CS486/586 Introduction to Databases
Spring 2022 Quarter

Assignment 5 – Storage and Indexing; Query Evaluation
Due: Friday, May 20, 11:59 pm

Submitted By: - Parth Parashar

Instructions & Notes:

- 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, including both of your names for each group.
- 100 points total.
- This assignment uses the postgresql EXPLAIN command. You can find info on the EXPLAIN command at: https://www.postgresql.org/docs/12/performance-tips.html
- **The instructions for this assignment are a bit more complex than previous assignments. Be sure to read through the questions carefully and completely.**

Part I - Index Matching (20 points total)

Schema: Stadiums(id, name, maximum_capacity, field_size)
Assume a clustered index on id and a multi-attribute index on (maximum_capacity, field_size).

For each selection predicate below, say if the index "matches" the predicate. If the index does not match the predicate, give a brief explanation as to why the index does not match the predicate. You can find the appropriate capacity and square foot information for the two records contained in the schema here:

- Providence Park at https://en.wikipedia.org/wiki/Providence_Park
- CenturyLink Field at https://en.wikipedia.org/wiki/CenturyLink_Field

For questions c and d, the field_size attribute is measured in square feet (**not yards**).

Question 1 (20 points):

a) name = 'Providence Park'
b) id < 3
c) field_size < 18000 AND maximum_capacity > 25000
d) maximum_capacity < 35000 OR field_size > 17000
e) Which of the above predicates will Providence Park satisfy? Which will the CenturyLink Field satisfy?

Answer: -

a) Name = 'Providence Park'

The index does not match the given predicate. This is because there is an index on the id attribute , on the maximum capacity and field size attributes, but there is not an index on the name attribute which is why the index does not match the given predicate.

b) Id < 3

The index does not match predicate.

c) Field_size < 18000 AND maximum_capacity > 25000

The index does match the predicate

d) Maximum_capacity < 35000 or field_size > 17000

The index does not match the predicate because the given clause is an OR clause.
Since the index couldn't be used for the field_size (index is first sorted by maximum_capacity), the index might be used to find all the stadiums with a maximum_capacity above 35000 but there is no surety of finding the desired results and hence it does not match the predicate.

e) Providence Park will satisfy the following predicates: -

   a. **Name = 'Providence Park'**
   b. **Id < 3.** This is conditional and will satisfy only if ids start with 1 and are present in sequential ascending order.
   c. **Maximum_capacity < 35000 or field_size > 17000**

CenturyLink will satisfy the following predicates: -

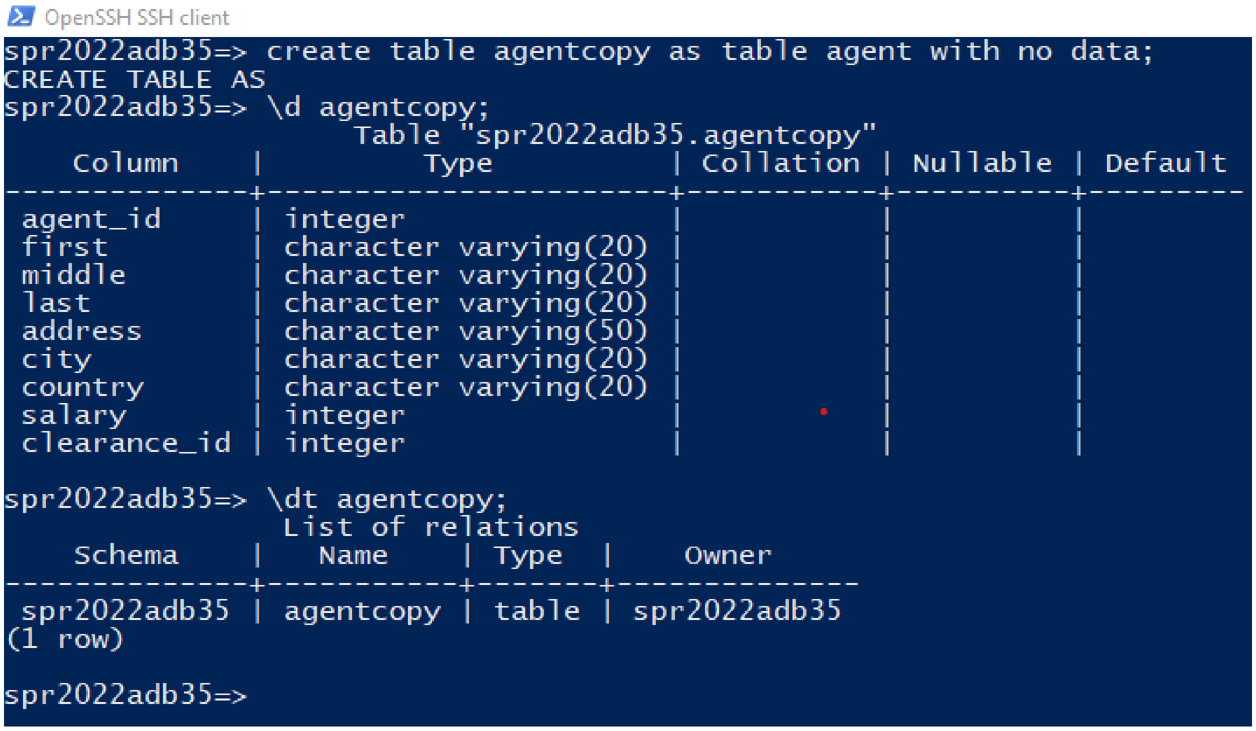
   a. **Id<3.** This is conditional and will satisfy only if ids start with 1 and are present in sequential ascending order.
   b. **Maximum_capacity < 35000 or field_size > 17000**

Part II: Query Plans – (80 points total)
Some questions in this section ask you to use the pg_class table which contains information about the number of pages and tuples in a relation. You can see information about the pg_class table here: [Documentation: 14: 52.11. pg_class]() (Also see supplementary video)

Question 2 (20 points):

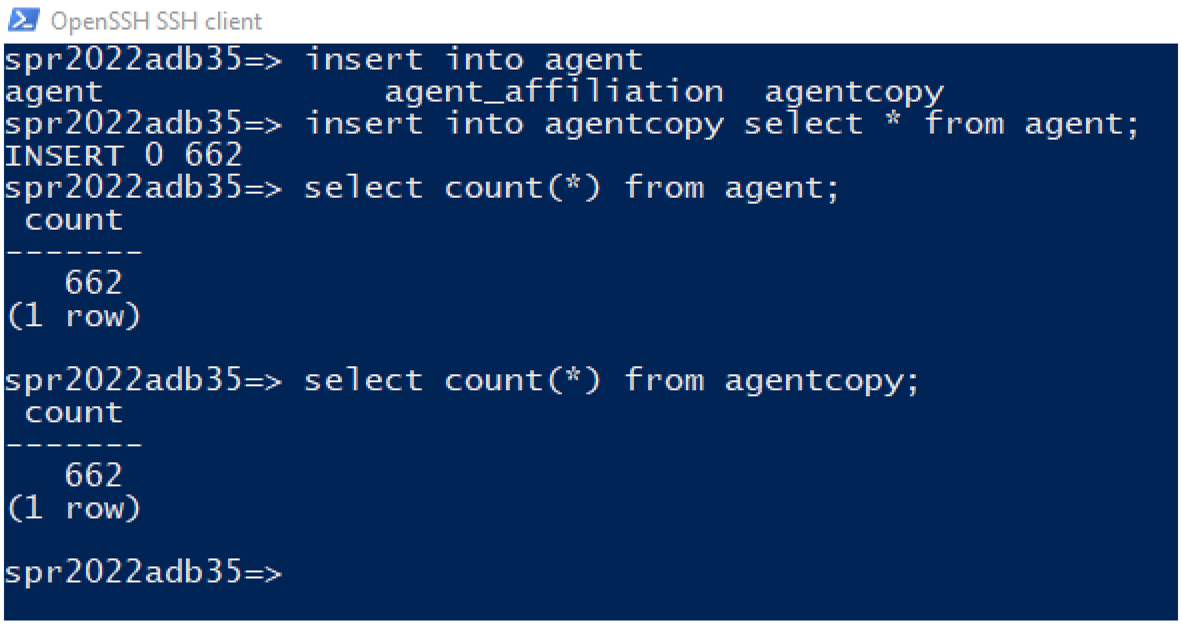
Make a copy of the agent table using the following set of commands. By running the first command and excluding the line 'WITH NO DATA' you can copy the schema and the data in one command; however, when dealing with data you are unfamiliar with it's a) good to know how to copy the schema and data separately, and b) generally considered a best practice to do so.

CREATE TABLE <new table name> AS
TABLE <existing table>
WITH NO DATA;


OpenSSH SSH client

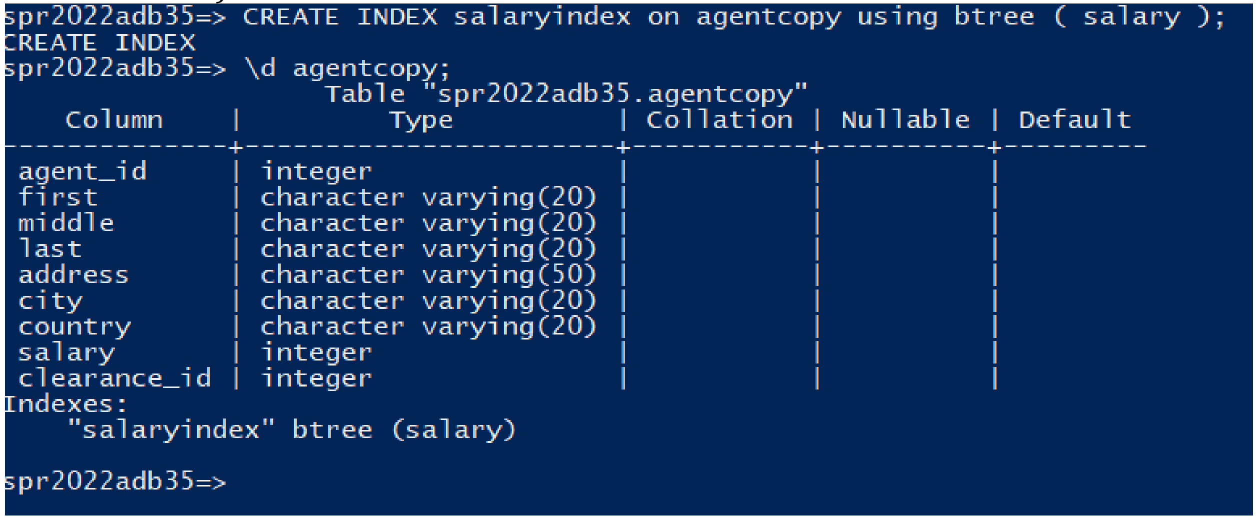
```
spr2022adb35=> create table agentcopy as table agent with no data;
CREATE TABLE AS
spr2022adb35=> \d agentcopy;
                  Table "spr2022adb35.agentcopy"
    Column     |         Type          | Collation | Nullable | Default
---------------+-----------------------+-----------+----------+---------
 agent_id      | integer               |           |          |
 first         | character varying(20) |           |          |
 middle        | character varying(20) |           |          |
 last          | character varying(20) |           |          |
 address       | character varying(50) |           |          |
 city          | character varying(20) |           |          |
 country       | character varying(20) |           |          |
 salary        | integer               |           |          |
 clearance_id  | integer               |           |          |

spr2022adb35=> \dt agentcopy;
                List of relations
    Schema     |   Name    | Type  |    Owner
---------------+-----------+-------+--------------
 spr2022adb35  | agentcopy | table | spr2022adb35
(1 row)

spr2022adb35=>
```

INSERT INTO <new table name>
SELECT *
FROM <existing table>;


OpenSSH SSH client

```
spr2022adb35=> insert into agent
agent                  agent_affiliation   agentcopy
spr2022adb35=> insert into agentcopy select * from agent;
INSERT 0 662
spr2022adb35=> select count(*) from agent;
 count
-------
   662
(1 row)

spr2022adb35=> select count(*) from agentcopy;
 count
-------
   662
(1 row)

spr2022adb35=>
```

Create an index on the salary attribute of the **copy** of the agent table. For the purposes of the SQL statements below, it will be referred to as 'agent2' though you may have named it something different.(You can use \help in the command line interface to get the syntax for CREATE INDEX).

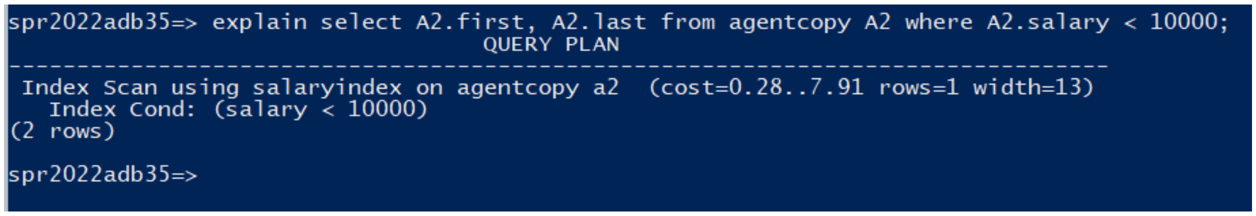


```
spr2022adb35=> CREATE INDEX salaryindex on agentcopy using btree ( salary );
CREATE INDEX
spr2022adb35=> \d agentcopy;
                  Table "spr2022adb35.agentcopy"
    Column     |          Type          | Collation | Nullable | Default
---------------+------------------------+-----------+----------+---------
 agent_id      | integer                |           |          |
 first         | character varying(20)  |           |          |
 middle        | character varying(20)  |           |          |
 last          | character varying(20)  |           |          |
 address       | character varying(50)  |           |          |
 city          | character varying(20)  |           |          |
 country       | character varying(20)  |           |          |
 salary        | integer                |           |          |
 clearance_id  | integer                |           |          |
Indexes:
    "salaryindex" btree (salary)

spr2022adb35=>
```

For each SQL statement below, use the EXPLAIN command to find the query plan that postgresql uses. For your answer, indicate if Postgres chooses an index or not.

SELECT A2.first, A2.last
FROM agent2 A2
WHERE A2.salary < 10000;

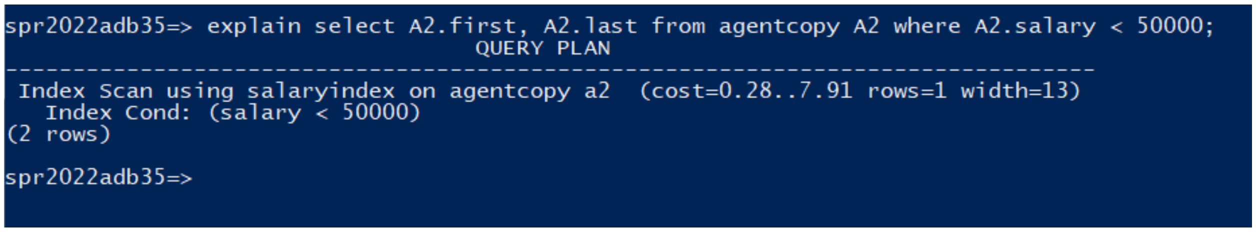
PostGre chose an index



```
spr2022adb35=> explain select A2.first, A2.last from agentcopy A2 where A2.salary < 10000;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using salaryindex on agentcopy a2  (cost=0.28..7.91 rows=1 width=13)
   Index Cond: (salary < 10000)
(2 rows)

spr2022adb35=>
```

SELECT A2.first, A2.last
FROM agent2 A2
WHERE A2.salary < 50000;

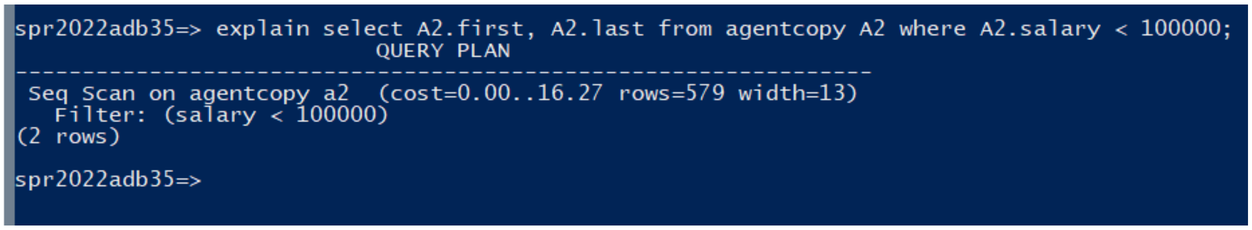
PostGre Chose an index



```
spr2022adb35=> explain select A2.first, A2.last from agentcopy A2 where A2.salary < 50000;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using salaryindex on agentcopy a2  (cost=0.28..7.91 rows=1 width=13)
   Index Cond: (salary < 50000)
(2 rows)

spr2022adb35=>
```

SELECT A2.first, A2.last
FROM agent2 A2
WHERE A2.salary < 100000;

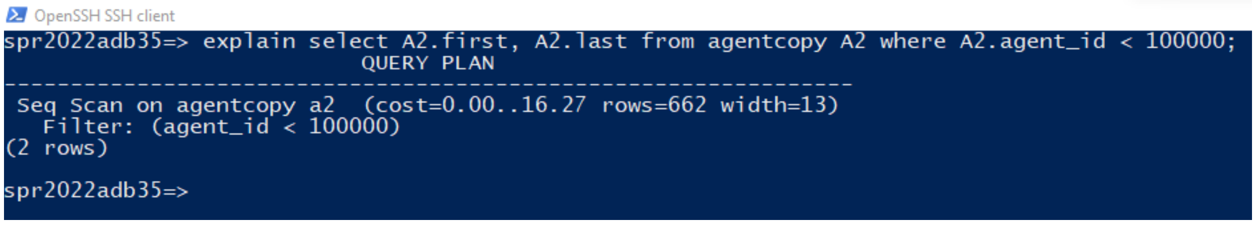
PostGre did not chose an index



```
spr2022adb35=> explain select A2.first, A2.last from agentcopy A2 where A2.salary < 100000;
                           QUERY PLAN
-------------------------------------------------------------
 Seq Scan on agentcopy a2  (cost=0.00..16.27 rows=579 width=13)
   Filter: (salary < 100000)
(2 rows)

spr2022adb35=>
```

SELECT A2.first, A2.last
FROM agent2 A2
WHERE A2.agent_id < 100000;

PostGre did not chose an index

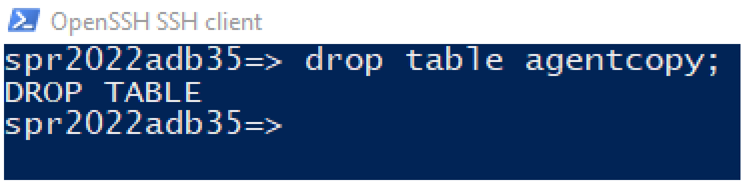


OpenSSH SSH client

```
spr2022adb35=> explain select A2.first, A2.last from agentcopy A2 where A2.agent_id < 100000;
                           QUERY PLAN
-------------------------------------------------------------
 Seq Scan on agentcopy a2  (cost=0.00..16.27 rows=662 width=13)
   Filter: (agent_id < 100000)
(2 rows)

spr2022adb35=>
```

When you have answered this question, delete the copied table using the following command:

DROP TABLE <name of copy of agent table>;


OpenSSH SSH client

```
spr2022adb35=> drop table agentcopy;
DROP TABLE
spr2022adb35=>
```

Questions 3 - 5

For each SQL query below, do the following:

a) Write a query for the pg_class table to get the number of pages and tuples in each relation. Show the query and the results. No need to repeat answers if a table is used multiple times in the questions below. That is, for each question, just get information about "new" tables.

b) List two types of joins that could be used for the query.

c) Using the formulas provided in the slides, calculate the cost of doing each type of join listed in b. **Note:** Keep in mind number of scans is a number of times of scanning; therefore, it needs to be in integer.

d) Use EXPLAIN to identify which join algorithm postgresql uses. (See Documentation: 14: EXPLAIN Also, see slide 19 in Slides 12 for and Activities for Slides 12 for information about Explain) (Also see supplementary video)

e) Report if your calculation of which join is cheapest matches postgresql's choice or not

f) Use work_mem / 8k to find # of buffer pages available to join (Documentation: 13: 19.4. Resource Consumption) (Also see supplementary video)

Question 3 (20 points):

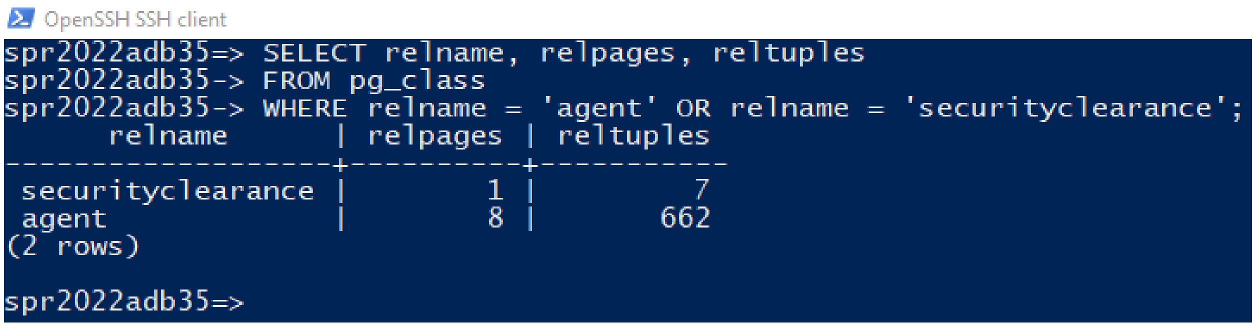
SELECT A.first, A.last, A.clearance_id
FROM agent A, securityclearance S
WHERE A.clearance_id = S.sc_id

Answer: -

a) Query: -

SELECT relname, relpages, reltuples
FROM pg_class
WHERE relname = 'agent' OR relname = 'securityclearance';

From the result of the above query, we know that
Number of pages in agent: 8
Number of tuples in agent: 662
Number of pages in securityclearance: 1
Number of tuples in securityclearance: 7



```
spr2022adb35=> SELECT relname, relpages, reltuples
spr2022adb35-> FROM pg_class
spr2022adb35-> WHERE relname = 'agent' OR relname = 'securityclearance';
      relname       | relpages | reltuples
--------------------+----------+-----------
 securityclearance |        1 |         7
 agent             |        8 |       662
(2 rows)

spr2022adb35=>
```

b) Index nested loop or Sort-Merge

c) Index Nested Loop

M = 8 pages in agent

M * p_{agent} = 662 tuples in agent

Cost = M + ((M * p_{agent}) * 2)

= 8 + (662 * 2)

= 1332 I/Os

Sort-Merge

M = 8 pages in agent

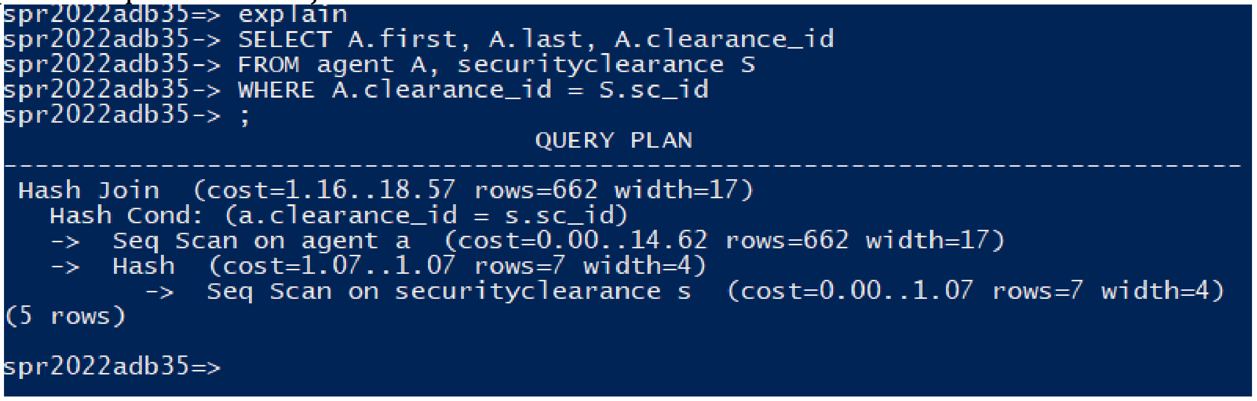
N = 1 page in securityclearance

Cost = 3 * (M + N)

= 3 * (8 + 1)

= 27 I/Os

d) postGresql uses a hash join.



```
spr2022adb35=> explain
spr2022adb35-> SELECT A.first, A.last, A.clearance_id
spr2022adb35-> FROM agent A, securityclearance S
spr2022adb35-> WHERE A.clearance_id = S.sc_id
spr2022adb35-> ;
                                  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)

spr2022adb35=>
```

e) sort-merge was the cheapest at 27 I/Os.

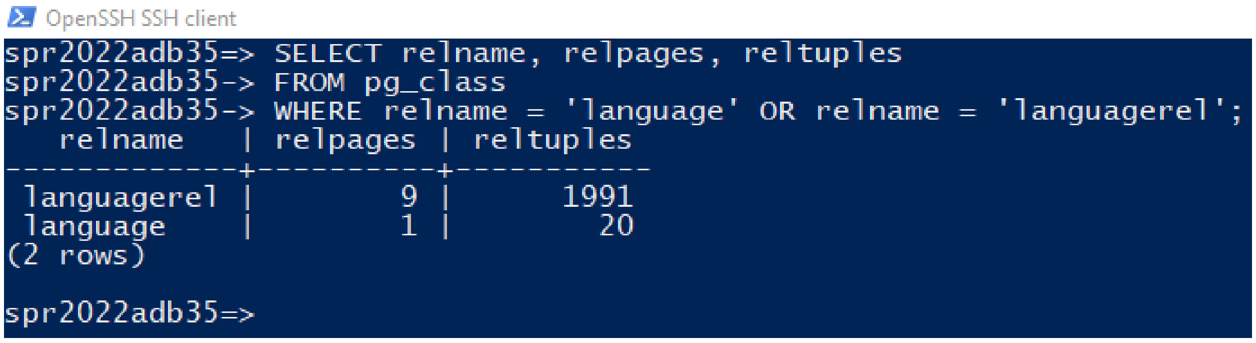
postGresql instead chose hash join which is more cheap at 18.57 I/Os.

Question 4 (20 points):
SELECT L.language L.lang_id
FROM language L, languagerel LR
WHERE L.lang_id = LR.lang_id;

Answer: -

a) SELECT relname, relpages, reltuples
   FROM pg_class
   WHERE relname = 'language' OR relname = 'languagerel';

   From the result of the above query, we know that
   Number of pages in language: 1
   Number of tuples in language: 20
   Number of pages in languagerel: 9
   Number of tuples in languagerel: 1991



OpenSSH SSH client

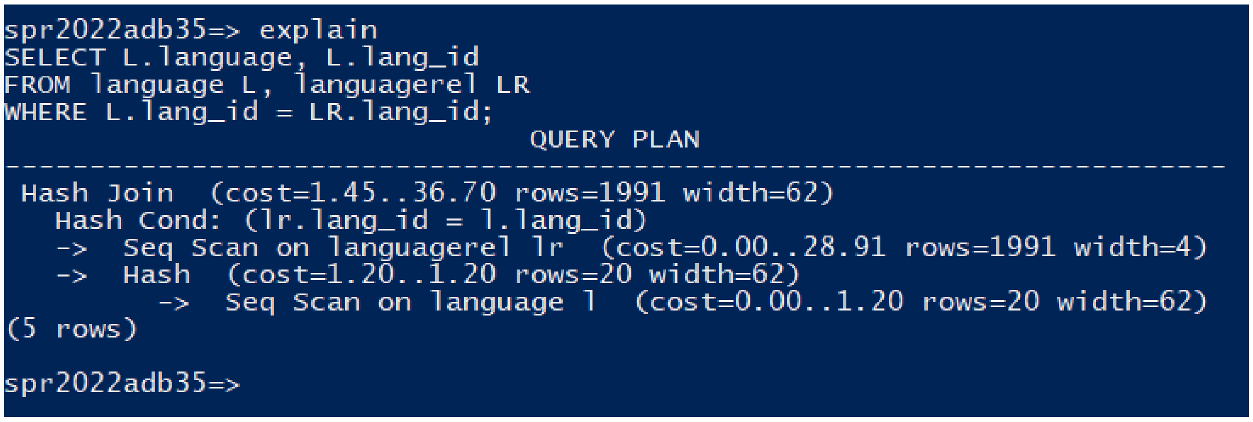
```
spr2022adb35=> SELECT relname, relpages, reltuples
spr2022adb35-> FROM pg_class
spr2022adb35-> WHERE relname = 'language' OR relname = 'languagerel';
   relname    | relpages | reltuples
-------------+----------+-----------
 languagerel |        9 |      1991
 language    |        1 |        20
(2 rows)

spr2022adb35=>
```

b) Simple Nested Loop or Page Nested Loop

c) Simple Nested Loop
   M = 1 page in language
   N = 9 pages in languagerel
   $p_{language}$ * M = 20 tuples in language
   Cost = M + ($p_{language}$ * M) * N
        = 1 + 20 * 9
        = 181 I/Os

   Paged Nested Loop
   M = 1 page in language
   N = 9 pages in languagerel
   Cost = M + M * N
        = 1 + 1 * 9
        = 10 I/Os

d) postGresql uses a hash join.



```
spr2022adb35=> explain
SELECT L.language, L.lang_id
FROM language L, languagerel LR
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)

spr2022adb35=>
```

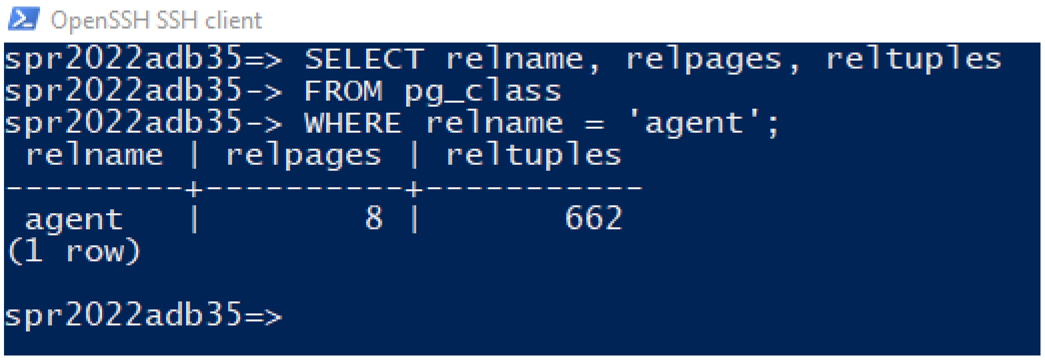
e) Of the two joins I chose, page nested loop was the cheapest at 10 I/Os. PostGresql instead chose hash join which is more expensive at 36.70 I/Os.

Question 5 (20 points):
SELECT A1.agent_id, A2.agent_id
FROM agent A1, agent A2
WHERE A1.salary > A2.salary

Answer: -

a) SELECT relname, relpages, reltuples
FROM pg_class
WHERE relname = 'agent';

From the results of the above query, we can say that: -
Number of pages in agent: 8
Number of tuples in agent: 662



OpenSSH SSH client

```
spr2022adb35=> SELECT relname, relpages, reltuples
spr2022adb35-> FROM pg_class
spr2022adb35-> WHERE relname = 'agent';
 relname | relpages | reltuples
---------+----------+-----------
 agent   |        8 |       662
(1 row)

spr2022adb35=>
```

b) The only option for this query is Block Nested Loop. This is because this query does not use equi-join.

c) Block-Nested Loop

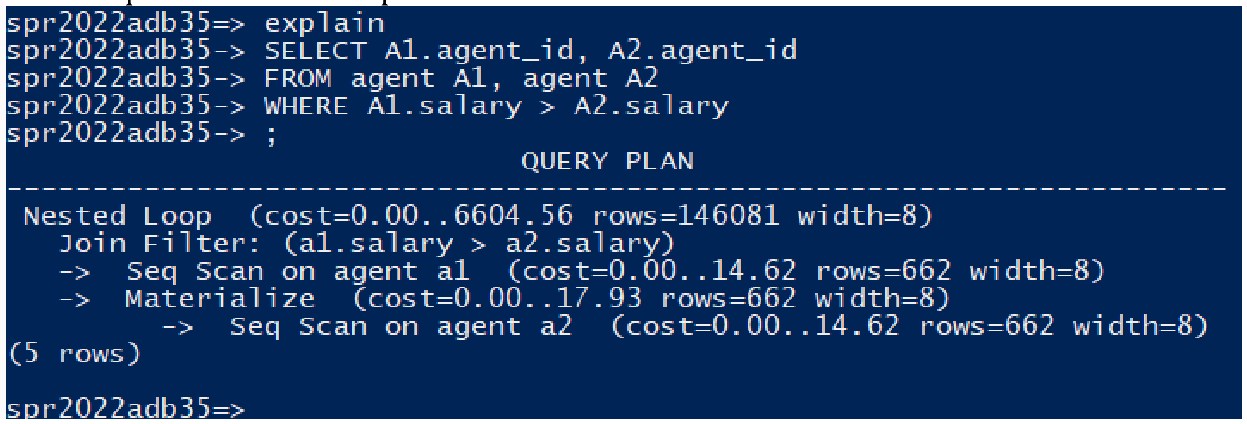
M = 8 pages in agent

N = 8 pages in agent

Now since we know that a total of 1024 buffer pages can fit in memory at one time. In our case though, we only need 10 pages in total. Therefore I am only going to use 10 pages for the requisite calculation.

BP = 10 buffer pages
Cost = M + (M / (BP - 2)) * N
= 8 + (8 / (10 - 2)) * 8
= 16 I/Os

d) PostGresql uses a nested loop.



```
spr2022adb35=> explain
spr2022adb35-> SELECT A1.agent_id, A2.agent_id
spr2022adb35-> FROM agent A1, agent A2
spr2022adb35-> WHERE A1.salary > A2.salary
spr2022adb35-> ;
                              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)

spr2022adb35=>
```

e) I chose to use block nested loop which has a cost of 16 I/Os.
PostGresql also chose a nested loop which was more expensive at 6604.56 I/Os

NOTE: Make sure that you are running question 5 against the original agent table and not the copy you created for question 2 (this is why you were supposed to drop the copy of the table at the end of question 2!)