Phase 2 Report
Joseph Camacho-Terrazas

Description of Methods/Techniques Used to Complete the Tasks

To complete this project, I used a few technologies which included: Python, MySQL, Faker, pymsql, and tkinter.

- Creating Random Files
  To accomplish this, I used a combination of Python and a library called Faker, as well as the random sampling within Python. Faker was used to generate random first and last male names, and random integers for touchdowns and total yards. The other fields were filled by creating an array of team names and positions and selecting a random sample from each. To create a file, the user runs the command "python Randomizer.py players 100000". The program takes in the argument "players" to execute the generation code for player data. The "100000" argument gets fed into the function and is used to generate a row of data 100,000 times in this instance. It also supplies the player ID's. Each attribute is separated by a comma and placed in a string, then the string is written to the file followed by a newline character. The file will then create a file with the name playersx.txt, where x is the number of rows the user requested.

  Note: I did not submit my randomizer code because it's not a specific requirement, but I can supply it if necessary.

- Connecting to the Database
  To connect to the database, I used the pymysql library. This allows for a simple connection to the database server. In my implementation, the user simply runs the command "python Phase2JCT.py 'ipaddress' 'username' 'password' 'database name'", where the quoted words correspond to the information of the server. Next, I create an init at the beginning of my class, so that I can pass these arguments into the program. To connect to the database, the connect command is run with the command line arguments passed in. It will then create a cursor that will allow queries to interact with the database and returns the connection and cursor.

- Insertion
  To complete an insertion, the technique is very similar for both. First, the function will connect to the database, and then store the given file name in a variable. To avoid case errors, I convert all user input to lowercase using .lower(). Next I used an if statement to check if the word "player" is present in the file name. This is so that the program would execute the appropriate code for each table in our schema. However, I removed the other table functionality and left only functionality for the players table. Next, for single insertion, the function will strip each row that is terminated by a newline character. Then for each row, it will split the data by commas and place each value in an array. It will then perform an INSERT INTO query using that row's data, and continue to the next line. Bulk insertion is handled similarly, but instead of splitting data out of the file, the LOAD DATA INFILE query is executed. Finally, the start and end times are recorded, and the file and database connection are closed. The function will output a success message into the command line as well as the UI.

  Note: For bulk insertion on my computer, MySQL had some security feature enabled that didn't allow for data inserts from a file that wasn't in a special directory.

I had to drop the input files in C:\ProgramData\MySQL\MySQL Server 8.0\Data\phase2. I read that C:\ProgramData\MySQL\MySQL Server 8.0\Uploads also works. But I promise bulk insert works.

- Deletion
  The method to complete the deletion of a table is the same as bulk insertion, the only difference is that it executes the delete MySQL commands instead of the insertion commands. The name checks and message displays are all the same though, minus file input and timing.

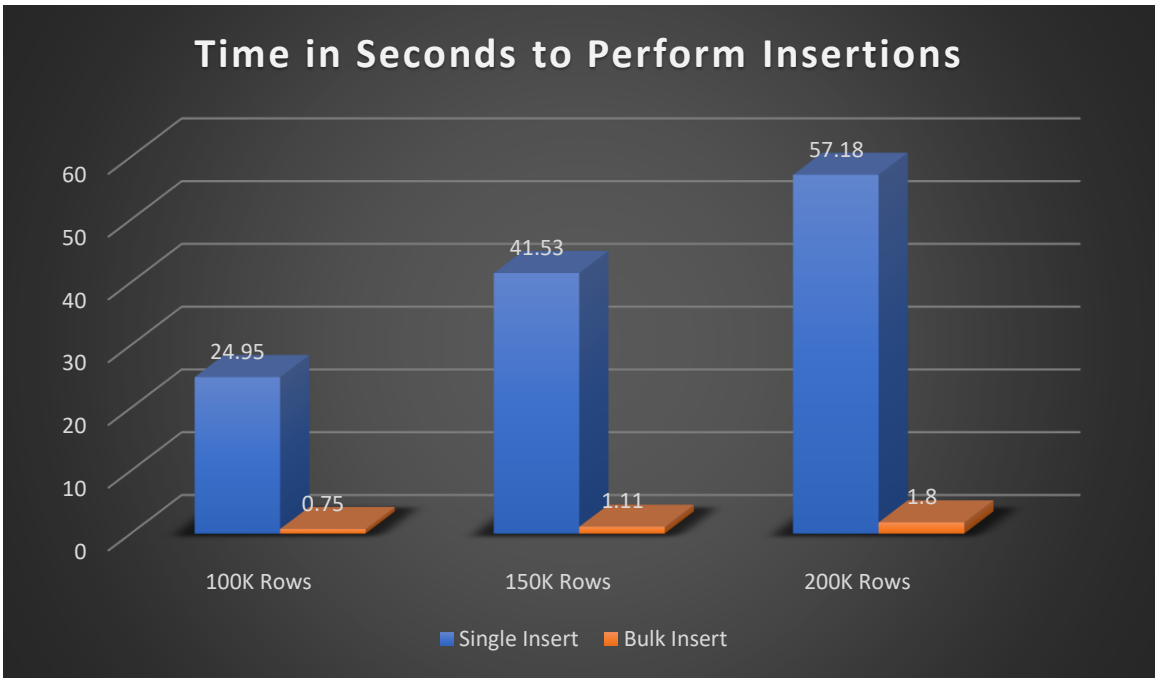
- Queries
  To complete a query, the method is mostly the same as the previously mentioned tasks. It connects to the database and will clear the canvas where the results are displayed for each new query. Next, it will check to see what table is mentioned in the user input and will execute the appropriate code. (Originally, my query code worked for all 3 tables, but I only left in player functionality.) The function will the execute the query using the pymysql cursor, and then fetch the name of each attribute and the data rows and store them in arrays. Then, each attribute name is sent to an output string in a formatted fashion, followed by each subsequent row of data terminated by a newline character. The output string is then printed to the UI canvas, and a success message is displayed.

  Note: Due to a limitation of the Tkinter entry boxes, this UI only processes single-line queries.

- UI
  To create the UI, I used Tkinter. This library makes it easy to input and display data and execute commands with buttons. The labels mostly just give instruction to the user. Some labels have functionality though. The top of the UI displays the database connection status. You can quickly see what database you're connecting to as well as the username and IP address. If you click the test button, it will run the connect command quickly and let you know if you've successfully connected to the database. The text also updates without the test button when you run any other function. I also implemented a label that reports the file you are inserting and displays how long each insertion method took for that run. Other than the canvas and scrollbars, the UI is simple.

- Error Handling
  To handle any errors, I used try and catch to handle exceptions. When an exception occurs, a window will pop up in the UI with the exception so the user doesn't have to look at the terminal, and the program will keep running. I also print a few words in the terminal for debug purposes, but these are not all errors, it also displays successes. To handle errors when making a text entry, I have the UI display an error box as well. My program is coded so that you can only perform tests on the players table. For example, if you try to query games, this is valid in MySQL, but my program will not allow it. It will not execute any query for any table other than players. If you enter a query for players, but happen to make a syntax error, the error box will display the exception. This happens for all MySQL related errors as well. Finally, if the program cannot connect the database, it will display the error, and exit the program once you click ok. I did this

because the UI is useless if you can't connect, since the information is entered via command line. That is why I added in the connection statistics so you can be sure you entered the correct information.



Raw Data:
Single Insertion 100k: 24.96s
Bulk Loading 100k: 0.75s
Single Insertion 150k: 41.54s
Bulk Loading 150k: 1.11s
Single Insertion 200k: 57.18s
Bulk Loading 200k: 1.43s

**Times varied based on the programs I had open.*