Cameron Tauxe
Jared Peterson
Kathleen Near
CS 482
4 November, 2018

Group Project: Phase 2

Assigned & Completed Tasks

Our team consisted of three members. Cameron Tauxe led software development, creating both the front and back end of our database-connected web application, Jared Peterson aided in writing Python code for the back end, and both with the addition of Kathleen Near tested functionality and wrote up documentation. Where documentation was concerned, Jared and Kathleen gathered the data and wrote up the analysis and Cameron fashioned the graphs.

Methods

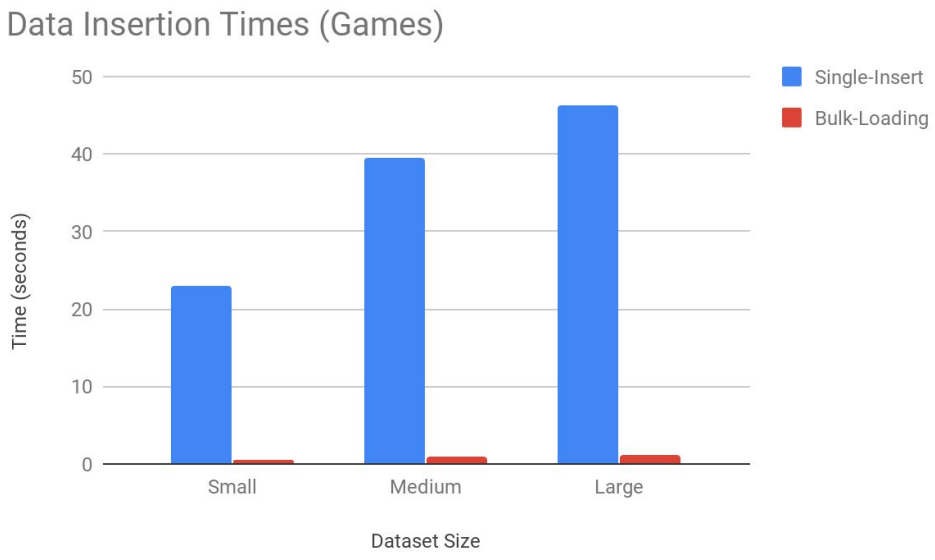
Our solution to the project roughly follows the LAMP (Linux Apache MySQL PHP) development stack (but in this case, the 'P' stands for 'Python' instead of 'PHP'). The database and the website for accessing it is hosted on an AWS server that Cameron is renting (mysql.camerontauxe.com). On that server is a running MySQL server instance which stores our database and an Apache web server instance which serves the web pages and executes the back-end code via CGI. On the front end, javascript code handles sending the necessary HTTP requests to the server and uses D3 (Javascript data-visualization library) for binding the received query results into a table.

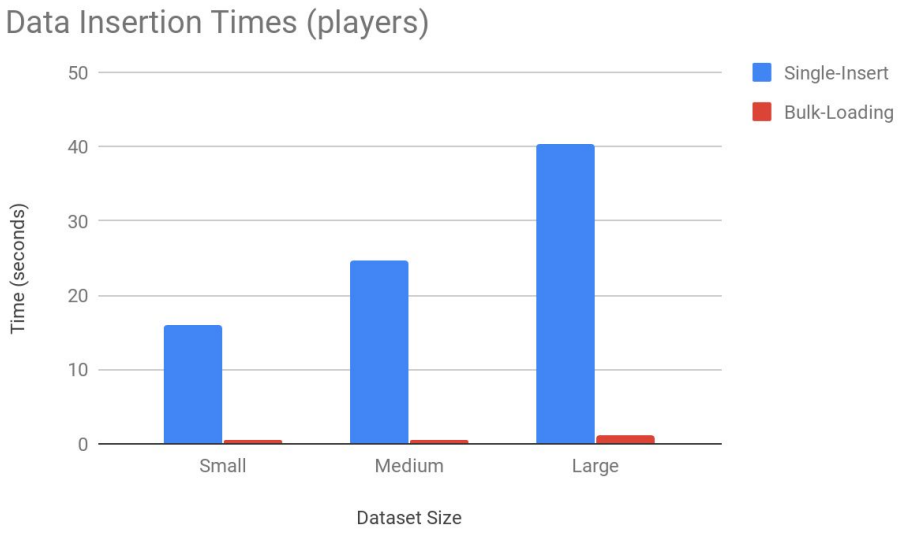
For security, elements of SQL queries are sanitized wherever possible. In the case of submitting a full query (which is impossible to sanitize), the back-end connects to the MySQL instance using a special "read-only" user which is only allowed to use the "SELECT" operation. Attempting to do anything else will result in an SQL error (which will be returned to the client as a 400 Bad Request error).

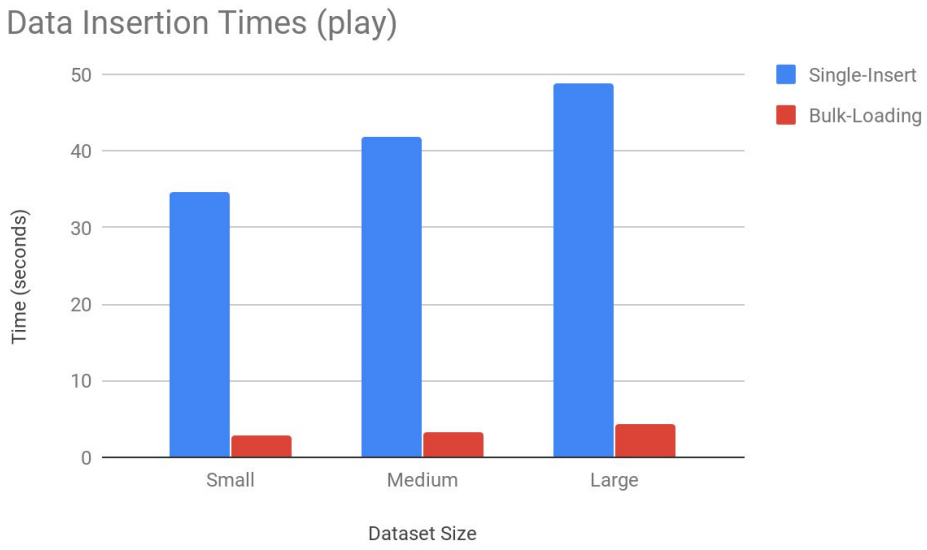
The two methods of inserting data (single-insert vs. bulk-loading) are handled differently on the server-side. In both cases, the text of csv file containing the data is provided to the server as the body of a HTTP request. In single-insert mode this data is parsed using Python's own csv-parsing utilities and inserted by sending individual "INSERT" statements to MySQL (one statement for each row). All of the changes are committed at the end. For bulk-loading, the data is written to a temporary file on disk and then we send a "LOAD DATA INFILE" statement to MySQL to load all of the data.

Insertion Time Analysis

Depicted in the following three graphs is the time required to insert new data into different tables of the database from files of varying size. This was done by selecting the table, file, and insertion type from the front end of the website. In blue, is the time taken for single insertion of the data, and in red is the time taken for bulk insertion. It quickly becomes clear that using single insertion is substantially slower than bulk insertion. At its most extreme time difference single insertion takes roughly 40 times as long as bulk insertion and its time rises at a higher rate with each size increase. Bulk insertion rises an average of 0.4 seconds between size increases whereas single insertion averages an increase of 10 seconds each size increase.





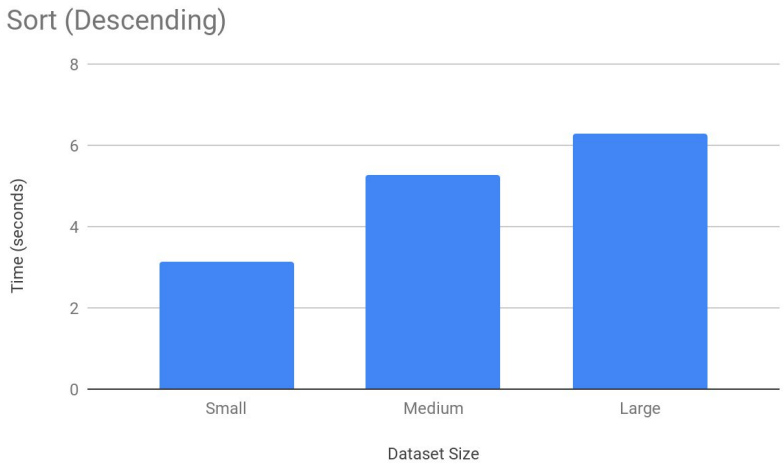


Scalability Time Analysis

We tested five different MySQL functions with datasets of three different sizes. The five functions tested were sorting (descending), simple join, count, maximum, and average. The three dataset sizes tested were small, medium, and large. The small dataset size held 150,000 elements for the games table and 250,000 for the play table. The medium dataset size held 250,000 elements for the games table and 300,000 for the play table. Lastly, the large dataset size held 300,000 elements for the games table and 350,000 for the play table. The times recorded timed not only the MySQL response, but also the time it took for the back-end to finish the request (i.e. formatting data returned by MySQL). The recorded times for all of the functions tested increased as we tested them with larger datasets. In order of fastest to slowest the the functions are count, maximum, average, sort (descending), and simple join. The count, maximum, and average functions all had similar times with the sort (descending) and the simple join functions being considerably slower than the other three.

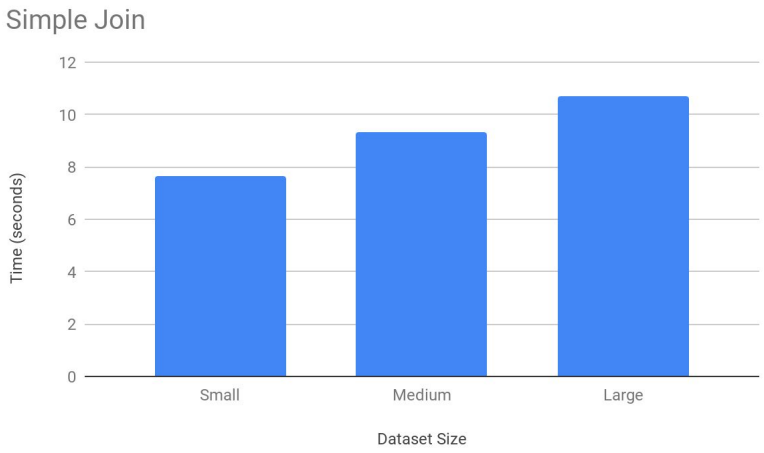
Sort (Descending)

Of the five functions tested sort (descending) was the second slowest among all datasets following only the simple join. The times recorded for the sort (descending) function for the small, medium, and large datasets were 3.145 seconds, 5.266 seconds, and 6.272 seconds respectively. The query used in this test was as follows “SELECT * FROM games ORDER BY ticketrevenue DESC”.



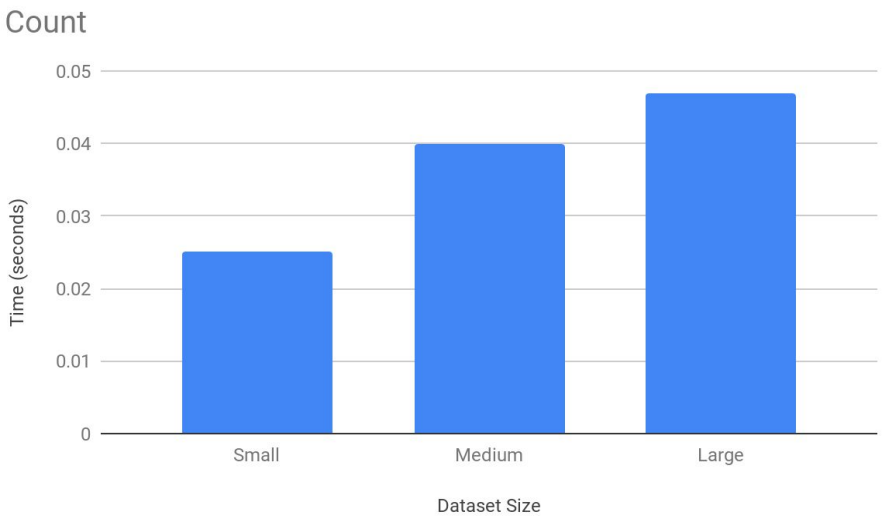
Simple Join

The simple join function was the slowest function we tested taking an average of 9.215 seconds to complete over our three dataset sizes. The times recorded for the simple join function for the small, medium, and large datasets were 7.63 seconds, 9.334 seconds, and 10.681 seconds respectively. The query used in this test was as follows "SELECT * FROM games G, play P WHERE G.gameid = P.gameid;".



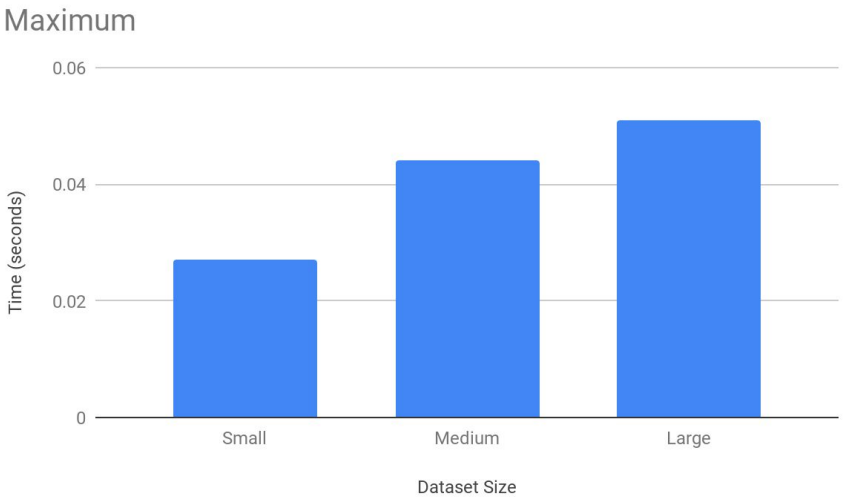
Count

The count function was the fastest of the five functions we tested averaging just 0.037 seconds to complete over our three dataset sizes. The times recorded for the count function for the small, medium, and large datasets were 0.025 seconds, 0.040 seconds, and 0.047 seconds respectively. The query used for these tests was "SELECT count(ticketrevenue) FROM games".



Maximum

The maximum function was the second fastest function tested after the count function. The times recorded for the maximum function for the small, medium, and large datasets were 0.027 seconds, 0.044 seconds, and 0.051 seconds respectively. The query used for these tests was "SELECT max(ticketrevenue) FROM games".



Average

The average function was the third fastest function tested, but had times far more comparable to the maximum function rather than the sort (descending) function. The times recorded for the average function for the small, medium, and large datasets were 0.033 seconds, 0.054 seconds, and 0.074 seconds respectively. The query used for these tests was “SELECT avg(ticketrevenue) FROM games”.