Data Structures Assignment: MapSQL (Version 1.0)

Introduction:

MapSQL is a simple in-memory database management system for Java. It consists of two subsystems:

- The core database engine
- A shell for interacting with the database engine (this is a command line based interface)

This worksheet requires that you complete a number of missing methods in order to finish the implementation of the database engine.

NOTE: As a first step, you should spend some time trying to understand how the existing code works. Please focus on the database engine (the code is in the mapsql.sql package) instead of the shell code as the shell code uses an advanced tool called JavaCC to handle the parsing of the SQL statements. Use of this tool is beyond the scope of this course.

For your convenience, I have implemented an example test program that shows how you can directly test the functionality of the database engine (without having to use the shell subsystem). This test program is located in the mapsql.sql.test package. Feel free to change the example statements in order to test each one of your implementations as you complete them.

Outline Marking Scheme:

The overall marking scheme for the assignment is given below:

1. Basic Condition Implementations: (5% per condition) 20%
    a. GreaterThan
    b. GreaterThanOrEqual
    c. LessThan
    d. LessThanOrEqual
2. Implement Like condition (3 case % at start, % at end, or % at start and end) 10%
3. Implement Table.update() method 20%
4. Implement Table.delete() method 20%
5. Implement TableDescription.checkForNotNulls() method 10%
6. Implement DropTable.execute() method 10%
7. Implement the execute method in Sources.execute() method 10%

Overview of Codebase:

The core components of the database engine can be found in the mapsql.core package. They are:

- **SQLManager**: This is the "front" of the database system. All operations are carried out by invoking the execute() method which is overridden for SQLStatements and SQLCommands .
- **SQLStatement**: This is the main interface for SQL statements (e.g. CREATE TABLE, SELECT, INSERT, UPDATE, DELETE, DROP TABLE). Implementations of these statement types can be found in the mapsql.sql.statement package (you will be modifying the DropTable class).
- **SQLCommand**: This is the main interface of SQL commands, which are commands that can be entered through the shell, but which are not actually part of the SQL language (e.g. QUIT and SOURCE). Implementations of these commands can be found in the mapsql.sql.command package (you will be modifying the Sources class).
- **SQLOperation**: This is a common super interface for SQLStatements and SQLCommands. It is included purely to allow you to work with both SQLStatements and SQLCommands seamlessly (you can create a list of operations that includes both statements and commands).
- **Table**: This is the core implementation of an SQL table. In combines two other core classes: a TableDescription and a list of Rows. NOTE: The list implementation used is the one developed on the course. You will be implementing two methods from this class.
- **TableDescription:** This class models the description of a table (i.e. its name and fields). You will be implementing one of the methods specified in this class.
- **Row:** This class implements an individual row in a table. It is basically a wrapper for a Map whose key is the field name and whose value is the data that is associated with that field in the row. *Notice that the table data is stored as a string*.
- **Field:** A field is a definition of a column in the table. This is an interface that should be implemented for each type of data that can be stored in the database. For this assignment, we support only CHAR's and INT's (the implementation of these can be found in the mapsql.field package).
- **Condition:** A condition is a comparison that is defined in the WHERE clause of an SQL statement. Conditions are basically comparisons that can be combined via AND or OR operations. This file is an interface that should be implemented for each type of condition that is supported (the full set of supported conditions can be found in the mapsql.condition package. You will be completing the implementation of some of these conditions as part of your assignment.
- **SQLException:** this is the standard exception that is thrown whenever anything goes wrong. The exception has a constructor that allows you to include an error message that will be printed out by the shell if the exception occurs.
- **SQLResult:** this is an interface that is used to return the result of executing a SQLStatement. It contains two methods that return the TableDescription and a list of rows (only in the case of a SELECT statement).

The main data structures used in this code are Maps (I have used the java.util.Map interface and the java.util.HashMap implementation) and Lists (I have copied the list implementation from the course into the mapsql.util package).

Trying to Manage the Number of Classes

As you start dealing with more complex systems, you need to adapt your way of thinking about the code you write. Here, I want to try to explain how I manage that increasing complexity with the hope of helping you to understand the code for this assignment.

When a system becomes large, concepts become increasingly important as the framework for maintaining a view of the system than individual classes. For example, in MapSQL, you can talk about the concept of statements, which covers 9 of the 38 classes that make up the database engine. Further, the best way to think about these classes is not to think of the classes, but to think only of the interface (which only has one method). So, you can say that statements are Java classes that contain an execute statement that returns a SQLResult object (notice that I am including this interface and class as part of the "statement" concept). Its only when you talk about a specific statement that you need more information than this…

Similarly, the concept of a table covers 7 classes (table contains a list of rows whose contents is described by a predefined set of fields), the concept of a condition for the WHERE clause of a statement covers 11 classes, and 5 classes are used to implement the core List ADT. In summary, while the system contains a lot of classes, most of the classes can be grouped about

Statement	9
Condition	11
Table	7
List	5
Command	3
Other	3
TOTAL:	**38**

As can be seen, we have reduced the 38 classes to only 5 concepts. As the system becomes even more complex, this view of the system may become a subview of a larger view in which the entire database engine is just a single concept….

Running the Complete System

To run the full system (database engine + shell), you need to run the mapsql.Shell class. This should result in a command prompt appearing:

mapsql>

You can start executing basic SQL statements via this prompt (CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, DROP TABLE). Also, there are two special commands: SOURCE (to load an SQL file) and QUIT (to exit the program). The system only supports two types: CHAR's and INT's, but supports the UNIQUE, NOT NULL, and AUTO_INCREMENT modifiers for field definitions.

Key Tasks

Task 1: Basic Conditions (20%)

The first task is to complete the implementation of a number of basic comparison operators, namely: greater than (>), greater than or equal (>=), less than (<), and less than or equal (<=). Specifically, you must complete the implementation of the evaluate() method in each of the corresponding classes in the mapsql.condition package. While you can complete these implementations any way you want to, I recommend that you look at the implementation of equals (=) and not equals (<>) as a starting point. For each operator, I suggest that you create a simple test program (add them to the mapsql.test package) that is based on the example test program.

For example, to test less than, create a contacts table and insert 2 rows into the table. Now, create a select statement that will only return one of the rows.

Task 2: The Like Condition (10%)

The like condition can be used to compare CHAR fields (Strings). Specifically, like allows you to use the % sign as a wildcard for matching. In MapSQL, the use of wildcards is limited to the first and last characters in the string that is being matched against. Specifically, you can match %xxx, xxx%, or %xxx%. The first wildcard pattern allows you to check if a string ends with a given substring; the second pattern allow you to check if a string starts with a given substring; and the final pattern checks whether a string contains a given substring. The implementation of this condition is more complex than those completed in task 1.

Again, remember to check your answer by creating a test program that checks all three permitted wildcard patterns.

Task 3: The Table.update() method (20%)

This method is supposed to be used in tandem with the Update statement. Specifically, it does the actual update of the any row that matches the WHERE clause. When updating a row, you should replace any existing values with the corresponding values that have been provided. The values of any columns that are not explicitly updates should stay the same.

Task 4: The Table.delete() method (20%)

Like task 3, this task is supposed to be used in tandem with the Delete statement. Specifically, it does that actual removal of any rows that satisfy the WHERE clause.

Task 5: TableDescription.checkForNotNulls() (10%)

This task is concerned with ensuring that any field that is declared as NOT NULL is included explicitly in any INSERT statement. The method is already called by the Insert statement class, and it is expected that an SQLException will be raised as soon as a field is found that is declared as NOT NULL, but has not been explicitly included in the INSERT statement

For example,given a table that has been created by the SQL statement:

```
CREATE TABLE contacts(
       id INT AUTO_INCREMENT,
       name CHAR(30) NOT NULL,
       email CHAR(30)
);
```

Then the insert statement:

INSERT INTO contacts (email) VALUES('Henry');

Should result in an exception being thrown that has the following error message:

Missing Value for NOT NULL field 'name'

Task 6: The DropTable.execute() method (10%)

The DROP TABLE statement results in the named table being removed from the database and this behaviour should be implemented in this method. If you examine the codebase then you will see that there is a "system" table called "mapsql.tables". You should generate an SQLException if anybody attempts to delete this table. The error message for the exception should be: "Table 'mapsql.tables' cannot be modified.".

If the statement does refer to an existing table then the table should be removed from both the tables Map and from the above system table.

Task 7: The Sources.execute() method (10%)

This last task is to implement the SOURCES command. This command basically loads a specified text file and attempts to parse it. If the file is parsed successfully, then the resultant SQL statements are executed.