COL 362/632 Introduction to Database Management
Assignment 3

Submit By: 26 March 2025 11:59 PM

Assignment Goals:

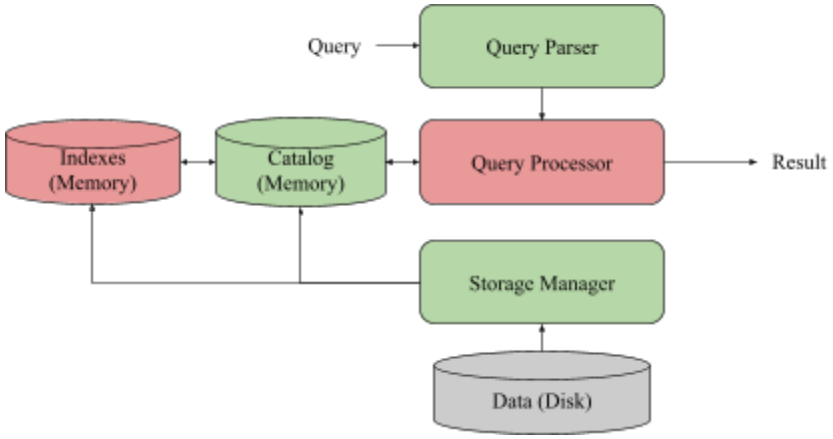
The goal of this assignment is for you to get some hands-on experience with:

- Implementing B+ Tree, Hash Index based on Extendible Hashing and BitMaps Indexes
- A toy database named DB362 for processing boolean queries on CSV files using the above indexes

You have been provided with a starter code for this assignment. The code is available here (you should be logged in with your IITD credentials to access it). In this assignment, you will further develop DB362.

DB362

DB362 is an **in-memory database system** for evaluating **boolean queries** on a **CSV file**. It is designed to be an in-memory system, i.e., it first loads the CSV file from disk to memory and then provides a basic data management framework to run boolean queries on it. The following figure illustrates the high-level system architecture and the flow.



Broadly, the green components have already been implemented, and you have to develop the ones in light red. Below is a description of each component, explaining the overall flow.

Data (Disk)

You will work with CSV file(s). The system supports (comma-separated) CSV files with the following format:

- The first line is a header.
- Each header field is of the form **attributeName:attributeType**. Examples include *"salary:double", "department:string", or "id:integer"*.
- Currently, supported attribute types are *integer, double, string, and date*.

Queries

The system supports boolean queries over a CSV file. The queries are based on the following grammar:

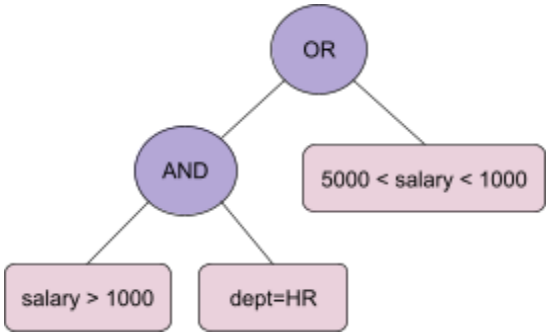
```
Expression     -> Term (OR Term)*
Term           -> Factor (AND Factor)*
Factor            ->  NOT  Factor  |'('  Expression  ')'  |
Predicate
Predicate      -> RangePredicate | SimplePredicate
RangePredicate :  NUMBER '<' IDENTIFIER '<' NUMBER
SimplePredicate:  IDENTIFIER ( '=' | '<' | '>' ) Literal
Literal        :  NUMBER | STRING | IDENTIFIER
```

For example, below are some valid (example) queries:

```
Q1: "NOT (department = HR)"
Q2: "(salary > 10000 AND department=HR) OR 5000 < salary < 10000"
Q3: "5000 < salary < 10000"
Q4: "age > 50 and department = CS"
```

Query Parser

The query parser parses a boolean query into a query tree comprising *Operator Nodes* and *Predicate Nodes*. Operator Nodes are non-leaf nodes and include AND, OR, and NOT. Predicate nodes are the leaf nodes and include range predicates or simple predicates. For example, the following illustrates a parse tree for Q2 above:



Storage Manager

The system has a simple storage manager with an I/O utility to parse a CSV file. Specifically, for a given CSV file, it creates multiple indexes including B+Tree, Hash Index based on Extendible Hashing, and Bitmap on specified attributes. After parsing the CSV file, it stores the available indexes for each attribute in the system catalog along with the indexes in memory.

Catalog

The catalog stores a list of indexes available for each attribute. It further provides a functionality to retrieve an index for a given attribute. This functionality is based on a simple heuristic, which first checks the predicate type. If it is a range predicate (e.g., 5000 < salary < 1000) or a simple predicate involving '>' or '<', it returns a B+Tree index for that attribute. Otherwise, for simple predicates involving '=', it prefers a BitMap index over HashIndex over a B+Tree index.

Query Processor

The processor takes a parse tree (its root node) as input and evaluates it by doing a post-order traversal on the query tree. For each leaf node (predicates), it computes the rowIds for which the predicate holds. For each non-leaf (binary) node, it intersects (for AND operator) or unions (for OR operator) the rowIDs from its children. For each non-leaf (unary) node (NOT), it computes the complement of the rowIds. In the end, the Query Processor returns the list of rowIds for which the boolean query holds.

Indexes

The system supports indexing attributes using B+Tree, a Hash Index based on Extendible Hashing, and a Bitmap Index. These indexes are used by the Query Processor to evaluate predicates.

Tasks

Install Java 11 and Maven 3.9 before proceeding

1. **Clone the project**
   a. Create directory `path/to/assignment_3/`
   b. cd into the newly created directory by `cd path/to/assignment_3/`
   c. Run
      `git clone git@git.iitd.ac.in:2402COL362/assignment-3.git .`
      to clone the project on your local machine (note the dot when cloning into the newly created directory)
   d. Run `mvn clean install -DskipTest`

2. **Import the project into your favorite editor. I strongly recommend Intellj.**

3. **Implement the B+Tree Index.**
   The starter code has been provided in
   `in/ac/iitd/db362/index/bplustree/BPlusTreeIndex.java`
   You should further develop this class by completing all TODOs. Your B+tree index should be right-biased. The node structure used in the B+Tree is specified in `in/ac/iitd/db362/index/bplustree/Node.java`
   Duplicate Keys: To handle duplicate keys, exploit the given node structure and implement the duplicate handling approach that allows leaf nodes to spill into overflow nodes.
   Study the node structure carefully and do not modify it. Note that the test cases will be based on B+tree logic discussed in the class.

4. **Implement Hash Index based on Extendible Hashing.**
   The starter code has been provided in
   `in/ac/iitd/db362/index/hashindex/ExtendibleHashing.java`
   You should further develop this class by completing all TODOs. There is a hashing scheme provided in `in/ac/iitd/db362/index/hashindex/HashingScheme.java`. The

scheme includes a `getDirectoryIndex` method that returns the offset of the bucket address table based on `globaldepth` least significant bits. Your logic should use this hashing scheme. Do not modify the hashing scheme. Further, the index uses a bucket structure specified in `in/ac/iitd/db362/index/hashindex/Bucket.java`. Study the bucket structure carefully and do not modify it. The test cases will be based on the hashing logic discussed in the class.

5. **Implement Search using BitMap Index.**
   The starter code has been provided in
   `in/ac/iitd/db362/index/BitmapIndex.java`
   The BitMap index is backed by an array of 32-bit integers, as Java does not natively support bit vectors. Study the insertion logic carefully to understand how this is done. You should implement the search functionality for this index.

6. **Implement Query Processor**
   The starter code has been provided in
   `in/ac/iitd/db362/processor/QueryEvaluator.java`
   You must complete this class by implementing the `evaluateQuery` method. Note that you implementation **must use** the `evaluatePredicate` for evaluating all predicates in a query. Do not change or remove this function.

Make sure to read all comments provided in the code carefully. There are certain classes, files, data structures, and variable names that you should not modify. Modifying code that you should not will lead to failing of test cases.

Testing your code

The starter code has been developed using Java 11, which will also be the version that will be used for testing. Make sure that you have Java version 11 before proceeding with the assignment. Also install `Maven` for installing and testing your code.

You can test your code by running `mvn test` from the command line. Two simple tests are provided in the `src/test/java/in/ac/iitd/db362`. The `CSVParserTest.java` contains a @Disabled annotation for the `testValidCSVParsing test`. After implementing the indexes to test parsing and index creation, you can remove it. To add new test cases, create new test files and follow a syntax similar to the ones already included. (should include a “@Test” annotation before the test function).

How and What to Submit

- `cd path/to/assignment_3`
- Create a patch by running the command: `git diff [COMMITID] > [ENTRYNO].patch`
- Replace `[ENTRYNO]` with your entry number.
- Replace `[COMMITID]` with the one that will be provided to you (this will be provided 2 days before the submission deadline)
- Upload your patch file on Moodle.

Follow all instructions carefully as given here and as comments in the code. Implement all TODOs and carefully read all notes mentioned in the comments. Do not rename any files, classes, function signatures, variable names, or any other unless requested. You may be called for an in-person demo of your code!

Happy Coding!