Team 1 SQL Database with PHP Project

Shane Grayson
New Mexico State University
Las Cruces, United States
shaneg@nmsu.edu

Ian Navarro
New Mexico State University
Las Cruces, United States
iannava@nmsu.edu

Andrew Baca
New Mexico State University
Las. Cruces, United States
abandrew@nmsu.edu

ABSTRACT
Phase 2 of the Database Management Project is an extension of Phase 1, where we made a SQL database for the offensive players of a football team. Prior to phase 2, we had a working database in the MySQL Workbench with three tables. These tables included players, games, and play. In phase 2 of the project, we built a web interface for the database that was made in phase 1. Some basic properties of the web interface include the ability to bulk-load data to the tables based on filename, the ability to enter queries, delete from tables, and provide runtime analysis of the different operations. This paper will cover the work done and the methods used on the project, each group members specific tasks in relation to the project, and a runtime analysis and conclusion based on the different methods within the interface.

Author Keywords
SQL, PHP, Schemas, Queries, Bulk-loading, Single-Insertion, Bootstrap, HTML,

INTRODUCTION
For the last phase of the project, we had made a functional schema in MySQL containing information regarding offensive players of a football team and the games that they have played in. This phase of the project, we have built a web interface for that schema with the basic abilities to insert into a table, delete from a table, run queries on the schema, print the results, and provide runtime analysis. We decided on PHP as the programming language used to undertake this project for a variety of reasons including its libraries to communicate with MySQL. We also used the Bootstrap framework for easy and clean handling of the front end user interface of our webpage. The following Sections will cover the methods used, the work done, and the overall analysis and conclusion of our web interface design to our database.

TOOLS USED
While working on this project, all of the group members used the linux computers located in Science Hall 118 for use of the MySQL database installed, as well as the public_html folder that linked to the cs.nmsu domain online. We chose PHP as the primary language for the back end of the website for a variety of reasons including the libraries included in PHP for communication with MySQL, as well as the group members wanting to undertake a language they were unfamiliar with for a new experience. Much of the front end of the web interface was generated using the Bootstrap framework, using HTML to give the interface an adaptable, professional and user-friendly feel to it. We also composed a random file generator in Java to test a variety of different fields of data.

ARCHITECTURE/DESIGN
Our project can be accessed from any computer at www.cs.nmsu.edu/~sgrayson. This will lead you to the front page of our interface, which is located in the index.php file as far as code is concerned.

Upon entering the main page of our project, you will find a traditional main page for a website, with a traditional navigation bar at the top leading to different database management methods. The main page also contains a project description, along with features, analysis of different insertion times, and developer info at the bottom of the page. The SQL-Query tab in the Navigation Bar will lead to a page with a textbox and a submit button where you can type in a single query and click submit, and the results will be printed just below. This was done using PHP code for a textbox, and linking the string typed in the text box to the sql_query.php file where the input was parsed and processed properly using mysqli_query() command as well as some fetch and row commands used for php code to communicate with the MySQL Database.

Another tab in the navigation bar of the web interface is the File Management tab, and this is where you can either bulkload or single-insert bulkload into certain tables of the schema depending on the file name. The bulkload operation is done in PHP by storing the file path into a string, and then using that string in the “load into” command so that SQL ran read the contents of the file into the chosen table based on the name chosen. The single-insert bulkload operation is similar to the regular bulkload operation and can be used to store mass amounts of data into the database. The regular bulkload operation is implemented by browsing for the intended filepath, opening the file, reading the file into a PHP variable, and using mysqli() functions to parse through the file and run each insert command accordingly, closing the chosen file at the end of the insertions process. Within the file management page, you will also be able to find a view tables dropdown menu, where you can view what is currently inside the table, and see if the insertion worked properly. there is also a time function implemented to show the total time it took for the MySQL operation to complete.

The data management link in the navigation bar leads to the page where you can either do single row insertion, as well as deletion functions to our tables within our schema.

The About tab in the navigation bar will lead to a detailed summary of the phase 2 of this project as well as some run time analysis charts of the bulkload versus the single insertion bulkload methods of insertion.

WORK DONE

The work done on this project was handled by three group members, stated as the authors above.

Shane has spent most of his time on this project working on the back end with PHP, as well as connecting the back-end PHP with the front end that Ian had worked on. Some of these main tasks included the initial connection of the website to the NMSU database as well as the initial interaction Bulk load In-file insertion functionality. Shane also set up many of the testing elements such as finding the time for SQL queries and Bulk insertions both Infile and Singleline. Some of the important functions used to allow most of the back end functionality to work for the website were $_POST which is a super global that allowed us to send over information from the HTML side of the code; mysqli_query() which, when connected to a database would allow us to draw one query against said database; microtime() that allowed us to track in real time the amount of time a query against the database took; mysqli_set_local_infile_handler() and mysqli_set_local_infile_default() change certain parameters in mysql to allow infile insertions to work with a mysqli_query function call with default reverting the infile settings; and many of the constraints for SQL query and data management sections. He also worked on getting the errors for SQL query and Data Management sections to correctly show the errors that a user would come across if making an error while querying.

Ian has spent most of his time on the project working on the front end of the Web Interface using the HTML framework and Bootstrap, initial SQL-query view, as well as integrating the front-end to the back-end with the aid of Shan. Some of his main tasks included the implementation of the website's interface in which he did through HTML. He extended the website's design and capabilities with Bootstrap. With this tool Ian was able to make the website's accessibility work with smartphones, desktop and different sized browsers. The website's main page is designed to show the project, the team's goals and the website's features. He implemented the SQL-query interface with an interactive text-box where users can type in queries and hit the button to submit. The file management page is designed to load files with the purpose of having the option of bulk-loading or single-insertion and a button for each section for submission. This page also contains the view-list, where the user can view the different tables within our database. The data management page was implemented with an search-input box where user can input single deletion, single insertion, and table deletion queries. As he implemented this page, he also included the view-list feature. Ian also added an about page where users can find more details about our project and each team member's responsibilities. The website's added footer is to show the website's professionalism and functionality. Ian also worked with Shane with the integration of the full front-end and back-end implementations. As Ian was designing different interfaces, he needed to know where the PHP codes needs deploying beforehand. He also did the bulk-load, single-insertion and function analysis with the help of Shane. Excel was the tool used to graph the data collected from our website.

Andrew has spent most of his time on this project working on the back-end with PHP, specifically with aspects of insertion such as single row insertion, bulkload insertions with the help of Shane, and single-insertion bulk loading. For the bulkload method, he used a string object in PHP to collect the file path via the browse function on the front end that Ian had created. From there, he assigned the table based on the file name to be loaded into, and used a mysqli_set_local_infiler() function in PHP to format the data to be loaded into MySQL. From there, all the loaded data is ran through the mysqli_query(). For the single-insertion bulk loading, he once again captured the file path through the browse function in the front end, form there he opened the file and read the whole file into a string object in PHP, and used a do while loop with a mysqli_next_result() function as the condition to check if there is another query to run. From there, and uses the store and fetch result functions to execute the single row query to MySQL. What this basically did was read substrings line by line, terminated at the end of the query (semicolon), and sent single row insertion queries, one after the other, to MySQL, until the file string was terminated, and then closed the file. He has also worked on the data end of things, making a random file generator in java that has the capabilities of generating various files for the players, games, and play table of scalable sizes for this project both for bulk-loading (where the files are generated to the size the user enters, and all the values are separated by commas, and new line from tuple to tuple) as well as the single insertion bulk loads (where there will be multiple “insert into” query structures.

ANALYSIS

We conducted run times analysis on multiple different operations of the web interface. These included analysis with single insertion (See fig. 2) and bulkload insertion (See fig. 1) using 100k, 200k, and 300k data elements. All query tests conducted were against 100k, 200k, and 300k sized tables. We also tested 6 queries against database. These quieres included a simple join between the relations players, games, and play; we tested average, orderby, count, min, and max queries.

The Queries that we tested the database on included (for the join query we limited to 100 to avoid long wait time on printing due to our website):

- **Sum Query (See Fig 3.)**
  select sum(p1.touchdowns)
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id;
- **Average Query (See fig. 3)**
  select avg(p1.touchdowns), p1.team_name
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id
  group by team_name;
- **Count Query (See fig. 3)**
  select count(g.results), p1.team_name
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id
  group by team_name;
- **Join Query (See fig. 4)**
  select *
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id
  limit 100;
- **Min Query (See fig. 4)**
  select min(p1.touchdowns), p1.name
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id
  group by team_name;
- **Max Query (See fig. 4)**
  select max(p1.touchdowns), p1.name
  from players p1, games g, play p2
  where p1.player_id = p2.player_id and g.game_id = p2.game_id
  group by team_name;

CONCLUSION

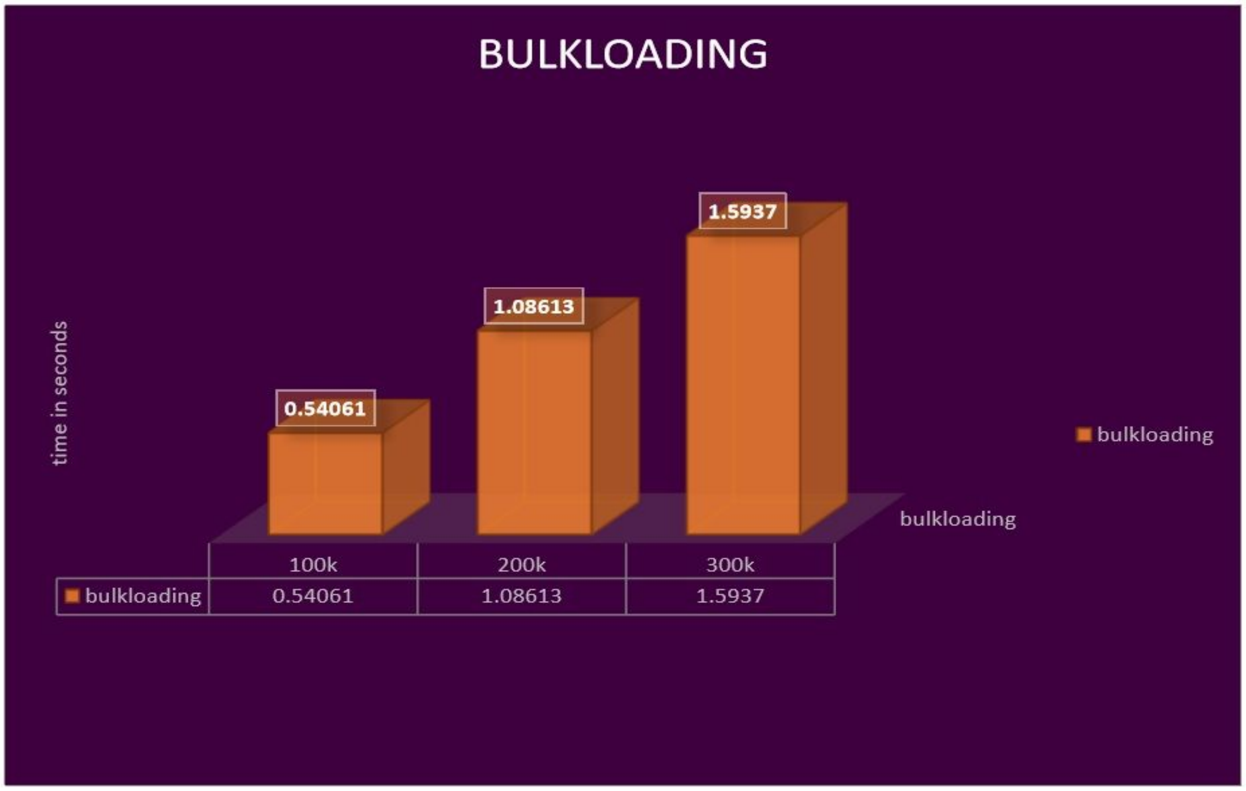
After the completion of phase 2 of our database project, we now see that most, if not all, databases need to have some sort of front end interface in order for the everyday user to be able to use the tool. We also see the various techniques and languages that you can use to link the user interface with the database. A typical project like this has to be structured carefully, and some strange issues can arise through strenuous testing of the system such as lag time from the web interface to MySQL, Nuances of learning a new language and the characteristics that come along with that, and using functions properly for MySQL to communicate with the front end properly.

With our runtime analysis, we see that bulk-loading is significantly faster than single-insertion bulk loading. We can also see from our results that all aggregates come at the same price of time, where joins come at a slightly higher time cost.

REFERENCES

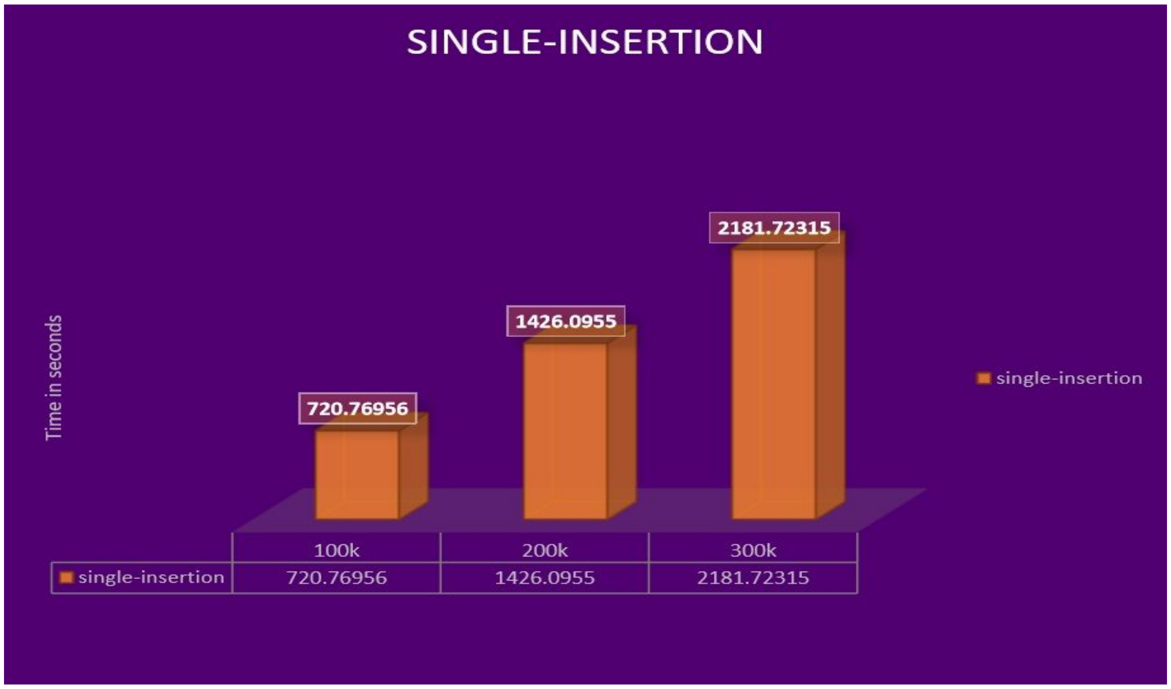
1. https://www.w3schools.com/php/showphp.asp?filename=demo_form_validation_escapechar
2. https://www.w3schools.com/php/func_mysqli_query.asp
3. https://razorsql.com/articles/mysql_column_names_values.html
4. http://php.net/manual/en/function.microtime.php
5. https://getbootstrap.com/docs/4.1/getting-started/introduction/
6. https://stackoverflow.com/questions/13579810/how-to-import-data-from-text-file-to-mysql-database
7. https://www.w3schools.com/php/func_mysqli_multi_query.asp

Graphs:



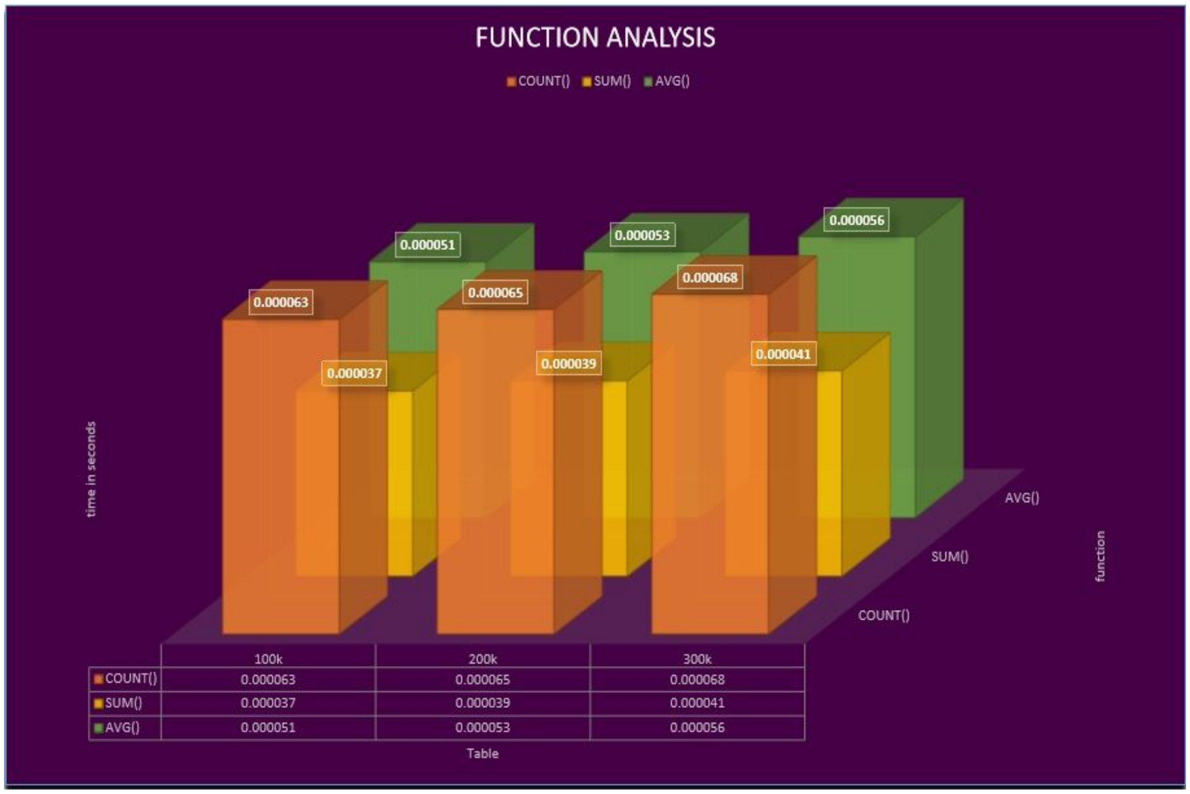
	100k	200k	300k
🟧 bulkloading	0.54061	1.08613	1.5937

Figure 1:Bulkloading Analysis



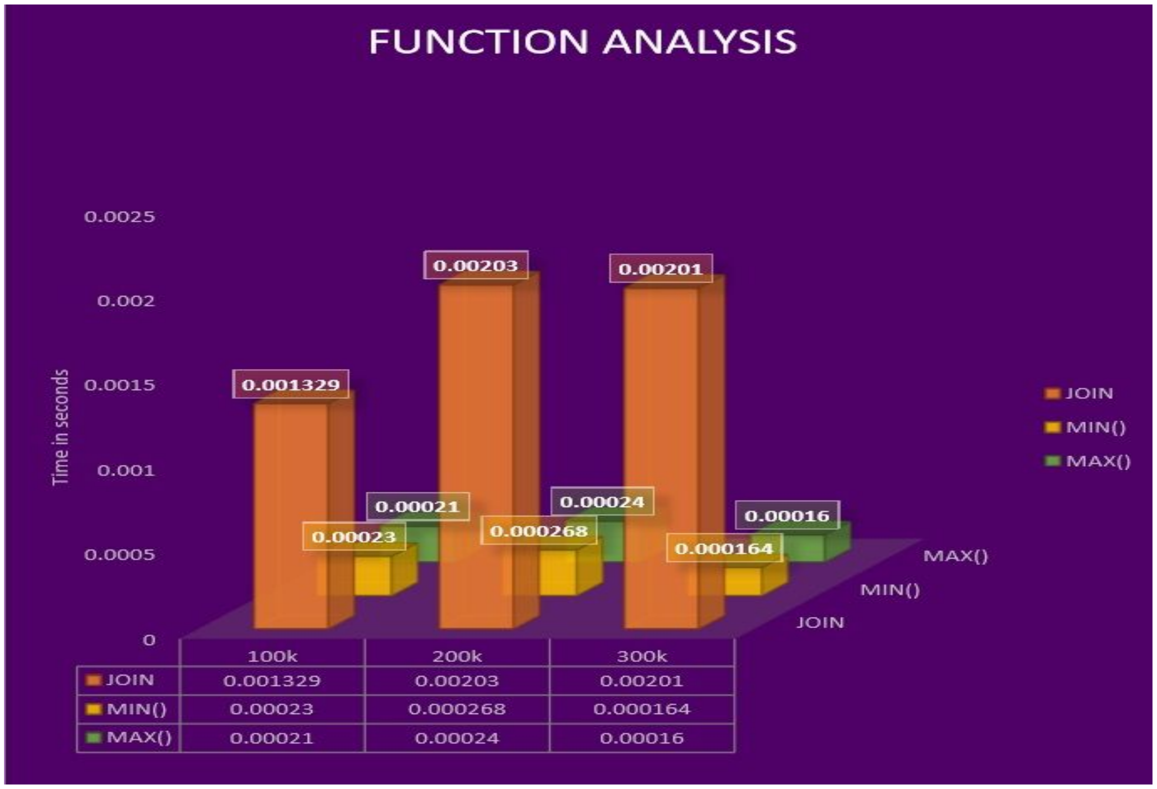
	100k	200k	300k
🟧 single-insertion	720.76956	1426.0955	2181.72315

Figure 2: Single-insertion Analysis.



	100k	200k	300k
🟧 COUNT()	0.000063	0.000065	0.000068
🟨 SUM()	0.000037	0.000039	0.000041
🟩 AVG()	0.000051	0.000053	0.000056

Figure 3: Function Analysis on Count(), Sum() and Avg().



	100k	200k	300k
🟧 JOIN	0.001329	0.00203	0.00201
🟨 MIN()	0.00023	0.000268	0.000164
🟩 MAX()	0.00021	0.00024	0.00016

Figure 4: Function Analysis on Join, Min() and Max().