CS111 SQL Database Project Description

Rutgers University New Brunswick

Spring 2015

In this project you will be using your existing knowledge gained through lectures and homeworks in order to explore an area of computer science that is utilized in essentially every piece of technology or software that you have ever encountered. This writeup is meant to give you a brief overview in order to get you started on the project, but you are highly encouraged to check out different sources to learn more about the topic at hand. This semester's alternative project will focus on implementing a very basic database engine that can be manipulated via a limited subset of the Structured Query Language (SQL)[1]. This is an individual project, and as such you will be held to the University's academic integrity policies. You are welcome to discuss strategies and ideas with other students (and will be encouraged to in recitation), but you must code the project on your own.

This project is what you make of it. Listed below are point guidelines as to how they relate to the number of points given to the project (total 100). However, you should not be limited by what we suggest the minimum work required here is. To get the most out of this project, I urge you all to push yourselves to find your limits. You might surprise yourself! Good luck!

Background

Databases are widely used in many software applications. The majority of the websites that you interact with on a daily basis use databases to store information such as user data, product data, site statistics data, etc. Just like you are currently utilizing Java in order to instruct an application to perform some sort of manipulation, SQL is a database language that aids in manipulating a relational database from a user program.

Relational Database

There are many databases that implement the SQL language such as MySQL, Oracle, or Microsoft SQL Server. What these database systems have in common is that they use a relational database structure apporach. A relational database organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row[2]. Figure 1 shows an example of how such a database table would look.

[1] Read more at `http://en.wikipedia.org/wiki/SQL`

[2] Source `http://en.wikipedia.org/wiki/Relational_database`

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 of employees

Each row represents a database entry with columns as fields or attributes of each row entry. For instance take a look at row where ID is 3. That entire row, which can be identified by the unique key of 3, describes Andrew Jones who lives in Austin, is of age 22, and earns $210.

This was just a basic example to show you how a basic database table looks like. Notice that there is a set number of columns while the number of rows can grow infinitely[3]. The reason for this is because a table is initially created by thinking of what type of data you want to store with what kind of attributes[4]. You can essentially store any sort of dataset in a database as long as you have an idea on how your dataset will fit nicely into a row-column table format.

SQL Language

Now that you understand what a database looks like, lets dive into how data can be manipulated and retrieved from a database via the help of SQL. We will focus on four main commands which are select, insert, delete, and update.

Select

The select keyword is used for viewing the database. The select keyword does not alter the data being stored. Here is a basic structure of a select statement.

```
SELECT [fields] FROM [table name];
```

Here fields can be any number of fields separated by a comma that are specified in the table or the wildcard character, represented by ' * ', which means select all fields. The table name represents the name that is given to your table[5]. So let's take a look at the following query.

```
SELECT Last,First FROM employees;
```

This will return to us only the first and last name of each row entry as shown in Figure 2 below.

[3]Or until space runs out

[4]Columns can be added after table creation, but this will create empty fields in row entries.

[5]There can be many tables in one database instance and they are identified by names specified by the developer.

Last	First
Smith	Tom
Doe	Lisa
Jones	Andrew
Howard	Richard
Cameron	Adam

Figure 2: First select query result

If instead we modified the query to

```
SELECT * FROM employees;
```

This would yield exactly all of the entries as shown in Figure 1.

Select queries can also be filtered using a 'where' keyword. For now we will focus on filtering by one field, such as ID. Here is an example

```
SELECT * FROM employees WHERE ID=1;
```

This will return all fields of the entry where ID is equal to one. This can be seen in Figure 3 below.

ID	Last	First	City	Age	Salary
1	Smith	Tom	Seattle	45	$280

Figure 3: Second select query result

Insert

The insert keyword allows you to insert a row entry into your table. The general structure is as follows

```
INSERT INTO [table name] VALUES ([comma separated values]);
```

The 'comma separated values' represent all the fields that you defined for your table[6] in the same order as they appear in your table. Here is an example

```
INSERT INTO customers VALUES (Jacobs, Michelle, Edison, 20, $390);
```

This will insert a new entry at the bottom of the table as shown in Figure 4 below.

[6] In full SQL you can insert a few field values instead of all.

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
6	Jacobs	Michelle	Edison	20	$390

Figure 4: Database table after insert query

Notice that the ID field was not a part of the query. This will be up to you to figure out how to maintain your database engine in a way that will automatically generate ID numbers.

Delete

The delete keyword allows you to delete a row entry by specifying an ID. Here the general structure is as follows

```
DELETE FROM [table name] WHERE [condition];
```

The condition refers to specifying the ID of entry to be deleted. So if the following query is ran

```
DELETE FROM customers WHERE ID=3;
```

The result would simply remove the row entry labelled with ID number 3. Hence a subsequent select query would not display row entry with ID number 3.

Update

The update keyword allows you to update one field of a row entry by specifying an ID. Here the general structure is as follows

```
UPDATE [table name] SET [field name]=[new value] WHERE [condition];
```

The condition works the same way as for the delete keyword. Here is an example to demonstrate

```
UPDATE customers SET Last=Jordan WHERE ID=6;
```

Now if we run a select query notice how the row entry with ID number 6 changed in Figure 5 below.

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
6	Jordan	Michelle	Edison	20	$390

Figure 5: Database table after update query

Note on WHERE clause

For this assignment the only type of condition that you will be able to specify in your queries will be 'WHERE ID=[number]'. SQL features give much more flexibility, but as mentioned before this project is meant to implement partial SQL functionality.

Design Phase (Milestone 1, Due ——)

In this phase you will not be doing any coding. You will be designing how you want your database to function. You will be given a library that provides you with the following capabilities

- Read query from user in a similar way that the IO module works
- Identify what type of query was entered (SELECT, INSERT, DELETE, or UPDATE)
- Parse and return to you fields specified by SELECT query
- Parse and return to you values to be inserted by the INSERT query
- Parse and return to you field name and new value specified by the UPDATE query
- Parse and return to you ID number filter from the WHERE clause in any type of query

Think about how you would you structure your database engine to hold data and respond to given commands accordingly. Also think about the type of data that you will hold. The choice that you make regarding the data type might affect the way your database engine will be structured.

At the very least, for full credit, your database engine should have the following functionality (this is the minimum set, you are encouraged to exceed these requirements).

1. Allows storage of row entries that consist of AT LEAST 3 fields
2. Viewing all entries in table via 'SELECT' query
3. Filtering 'SELECT' query via 'WHERE' clause
4. Adding data via 'INSERT' query
5. Being able to generate new ID's for newly inserted data
6. Proper implementation of 'DELETE' query

7. Proper implementation of 'UPDATE' query

You should also, in your submitted design, answer the following questions. These should give you some inspiration as well. This is a non exhaustive list and not every question will be applicable to your project. You may need to come up with additional explanations that are not in this list to make it clear to the reader (grader) that you have given sufficient thought about the direction of this project.

1. What type of data will you store in your row entries? This could be all of one type (int, float, double, String) as long as the data has some sort of theme or meaning and there are AT LEAST 3 fields. List the fields that will be stored by your database. For instance, in this writeup data about 'employees' was stored.
2. What kind of structures will you use to store this data? Will this be a 2D array structure of the data type specified in the first question? Array of custom objects? Explain your approach.
3. How will you relate a column with a field name? In other words, if you run 'SELECT Last,First FROM employees;' how will you know which column in your entry row corresponds to the field 'Last' and field 'First'?
4. How will generating new row ID's work for the 'INSERT' query?
5. Will your database limit the number of entries that can be inserted? If so, why? If not, how will this be done?
6. How will deleting a row work?
7. How will the 'WHERE' clause work? Will it differ from query to query? Explain.