COL 362/632 Introduction to Database Management
Assignments 4 & 5 (Total Marks 14)

Submit by: 25th April 2025 11:59PM

Assignment Goals:

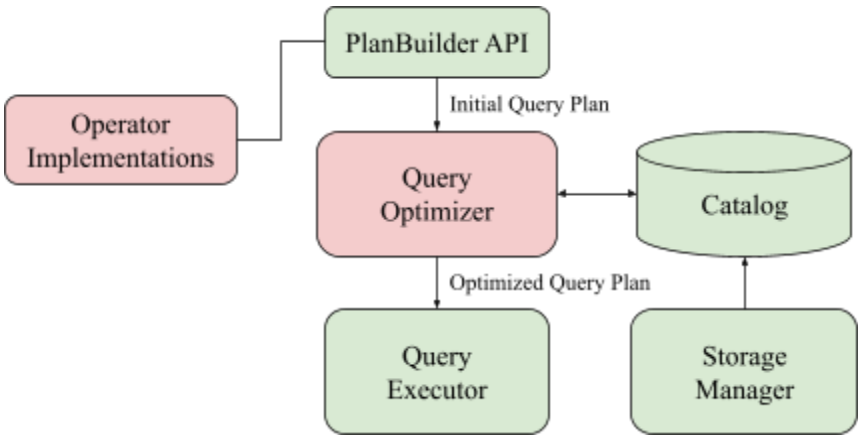
This programming assignment focuses on building a simple query execution and optimization engine for processing analytical queries over CSV files. The system will allow students to implement core relational operators based on the iterator model and a query optimizer. The CSV files act as tables, and students will implement operators that process these tables efficiently.

A starter code for this assignment has been provided, which is available here (you should be logged in with your IITD credentials to access it). In this assignment, you will further develop the system by implementing

1. Core relational operators (Assignment 4)
    a. Project Operator (that also allows duplicate elimination)
    b. Filter Operator
    c. HashJoin Operator
2. Query Optimizer (Assignment 5)
    This is an open-ended task! You are free to use your creativity and problem solving skills to develop one.

DB362 V2

DB362 V2 is an (toy) in-memory data analytics system that supports basic analytical queries on CSV files. It provides a fluent API that makes it convenient for analysts to specify basic data manipulation tasks. The following figure illustrates the high-level system over, where the green ones have already been developed and the red ones are what you'll have to develop as a part of the assignment.



CSV File Format

Each CSV file represents a table and follows these rules:

- The first row contains the column names and types in the format: `attributeName:attributeType`.
- Supported data types: `integer, string, and double.`
- **The system expects that every attribute name across CSV files is unique.**

Some (toy) example csv files are provided in `data/csvTables/`

Examples:

`customer.csv`

```
c_customer_id:integer,c_name:string,c_age:integer
1,John,30
2,Jane,25
3,Mark,40
4,Alice,35
5,Bob,28
6,Eve,22
7,Tom,45
8,Sara,33
9,Alex,27
10,Linda,31
```

`orders.csv`

```
o_order_id:integer,o_customer_id:integer,o_product_id:integer
101,1,201
102,2,202
103,1,203
104,3,204
105,5,205
106,7,206
107,2,207
108,8,208
109,10,209
110,4,210
```

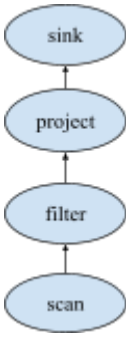
PlanBuilder API

The DB362 V2 provides a *fluent API* that allows users to specify analytical queries via a custom domain specific language (DSL) over CSV files. The API directly allows users to build query execution plans (internally represented as operator trees).

Examples:

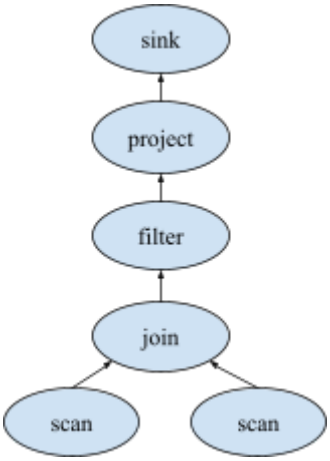
1. The following query plan reads customer.csv to write the name and age of all customers over the age of 30 to output.csv. The plan builder API directly builds the operator tree bottom-up. For instance, the operator tree for the above query can be visualized as

```
Operator plan = PlanBuilder.scan("customer.csv")
        .filter("c_age > 30")
        .project("c_name", "c_age")
        .sink("output.csv")
        .build();
```



2. The following query plan finds names of customers older than 30 and their order id and writes it to output.csv

```
Operator plan = PlanBuilder.scan("customer.csv")
        .join(PlanBuilder.scan("orders.csv"),
                "c_customer_id = o_customer_id")
        .filter("c_age > 30")
        .project("c_name", "o_order_id")
        .sink("output.csv")
        .build();
```



Operators and Semantics

1. **`scan`**: Reads a CSV file and returns tuples. Each tuple is modeled as a list of values and list of column names.
2. **`filter`**: Evaluates a boolean condition and filters tuples. Each filter operator only allows specifying **one atomic boolean condition** of the form `A op c` where `A` is a column name, `op` is one of `=, >, >=, <, <=, !=`, and `c` is a constant or another column reference. For example:
   `filter(age > 20 and depth = cse)` is **not** allowed
   `filter(age = 20).filter(dept = cse)` is allowed

3. **`project`**: Selects specific attributes from tuples. It expects attributes provided as a list of comma separated strings. For example, `project("age", "name")`.
4. **`projectDistinct`**: Removes duplicate tuples based on projected attributes.
5. **`join`**: Performs an in-memory hash-based equi-join, specified as `leftOperator.join(rightOperator, joinPredicateString)`. The joinPredicateString is specified as "`leftcolumn = rightcolumn`"
6. **`sink`**: Writes query results to an output CSV file.

See `in/ac/iitd/db362/api/PlanBuilder.java` and `in/ac/iitd/db362/query/QueryTest.java` for some examples on using the api and operators. You can also use the PlanPrinter at `in/ac/iitd/db362/api/PlanPrinter.java` to print plans on stdout.

Operator Interface

Each operator implements the Operator interface, which includes:

- `open()`: Initializes resources.
- `next()`: Returns the next tuple.
- `close()`: Releases resources.

Catalog and Storage Manager

The storage manager includes a utility to preprocess a CSV file and collects various statistics about the data that it stores in an in-memory catalog. In particular, statistics include: For each table, it stores information about the number of rows (excluding the header line), number of columns. For each column, it stores information about the minimum value, maximum value, number of distinct values (i.e., cardinality), total number of values, and an histogram. You can check the Catalog package `in/ac/iitd/db362/catalog` to see what statistics are collected.

Query Optimizer

The query optimizer takes as input an execution plan and catalog, and outputs an optimized plan.

Query Executor

The query processor performs a pipeline execution of the query plan. It upon receiving the plan simply calls the open() method of the root operator (i.e., the sink), and repeatedly calls next() of the root operator to get all tuples, writes it to the output csv file, and finally calls close() of the root operator.

Assignment Tasks

Task 1 (Assignment 4) Weight 30%

Implement Project, Project Distinct, Filter, and Hash Join operators by implementing the operator interface. For filter and join operators, you must also implement the Predicate and the JoinPredicate interfaces. The interface provides an evaluate method that the operators must use in their implementation to evaluate a tuple. See. `in.ac.iitd.db362.operators.ComparisonPredicate` and `in.ac.iitd.db362.operators.EqualityJoinPredicate`, that provides a started code for implementing the Predicate and the JoinPredicate interface.

Starter code templates for these operators have been provided to you. In addition, the scan and the sink operators are already implemented for you. You can try writing the following query to test scan and sink operators.

```
Operator plan = PlanBuilder.scan("/path/to/input.csv")
                           .sink("/path/to/output.csv")
                           .build();

QueryExecutor.execute(plan);
```

Task 2 (Assignment 5) Weight 70%

Implement a query optimizer by implementing the Optimizer interface. A starter code template has been provided in `in/ac/iitd/db362/optimizer/BasicOptimizer.java`. Upon successful implementation, one could use the optimizer as follows

```
// Initialize and collect statistics
Catalog catalog = new Catalog();
DataLoader.createStatistics("/path/to/input1.csv"), catalog);
DataLoader.createStatistics("/path/to/input2.csv"), catalog);
…

// Build a plan
Operator plan = PlanBuilder.scan(...)
                  .filter(...)
                  .join( PlanBulder.scan(…). project (...), "join_predicate")
                  .filter(...)
                  .sink(...)
                  .build()

// Optimize plan
Optimizer optimizer = new BasicOptimizer(catalog);
Operator optimizedPlan = optimizer.optimize(plan);

// Execute
QueryExecutor.execute(optimizedPlan)
```

How to proceed?

1. Clone the project
    a. Create directory `path/to/assignment_45/`
    b. cd into the newly created directory by `cd path/to/assignment_45/`
    c. Run
    `git clone git@git.iitd.ac.in:kbeedkar/assignment-45.git .` to clone the project on your local machine (note the dot when cloning into the newly created directory)
    d. Run `mvn clean install -DskipTests`
2. Import the project into your favorite editor. I strongly recommend Intellj.
3. Implement operators
4. Implement optimizer

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.

DO NOT REMOVE ANY OF THE LOGS

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.

There are some simple tests and examples provided
`in/ac/iitd/db362/query/QueryTest.java`.

How and What to Submit

- `cd path/to/assignment_45`
- Clone the repository, implement operators and the optimizer.
- **REMOVE ANY CSV I/O FILES that you used for testing other than those provided in `/data` folder !**
- Create a patch by running the command: `git diff [COMMITID] > [ENTRYNO].patch`
- Replace `[ENTRYNO]` with your entry number.
- Replace `[COMMITID]` with one that will be provided 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. DO NOT PLAGIARIZE. You may be called for an in-person demo of your code!

Happy Coding!

Grading

- For the first task (Assignment 4), your code will be evaluated for correctness using multiple test cases.

- For the second task (Assignment 5), there will be **relative grading**.