**SwingBench: A Project Write-Up**

Matthew Monteleone, Justin Doonan

**Introduction**

Our very ambitious project, SwingBench, was to create a Java application that interfaces between a user and his/her MySQL database to give said user a more pleasant and interactive UI to work from instead of the command line. By working on such a middleman application, we hoped that we could glean some knowledge about databases and user interaction with them as well as gain a greater appreciation for and insight into the work that the DBMS and other GUI applications, such as phpmyadmin and MySQL Workbench, do for the user. We decided on this project over some others because we both have experience working together with MySQL-backended applications and building Java GUIs from some of our previous classes. Since we are both well-versed in such topics, we decided to use NetBeans to speed up the process of UI building to give us more time to work on application functionality.

As the semester passed us by, we decided to narrow the scope of the project to creating a simple UI that a user who is new to using databases could find easier to use and understand. Since our application would be simpler, there would be less distracting and complex features filling up the screen and, in their place, a short list of straightforward features. We decided that the most important features to include would be query building-eqsqe functions to drop and create databases and tables and to import and export files. Along with those functions, we had to include a function that allowed any SQL command to be entered and executed. To accompany these, we would need a good way for users to see all of the databases and tables they have access to along with the data that resides in those tables.

**Model**

Since this project didn’t require any complex algorithms, performance checking, platform comparisons, or things of that sort, we focused most of our time to ensuring a pleasant user experience within our application by making menus that are straightforward enough for the newest database user to understand yet quick enough for the expert user to rapidly fly through and appreciate. Thus, this section will be mainly about what the interface looks like and how we believed a user would use a specific window or function.

Before a user has access to a database, they must first log in! When SwingBench first launches, a small login screen that requires a url to a working MySQL server along with a username/password combination that works on that server. We also made it a point to include a way for the program to remember login credentials, just incase the server url or password is hard for a user to remember. If a set of credentials does not work, a small red-text alert pops up on the screen to let the user know they have incorrect credentials. To reduce any security risks for remote servers, we do not report the exact type of login error in the event that a person is guessing usernames that are valid on a particular remote server.

After a user has successfully logged in, we wanted a larger main window to appear in place of the login screen that displays the file structure of their databases along with links to the other functions in the application. On top of that, we wanted some way to report success or failure to the user. Because we decided to use Java and NetBeans to build our application, we were able to make use of a JTree file directory UI structure to display the databases and tables/views to a user in a way that looks similar to the directory system of a regular computer. This takes up very little space on the main application window in favor of a large table display that updates its rows and columns whenever a user double clicks on a table in the JTree. For larger tables, an alert window pops up and shows real-time loading information that way the user won’t panic when a table doesn’t instantly appear. The executed SELECT statement is displayed below the table in the form of a debugging console.

At the top of the main window is a small menu bar that has File, I/O, and Query tabs. We needed a place to put links to the drop/create database and table and import/export file functions. We also needed to place a link for the user to log out of the application, disconnect and re-login with new credentials, and manually refresh the main file structure. Underneath the File tab resides Disconnect, Exit, and Refresh options, and their functionality is extremely self-explanatory.

Underneath the I/O tab resides links to import files from .txt and .sql files and links to export data into .txt and .sql files. When the Import .sql option is selected, a file explorer appears and the user must select a file to execute. The information in the file is read until the SQL end of line character, ’;’, is encountered and executed on the basis that the end of a full SQL command will end with the first appearance of the next semicolon. When the Import .txt option is selected, a window is displayed that has fields for the user to enter any file delimiting information that their file follows and a button to select a file to be executed. Since data to be imported from a .txt file requires a database and table to be selected fist, we require a database and table to be selected from two dropdown menus. Also, the MySQL defaults for line delimiting in file importing are present in the fields when the window is initially opened. When the Export .sql function is selected, a window appears that requires the user to select a database, landing directory for the .sql file, and a file name for the .sql file. When the Export .txt function is selected, the same window appears except it also requires a table name from the selected database to be selected. This function prints the contents of a table to a file in CSV format.

The last menu option, Query, gives the user a list of 5 options: Custom Query, Create DB, Drop DB, Create Table, and Drop Table. While the functionality of each is easy to assume, it is still important to note the details we put into each function and associated window. The custom query window not only includes a large text area for the user to type queries, it also has its own debugging console and table display. This allows the user to compare 2 table’s data side by side without requiring anything more than two simple SELECT statements, the ability to see the returning tables from SHOW and DESC commands, and the chance to view verbose MySQL error reporting from any kind of query. The text area where queries go does not execute them when the return key it used. Instead, it relies on a button and lets the user enter multiple queries for a batch execution.

The Drop DB option displays a window with a dropdown of all available databases to drop. Similarly, the Drop Table option displays a window with dropdowns to select a database and a table from the selected database to drop. The Create DB option displays a window with a lone field for the user to enter a name. The Create Table option displays the most complicated appearing window of the whole application. It is divided into three sections: naming, filling, and executing.

The name section contains a field for a table name as well as a dropdown of all databases to place the table in. Once these are selected and the Add button is pressed, the beginnings of the CREATE statement are printed to a large text area below and the user can now move onto the filling section. Here is a field for column name, a dropdown of suggested types, and a field for any extra constraints a more advanced user could tack onto the end of a column declaration. There is an Add button here as well that is only enabled once the previous Add button has been used. Using this Add button appends the column described by the above three fields to the CREATE statement. The user can add multiple columns this way, or the user can simply edit the text area on their own. We decided to leave this text area editable to give the user control over their CREATE statement in the event of a mind change or misspelling. Once the CREATE statement has been completed, the user can move on to the execute section. Here is the button to execute the query as well as another debugging console to report verbose responses to their statement. The JTree on the main application window is also refreshed because it detected a new table being created.

For each of the separate displays that our application produces, we tried our best to place each element in a logically flowing manner from the top of a window to the bottom. Every window aside from the main application window does not require the user to look directly at the top of the window to begin usage and end by looking at the bottom. In this way we are semi-forcing the user to miss no detail about the window they are currently focused on because, when working with databases, it can be easy to mistype one thing or leave a small detail out when you are working with a large amount of data or huge job to finish.

**Datasets Used**

Since this project was designed to work with every data model possible in MySQL, we didn’t have a strict set of data to use. In testing, we used the data that was already present in out own databases. To test the ability of the program to keep up with large tables, the infamous ‘employees’ database from the MySQL website was used. When working with the UI, the decisions we made were not with some collection of data gathered from user surveys but with information we gathered from this class and others, mainly Data-Info-Knowledge.

**Experimental Setup**

This project did not necessarily require an experiment to test it, but we did actually test the application. Since it ended up being simplified in functionality, the testing was not rigorous in the least because we only had to test for correctness and speed as opposed to the same actions in the command line. Every function was tested after implementation and then again in another round all together. Both rounds of testing were completed on an OSX 10.7.5 platform and an Ubuntu 14.04 platform. We also gathered other student’s opinions about the look and feel of the application. Technically, discovering the difficulty of designing this type of database interfacing application could also be viewed as an overarching experiment that the both of us participated in. The results from both are discussed below.

**Results**

In most cases, the response times of our application were the same as running queries on the command line. As result set tables grow larger, the amount of time it takes to return in our application slightly increases due to the fact that the data has to be processed twice, essentially: once by MySQL and then again by our Java code. As far as correctness, our application replicated each feature that we set out to implement very well. It properly connects to local and remote MySQL servers as well as display the information in tables properly. The Java MySQL driver correctly provides the verbose error reporting. From all of our internal testing of the application, the application properly, correctly, and tolerably executes all of the functions we promise it will.

The difficulty of building this type of application ended up being surprisingly high. The amount of small details that you must track with each table and each row in that table is almost as much data as resides in the table itself, such as the column that holds a primary key or the foreign key constraints being upheld by a particular table. Building an application that can correctly keep track of all of the tables’ metadata would require (at least for these fellows) more time, practice, and patience.