COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CMPS 451 – Database Management Systems
Fall 2023
Course project – Phase 2 (query cost estimator)

The submission details:
Your report should include:

a. Group members names, number and emails
b. A description of your application, in terms of software, APIs, …, etc. that you have chosen.
c. A link to your application (you will demonstrate this application via a discussion session between you and me starting Wednesday 8/11/2023 over MS Teams or in class)
d. Screen shots which show samples of the statistics and samples of query execution plans
e. Weight of this phase is 15 grades.

Grading Rubrics

1. A working application
2. Proper coverage of metadata and statistics.
3. Cover of algorithms that can be applied to the select operation
4. Cover algorithms that can be applied to join algorithm
5. Well organized and thorough report describing your work
6. Perform well in the discussion session.

Structure of the query cost estimator

Meta data and statistics:

(1) From your relational database schema you chose in phase 1, pick two tables that have at least one relationship between them.
(2) For each of the two tables and their respective relationships store metadata and statistics necessary for the query optimizer to choose a proper execution plan. You can refer to the textbook and slides to determine the list of features that should be stored. Assume reasonable values for each of the features you decide to store.
(3) The arrangement of these statistics could be done as relational tables and stored in a DBMS, the one you chosen for phase 1, or could be designed as files. In the first case, your metadata will be like an application hosted on the DBMS; you need to connect your frontend application with the database using proper API. In the second case, your cost-estimator is a stand-alone application which means all your metadata is stored in local files and accessed by your program. **I prefer the first option**. There is a third option in which you will rely on the catalog created by the DBMS for your schema. In this case,

you need to provide reasonable data sizes, enforce minimum number of page buffers, and create indexes on the tables. You are free to choose the programming language for the application.

The application

The cost estimator

1. It is a standalone application or could be a front end to a DBMS.
2. The cost-estimator will accept queries that involve SELECT or JOIN operations. For the SELECT operation, focus only on selection using a primary key and equality operator, selection using a primary key with range operator, selection on a non-primary key using equality operator and selection on non-primary key with range operator. For JOIN operation consider only equi-join operation.
3. The format of proving the query to your cost estimator is left to you. Your estimator can support a natural language-based interface which allows you to write the SQL SELECT statement and then parse this statement to extract the proper fields and operations. On the other hand, you can rely on a graphical user interface to read the required details. Last, you can use a series of input/read statements that will eventually provide the necessary data to your cost estimator.
4. Once the query is fed to your query estimator, the estimator will explore all possible query plans based on the form of the query and the stored statistics.
5. The output of the query estimator will be the query information, the list of possible execution plans with the estimated cost for each of them.
6. The estimator should also support utility functions such displaying the statistics associated with relations.

how we pass the query to our app . program

we feed the input, run it to all the query plans and chose the best

time it took to execute query

SELECT OPERATION:
primary key and equality operator,
primary key with range operator,
non-primary key using equality operator
non-primary key with range operator.

JOIN OPERATION:
operation consider only equi-join operation

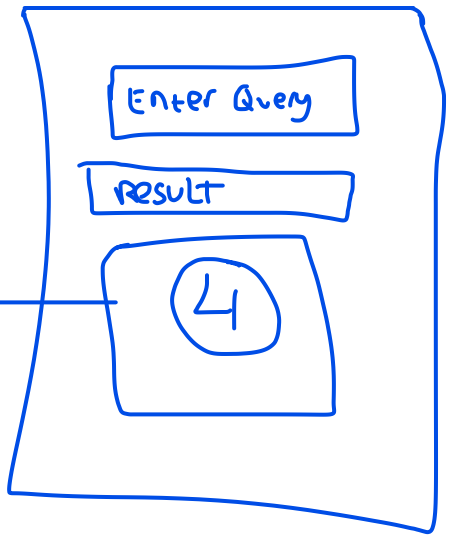
S1, S4, J2 -> Moh
S2, S6, J3 -> Ridhwan
S3b, J1, J4 -> Diyan

Fname, Lname make it unique, and dont use Minit
then convert to JSON, (both employee and project)

convert pseudocode algorithm to kotlin language

S1 - Linear Search
S2 - Binary Search
S3b - hash key
S4 - (?) Index searches needs double checking
S6 - Secondary index (B+Tree) for equality

J1 - Nested loop join
J2 - (?) Indexed Based nested loop join
J3 - Sort Merge
J4 - Partition hash join



In descending order