CS4432 - Project 3: Implementing a Simplified Version of Join and Aggregation Algorithms in DBMSs

Project to be done individually.
Programming Language: Java (Better use Java 8 (JRE 1.8))
Release Date: April 24, 2025
Due Date: May 5, 2025 (11:59PM)
Total Points: 100 Points

****SELECT ONLY TWO of the THREE PROBLEMS BELOW****

Important: Read the entire document to understand all the requirements before you start your design and implementation.

Generative AI Usage Instruction

You are permitted to use generative artificial intelligence tools (such as ChatGPT, etc.) to assist in completing the assignment. However, when using these tools, you must clearly state the following in your assignment:

1. How the generative artificial intelligence tool was used.
2. How the generated answers were verified to be correct.

Section 1: Overview

In this project, you will implement some of database operators, more specifically *join* and *aggregation*, to answer a SQL query.

Dataset Format:
In this project, you will work with two datasets (Dataset-A, Dataset-B)

Dataset-A

- The directory containing the data is named "**Project3Dataset-A**". You can hardcode this name in your project. It is the name that TAs will use when doing the testing. This directory should be under the working directory.
- The dataset directory contains 99 files (A1…A99), each file contains 100 records, and each record is 40 bytes. The record format for a record #j in file #i is (for *i* use two digits like 01, and for *j* use three digits like 001):
  //The only difference from Project 2 is the 1st letter in the record (it is "A" instead of "F")

  Ai-Recj, Namej, addressj, *RandomV*…

  where *RandomV* is a four digit random number between 0001 and 0500. Since the number of records in the entire dataset is around 10,000, then each value within the range of 0001 and 0500 is expected to appear in the dataset (on average) 20 times, but it can be more or less. Also, each record ends with three dots "…" to complete to 40 bytes.
  //Compared to Project 2, the range for RandomV is much smaller (only from 1 to 500).

- It is important to highlight that index "j" resets and starts from "001" in each file. This is important especially in Section 4 (the aggregation section).

Dataset-B

- The directory containing the data is named "**Project3Dataset-B**". You can hardcode this name in your project. It is the name that TAs will use when doing the testing. This directory should be under the working directory.
- The dataset will contain the same exact content as Dataset-A. The only difference is that the 1st letter of each record is "B" instead of "A". Also the file names will be B1…B99

******SELECT ONLY TWO of the THREE PROBLEMS BELOW*****

Section 2 (Building Hash-Based Join) 50 Points

Command: The following command (SQL statement) is what your program will receive to trigger the execution of the hash-based join.

```
"SELECT A.Col1, A.Col2, B.Col1, B.Col2  FROM  A, B WHERE A.RandomV = B.RandomV"
```

where "A" refers to Dataset-A, "A.Col1" and "A.Col2" refer to columns 1 and 2 in the dataset. The same applies to "B" and its dataset. The syntax of the command is fixed, i.e., nothing will change when testing.

In this part, you need to write code that:

- Builds a hash table on Dataset-A. The hash table should have 50 buckets. The buckets will store the entire record content.

- The hashing of each record to determine the corresponding bucket should be based on the join column. Refer to the SQL commands to know which column is the join column.
- Then, make a loop to read Dataset-B file-by-file and record-by-record. For each record (say *r*) apply the same hash function on the join column to know which bucket you should check from Dataset-A (say bucket# *K*)
- Now you need to apply the join condition between *r* and each records in bucket K. If the join condition is met, i.e., `A.RandomV = B.RandomV,` then you need to produce an output record with the needed columns (see the SQL command for the columns).
- It is up to you to either maintain the records in each bucket sorted (based on the join column) or you leave them unsorted. If you keep them sorted, then the search in the previous step should be more efficient (use binary search). //In your report indicate which design choice you made

What to produce as output:

1) Print out the execution time taken to execute the command (in millisecond)
2) Print out the qualifying records (only the columns specified in the query)

Section 3 (Building Block-Level Nested-Loop Join) 50 Points

Command: The following command (SQL statement) is what your program will receive to trigger the execution of the Nested-Loop join.

```
"SELECT count(*)  FROM  A, B WHERE A.RandomV > B.RandomV"
```

This command will report the count of records satisfying the join condition. Since the join condition is not equality, then hash-based join cannot be used. But nested loop join can be used. The syntax of the command is fixed, i.e., nothing will change when testing.

In this part, we assume the available memory is limited and can only hold at most the content of one file. Therefore, you need to write code that:

- Loops over each file in Dataset-A, and for each file do:
  - Store the records of that file in memory (put them in some data structure such as an array).
  - Read the entire Dataset-B, file-by-file and record-by-record, and compare each record with the records you have in memory from Dataset-A
  - Maintain the count of the records matching the join condition
- Then, retrieve the next file from Dataset-A and repeat the process.

What to produce as output:

1) Print out the execution time taken to execute the command (in millisecond)
2) Print the count of the qualifying records

Section 4 (Hash-Based Aggregation) 50 Points

Command: The following command (SQL statement) is what your program will receive to trigger the execution of the aggregation operator.

```
"SELECT Col2, <AggregationFunction(ColumnID)>  FROM  <Dataset> GROUP BY Col2"
```

where:

- *<Dataset>:* Is a placeholder. The actual value put in the command can be either **A** or **B**, referring to Dataset-A or Dataset-B, respectively.
- *<AggregationFunction(ColumnID)>*: Is a placeholder. The actual value can be any of the aggregation functions **SUM(RandomV)** or **AVG(RandomV)**
- Col2: Is part of the syntax, i.e., it will not change. It is referring to the 2nd column in the dataset (the "*name*" column).
- The meaning of the query is that all records in the dataset having the same value in the 2nd column should form one group on top of which the aggregation function is applied. Each group should produce one output record consisting of Col2 value along with the output from the aggregation function. (Similar to the standard SQL).
- To implement the aggregation query, you should maintain a hash table, where each distinct group should have an entry in this table. Then, as you scan the dataset, you need to keep updating the aggregation value maintained in the hash table. After scanning the entire dataset, you start printing the content of the hash table.

What to produce as output:

1) Print out the execution time taken to execute the command (in millisecond)
2) Print out the output records from the SQL statement

What to Deliver

- The entire source code package
- Readme.txt, in which you must include:
  - Your name and student ID
  - Section I: section on how to compile and execute your code. Include clear easy-to-follow step by step that TAs can follow
  - Section II: State clearly which parts are working, and which parts are not working in your code. This will help the TAs give you fair points.
  - Section III: section describes any design decisions that you do beyond the design guidelines given in this document.

What and Where to Submit

A single file .zip to be submitted in the Canvas system