# CS 5200 - Database Management Systems

# Final Report

| | |
|---|---|
| **Project Name** | **Musify - Your Music Player** |
| **Group Name** | **TanYParkMLiC** |
| **Group Members** | **Yuqian Tan, Myungkeun Park, Chengjun Li** |

**README**

Files included in our submission:

| **File Name** | **Description** |
|---|---|
| TanYParkMLiC_dump.sql | This is a self-contained schema file that includes tables, data, and stored procedures. |
| TanYParkMLiC_app.py | This Python file contains the access point for the application |
| Final Report__TanYParkMLiC.pdf | This is the final report for our database project. |

Installation Process & Library requirement:

1. Import and create the database in the user's local MySQL host server using the provided dump file. TanYParkMLiC_dump.sql is a self-contained file with the CREATE command. In the local MySQL server, go to "Server > Data Import > Import from Self-Contained File > Start Import." After importing, the database named “**musify**” should appear on the user’s local server. Verify that the database contains tables, data within each table, stored procedures, and functions.

2. The application code is written in Python and uses the PyMySQL library to connect to the MySQL Server. To run this application code, the user must have a compatible Python version (3.7 or above) and install the PyMySQL library package using the pip command before starting the application.

3. The application code also requires the user to install Pandas and Matplotlib library. We use these two libraries to do the data visualization work.

4. After installing the database and all the Python libraries mentioned above, the user can open the application code in any IDE that supports the Python language.

## Technical Specifications

Our project employs SQL alongside MySQL Workbench for efficient storage and management of the application's database. In the backend, we leverage the PyMySQL library within Python for database interactions, and utilize matplotlib for creating data visualization charts. The raw data retrieved from the database is structured into an organized format using Pandas. Furthermore, we have incorporated a command line interface to facilitate user interactions with our application.

## Project Introduction

Musify is a personal music player platform that empowers users to curate their playlists and populate them with their favorite songs. In addition, users can view song details within their playlists and share personal reviews and feedback.

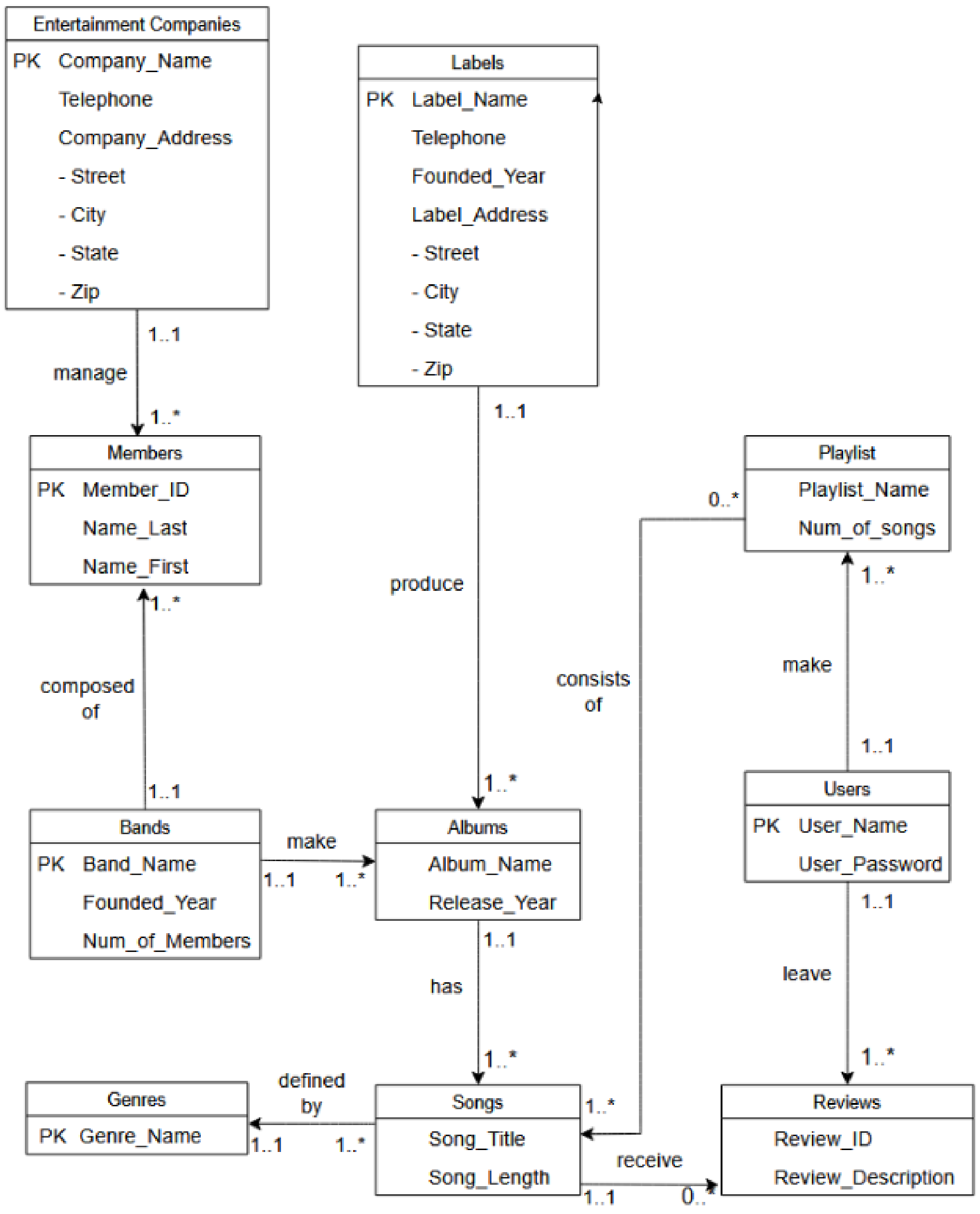
## Description of Database

We are creating a relational database for a music app similar to Spotify and will use SQL to manage and query the data. The information in our database includes entertainment companies, members (artists), bands, albums, album record labels, songs, genres, users, playlists, and user reviews.

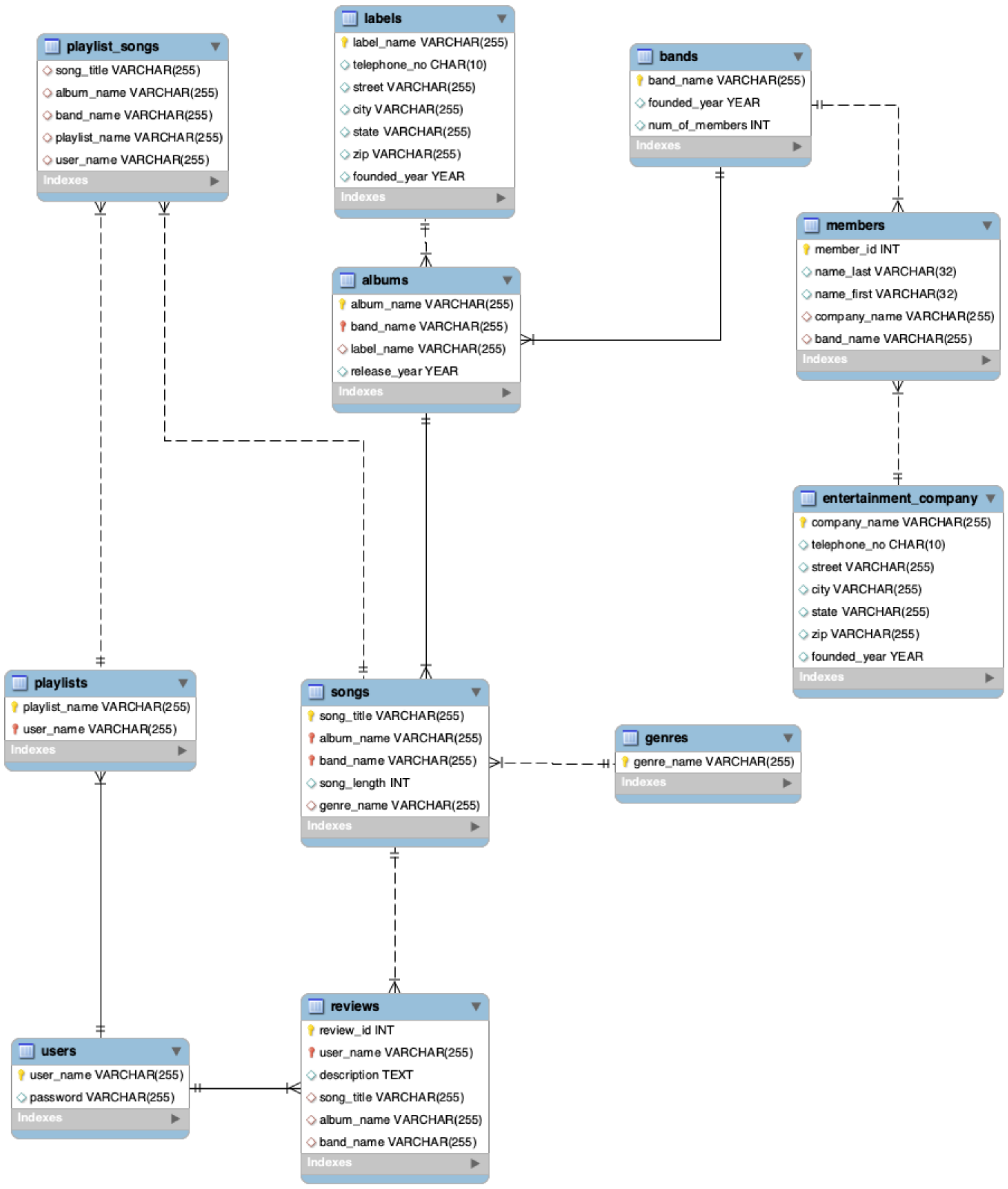
Each entertainment company has a unique name. Each member (artist) under the entertainment company will have a unique name. Each entertainment company will manage one or more artists, but each artist will only work for one entertainment company. Artists usually form bands. Each artist will only be in one band. Each band has a unique name. Each band will have one or more artists. Each band will make one or more albums, and an album can be associated with only one band. Each album is associated with one record label. However, each record label can produce one or more albums. A record label will have a unique name. An album will have one or more songs. Each song has a title and length. Each song will only be recorded in one album. Each song is classified by one genre. Each genre can be associated with 0 or more songs.
Our users will have unique usernames and passwords. Users can create one or more playlists. Users can name their playlists. A playlist can collect many songs. Each song can also be added to 0 or more playlists. Users can leave reviews of a song. A review has a description. Each review

will only belong to one user, but each user can write 0 or more reviews. Each review will only be tagged for a single song, but each song can receive 0 or more user reviews.

## Conceptual Design



## Logical Design



**System User Flow**

Use the application - TanYParkMLiC_app.py

To start the application, Go to the main function and click run.

The user will be prompted to enter the local server username (default is 'root') and password. This application assumes the user uses localhost for the connection. A message, "You have successfully connected to the database," indicates a successful connection. If the connection fails, an "Error connecting to the database" message will appear. Remember, both input fields for username and password are **case-sensitive**.

After the user connects to the database, we provide a menu for the user. The user can choose to log in to the existing music account, create a new account, update the password associated with the current account, or close the connection with the database. In our database, we initially stored eight users and their passwords. For grading purposes, we shared all the user names and their passwords in the table below. Graders can choose to create a new account to test our "C" operation, update the password to test our "U" operation, or log into the existing account below to test more operations.

| **User Name** | **Password** |
|---|---|
| user1 | password1 |
| user2 | password2 |
| user3 | password3 |
| user4 | password4 |
| user5 | password5 |
| user6 | password6 |
| user7 | password7 |
| user8 | password8 |

After the user logs into their music account, we provide a menu for the user to perform the below tasks: manage playlists, explore music, view band distribution graph, view album release year distribution graph, discover interesting facts, and log out.

Manage Playlists - The user can create a new playlist, update the current playlist name, delete a playlist associated with his/her account, insert a song into the playlist, delete a song from the playlist, or exit the current menu. Please note that all input box is **case sensitive**.

Explore Music - This function allows the user to explore the music in our database. The user can find the song by band name, album name, genre type, and song name or exit the current menu. Please note that all input box is **case sensitive**.

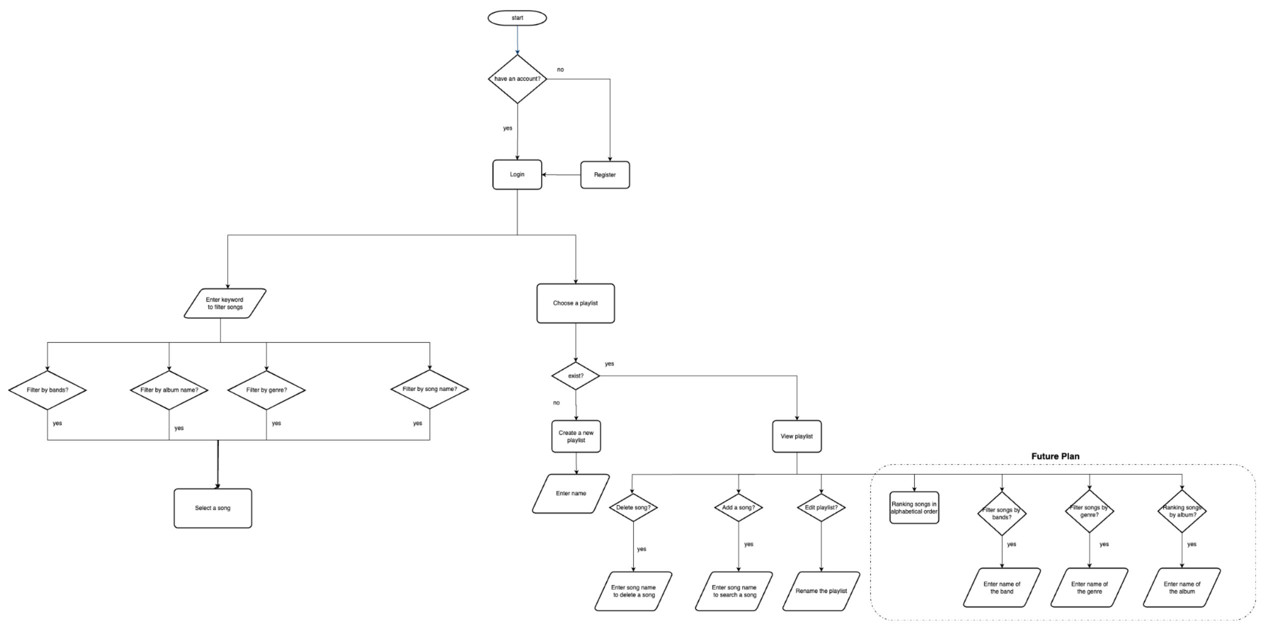
View Band Distribution Graph - Here, we provide the user an opportunity to quickly learn the music in our database through a graph of the bands' founded years.

View Album Release Year Distribution Graph - Here, we provide the user an opportunity to quickly learn the music in our database through a graph of the albums' release year.

Discover interesting facts - Here, we provide the user another opportunity to learn about our database by listing four fun facts: the top five albums with the longest playtime, a list of bands with their latest album, the top five most popular genres by the number of songs, top five most active users by the number of reviews.

Log out of your music account - This option allows the user to log out of the account and back to the last menu where the user can choose to log in to the music account again, create a new account, update the password for their current account, or close the connection with the database.

We provide the flow chart to provide an overview of our application's user flow:



## Lessons Learned

*Technical expertise gained*

Throughout this project, we've significantly enhanced our skills in several technical areas. Firstly, our proficiency with SQL and database management has grown through hands-on experience with MySQL Workbench. We learned to efficiently structure and query a complex relational database. In addition, integrating Python with SQL, particularly using the PyMySQL library, improved our ability to create dynamic, data-driven applications. The use of Pandas and Matplotlib for data handling and visualization also deepened our understanding of these libraries.

*Time management insights & data domain insights*

This project offered valuable insights into the practical challenges of developing a database-driven application. Balancing the workload among team members and adhering to our project timeline was crucial. We learned the importance of regular meetings and updates, which helped in identifying and resolving issues promptly. Time management was key, especially in the phases of schema design and debugging.

Working with a music-themed database gave us insights into the complexities of the entertainment industry's data management needs. The intricacies of managing relationships between artists, albums, and songs highlighted the necessity of a well-thought-out database schema to handle real-world complexities.

*Realized or contemplated alternative designs/approaches to the project*

Our team has identified several areas for improvement. For instance, the current process for adding or removing a song from a playlist is overly complicated. Users are required to specify all information related to the song. To enhance the user experience, it would be more efficient to simplify this process into a 'Yes or No' question. For example, after a user provides the name of a song, we could present a list of songs with that name, considering that the same song title could be used by different bands. This approach would allow the user to select the correct one instead of having to type in the band name and album name themselves. Additionally, our current application enables users to explore songs by band, album, or song name. We could enhance this feature by integrating a 'Yes/No' option, allowing users to add songs to their playlists directly from this section.

Furthermore, we believe that relying on a local database may impact performance and pose challenges in supporting multiple users online simultaneously. Exploring cloud-based database solutions could provide the scalability needed for future expansions.

*Document any code not working in this section*

Our team has conducted rapid tests on our application code, and we are confident that our code can connect with the local database server to perform all the functions listed, especially the core 'CRUD' (Create, Read, Update, Delete) operations for users to manage their playlists and personal accounts.

**Future Work**

First, as mentioned in the lessons learned section, our current application is user-input-heavy. We believe the priority of the next step is to reduce the number of the input box to improve the user experience.

Moreover, in the current application, our group focuses on developing basic functions and meeting the 'CRUD' requirements. In our market analysis, we found that many music apps, like Spotify, allow users to do more with their playlists. As outlined in our user flow chart, functions such as ranking songs in a playlist by band, genre, album, release date, etc., could offer a more personalized user experience. From the users' perspective,  a user-friendly interface should be added for a better user experience. We believe that incorporating such functions in our future work could provide a more comprehensive user experience.

With the data already collected in our database, we can leverage advanced Business Intelligence tools like Power BI or Tableau for deeper insights. These tools will enable us to analyze trends and patterns more effectively. Specifically, they can enhance our understanding of music preferences and sentiments by analyzing customer reviews. This approach will not only provide a quantitative analysis but also a qualitative understanding of what drives people's choices in music.

Finally, we are considering integrating a play button feature. By utilizing available online APIs, we can enable direct playback of songs from the playlists. This functionality will transform our service from a mere playlist management tool into a comprehensive music player. This integration will provide a seamless and interactive experience for users, allowing them to enjoy their favorite music and discover new tracks within the same platform.