Team: 032
Stage 6: Project Report
05/04/2022
Rithik Morusupalli, Daniel Dilan, Qiyuan Cheng, Aryaman Joshi

1. **Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

The biggest visual difference is that the comments do not appear directly under the song they belong to, instead they are in a separate section. Additionally we added a top artist list and a hot songs list to show off our SQL databases and add recommendations for the user.

2. **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

We successfully were able to get users to be able to login, and search for songs, artists, or albums in our database. They could also choose a song and make a comment on it. A user could see all the comments that were made. We failed to organize these comments, so that it was more accessible to make a comment on a single song without the song id, and being able to see all the comments of one song.

3. **Discuss if you changed the schema or source of the data for your application**

We did not change the source of the data, but we added a few things to our schema. We added a password table, so that we could have login functionality instead of through Spotify. We made the comment ID's integers instead of strings so that we could more easily add comments. We also got rid of the sub comments as it was too complicated to integrate into the application within the timespan.

4. **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

We added a table for passwords to support our login function. This seemed important to differentiate between which user is making a comment. Additionally we removed the comment to a comment table, because in our current UI it became redundant.

5. **Discuss what functionalities you added or removed. Why?**

We added a new functionality named "Hot Songs Right Now", it is related to our trigger. If a song has more reviews and likes than others it means that this song is currently followed by many users, it is the hottest. Hence, when a song is reviewed, it will automatically obtain 5 hotness, and it will also obtain hotness based on the range of the number of likes. We design this

functionality to help users save their time if they want to know the hottest song currently. We removed the functionality to show the comments directly under the song they belong to, because we need to use a linked page on this, but at stage 5, we focused on the trigger and stored procedure, and did not totally know how to make a linked page.

6. **Explain how you think your advanced database programs complement your application.**

We had four advanced database programs. The two that we implemented in stage 3 as well as the stored procedure and trigger we implemented in stage 5. One of our first advanced programs we implemented using an advanced SQL query was the ability to sort and find the top artists right now using our likeCounter value assigned to each song. Through the use of keys we then sort the top ten artists based on who has the greatest amount of likes on all of their songs. This compliments our program as in every song based application, there is always an option to sort for the top artists and so we wanted to implement that in our application as well.

The second advanced program that we implemented was our second advanced SQL query from stage 3, our searching method. This advanced program became the basis of our application. Using an advanced query that returned all songs, albums, and artists found when the user searches for a string, this program achieved our most important function of a user not having to know the entire name of an artist, song, or album in order to “discover” it in our database.

Between stage 4 and our final demo for stage 5 we also implemented our final two advanced database programs: one trigger and one stored procedure based program. Our tigger based advanced application allowed us another way for users to directly interact with our database and change a ranking system. While we already allowed users to update the database through comments and likes we realized that we had no way to alter the order in which users were presented the songs in terms of how popular they were. That is why we designed our trigger to fit the functionality of a “Hot Songs Right Now” button. When the user interacted with a song (i.e. commenting on it) it would cause the trigger to find out how popular the song was through the amount of likes it had and then assign it a “hotness” value which was ordered and returned when the users clicked on the “Hot Songs Right Now” button.

The final advanced database program we implemented was our stored procedure function. We wanted users to be able to login to the application as a way to personalize their experience and have access to the amount of times they have interacted with our database. That is why we decided to have our stored procedure be used to store login information from users and return messages if the login was successful or unsuccessful and if successful, show the amount of comments a user has made. We believe this improved our application as it adds a personalized touch to each user’s experience almost like a user profile.

7. **Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.**

    a. Qiyuan: I tested our database sql file in localhost MySQL first and then uploaded it to GCP, the sql file ran successfully in localhost but failed in GCP. I found that in the Windows system, MySQL is not case sensitive, but GCP is a Linux system, it is case sensitive so the sql file ran failed, it took me a lot of time to check the case. I used SQLyog, a GUI SQL tool, to test whether the sql query can work on our database, which is much earlier than testing it in google shell or MySQL workbench. It also can show the trigger and stored procedure we make, which saves me a lot of time.
    b. Aryaman: Debugging our stored procedure was difficult. Because there were many pieces within the stored procedure, especially queries. It was helpful unit testing the stored procedure by testing the queries independently. Once the components of the stored procedure work as expected, testing and debugging the output of the whole stored procedure becomes easier.
    c. Daniel: I had challenges inserting new comments into the database. This was because our schema originally had the comment primary key as strings, so it was hard to ensure that the new comment would have a unique primary key. By making it integers, we could ensure that the primary key would not be a duplicate by finding the max id of the integer keys that were already in the database, and that fixed the problem.
    d. Rithik: One other challenge that we faced was during the creation of our first SQL database. Specifically when gathering the data to fill out our songs, albums, and artists tables. To get this data we used a Spotify API that would find slightly over 1000 artists, then using that artist list find songs linked to that artists to get slightly over a total of 1000 albums, and finally use those albums and find a total of slightly over a 1000 songs. However, when we ran the API using a python script we ran into issues with our gathered data as there were duplicate songs linked to multiple artists which caused errors due to the way we assigned primary keys and foreign keys in our tables. To solve this issue we resorted to breaking our database into smaller chunks in order to isolate the duplicate songs and manually removed them from our data. To avoid this issue, I would suggest future teams look to test an API that they would use to gather data in smaller batches so

they do not have to go through thousands of lines in a database by small steps as we had to.

8. **Are there other things that changed comparing the final application with the original proposal?**

We added a hot chart that's based on total interaction (likes and comments) that the user can see and increases exponentially based on the current hotness of the song. We originally were not going to include this at all.

9. **Describe future work that you think, other than the interface, that the application can improve on**

Adding in comments to comments seemed redundant for our current application, but in the future with a cleaner interface adding this functionality would be an improvement. Also, using NoSQL would allow the application to run faster when considering expanding the users and the size of the available song library.

10. **Describe the final division of labor and how well you managed teamwork.**

We worked on the project together and in person. We would usually pair code with two members while the other two helped answer any questions the coders might have. If there were non coding tasks or multiple tasks we would still come together to complete them. We believe our teamwork was good throughout the entire project as we came to decisions regarding the project together and chose to proceed with whatever option was agreed upon fully by all four members.