Team 12
CS482
11/4/2018

# Project: Phase 2 Report

Lucas Herrmann, Victor Lozoya, Michael Curtis

Tasks: For this project, Lucas Herrmann wrote the PHP script for querying the database and displaying the results in an HTML table wrote the first draft of the report, and performed scalability analysis. Victor Lozoya wrote the PHP script for file upload and bulk insertion as well as creating test data, and worked on the report. Michael Curtis wrote the PHP script for single row insertion and deleting the contents of a given table.

Methods/Techniques: For phase 2 of project 1, creating a web interface for the mySQL database created in phase 1, we chose to use PHP and HTML for our programming language. HTML forms was largely used for the visual page content and formatting, and PHP along with the build-in PHP extension mySQLi was used to connect, insert rows, delete table contents, and query the database. Victor used PHPStorm IDE to write the files which comes with a built in web server. After experiencing issues and doing research he determined it would be better to use XAMPP which has an Apache web server that worked for the localhost.

File Upload single insertion: We use an xml text box for the user to enter a text file. Then we start parsing the data. We begin by checking the filename and storing it into a variable then we check the file name to determine what table the data will be inserted. We used php's fopen method to get a pointer to the begin of the file. Then we used a while loop to parse the file until the end of the file has been reached. Since the values of the table are formatted on a line we use php fgets to read each line and set that to a variable. Since the values are separated by commas we use the php explode method to store each value separated by commas to an array. To ensure each line has the correct number of arguments we use php array_pad method which allows you to specify the limit and length of the array and returns null values if it doesn't find one on the line. Once the values are extracted into the array we make sure they aren't null and insert them into the table using mysqli_query method which has two parameters which is the mysql connection and the query. We took advantage of mysqli that is part of php and used that to output errors when they occur. The method is mysqli_error.

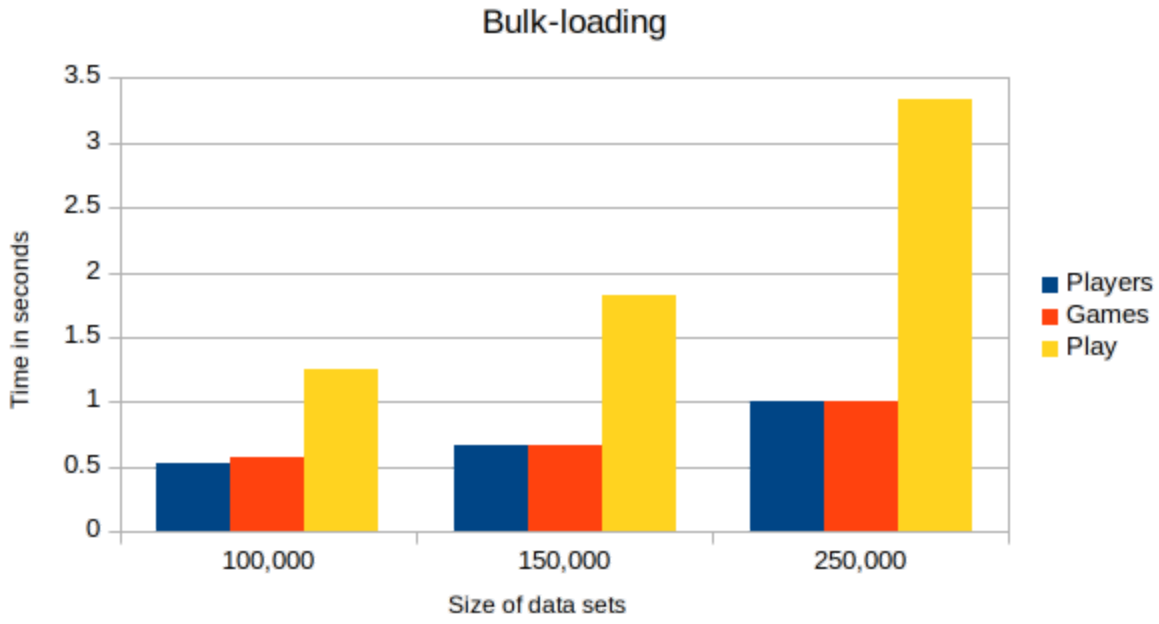
Bulk-loading: For the file upload we used an html upload form to allow users to browse through the directory folder and select the text file. Once the file is uploaded and we run a check to make sure then we extract the name from $_FILES array where files are automatically stored using xml forms. The depending on the file name we insert the data

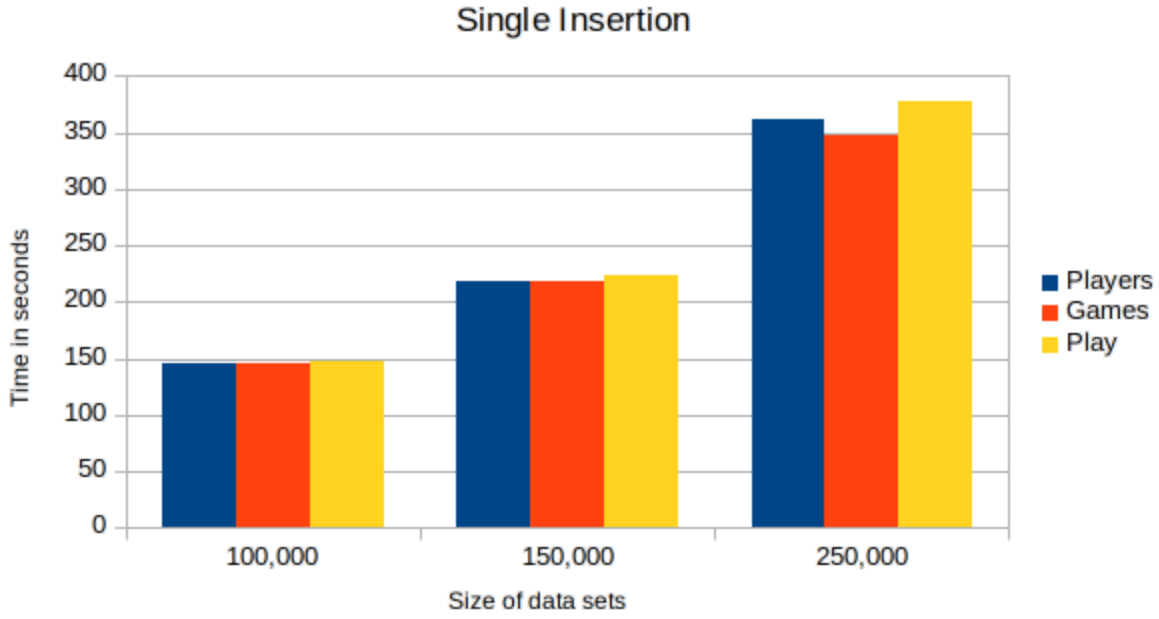
to the corresponding table. We use php LOAD DATA LOCAL INFILE method which returns a string as a query. So the good thing about this method is we can separate values by , and lines are terminated by \n line. Once we have the query we use mysqli_query method which we used for single insertion and enter it into the database.

Querying: For the query textbox, we stored the text entered as a string, and sent the string to a separate php file. That file sent the string to the database using the mysqli_query built-in function. This returns an array of arrays, which we used simple foreach loops to extract into each individual element. Finally, the results were displayed in a simple HTML table with a 1 pixel black outline, and a “go back” javascript hyperlink was added to the bottom of the results page.

Performance: To test performance we used php’s microtime method which allows you to time how long a function takes to execute. We recorded the time before the function, then immediately after and subtracted to get the elapsed time of the function. During the development process we simply echo out variable to ensure we got the right info. For example once the file is uploaded we used php file_get_contents method to store the data to a string then we echo that string. This helped determine if we needed to keep working on our functions. We used the isset on the $_POST array to ensure that the file has been submitted through the html upload form. To ensure it was uploaded we used the is_uploaded_file method on the $_FILES array to ensure the file was uploaded after submission. To create test data for the file upload we used a java program which generated a text file.

[Analysis of single vs. bulk insertion here]

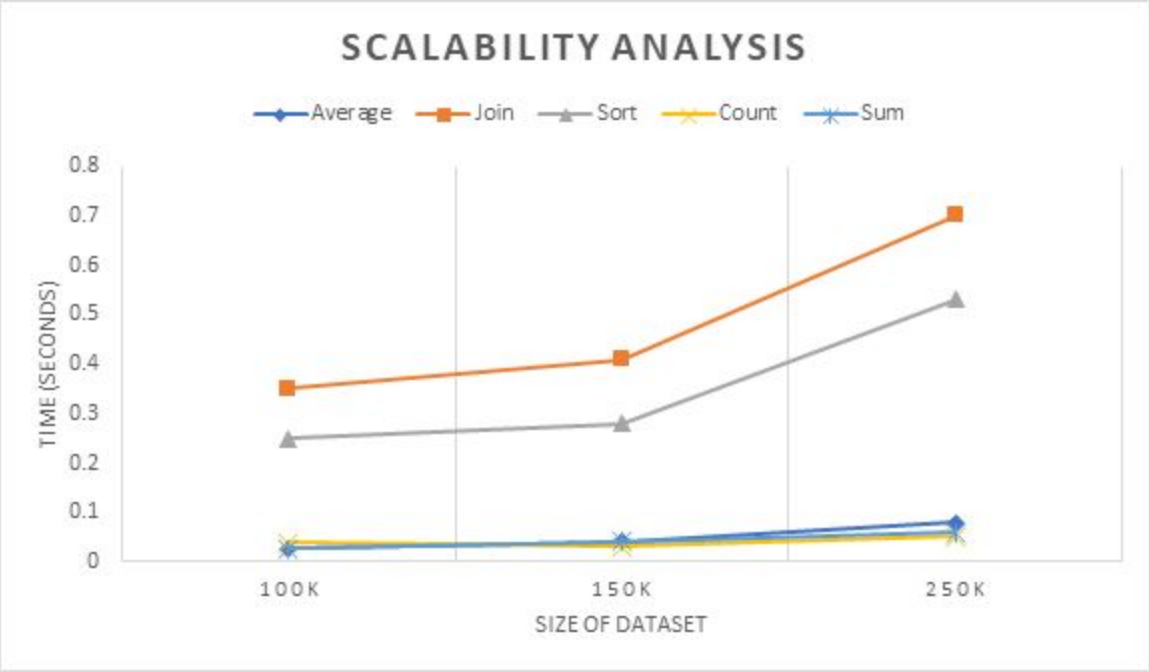



From the chart above we can infer that when inserting to the different tables it takes approximately the same time for each data set  when doing single insertion. For bulk insertion we have examined that when inserting into the table Play it takes almost double the time but players and games are the same. We conclude this is because of the foreign key constraints on the table Play PlayerID and GameID refer to the PlayerID and GameID from the corresponding tables. Before it can be inserted it must check that

the ID exist in both tables. That is the only thing we can think of why Play takes longer. As the data increases so does the time to insert the data into the database. There is about a 50 second increase when the data increases by 50,000 from 100,000 for single insertion which is about a 33% increase. If you increase the data by 150,000 from 100,000 we observed that the time to execute is greater than double the time we calculated a 133% increase. From this we can infer that as data sets start to increase so will the time to insert it into the database through single insertion at an exponential growth rate. We observed that bulk-insertion is much faster than single insertion as we originally expected. We conclude this is because of the single query for bulk-loading vs multiple inserts for single insertion. For bulk-loading when you increase the data by 50,000 from 100,000 there is a very small difference not even a second. When increasing from 100,000 to 250,000 the time doubles. There is a slight difference with the Plays table with a 50,000 increase in the data there is about half a second difference but when you increase data from 100,000 to 250,000 there is almost a 2 second difference. Again we are concluded this is because of the foreign key constraint which requires more work to check if the ID exist.

Scalability Analysis:




We tested the following functions: average of a column, joining two tables, sorting tables by a column, counting the number of entries in a column, and gathering the sum of the column. We used dataset sizes of 100,000, 150,000, and 250,000 rows.

From our results: average, count, and sum scale rather linearly in proportion to the size of the dataset. By contrast, the time it took for joining and sorting tables increased dramatically in proportion of the size of the dataset, moving from 0.35s and 0.25s for 100k entries to 0.7s and 0.5s for 250k entries respectively. From this data, we can determine that joining and sorting takes about 10x the time of the mathematical functions, and increases at a much steeper rate as the dataset size increases.