JAVA MySQL API & Command-Line Interface

Facilitating a faster, and intuitive method for quick MySQL integration within personal Java applications

ZACHARIE HAPPEL

Preface:
This application was developed on Java SE 8, a runtime environment developed by Oracle. Personal design and development of the software was done solely by myself, Zacharie Happel, on a late-2015 MacBook Pro using Eclipse Photon.

Purpose:
This application was developed and submitted as the final project to my Software Development I course. The individual goal and direction of the project was one that we were allowed to choose for ourselves. I saw this assignment as an opportunity to build the previously existing foundation of knowledge and familiarity I have for software development platforms and their applications.

Application Architecture Displayed via Tiered Model:

- Server.class
  - Database.class
    - Table.class
      - Column.class

[Server.Class]

This class is contains the details pertaining to the server in which you would like to access. This class requires no imported libraries as the server object is simply used as an information vessel that gets passed along to the Database object.

Notes

- String variable *`name`* refers an alias/nickname the user assigns to the Server object being defined.
- String variable *`full_address`* refers to the server path

```
e.g jdbc:mysql://localhost:3306
```

[Database.Class]

The arg-constructor requires *`Server.full_address`* to create address to the database, *`this.database-address`*. If the Database object's parent Server was defined using the no-arg constructor, then the use of the Server's *`updateFullAddress()`* method is required to obtain the proper server address.

The method *formConnection* established a connection from client to host. Upon a successful connection, System.out will display a message in the following format: *`Connected: [Database Name] on [Server Name].`*

The Database objects contains within it methods designed and purposed with the developer in mind. These methods allow the easy integration of MySQL within future applications.

Listing the tables within a database is done using the *`findTables()`* method. Importing a table's columns, values, as well as attributes associated with the table, such as size, is an action performed by the *`importTable()`* method. Tables are stored within the HashMap *`ImportedTables`*. The key of the HashMap is the name of the table, and the value is the Table object that was imported. HERE! The method *`getTable()`* performs the same functionality that would come from

`ImportedTables.get()` without having the user need to refer to the list itself. Additionally, there are several other table parsing methods included within the Database object. These include:

`listImportedTables()`, *`listColumnValues()`*, *`showTable()`*, and *`showTablesAll()`*

While the list methods are self explanatory, showTable is not so much. What *`showTable()`* does is output a specified imported table to System.out in tabular form.

How it works: showTable() and showTablesAll()
Using the Column objects that are stored within the Table object of the to-be-displayed table, a local HashMap is defined.The depth of the table is acquired and used as the range in which to traverse the y-axis. Similarly, the count of columns determines that of our x-axis. Defined by the name Rows, an ArrayList is defined as a List object comprised of other List objects, of which contain String values. Defining two for-loops, the outer being that of the x-axis, and the inner of y — applying the range constraints found earlier to each respective loop — the found values are added to a separate ArrayList that is initialized upon each iteration of the outer loop.

Using a Java table formatting library named *Wagu*, the table is defined using column names, rows, and a table size — of which is dynamically calculated through a private *`Database.class`* method, *`calcBoardSize()`*.

[Table.Class]

Passing along the required connection to the server, as well as the database name and the name of the Table object that is to-be, is formed. Upon defining the table using the aforementioned technique, the Table object then calls upon the method *`ImportColumns()`*. With said method, a query defined with the parameters passed to table is executed. This query polls for the names of the column, and upon retrieving its answer, creates a Column object consisting of the gathered information. An ArrayList is instantiated with the values returned from the method *`retrieveColumnValues()`*. Finally, both the name of the column and its values are added to the LinkedHashMap, columns, that contains all of the columns within the table.

Additionally within this class are getters, setters, and listers. While the methods within the Database class were designed for users of the API, the methods within the Table and Column class provide additional functionality when attempting to access values with their respective objects.

[Column.Class]

Unlike Database and Table whom rely upon methods consisting within classes seated above them within the hierarchy of objects. Column does not, and as such, does not extend its predecessors. Consisting of getters and setters, the purpose fo the Column object is to consist of the values contained within the column in which it is defined.

Command Line Interface:

The usage of the command line interface is fairly explanatory. You enter the server address, port, connection name, username, and password. Upon connection you are greeted with the same messages verifying the connection that you would have had you used the API.

You are then greeted with a menu which allows you to perform the core operations defined with the database class.

To run the CLI, you may either compile and run the *`JavaMySQL_CLI`* through Terminal, CMD, etc.   or   You may simply run the .class file in Eclipse.

Structure of Data Objects:

Server

- Provided parameters for connection to a MySQL server.

Database

- Relies on the parameters passed to it by Server, and forms a connection to a database
- Keeping Database separate from object, multiple databases can be defined and connected to at once
- Maintains a list of all imported tables, and methods that are purposed for further java application

Table

- Representation of a table within a database.
- Once specified, the names of the columns as well as the values within them are found.
- Values are stored within a LinkedHashMap so that data stored in the order in which it was was added while also still providing access to its elements via name (keys)

Column

- Representation of a column within a table.
- An ArrayList maintains the value of the data within the column.