Ajay Babu Gorantla
Kelsey Werner

CS486/586 Introduction to Databases
Fall 2021 Quarter

Assignment 5 – Storage and Indexing; Query Evaluation
Due: Friday, Nov 19, 11:59 pm

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.
- 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)

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'**
The index does not match the predicate. There is an index on the id attribute and on the maximum capacity and field size attributes, but there is not an index on the name attribute.

b) **id < 3**
The index does match the 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 this is an OR clause. The index could be used to find all of the stadiums with a maximum_capacity above 35000, but the index couldn't be used for the field_size given that the index is first sorted by maximum_capacity.

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

Providence Park will satisfy the following predicates:
- a
- b (assuming that the ids start at 1 and are assigned sequentially in ascending order)
- d

Century Link will satisfy the following predicates:
- b (assuming that the ids start at 1 and are assigned sequentially in ascending order)
- d

Part II: Query Plans – 80 points

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: https://www.postgresql.org/docs/12/catalog-pg-class.html

Question 2 (10 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;

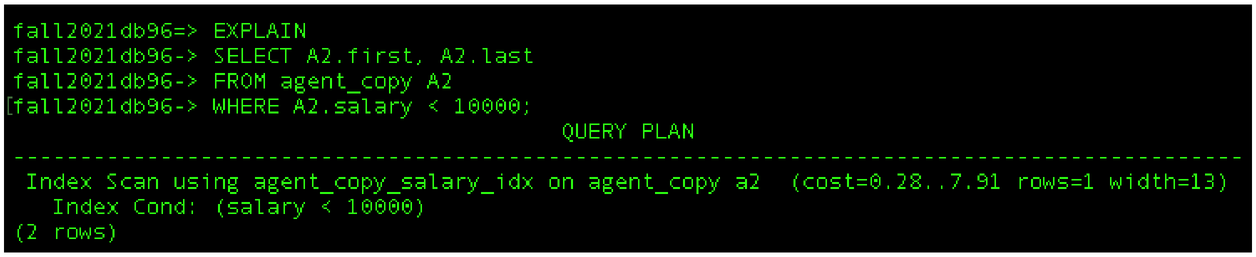
INSERT INTO <new table name>
SELECT *
FROM <existing table>;

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).

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;

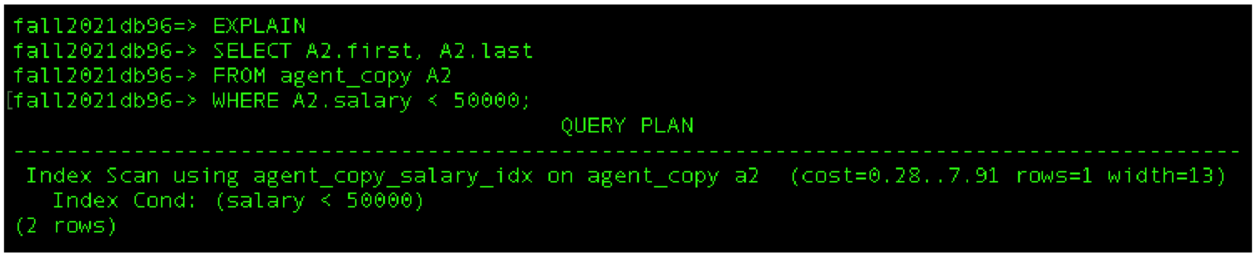
Postgres chose an index.



```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A2.first, A2.last
fall2021db96-> FROM agent_copy A2
[fall2021db96-> WHERE A2.salary < 10000;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using agent_copy_salary_idx 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 agent2 A2
WHERE A2.salary < 50000;

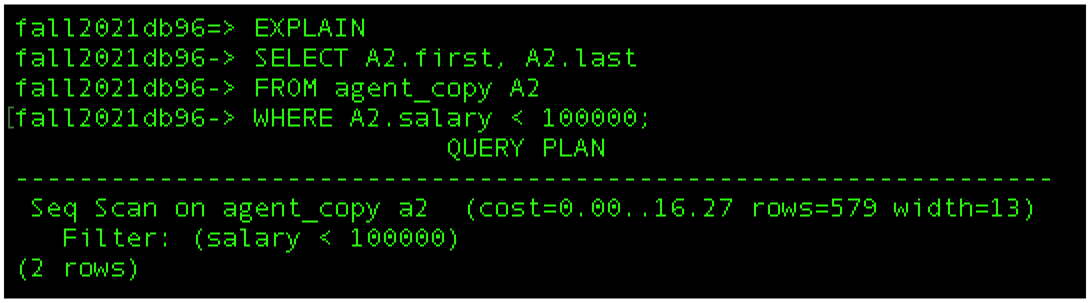
Postgres chose an index.



```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A2.first, A2.last
fall2021db96-> FROM agent_copy A2
[fall2021db96-> WHERE A2.salary < 50000;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using agent_copy_salary_idx 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 agent2 A2
WHERE A2.salary < 100000;

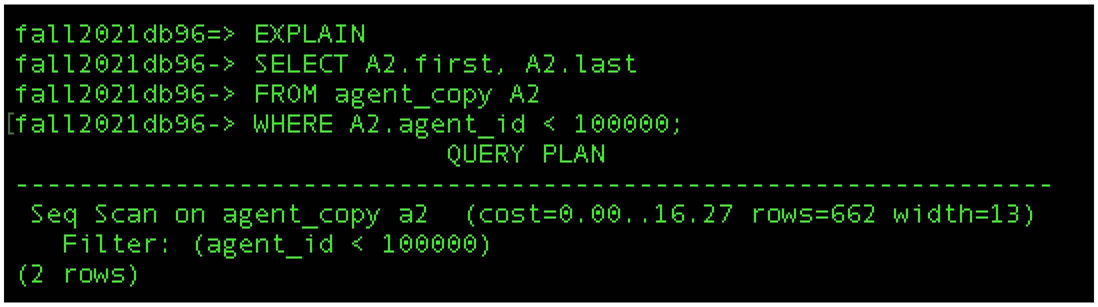
Postgres did not choose an index.



```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A2.first, A2.last
fall2021db96-> FROM agent_copy A2
[fall2021db96-> 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 agent2 A2
WHERE A2.agent_id < 100000;

Postgres did not choose an index.



```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A2.first, A2.last
fall2021db96-> FROM agent_copy A2
[fall2021db96-> 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>;

Questions 3 - 5
For each SQL query below, do the following:

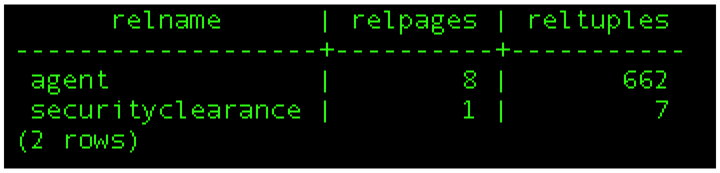
a) **Write a query for the pg_class table to get the number of pages and tuples in each relation. Show the query and the results. No need to repeat answers if a table is used multiple times in the questions below. That is, for each question, just get information about "new" tables.**
b) **List two types 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.
d) Use EXPLAIN to identify which join algorithm postgresql uses
e) Report if your calculation of which join is cheapest matches postgresql's choice or not

Question 3 (20 points):
SELECT A.first, A.last, A.clearance_id
FROM agent A, securityclearance S
WHERE A.clearance_id = S.sc_id;

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

Number of pages in agent: 8
Number of tuples in agent: 662
Number of pages in securityclearance: 1
Number of tuples in securityclearance: 7



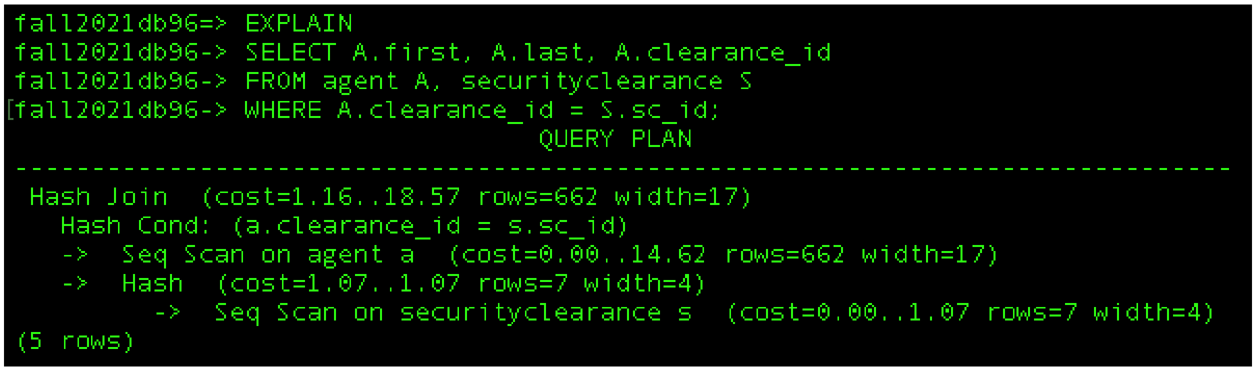
```
      relname      | relpages | reltuples
-------------------+----------+-----------
 agent             |        8 |       662
 securityclearance |        1 |         7
(2 rows)
```

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.



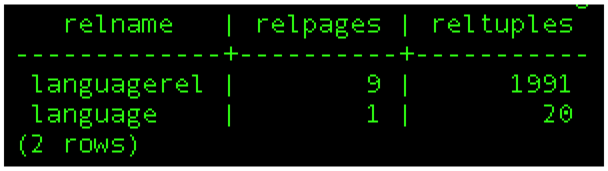
```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A.first, A.last, A.clearance_id
fall2021db96-> FROM agent A, securityclearance S
[fall2021db96-> WHERE A.clearance_id = S.sc_id;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 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)
```

e) Of the two joins I chose, sort-merge was the cheapest at 27 I/Os. postgresql instead chose hash join which is even cheaper 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;

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

Number of pages in language: 1
Number of tuples in language: 20
Number of pages in languagerel: 9
Number of tuples in languagerel: 1991



```
   relname    | relpages | reltuples
-------------+----------+-----------
 languagerel |        9 |      1991
 language    |        1 |        20
(2 rows)
```

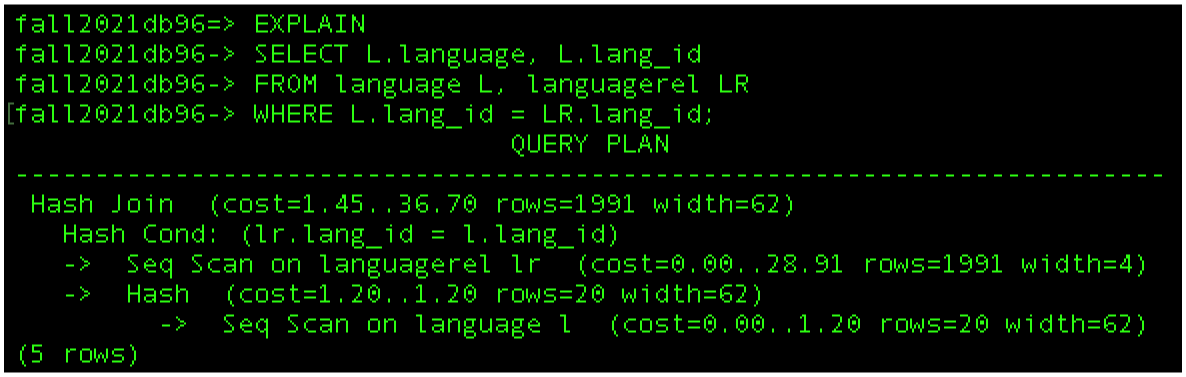
b) Simple Nested Loop or Paged Nested Loop

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.



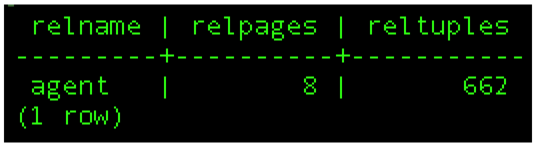
```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT L.language, L.lang_id
fall2021db96-> FROM language L, languagerel LR
[fall2021db96-> WHERE L.lang_id = LR.lang_id;
                                QUERY PLAN
--------------------------------------------------------------------------
 Hash Join  (cost=1.45..36.70 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) Of the two joins I chose, page nested loop was the cheapest at 10 I/Os. postgresql instead chose hash join which is actually 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;

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

Number of pages in agent: 8
Number of tuples in agent: 662



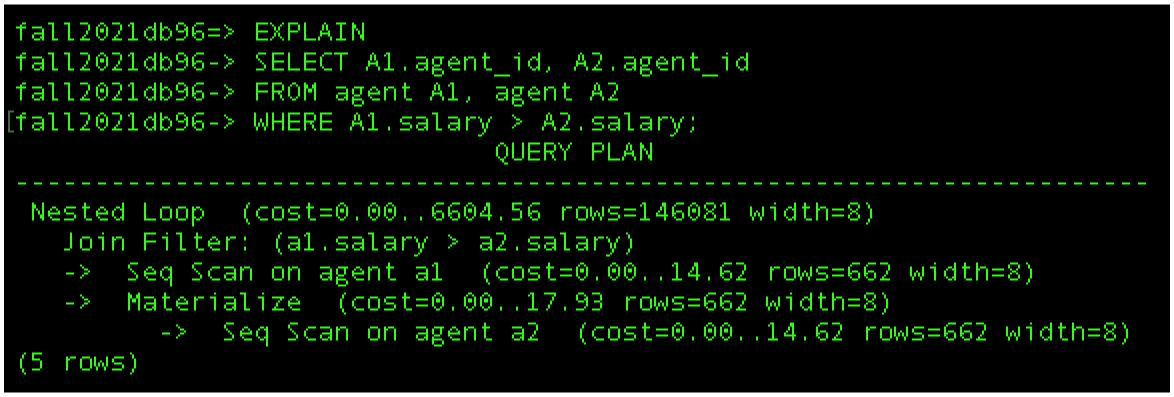
```
 relname | relpages | reltuples
---------+----------+-----------
 agent   |        8 |       662
(1 row)
```

b) Block Nested Loop is the only option for this query because this query does not use an equi-join. A Block Nested Loop is the only join algorithm we've studied so far that can be used with any joins besides equi-joins.

c) Block Nested Loop

M = 8 pages in agent
N = 8 pages in agent
BP = 10 buffer pages (total of 1024 buffer pages can fit in memory at one time, but since we only need 10 pages total, we are only using 10 pages)
Cost = M + (M / (BP - 2)) * N
= 8 + (8 / (10 - 2)) * 8
= 16 I/Os

d) postgresql uses a nested loop.



```
fall2021db96=> EXPLAIN
fall2021db96-> SELECT A1.agent_id, A2.agent_id
fall2021db96-> FROM agent A1, agent A2
[fall2021db96-> 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)
```

e) I chose the block nested loop which had a cost of 16 I/Os. postgresql also chose a block nested loop which was actually 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!)