**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):***

1. name = 'Providence Park'
2. id < 3
3. field\_size < 18000 AND maximum\_capacity > 25000
4. maximum\_capacity < 35000 OR field\_size > 17000
5. Which of the above predicates will Providence Park satisfy? Which will the CenturyLink Field satisfy?

**Answer: -**

1. **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.

1. **Id < 3**

The index does not match predicate.

1. **Field\_size < 18000 AND maximum\_capacity > 25000**

The index does match the predicate

1. **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.

1. **Providence Park** will satisfy the following predicates: -
   1. **Name = ‘Providence Park’**
   2. **Id < 3.** This is conditional and will satisfy only if ids start with 1 and are present in sequential ascending order.
   3. **Maximum\_capacity < 35000 or field\_size > 17000**

**CenturyLink** will satisfy the following predicates: -

1. **Id<3.** This is conditional and will satisfy only if ids start with 1 and are present in sequential ascending order.
2. **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](https://www.postgresql.org/docs/14/catalog-pg-class.html) (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;

A picture containing diagram

Description automatically generated

INSERT INTO <new table name>

SELECT \*

FROM <existing table>;

Graphical user interface, text, application

Description automatically generated

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).

A picture containing table

Description automatically generated

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**Text

Description automatically generated

SELECT A2.first, A2.last

FROM agent2 A2

WHERE A2.salary < 50000;

**PostGre Chose an index**

Text

Description automatically generated

SELECT A2.first, A2.last

FROM agent2 A2

WHERE A2.salary < 100000;

**PostGre did not chose an index**

Text

Description automatically generated with medium confidence

SELECT A2.first, A2.last

FROM agent2 A2

WHERE A2.agent\_id < 100000;

**PostGre did not chose an index**

Text

Description automatically generated

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

DROP TABLE <name of copy of agent table>;

A picture containing text

Description automatically generated

***Questions 3 - 5***

For each SQL query below, do the following:

1. 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.
2. List two types of joins that could be used for the query.
3. 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.
4. Use EXPLAIN to identify which join algorithm postgresql uses. (See [Documentation: 14: EXPLAIN](https://www.postgresql.org/docs/14/sql-explain.html) Also, see slide 19 in Slides 12 for and Activities for Slides 12 for information about Explain) (Also see supplementary video)
5. Report if your calculation of which join is cheapest matches postgresql’s choice or not
6. Use work\_mem / 8k to find # of buffer pages available to join ([Documentation: 13: 19.4. Resource Consumption](https://www.postgresql.org/docs/14/runtime-config-resource.html)) (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: -**

1. 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

**Timeline

Description automatically generated**

1. Index nested loop or Sort-Merge
2. Index Nested Loop

M = 8 pages in agent

M \* pagent = 662 tuples in agent

Cost = M + ((M \* pagent) \* 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

1. postGresql uses a hash join.

**Text

Description automatically generated**

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

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

1. 500  
   Chart

   Description automatically generated with low confidence

Number of buffer pages = 500

***Question 4 (20 points):***

SELECT L.language L.lang\_id

FROM language L, languagerel LR

WHERE L.lang\_id = LR.lang\_id;

**Answer: -**

1. 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

Text

Description automatically generated

1. Simple Nested Loop or Page Nested Loop
2. Simple Nested Loop

M = 1 page in language

N = 9 pages in languagerel

planguage \* M = 20 tuples in language

Cost = M + (planguage \* 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

1. postGresql uses a hash join.

Text

Description automatically generated

1. 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.

1. 500

Chart

Description automatically generated with low confidence

Number of buffer pages = 500

***Question 5 (20 points):***

SELECT A1.agent\_id, A2.agent\_id

FROM agent A1, agent A2

WHERE A1.salary > A2.salary

Answer: -

1. 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

Timeline

Description automatically generated

1. The only option for this query is Block Nested Loop. This is because this query does not use equi-join.
2. 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

1. PostGresql uses a nested loop.

Text

Description automatically generated

1. 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

1. 500

Chart

Description automatically generated with low confidence

Number of buffer pages = 500

**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!)**