CS111 SQL Database Project
Milestone 3

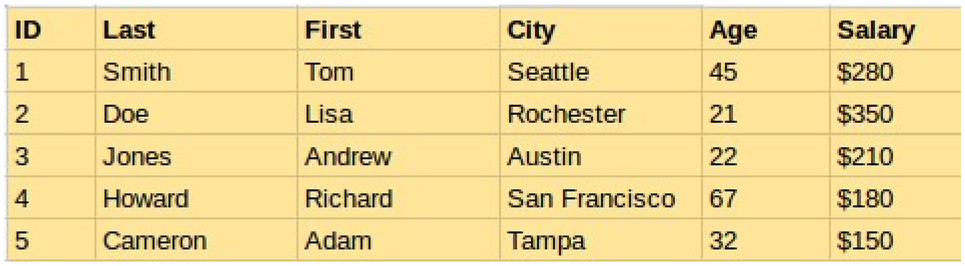
Rutgers University New Brunswick

Due: 05/04/2015 11:59pm
Spring 2015

For this milestone you will implement functional UPDATE and DELETE queries. The interface you have from MS2 should be modified accordingly to allow for these two new queries to be executed.

Your implementation must utilize the Query module provided to you. This module can be downloaded here `http://sql.pawel.pw/Query.java`. The functionality of this module is similar to the IO.java module that you are all already familiar with. The documentation for the Query module can be found here `http://sql.pawel.pw/Query.html`. Please make sure to READ THE TOP SECTION of the Query documentation. It specifically outlines the differences in correct query vs. incorrect query. Parsing languages programatically is no easy task hence limitations had to be specified.

Throughout the document the 'employees' relation in Figure 1 below will be used for examples and explanations.



ID	Last	First	City	Age	Salary
1	Smith	Tom	Seattle	45	$280
2	Doe	Lisa	Rochester	21	$350
3	Jones	Andrew	Austin	22	$210
4	Howard	Richard	San Francisco	67	$180
5	Cameron	Adam	Tampa	32	$150

Figure 1: Database table/relation of employees

Your fields should be CASE SENSITIVE. Hence 'last' and 'LaSt' should not be thought of as the same field.

DELETE Query Specifications

The DELETE query should properly perform a remove operation as specified by the query. If a single row is to be deleted, it can no longer be accessed by any other issued queries. For instance, if you delete an entry with ID=5, then a SELECT/UPDATE query with 'WHERE ID=5' should print a message saying that entry with ID #5 does not exist in a similar way that has been done for SELECT. How this delete actually works internally will be up to you to decide.

Your implementation of DELETE queries should be able to support the following functionality:

1. `DELETE FROM employees;`

   This should remove everything. After this query your table/relation should not have any entries. After this query is issued you should NOT reset your 'ID' generation. In other words, if your last row had ID #5, after issuing this delete query the next INSERT query should generate #6 for the next 'ID'. With this 'ID' generation approach, this implies that once an entry with an 'ID' is deleted, that 'ID' will never exist again during your program execution, but can still appear in queries.

2. `DELETE FROM employees WHERE ID=1;`

   This will remove the row specified by ID #1. In our employees relation, the subsequent SELECT query would return rows #2 through #5.

Keep in mind that the queries shown above have been specifically written for the 'employees' relation, hence the fields should correspond to your specific relation.

Use the following function header for the implementation of the DELETE query:

```
public static boolean delete(Query deleteQuery,[your database]) {
        // your code here
}
```

The first parameter should be acquired through the help of the Query module. The second parameter passed to the function labeled as [your database] should be the data structure that holds your database. The function should return 'true' on a successful query execution, 'false' otherwise.

UPDATE Query Specifications

The UPDATE query should properly perform an update operation on the data present in your database.

Your implementation of UPDATE queries should be able to support the following functionality:

1. `UPDATE employees SET Last=Dijkstra;`

   This query updates all entries in your table/relation since no 'WHERE' clause was specified. After this query, every entry's 'Last' field should have 'Dijkstra' in it.

2. `UPDATE employees SET Last=Dijkstra WHERE ID=1;`

   This will update a specific row identified by ID #1. In the case of the 'employees' table/relation, the 'Last' field which holds the value 'Smith' will get overwritten by the value 'Dijkstra'.

Keep in mind that the queries shown above have been specifically written for the 'employees' relation, hence the fields should correspond to your specific relation.

Use the following function header for the implementation of the UPDATE query:

```
public static boolean update(Query updateQuery,[your database]) {
        // your code here
}
```

The first parameter should be acquired through the help of the Query module. The second parameter passed to the function labeled as [your database] should be the data structure that holds your database. The function should return 'true' on a successful query execution, 'false' otherwise.

Interface & Implementation

At this point you should already have a functional implementation of an interface that accepts infinite number of queries from milestone 2. Your SELECT/INSERT implementation should also be able to at the very least insert data and view all data. It is fine if you did not pass certain edge cases for SELECT/INSERT. If milestone 2 is not at the basic functionality level described then you should first go back and fix/complete it before proceeding forward with this milestone.

For this milestone you should be able to simply modify your interface in order to recognize and properly handle DELETE/UPDATE queries.

You can continue your implementation in 'SQLDatabase.java' file.

Warning

Just as with any other coding submission make sure that your code compiles! If your code does not compile for whatever reason you will not receive any credit. NO EXCEPTIONS! So make sure to remove any package statements.

Submission

Submit 'SQLDatabase.java' file only, you do not need to attach 'Query.java' module. If you happen to have additional files on top of 'SQLDatabase.java' please zip (archive) your submission into a file called MS3_SQL.zip. Once again submit in the same place as the regular MS3, but just indicate in the submission box that you are doing the alternate project. Good luck!