

CS486/586 Introduction to Databases

Fall 2021 Quarter

Submitted By: Agnita, Raghavendra

Assignment 6 – Query Optimization, Transactions, Recovery
Due: Monday, Nov 29th at 11:59PM.

Instructions & Notes:

- Do this assignment in groups of 2.
- Ensure that each group member's name is listed on the assignment, and in the notes field of Canvas to ensure credit.
- Submit your assignment in PDF format.
- Submit your completed assignment on Canvas, one submission per group.
- 100 points total.

Part I: Join Implementation – 15 points

Question 1 (15 points):

- A. Write a SQL query using the spy schema for which you believe it would be efficient to use hash join. Include the query here.

Solution:

```
SELECT * FROM agent, securityclearance  
WHERE agent.clearance_id = securityclearance.sc_id;
```

- B. Explain why you believe the hash join algorithm could or would be used.

Solution:

In the above mentioned query, we are using the equality operator as a comparison operator.

Although Sort-merge, Hash Join, and Index Nested Loop are used for equi joins, the cost for the Hash Join (simpler case because $M < BP$ OR $N < BP$) is much cheaper than the other two i.e., $M+N$. And hence, I believe Hash Join would be used for this query.

- C. Use EXPLAIN to see (and report) which join algorithm postgresql actually uses.
(Note: It's fine if the join algorithm postgresql uses does not match the join algorithm named in the question.)

Solution:

```
EXPLAIN SELECT * FROM agent, securityclearance
WHERE agent.clearance_id = securityclearance.sc_id;
```

Using EXPLAIN, we could see in the below screenshot that PostgreSQL actually uses the Hash Join algorithm that we predicted in part 2 of this question.

```
fall2021db78=> EXPLAIN SELECT * FROM agent, securityclearance
fall2021db78-> WHERE agent.clearance_id = securityclearance.sc_id;
               QUERY PLAN
-----
Hash Join  (cost=1.16..18.57 rows=662 width=234)
  Hash Cond: (agent.clearance_id = securityclearance.sc_id)
    -> Seq Scan on agent  (cost=0.00..14.62 rows=662 width=54)
    -> Hash  (cost=1.07..1.07 rows=7 width=180)
          -> Seq Scan on securityclearance (cost=0.00..1.07 rows=7 width=180)
(5 rows)
```

Part II: Statistics – 25 points

Question 2 (5 points): Determine the min and max salary values in the agent table, and the number of rows in that table.

Solution:

```
SELECT MIN(salary), MAX(salary) FROM agent;
```

```
[fall2021db78=> SELECT MIN(salary), MAX(salary) FROM agent;
 min | max
-----+-----
50008 | 366962
(1 row)
```

Agent table has overall 662 tuples/rows in it as captured in the below screenshot.

```
[fall2021db78=> SELECT count(*) FROM agent;
 count
-----
  662
```

Question 3 (5 points): Give an estimate for the number of rows in agent with salary < 92000, assuming a uniform distribution of salaries between min and max salary. Explain how you derived your estimate.

Solution:

From the previous observation in Question 2 of Part II, we know that the MIN Salary is \$50008 & MAX salary is \$366962.

We also know the number of total rows in the table: 662

1 - 10, uniform, 10. <9, 8

Difference between MIN Salary & MAX Salary is = $366962 - 50008 = 316954$

Assuming the salary is uniformly distributed, difference of salary between each agent is = $316954 / 662 = 478.78$

Difference between minimum salary and the condition of <92000 is = $92000 - 50008 = 41992$

We have these many agents between minimum salary and <92000 = $41992 / 478.78 = 87.7$

Approximately, we have 88 agents (including the person that makes 50008) that have salary < 92000.

Question 4 (5 points): Find the 25th, 50th and 75th percentile values for salaries in the agent table. (The 50th percentile value, for instance, is the smallest number such that 50% of the rows have salary value less than or equal to s.)

Solution:

SELECT

percentile_disc(0.25) within group (order by salary) as percentile_count_25,
percentile_disc(0.50) within group (order by salary) as percentile_count_50,
percentile_disc(0.75) within group (order by salary) as percentile_count_75
FROM agent;

```
fall2021db78=> SELECT
fall2021db78-> percentile_disc(0.25) within group (order by salary) as percentile_count_25,
fall2021db78-> percentile_disc(0.50) within group (order by salary) as percentile_count_50,
fall2021db78-> percentile_disc(0.75) within group (order by salary) as percentile_count_75
fall2021db78-> FROM agent;
 percentile_count_25 | percentile_count_50 | percentile_count_75
-----+-----+-----
          54802 |          58430 |          89643
(1 row)
```

Note: percentile_disc calculates the percentile based on a discrete distribution of the column values. Whereas percentile_cont gives percentile values that may or may not match the salaries of agents in the table. Hence we chose to use percentile_disc

Question 5 (5 points): Give an estimate of the number of rows in agent with salary < 92000, assuming in a uniform distribution in each quartile determined in Question 4. Explain how you derived your estimate. (Uniform distribution means values are evenly distributed between the min and max.)

Solution:

Based on the above screenshot in Solution 4, we can conclude that the 92000 will be in the 4th quartile.

So, for the 4th Quartile, minimum salary is 89643 and the maximum salary is 366962.

Difference between minimum and maximum salary is = $366962 - 89643 = 277,319$

Number of agents in 4th quartile = $662 / 4 = 165.5 =$ approximately 166.

So, uniform distribution of the above difference over 166 agents will give = 1670.59

Number of agents between 92000 and 89643 = $(92000 - 89643) / 1670.59 = 1.4108$ i.e., approximately 1.

Number of agents whose salary is less than 89643 (until end of 3rd quartile) = $662 * 0.75 = 497.5$ (approximately 497 agents).

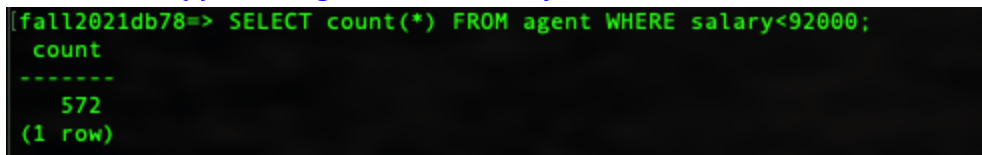
So, total number of agents whose salary is less than 92000 is = $497 + 1 = 498$.

Question 6 (5 points): How many rows in the agent table actually have salary < 92000?

Solution:

We have 572 rows in the agent table with salary < 92000.

SELECT count(*) FROM agent WHERE salary < 92000;



```
|fall2021db78=> SELECT count(*) FROM agent WHERE salary < 92000;
count
-----
    572
(1 row)
```

Part III: Query Plans, cont'd – 60 points

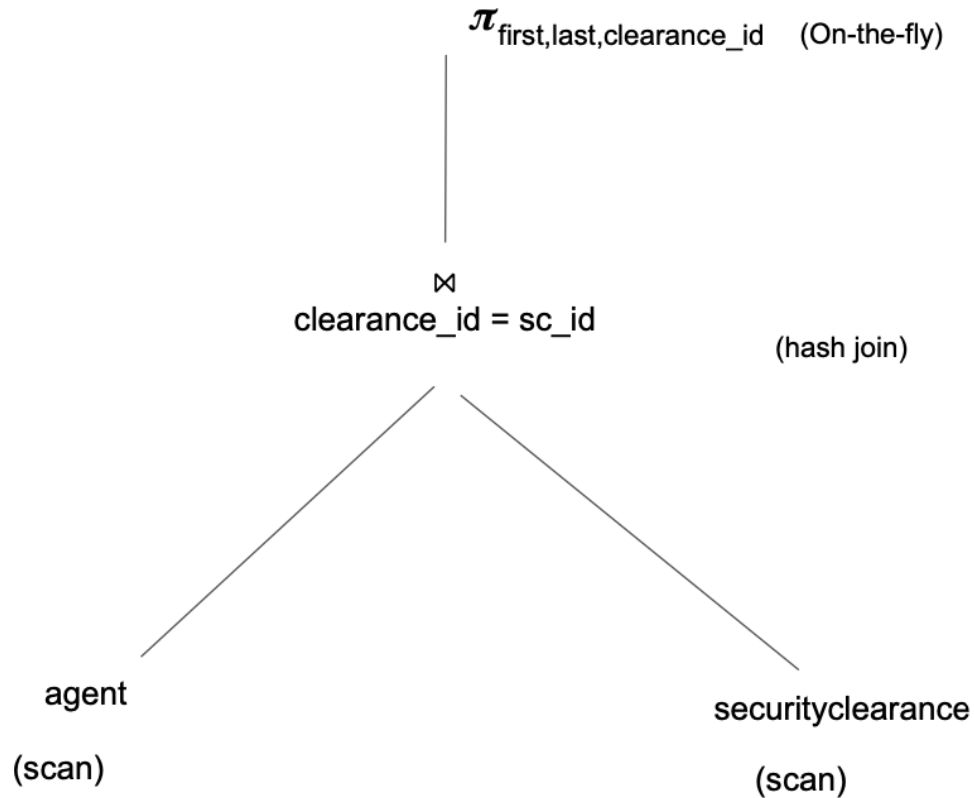
Following on the query plans work from HW 4 and using the same set of queries, please do the following. For each SQL statement below, draw the query plan PostgreSQL uses (you can find the query plan with the EXPLAIN command See [Documentation: 14: EXPLAIN](#) Also, see slide 19 in Slides 12 for and Activities for Slides 12 for information about Explain). For each plan, suggest a reason that the particular join algorithms were chosen.

Question 7 (20 points):

SELECT A.first, A.last, A.clearance_id
FROM agent A, securityclearance S
WHERE A.clearance_id = S.sc_id

Solution:

Query Plan:



EXPLAIN SELECT A.first, A.last, A.clearance_id
FROM agent A, securityclearance S
WHERE A.clearance_id = S.sc_id;

```
fall2021db78=> EXPLAIN SELECT A.first, A.last, A.clearance_id
fall2021db78-> FROM agent A, securityclearance S
fall2021db78-> WHERE A.clearance_id = S.sc_id;
               QUERY PLAN
-----
Hash Join  (cost=1.16..18.57 rows=662 width=17)
  Hash Cond: (a.clearance_id = s.sc_id)
    -> Seq Scan on agent a  (cost=0.00..14.62 rows=662 width=17)
    -> Hash  (cost=1.07..1.07 rows=7 width=4)
          -> Seq Scan on securityclearance s  (cost=0.00..1.07 rows=7 width=4)
(5 rows)
```

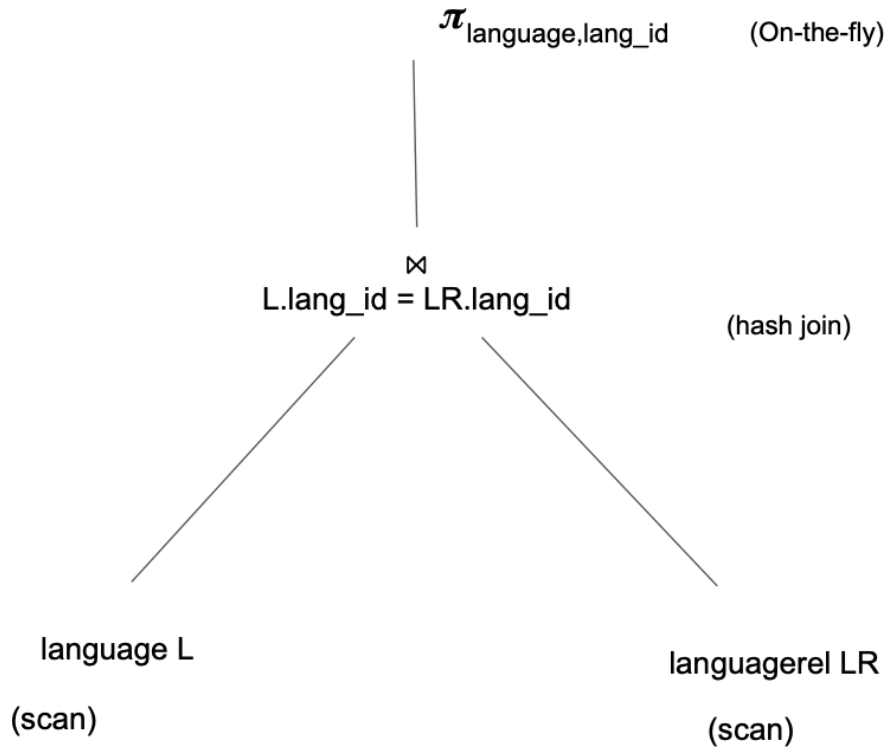
Reason: There is a significant difference between the tables agent and securityclearance (662 rows in agent and 7 rows in securityclearance). Table securityclearance can easily fit into the memory. And hence the Hash Join is being used for this query.

Question 8 (20 points):

```
SELECT L.language, L.lang_id  
FROM language L, languagerel LR  
WHERE L.lang_id = LR.lang_id;
```

Solution:

Query Plan:



```
EXPLAIN SELECT L.language, L.lang_id  
FROM language L, languagerel LR  
WHERE L.lang_id = LR.lang_id;
```

```
fall2021db78=> EXPLAIN SELECT L.language, L.lang_id  
fall2021db78-> FROM language L, languagerel LR  
fall2021db78-> WHERE L.lang_id = LR.lang_id;  
                QUERY PLAN  
-----  
Hash Join (cost=1.45..36.70 rows=1991 width=62)  
  Hash Cond: (lr.lang_id = l.lang_id)  
    -> Seq Scan on languagerel lr (cost=0.00..28.91 rows=1991 width=4)  
    -> Hash (cost=1.20..1.20 rows=20 width=62)  
          -> Seq Scan on language l (cost=0.00..1.20 rows=20 width=62)  
(5 rows)
```

Reason: There is a significant difference between the tables language and languagerel (20 rows in language and 1991 rows in languagerel). Table language can easily fit into the memory. And hence the Hash Join is being used for this query.

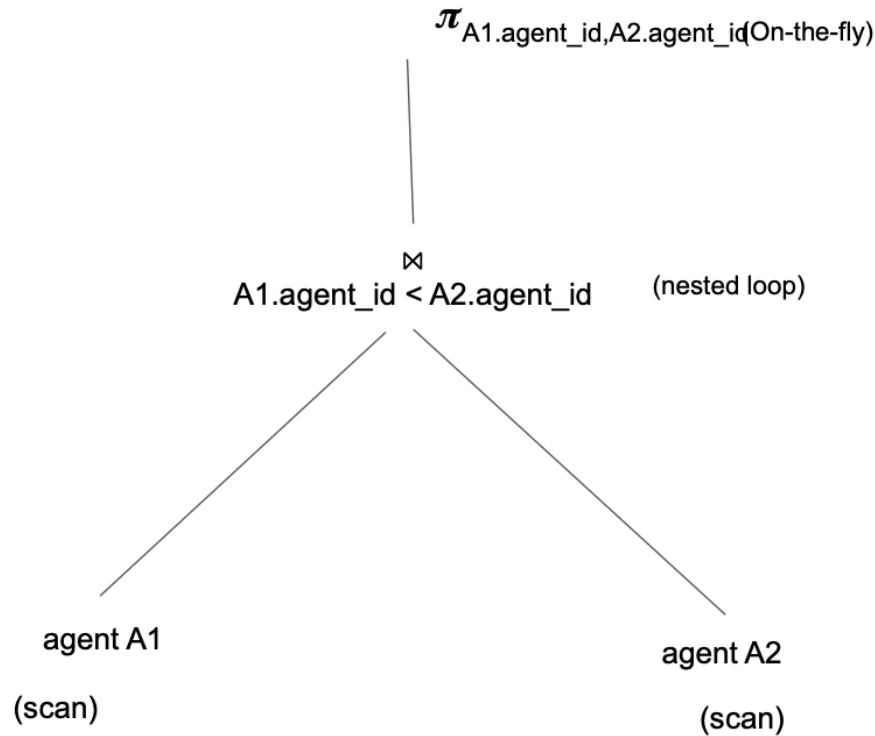
And also, partitioning is not required that makes the hash join cheapest of available ones.

Question 9 (20 points):

```
SELECT A1.agent_id, A2.agent_id
FROM agent A1, agent A2
WHERE A1.salary < A2.salary
```

Solution:

Query Plan:



```
EXPLAIN SELECT A1.agent_id, A2.agent_id
FROM agent A1, agent A2
WHERE A1.salary < A2.salary;
```

```
fall2021db78=> SELECT * from languagerel;
fall2021db78=> EXPLAIN SELECT A1.agent_id, A2.agent_id
fall2021db78-> FROM agent A1, agent A2
fall2021db78-> WHERE A1.salary < A2.salary;
               QUERY PLAN
-----
Nested Loop  (cost=0.00..6604.56 rows=146081 width=8)
  Join Filter: (a1.salary < a2.salary)
    -> Seq Scan on agent a1  (cost=0.00..14.62 rows=662 width=8)
    -> Materialize  (cost=0.00..17.93 rows=662 width=8)
        -> Seq Scan on agent a2  (cost=0.00..14.62 rows=662 width=8)
(5 rows)
```

Reason: This is not an equi join query. In this query, a comparative operator (<) is being used so the join algorithm is a nested loop. The comparative operator checks every tuple of the inner table with every tuple of the outer table.