Name: Chitradevi Maruthavanan

CS486/586 Introduction to Databases
Summer 2022 Quarter

Assignment 5 – Storage and Indexing; Query Evaluation
Due: Monday, August 8th, 11:59 pm

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)

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

Answer:

a) **name = 'Providence Park'**

The index **does not match** the name predicate. Since, we have the index on the attributes are id, maximum_capacity, field_size, but we do not have index on the name attribute.

b) **id < 3**

The index **does not match** the predicate.

c) **field_size < 18000 AND maximum_capacity > 25000**

The index **matches** the predicate.

d) **maximum_capacity < 35000 OR field_size > 17000**

The index **does not match** the predicate.

e) **Which of the above predicates will Providence Park satisfy? Which will the CenturyLink Field satisfy?**

Providence Park Predicates are:

a. name = ‘Providence Park’

b. id < 3

c. maximum_capacity < 35000 OR field_size > 17000

Century Link Predicates are:

a. id < 3

b. maximum_capacity < 35000 OR field_size > 17000

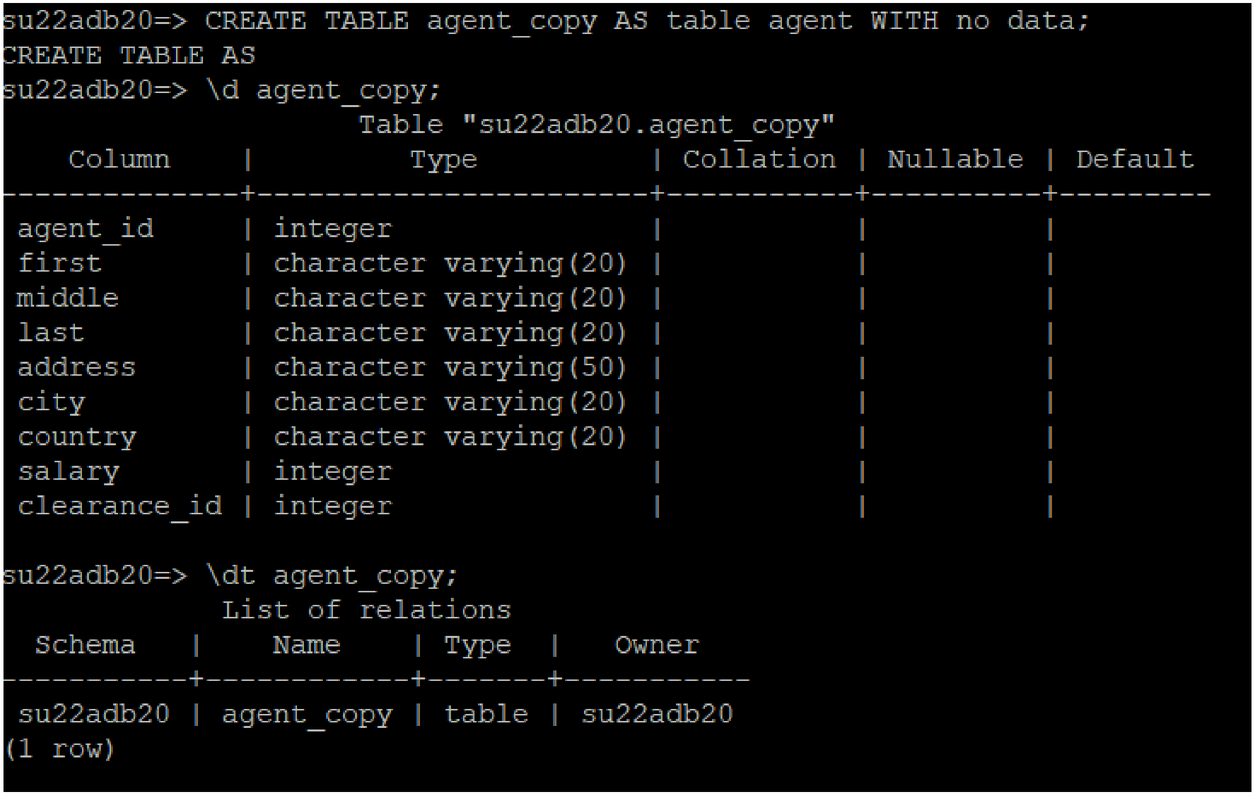
Part II: Query Plans – (80 points total)

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.

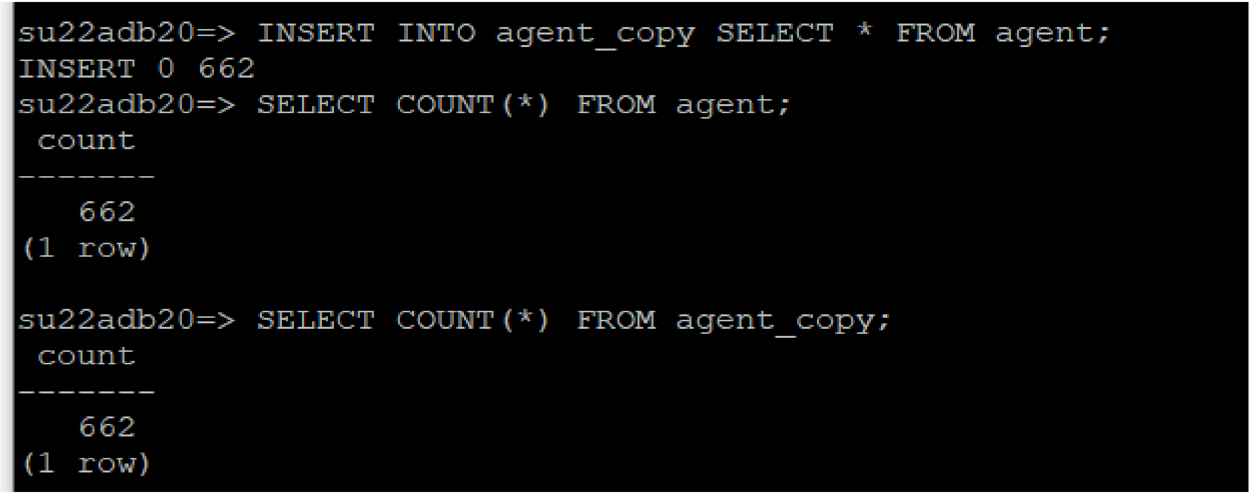
CREATE TABLE <new table name> AS TABLE <existing table> WITH NO DATA;



```
su22adb20=> CREATE TABLE agent_copy AS table agent WITH no data;
CREATE TABLE AS
su22adb20=> \d agent_copy;
                     Table "su22adb20.agent_copy"
    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               |           |          |

su22adb20=> \dt agent_copy;
             List of relations
  Schema   |    Name    | Type  |   Owner
-----------+------------+-------+-----------
 su22adb20 | agent_copy | table | su22adb20
(1 row)
```

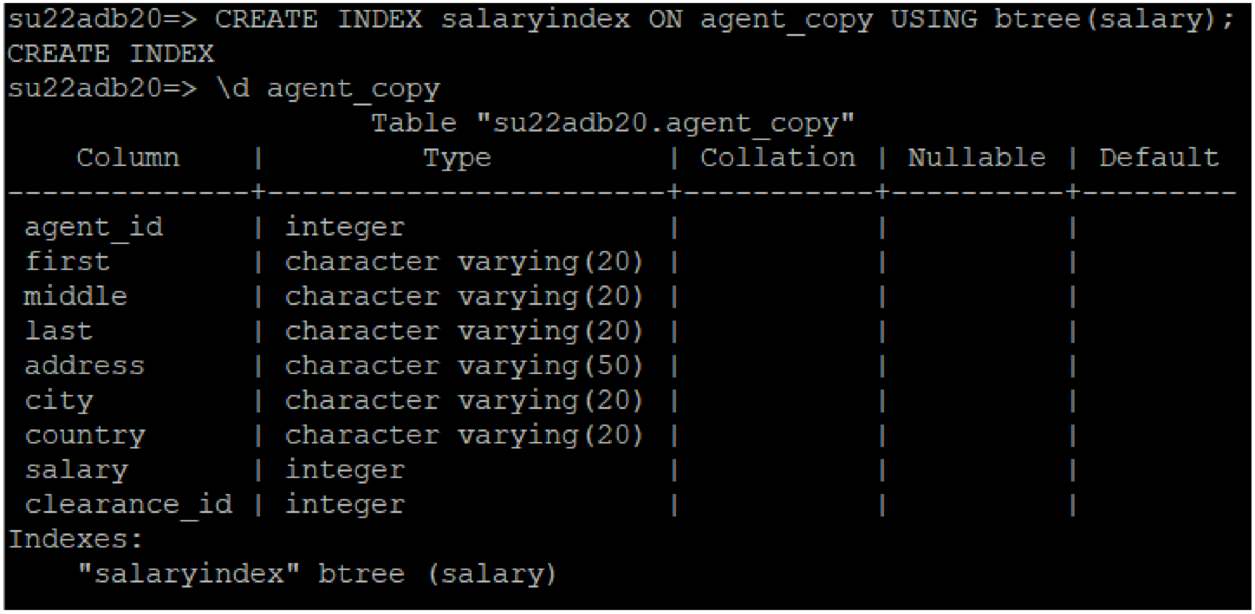
INSERT INTO <new table name> SELECT *FROM <existing table>;



```
su22adb20=> INSERT INTO agent_copy SELECT * FROM agent;
INSERT 0 662
su22adb20=> SELECT COUNT(*) FROM agent;
 count
-------
   662
(1 row)

su22adb20=> SELECT COUNT(*) FROM agent_copy;
 count
-------
   662
(1 row)
```

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 **'agent_copy'** though you may have named it something different.(You can use \help in the command line interface to get the syntax for CREATE INDEX).

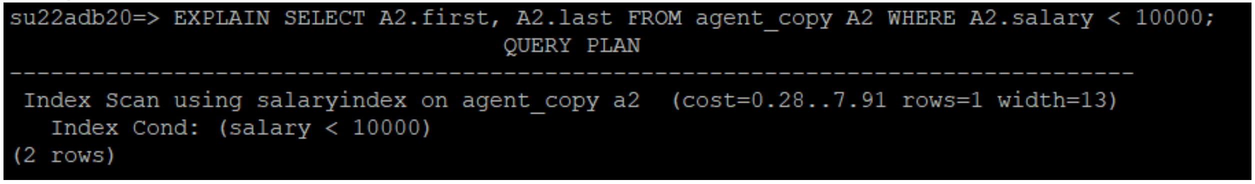


```
su22adb20=> CREATE INDEX salaryindex ON agent_copy USING btree(salary);
CREATE INDEX
su22adb20=> \d agent_copy
                     Table "su22adb20.agent_copy"
    Column    |         Type          | Collation | Nullable | Default
--------------+-----------------------+-----------+----------+---------
 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)
```

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 agent_copy A2 WHERE A2.salary < 10000;

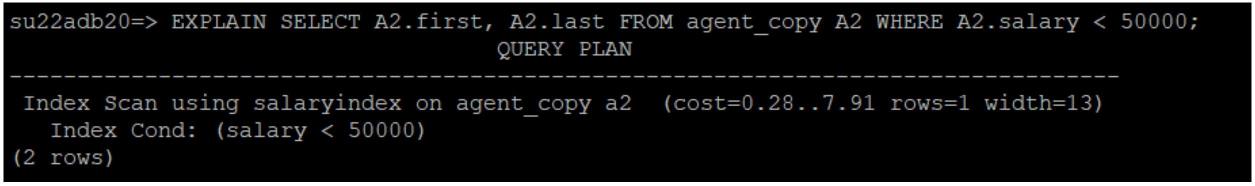
Postgres chose an index



```
su22adb20=> EXPLAIN SELECT A2.first, A2.last FROM agent_copy A2 WHERE A2.salary < 10000;
                                  QUERY PLAN
--------------------------------------------------------------------------------
 Index Scan using salaryindex on agent_copy a2  (cost=0.28..7.91 rows=1 width=13)
   Index Cond: (salary < 10000)
(2 rows)
```

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

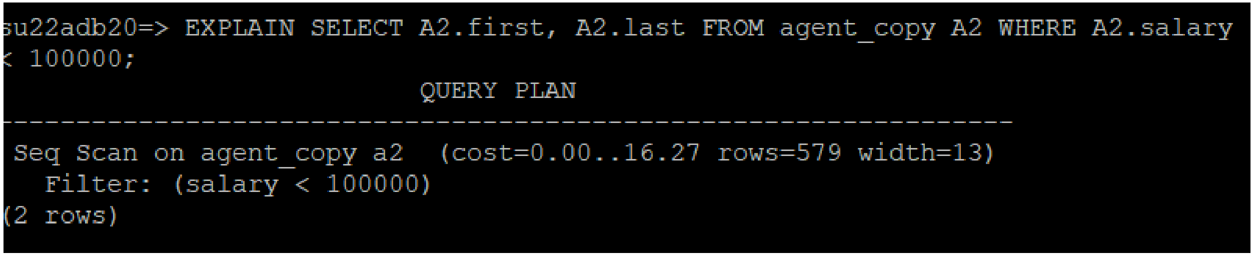
Postgres chose an index



```
su22adb20=> EXPLAIN SELECT A2.first, A2.last FROM agent_copy A2 WHERE A2.salary < 50000;
                                  QUERY PLAN
--------------------------------------------------------------------------------
 Index Scan using salaryindex on agent_copy a2  (cost=0.28..7.91 rows=1 width=13)
   Index Cond: (salary < 50000)
(2 rows)
```

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

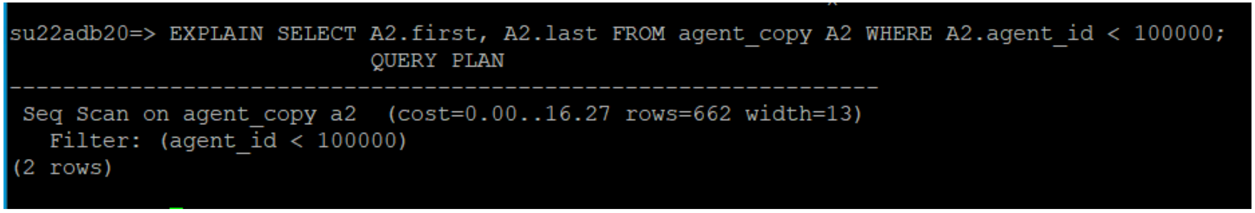
Postgres did not choose an index



```
su22adb20=> EXPLAIN SELECT A2.first, A2.last FROM agent_copy A2 WHERE A2.salary
< 100000;
                              QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on agent_copy a2  (cost=0.00..16.27 rows=579 width=13)
   Filter: (salary < 100000)
(2 rows)
```

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

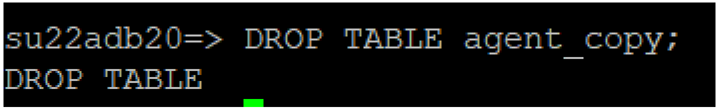
Postgres did not chose an index



```
su22adb20=> EXPLAIN SELECT A2.first, A2.last FROM agent_copy A2 WHERE A2.agent_id < 100000;
                              QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on agent_copy a2  (cost=0.00..16.27 rows=662 width=13)
   Filter: (agent_id < 100000)
(2 rows)
```

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

DROP TABLE <name of copy of agent table>;



```
su22adb20=> DROP TABLE agent_copy;
DROP TABLE
```

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)

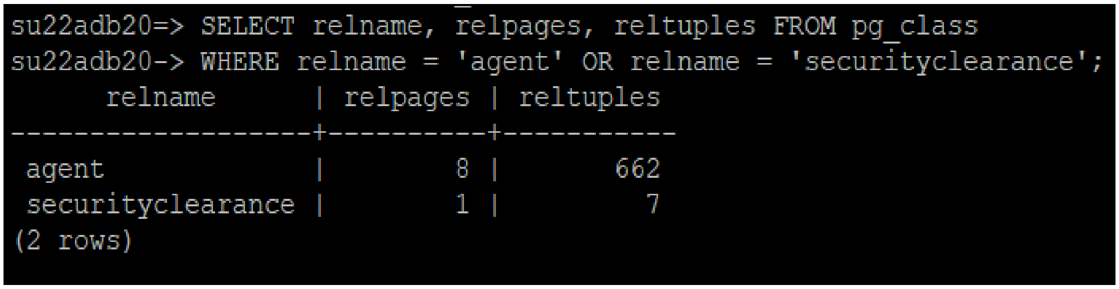
Question 3 (20 points):

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';



```
su22adb20=> SELECT relname, relpages, reltuples FROM pg_class
su22adb20-> WHERE relname = 'agent' OR relname = 'securityclearance';
      relname      | relpages | reltuples
-------------------+----------+-----------
 agent             |        8 |       662
 securityclearance |        1 |         7
(2 rows)
```

Number of pages in agent: 8

Number of tuples in agent: 662

Number of pages in security clearance: 1

Number of tuples in security clearance: 7

b) Index nested loop or Sort-Merge

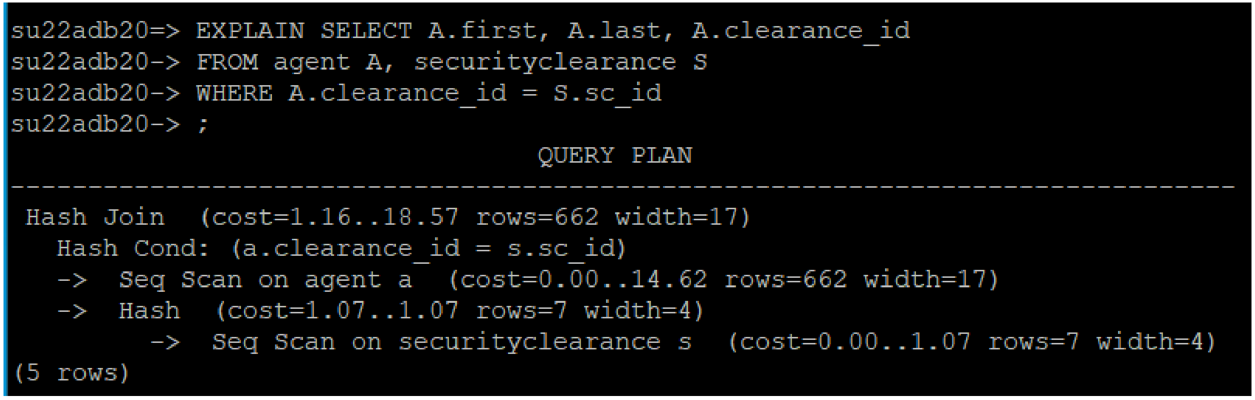
c) **Index Nested Loop**

M = 8 pages in agent
$M * P_A$ = 662 tuples in agent
Cost = $M + ((M * P_A) * 2)$
= 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

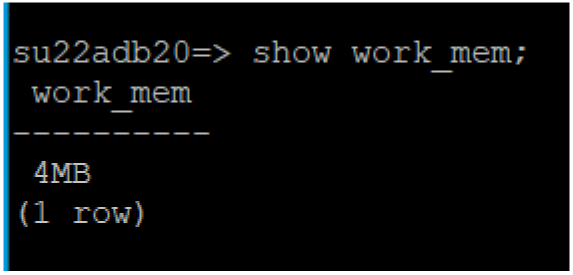
d) postgresql uses a hash join.



```
su22adb20=> EXPLAIN SELECT A.first, A.last, A.clearance_id
su22adb20-> FROM agent A, securityclearance S
su22adb20-> WHERE A.clearance_id = S.sc_id
su22adb20-> ;
                                  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)
```

e) Here Postgre hash join was the cheapest at 18.57 I/Os.
Of my 2 choices Sort-Merge was the cheaper at 27 I/Os than Index-nested loop at 1332 I/Os

f) **To find # of buffer pages available to join**



```
su22adb20=> show work_mem;
 work_mem
----------
 4MB
(1 row)
```

Number of buffer pages = $\frac{\text{show work\_mem}}{\text{8KB}} = \frac{\text{4MB}}{\text{8KB}}$ = 512 Buffer Pages

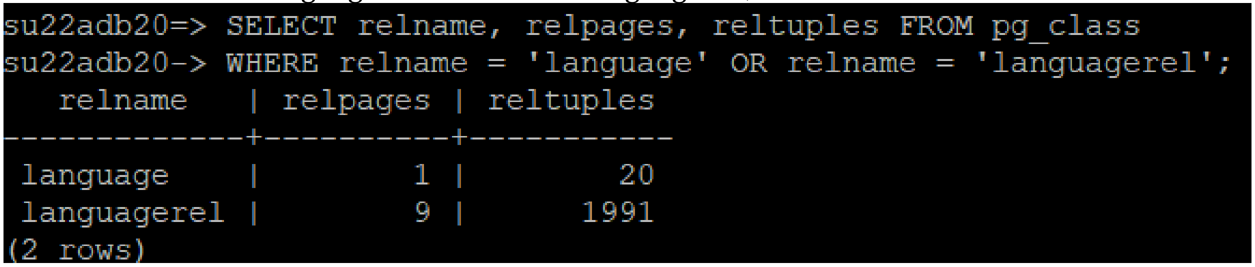
Question 4 (20 points):

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

Answer:

a) Query:

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



```
su22adb20=> SELECT relname, relpages, reltuples FROM pg_class
su22adb20-> WHERE relname = 'language' OR relname = 'languagerel';
   relname    | relpages | reltuples
--------------+----------+-----------
 language     |        1 |        20
 languagerel  |        9 |      1991
(2 rows)
```

Number of pages in language: 1

Number of tuples in language: 20

Number of pages in languagerel: 9

Number of tuples in languagerel: 1991

b) Simple Nested Loop or Page Nested Loop

c) Simple Nested Loop
M = 1 page in language
N = 9 pages in languagerel
P_A * M = 20 tuples in language

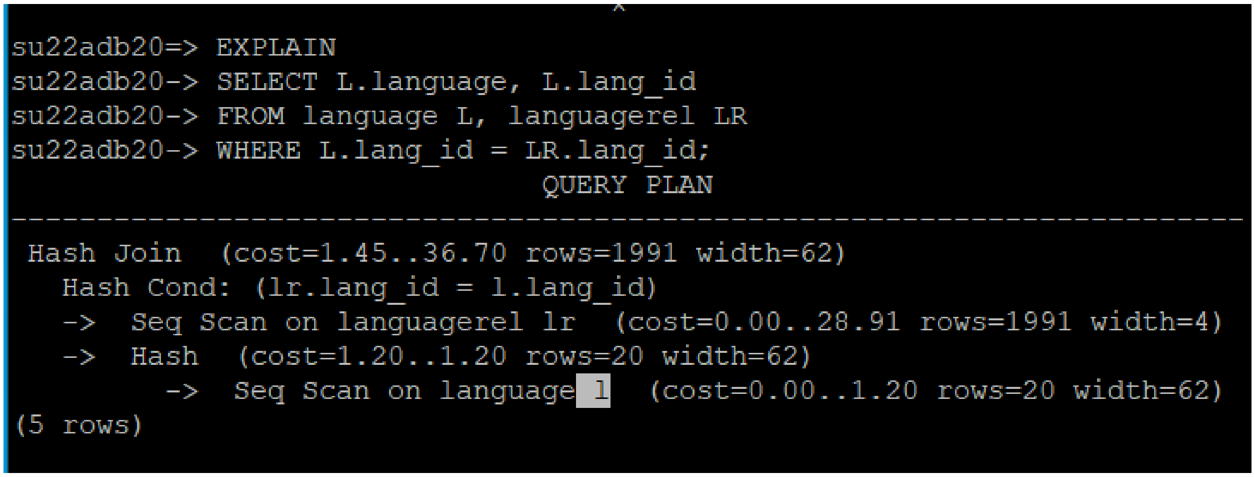
Cost = M + (P_A * M) * N
= 1 + 20 * 9
= 181 I/Os

Paged Nested Loop

M = 1 page in language
N = 9 pages in languagerel

Cost = M + M * N
= 1 + 1 * 9
= 10 I/Os

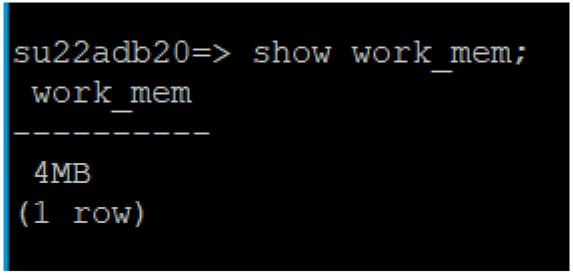
d) postgresql uses a hash join.



```
su22adb20=> EXPLAIN
su22adb20-> SELECT L.language, L.lang_id
su22adb20-> FROM language L, languagerel LR
su22adb20-> 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)
```

e) Here Postgresql chose hash join. The cost is more expensive at 36.70 I/Os.
Of my 2 choices, page nested loop was the cheapest at 10 I/Os.
(simple nested loop was 181 I/Os)

f)To find # of buffer pages available to join



```
su22adb20=> show work_mem;
 work_mem
----------
 4MB
(1 row)
```

Number of buffer pages = $\frac{\text{show work\_mem}}{\text{8KB}} = \frac{\text{4MB}}{\text{8KB}}$ = 512 Buffer Pages

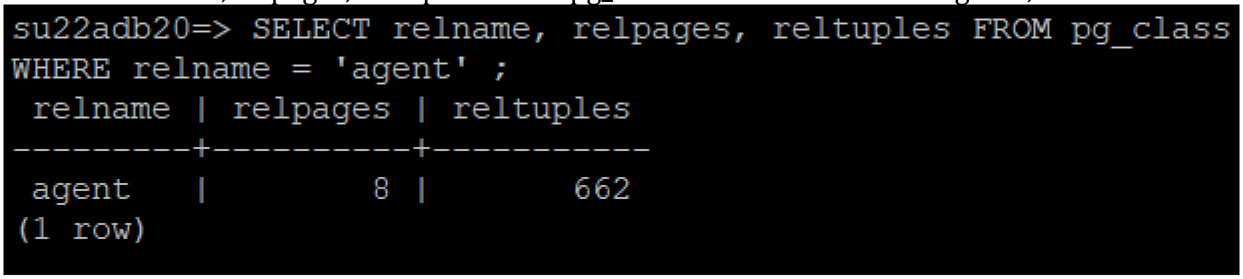
Question 5 (20 points):

SELECT A1.agent_id, A2.agent_id
FROM agent A1, agent A2
WHERE A1.salary > A2.salary

Answer:

a) Query:

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



```
su22adb20=> SELECT relname, relpages, reltuples FROM pg_class
WHERE relname = 'agent' ;
 relname | relpages | reltuples
---------+----------+-----------
 agent   |        8 |       662
(1 row)
```

Number of pages in agent: 8

Number of tuples in agent: 662

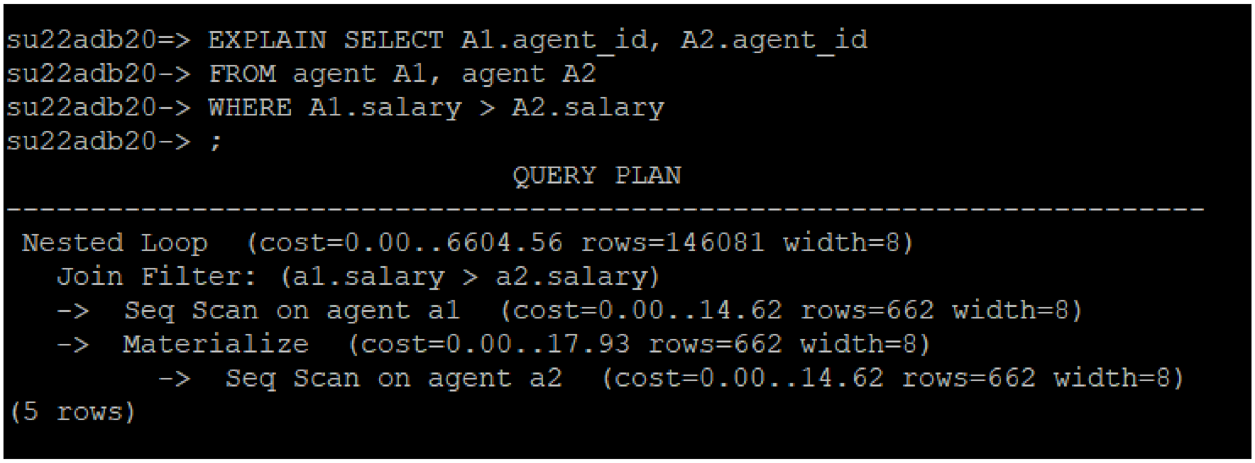
b) Block Nested loop is the only option because this query does not use equi-join.

c) Block Nested loop

M = 8 pages in agent
N = 8 pages in agent
BP = 10 buffer pages

$$\begin{aligned} \text{Cost} &= M + (M / (BP - 2)) * N \\ &= 8 + (8 / (10 - 2)) * 8 \\ &= 16 \text{ I/Os} \end{aligned}$$

d) postgresql uses a NESTED LOOP join.

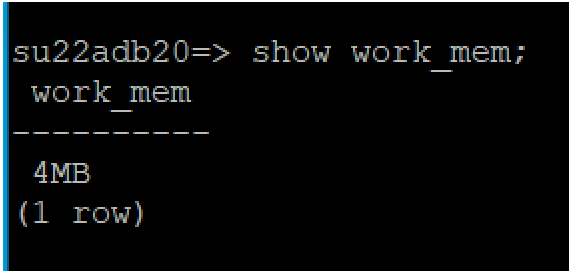


```
su22adb20=> EXPLAIN SELECT A1.agent_id, A2.agent_id
su22adb20-> FROM agent A1, agent A2
su22adb20-> WHERE A1.salary > A2.salary
su22adb20-> ;
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop  (cost=0.00..6604.56 rows=146081 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)
```

e) Here Postgre chose a nested loop. The cost was more expensive at 6604.56 I/Os

Of my choice, **block nested loop** was cheapest at 16 I/Os

f) **To find # of buffer pages available to join**



```
su22adb20=> show work_mem;
 work_mem
-----------
 4MB
(1 row)
```

Number of buffer pages = $\frac{\text{show work\_mem}}{\text{8KB}} = \frac{\text{4MB}}{\text{8KB}}$ = 512 Buffer Pages

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!)