Team Members:

Zachary Garcia

Moham’d Al-Tahat

Hector Rodriguez

**Database Project: Phase 2**

In Phase 2 of our project we developed a desktop application that allowed users to interact with the database created in Phase 1. The application was written in Java and took advantage of Java’s user interface libraries and the Java Database Connectivity API. Our application’s main features included the ability to load data from a text file using two different methods, the ability to delete data from tables, and an interface to execute queries.

To efficiently develop our application and complete Phase 2 of the project, the tasks were divided in the following way: Zachary Garcia was responsible for implementing the user interface to execute queries and preparing the project report. He also implemented the database connection class that established a connection to our local database. Moham’d Al-Tahat implemented the user interface components that allowed the user to load and delete data. He also recorded the results of our scalability analysis and assisted with the database connection class. Hector Rodriguez implemented a program that prepared the data files to be inserted into our database. These text files contained 100 to 500000 data entries that were later inserted into our database for testing. Hector also prepared the graphs for our runtime analysis.

When developing our desktop application, there were two major tasks we had to complete. First, we needed our Java application to be able to connect to a database. After establishing a connection with our database, we needed to create user interface that allowed the user to interact with the database. To establish a connection with our database, we used the Java Database Connectivity API. Using JDBC, we created a class file that created a connection to a database. Once the connection was created we were able to create statement objects. To update data in the database, we used the executeUpdate function on our statement. A string was passed to this function which contained the appropriate SQL statement to update the data. We used the executeQuery function in a similar way, and passed a string to the function that contained the SQL query to be executed. When performing queries, the executeQuery function returned a ResultSet object which contained the results of our query. Data could then be read from the result set and displayed.

To develop our user interface, we used Java’s swing and awt libraries. Our user interface was designed using Eclipse’s window builder feature. This allowed us to drag-and-drop user interface components and then have the appropriate code generated to display those components. Once this was done, we were able to implement the appropriate actions for each of our user interface components. When our application is launched, a drop down menu is displayed where the user can select whether they wish to insert data, delete data, or write a query. If the user wishes to insert data, they are then asked if they wish to use single insertions or bulk load the data. Either selection will display a window in which the user selects a file to load data from. Single insertions were implemented by using a while loop to read each line of the file. Each line was then translated into the appropriate insert statement and the statement was executed with executeUpdate. Bulk insertions were implemented using the LOAD DATA LOCAL INFILE syntax. A single statement was prepared which read all lines from the file. When executeUpdate was executed, all data was inserted into the database. The performance of these two methods is discussed further in our runtime analysis.

If the user wishes to delete data, a drop down menu is displayed which allows the user to choose a table to delete data from. When a table is selected, the deleteTable function is executed to clear all data from the table.

If the user wishes to execute a query, a window is displayed with a text field where the user can enter a SQL query. When the query is executed, a result set is returned. Data is read from the result set and then displayed in a table that is included in the query window. A pop-up is then displayed which shows the execution time of the query. If the query entered is invalid, the appropriate error message is displayed.

When testing our application, it was necessary to generate text files with meaningful data that could be inserted into our database. To accomplish this, a program was written that randomly generated meaningful data and then wrote it into a text file. These test files contained a minimum of 100 data entries and a maximum of 500000 entries.

After our desktop application was completed, we analyzed the runtimes of our two methods for inserting data into our database. Using the LOAD DATA INFILE syntax, we were able to bulk load the data from our text files into the database. Our single insertion method generated a SQL statement for every row that was inserted from the text file. As expected, the runtime for the bulk loading method was significantly faster. For every insert operation that is performed, a connection is established to the database, the query is sent to the server, the query is parsed, the rows are inserted, and the connection is closed (MySQL 8.0 Reference Manual). When performing many single insertions, this sequence of events is repeated for every row that is inserted into the table. When bulk-loading the data, a connection with the database is only established once and then all rows are added to the table. By eliminating the unnecessary repetition of creating many connections to the database, bulk-loading the data is much faster than performing single insertions. The runtimes for input sizes of 1000, 10000, and 100000 are shown below:

We also analyzed the runtimes of five different MySQL functions using data set sizes of 100000, 200000, and 500000. The functions tested were join, sort, average, count, and min/max. The count and min/max functions finished the fastest due to the small number of operations required to perform these functions. The average, join, and sort functions took significantly longer since these functions require a larger number of operations. All functions finished in fewer than 30 seconds with a data set size of up to 500000. This indicates that these functions are practical for use with very large databases. The runtimes for each of the functions tested are shown below:

Documentation:

(a) https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html

(b) https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

(c) https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

(d) https://dev.mysql.com/doc/refman/8.0/en/load-data.html