a = r1.a) | 129 |
|  | 5. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 6. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 7. | Seq Scan on public.r as r1 (cost=0..2 rows=100 width=8) | 100 |
|  | 8. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 9. | Seq Scan on public.r as r2 (cost=0..2 rows=100 width=8) | 100 |
|  | 10. | Hash (cost=7.35..7.35 rows=135 width=4) | 135 |
|  | 11. | Hash Inner Join (cost=3.25..7.35 rows=135 width=4)   **Hash Cond**: (r3.b = s.b) | 135 |

**Execution plan for query 4**

**A screenshot of a computer

Description automatically generated with low confidence**

| **#** | **Node** | **Timings** | | **Rows** | | | **Loops** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Exclusive** | **Inclusive** | **Rows X** | **Actual** | **Plan** |
|  | 1. | Aggregate (cost=26.78..27.45 rows=67 width=4) (actual=0.141..0.144 rows=33 loops=1)   **Buckets**: **Batches**: **Memory Usage**: 24 kB | 0.016 ms | 0.144 ms | ↑ 2.04 | 33 | 67 | 1 |
|  | 2. | Hash Inner Join (cost=15.54..26.05 rows=290 width=4) (actual=0.087..0.128 rows=117 loops=1)   **Hash Cond**: (r2.b = r3.a) | 0.015 ms | 0.128 ms | ↑ 2.48 | 117 | 290 | 1 |
|  | 3. | Hash Inner Join (cost=6.5..12.88 rows=166 width=8) (actual=0.045..0.074 rows=103 loops=1)   **Hash Cond**: (r1.b = r2.a) | 0.016 ms | 0.074 ms | ↑ 1.62 | 103 | 166 | 1 |
|  | 4. | Hash Inner Join (cost=3.25..7.17 rows=129 width=8) (actual=0.02..0.036 rows=95 loops=1)   **Hash Cond**: (p.a = r1.a) | 0.017 ms | 0.036 ms | ↑ 1.36 | 95 | 129 | 1 |
|  | 5. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) (actual=0.004..0.007 rows=100 loops=1) | 0.007 ms | 0.007 ms | ↑ 1 | 100 | 100 | 1 |
|  | 6. | Hash (cost=2..2 rows=100 width=8) (actual=0.012..0.012 rows=100 loops=1)   **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB | 0.006 ms | 0.012 ms | ↑ 1 | 100 | 100 | 1 |
|  | 7. | Seq Scan on public.r as r1 (cost=0..2 rows=100 width=8) (actual=0.003..0.006 rows=100 loops=1) | 0.006 ms | 0.006 ms | ↑ 1 | 100 | 100 | 1 |
|  | 8. | Hash (cost=2..2 rows=100 width=8) (actual=0.022..0.022 rows=100 loops=1)   **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB | 0.008 ms | 0.022 ms | ↑ 1 | 100 | 100 | 1 |
|  | 9. | Seq Scan on public.r as r2 (cost=0..2 rows=100 width=8) (actual=0.01..0.014 rows=100 loops=1) | 0.014 ms | 0.014 ms | ↑ 1 | 100 | 100 | 1 |
|  | 10. | Hash (cost=7.35..7.35 rows=135 width=4) (actual=0.04..0.04 rows=96 loops=1)   **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB | 0.006 ms | 0.04 ms | ↑ 1.41 | 96 | 135 | 1 |
|  | 11. | Hash Inner Join (cost=3.25..7.35 rows=135 width=4) (actual=0.019..0.034 rows=96 loops=1)   **Hash Cond**: (r3.b = s.b) | 0.015 ms | 0.034 ms | ↑ 1.41 | 96 | 135 | 1 |
|  | 12. | Seq Scan on public.r as r3 (cost=0..2 rows=100 width=8) (actual=0.003..0.006 rows=100 loops=1) | 0.006 ms | 0.006 ms | ↑ 1 | 100 | 100 | 1 |
|  | 13. | Hash (cost=2..2 rows=100 width=4) (actual=0.014..0.014 rows=100 loops=1)   **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB | 0.007 ms | 0.014 ms | ↑ 1 | 100 | 100 | 1 |
|  | 14. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) (actual=0.004..0.007 rows=100 loops=1) | 0.007 ms | 0.007 ms | ↑ 1 | 100 | 100 | 1 |

***Conclusion***

Here we are not able to see any drastic improvements in query performance since both the queries uses hash joins to perform the selection operation. However, performance may vary as data size increases significantly.

***Problem 3:***

* A) Optimized Q 6

-- step 1 Optimization

-- step 2 Optimization

-- step 3: Optimization

B) Compare Q5, Q6, Q7

|  |  |  |
| --- | --- | --- |
| Q5 | Q6 | Q7 |
|  |  |  |

**Execution plan for Q5,**

**Chart

Description automatically generated with medium confidence**

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Hash Semi Join (cost=5.89..8.16 rows=1 width=4)   **Hash Cond**: (p.a = r.a) | 1 |
|  | 2. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 3. | Hash (cost=5.88..5.88 rows=1 width=4) | 1 |
|  | 4. | Hash Anti Join (cost=3.25..5.88 rows=1 width=4)   **Hash Cond**: (r.b = s.b) | 1 |
|  | 5. | Seq Scan on public.r as r (cost=0..2 rows=100 width=8) | 100 |
|  | 6. | Hash (cost=2..2 rows=100 width=4) | 100 |
|  | 7. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |

**Execution plan for Q6, Diagram

Description automatically generated**

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Aggregate (cost=28.75..30.04 rows=129 width=4) | 129 |
|  | 2. | Subquery Scan (cost=3.25..28.43 rows=129 width=4) | 129 |
|  | 3. | Hash Except (cost=3.25..27.14 rows=129 width=16) | 129 |
|  | 4. | Append (cost=3.25..24.86 rows=304 width=16) | 304 |
|  | 5. | Subquery Scan (cost=3.25..8.46 rows=129 width=16) | 129 |
|  | 6. | Hash Inner Join (cost=3.25..7.17 rows=129 width=12)   **Hash Cond**: (p.a = r.a) | 129 |
|  | 7. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 8. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 9. | Seq Scan on public.r as r (cost=0..2 rows=100 width=8) | 100 |
|  | 10. | Subquery Scan (cost=6.5..14.88 rows=175 width=16) | 175 |
|  | 11. | Hash Inner Join (cost=6.5..13.13 rows=175 width=12)   **Hash Cond**: (r\_1.b = s.b) | 175 |
|  | 12. | Hash Inner Join (cost=3.25..7.17 rows=129 width=12)   **Hash Cond**: (p\_1.a = r\_1.a) | 129 |
|  | 13. | Seq Scan on public.p as p\_1 (cost=0..2 rows=100 width=4) | 100 |
|  | 14. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 15. | Seq Scan on public.r as r\_1 (cost=0..2 rows=100 width=8) | 100 |
|  | 16. | Hash (cost=2..2 rows=100 width=4) | 100 |
|  | 17. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |

**Execution plan for Q7,**

Chart

Description automatically generated

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Subquery Scan (cost=9.82..11.5 rows=67 width=4) | 67 |
|  | 2. | Result (cost=2..2.01 rows=1 width=32) | 1 |
|  | 3. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |
|  | 4. | Aggregate (cost=7.81..8.82 rows=67 width=36)   **Filter**: (NOT (array\_agg(r.b) <@ $1)) | 67 |
|  | 5. | Hash Inner Join (cost=3.25..7.17 rows=129 width=8)   **Hash Cond**: (p.a = r.a) | 129 |
|  | 6. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 7. | Hash (cost=2..2 rows=100 width=8) | 100 |

*Conclusion*

Here we can see substantial improvement using RA optimized query and worst performance with array aggregation query Q7. Since all the three queries were implemented using hash, the RA optimized in super in terms of multiple hash tables and hash except.

***Problem 4:***

* A) Optimized Q 9

-- step 1

-- step 2

-- step 3

* Compare Q8, Q9, Q10

|  |  |  |
| --- | --- | --- |
| Q8 | Q9 | Q10 |
|  |  |  |

**Execution plan Q8**

**Diagram

Description automatically generated**

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Nested Loop Semi Join (cost=0..12815.51 rows=50 width=4)   **Join Filter**: (NOT (alternatives: SubPlan 1 or hashed SubPlan 2)) | 50 |
|  | 2. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 3. | Materialize (cost=0..2.5 rows=100 width=4) | 100 |
|  | 4. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |
|  | 5. | Seq Scan on public.r as r (cost=0..2.5 rows=1 width=0)   **Filter**: ((p.a = r.a) AND (r.b = s.b)) | 1 |
|  | 6. | Seq Scan on public.r as r\_1 (cost=0..2 rows=100 width=8) | 100 |

**Execution plan Q9**

**Diagram

Description automatically generated**

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Aggregate (cost=399.48..401.48 rows=200 width=4) | 200 |
|  | 2. | Subquery Scan (cost=0..388.76 rows=4288 width=4) | 4288 |
|  | 3. | Hash Except (cost=0..345.88 rows=4288 width=12) | 4288 |
|  | 4. | Append (cost=0..295.01 rows=10175 width=12) | 10175 |
|  | 5. | Subquery Scan (cost=0..229.25 rows=10000 width=12) | 10000 |
|  | 6. | Nested Loop Inner Join (cost=0..129.25 rows=10000 width=8) | 10000 |
|  | 7. | Seq Scan on public.p as p (cost=0..2 rows=100 width=4) | 100 |
|  | 8. | Materialize (cost=0..2.5 rows=100 width=4) | 100 |
|  | 9. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |
|  | 10. | Subquery Scan (cost=6.5..14.88 rows=175 width=12) | 175 |
|  | 11. | Hash Inner Join (cost=6.5..13.13 rows=175 width=8)   **Hash Cond**: (r.b = s\_1.b) | 175 |
|  | 12. | Hash Inner Join (cost=3.25..7.17 rows=129 width=8)   **Hash Cond**: (p\_1.a = r.a) | 129 |
|  | 13. | Seq Scan on public.p as p\_1 (cost=0..2 rows=100 width=4) | 100 |
|  | 14. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 15. | Seq Scan on public.r as r (cost=0..2 rows=100 width=8) | 100 |
|  | 16. | Hash (cost=2..2 rows=100 width=4) | 100 |
|  | 17. | Seq Scan on public.s as s\_1 (cost=0..2 rows=100 width=4) | 100 |

**Execution plan Q10**

Diagram

Description automatically generated

| **#** | **Node** | **Rows** |
| --- | --- | --- |
| **Plan** |
|  | 1. | Aggregate (cost=18.01..19.18 rows=117 width=4) | 117 |
|  | 2. | Aggregate (cost=7.81..8.65 rows=67 width=36) | 67 |
|  | 3. | Hash Inner Join (cost=3.25..7.17 rows=129 width=8)   **Hash Cond**: (p\_1.a = r.a) | 129 |
|  | 4. | Seq Scan on public.p as p\_1 (cost=0..2 rows=100 width=4) | 100 |
|  | 5. | Hash (cost=2..2 rows=100 width=8) | 100 |
|  | 6. | Seq Scan on public.r as r (cost=0..2 rows=100 width=8) | 100 |
|  | 7. | Result (cost=2..2.01 rows=1 width=32) | 1 |
|  | 8. | Seq Scan on public.s as s (cost=0..2 rows=100 width=4) | 100 |
|  | 9. | Append (cost=1.53..7.06 rows=117 width=4) | 117 |
|  | 10. | Result (cost=1.53..3.78 rows=50 width=4) | 50 |
|  | 11. | CTE Scan (cost=0..0.02 rows=1 width=32) | 1 |
|  | 12. | Seq Scan on public.p as p (cost=1.53..3.78 rows=50 width=4)   **Filter**: (NOT (hashed SubPlan 4)) | 50 |
|  | 13. | CTE Scan (cost=0..1.34 rows=67 width=4) | 67 |
|  | 14. | CTE Scan (cost=0.02..1.53 rows=67 width=4)   **Filter**: (NOT ($5 <@ nestedr.bs)) | 67 |
|  | 15. | CTE Scan (cost=0..0.02 rows=1 width=32) | 1 |

***Conclusion:***

Here the optimized RA query the longest time with data size above 10 ^ 5 impossible to execute, this is attributed presence of multiple joins including nested inner loop join and hash inner join, compared to original query which had single nested inner loop join. The array agg method performing way better than the RA optimized query as well.

***Problem 5:***

For queries in problem 3, all queries had hash inner joins and variations of it. Resulting in Fastest query being RA optimized query this can be attributed to multiple hash tables being used to speed up the performance. Compared to queries in problem 4, all queries involved nested inner loop join resulting in poor performance with RA optimized query performing worst. This is due to presence of nested inner loop joins in RA optimized query. The array agg query used Cte scan resulting in better performance over the RA query.