CS4432 - Project 2 Implementing a Simplified Version of Indexing in DBMSs

Release Date: April 10, 2025
Total Points: 100

Project to be done individually.
Programming Language: Java (Better use java 8 or later) Or C

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:

- How the generative artificial intelligence tool was used
- How the generated answers were verified to be correct

Section 1: Overview

In this project, you will create a simple index structure to speed up the performance of the lookup queries. A lookup query is a query that involves a predicate (condition) on a certain column.

Ways to answer a lookup query: As covered in class, there are two main approaches as summarized below.

- ***Full Table Scan:*** In this approach, the database system needs to read the entire table (all its blocks) one by one. Then, for each block, the records are scanned one by one and checked against the query predicate.
    - This approach is used ONLY if there is no index available.

- **Index Lookup:** In this approach, the database system will first check the index, and figure out whether the value exists or not. If the value exists (or potentially exits), then the index will specify which data block to read. The DBMS will then read this data block only without scanning all other blocks.
    - This approach is used if there is an index available on the column involved in the condition

Dataset Format:

- You are given a dataset that you can directly use.
- The directory containing the data is named "**Project2Dataset**". You can hardcode this name in your project. It is the name that TAs will use when doing the testing.
- Put this directory "**Project2Dataset**" under the working directory of the Java program (the directory from which the program will run). This is the location from which the data is read.
- The dataset directory contains 99 files, each file contains 100 records, and each record is 40 bytes. This very similar to Project 1 dataset with slight differences as follows. For this project 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):

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

  where *RandomV* is a four digit random number between 0001 and 5000. Since the number of records in the entire dataset is around 10,000, then each value within the range of 0001 and 5000 is expected to appear in the dataset (on average) twice, but it can be more or less. Also, each record ends with three dots "…" to complete to 40 bytes.

- The *RandomV* column is the column on which you will do the search and build the index.
- As in Project 1, all records are of the same length (40 bytes), they are concatenated after each other, and there are no "new line" characters. The record boundaries are computed based on the 40-byte length.

Section 2: Building Hash-Based Index Structure [20 Points] (See the "CREATE INDEX …" command below)

In this part, you need to write code that builds a hash-based index on the *RandomV* column. The code will do the following functionalities:

- Read all files in the dataset directory, one by one, and for each file, read record by record.
- For each record, extract the *RandomV* value, and put it in a hash table. A hash table entry should have two components (key *k*, value *v*), where *k* = *RandomV* value, and *v* = the record locations (file number and the offset at which the record begins within this file).
    - It is up to you to design the appropriate data type (or structure for *v)* to keep multiple locations associated with a certain key.
    - *Hint: You may concatenate multiple locations in a single string.*
- The hash table should be kept in memory, and this is your hash-based index.

Section 3: Building Array-based Index Structure [20 Points] (See the “CREATE INDEX …” command below)

In this part, you need to write code that builds an array-based index on the *RandomV* column. The code will do the following functionalities:

- Allocate an array of size 5,000, each entry should store record locations (file number and the offset at which the record begins within this file).
    - Keep in mind that for a single value, there can be multiple records with that value
    - It is up to you to design the appropriate structure
- Read all files in the dataset directory, one by one, and for each file, read record by record.
- For each record, extract the *RandomV* value, say the value = i . go to the ith slot in the array and add the record location information.
- The array should be kept in memory, and this is your array-based index.

<u>Command to build the hash and array-based indexes</u>

- When your program executes, it should do nothing except printing the following sentence:
  ```
  Program is ready and waiting for user command.
  ```
  You program should be designed to loop and whenever it completes a command, it should print the same sentence indicated above and wait for the next command:
- If the user wants to create the hash-based and array-based indexes highlighted above, the user will enter the following command:
  ```
  CREATE INDEX ON Project2Dataset (RandomV)
  ```
    - The text of the command is fixed including the dataset name (`Project2Dataset`) and column name (`RandomV`)
    - This command should build both indexes and keep them in memory
    - The data files should be read ONCE to build both indexes concurrently
    - Once the two indexes are built, print out message
      ```
      “The hash-based and array-based indexes are built successfully.
       Program is ready and waiting for user command.”
      ```

<u>Note:</u> You may receive SELECT commands (see below) without indexes being created.

Section 4: Equality-Based Query Lookup [20 Points]

To receive the query, your program should support this command

```
SELECT * FROM Project2Dataset WHERE RandomV = v
```

- If there are no indexes built, then you should perform a full table scan.
- If there are indexes, then for equality search use the hash-based index.
- If you will use an index, make sure to leverage the record location information (both the fileId and byte offset) to minimize the I/O and CPU.
- “v” is any constant number.
- The syntax for the SELECT command is fixed (nothing will change) except value “v”

- The output that you should generate is:
    - Print out the record(s) matching the query
    - Indicate the index type you used (if any) or Table Scan.
    - Report the time taken to answer the query (in milli sec)
    - Indicate how many data file(s) (which are equivalent to disk blocks) did you need to read

Section 5: Range-Based Query Lookup [20 Points]

To receive the query, your program should support this command

```
SELECT * FROM Project2Dataset WHERE RandomV > v1 AND RandomV < v2
```

- If there are no indexes built, then you should perform a full table scan.
- If there are indexes, then for range search use the array-based index.
- If you will use an index, make sure to leverage the record location information (both the fileId and byte offset) to minimize the I/O and CPU.
- The syntax for the SELECT command is fixed (nothing will change) except values "v1" and "v2" which are random constants like "1" and "15"

- The output that you should generate is:
  - Print out the record(s) matching the query
  - Indicate the index type you used (if any) or Table Scan.
  - Report the time taken to answer the query (in milli sec)
  - Indicate how many data file(s) (which are equivalent to disk blocks) did you need to read

Section 6: Inequality-Based Query Lookup [20 Points]

To receive the query, your program should support this command

```
SELECT * FROM Project2Dataset WHERE RandomV != v
```

- With inequality operator, indexes should NOT be used (even if they exist)
- The syntax for the SELECT command is fixed (nothing will change) except value "v"

- The output that you should generate is:
  - Print out the record(s) matching the query
  - Report the time taken to answer the query (in milli sec)
  - Indicate how many data file(s) (which are equivalent to disk blocks) did you need to read

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