Concert Ticket Reservation System

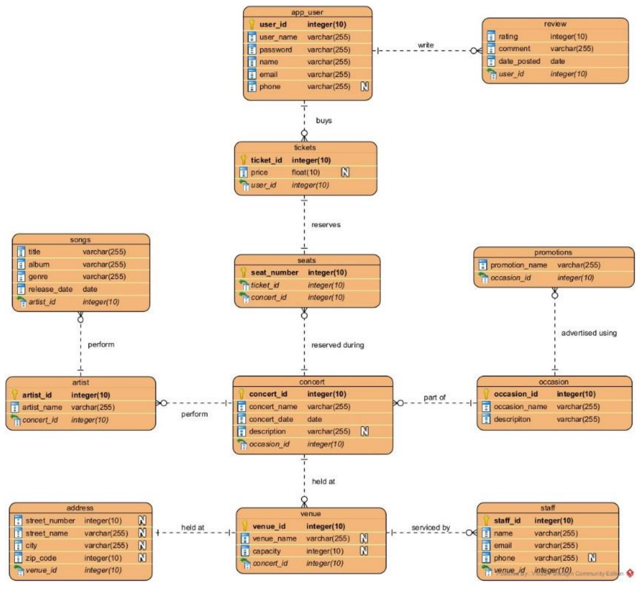
Hasan Khwaja, Sreeja Vepa, Pranav Kanth Anbarasan

Introduction

The Concert Ticket Reservation System was designed to help users book tickets to local concerts. The key objectives of this system are to manage ticket sales, concert information, user reviews, promotion schedules, and staff management. Inspired by connecting audiences with local artists, the project allows users to centralize their concert experience by enabling finding concerts, buying tickets, and sharing experiences through reviews using a graphical user interface (GUI), tkinter.

Additional to a user view, the system also enables staff to centralize their management practices using an admin view. The overarching goal of this system was to improve the general concert experience for both users and staff.

Database Design



The diagram above depicts the final ERD design of the music ticket reservation system. There were various design decisions that took place due to our self-generating of the data using Faker. One decision was that a user should be able to buy multiple tickets, and if the ticket is bought it will reserve a single seat. This ensures a user's ability to buy multiple tickets at once while maintaining proper individual seat allocations. As the concerts being currently considered are smaller in size, seats are assigned randomly. Another unique decision was to allow artists to sing any song of their choice without restriction from the songs table in the database. This also simplifies the process of inserting data into the database as not every artist will be given a personalized set list, but rather a pool of songs all of them are able to perform. The database also tracks if a concert being performed is part of a tour and/or occasion. This may be seen where each concert being a part of an occasion is documented. Along with the occasion, the method of how those occasions are advertised and promoted are also documented in the database.

This database is normalized to the 3NF form. One example of how we actively managed database normalization can be seen through the address and staff tables. Both tables were separated from the larger original venues table to maintain 3NF as neither were dependent on the primary key, venue_id, of the venues table. Another example is exemplified in the app_user table which has both user_id and user_name columns. To maintain user_id as the sole unique primary key, user_name was made a not unique field. Therefore, it is possible for two users to have the same user_name but they remain identifiable by their user_id. The same applies for all the other columns in the app_user table.

Data Collection

As mentioned earlier, this databases's data was primarily generated by python's Faker library. The Faker library generates fake data effectively for a great variety of fields for when realistic but not real data is needed. Additonal libraries used were pymysql, datetime, cryptography, and random. The general procedure of data generation was similar for all tables. The first step was to use pymysql to connect to the MySQL server.

After the connection and cursor have been established, and creating a loop iterating over the primary key, fake data was generated for each primary key in the iteration. While still in the loop, each iteration of fake data was inserted into MySQL using the cursor. Outside the loop, there is a print statement for each newly inserted row of the table to verify the data. Once the proper insertion of data has been verified the changes are committed to the MySQL server and the cursor and connection are closed.

There are several key differences that can be observed across the different data insertion functions. When looking at attributes with values that may be repeated, such as genres in the songs table, a list of values was made by hand. This was done due to how some attributes needed fixed values to be repeated at random across multiple rows in each table.

In addition to this, the foreign keys in each table also needed to manually be inserted into each table. This was done due to a technical limitation of the faker library, where, when trying to create fake foreign keys, the foreign key values would be skipped during some iterations, leaving null values in their place. When trying to solve this, one solution would have been to run the loop multiple times to give the program the chance to fill in the missing values. This solution, however, did not prove to be reliable. This conclusion was made with the understanding that there was no guarantee the null values would be filled in after running the program any number of times.
Since the data was generated at our discretion, there was no need for specific cleaning or preprocessing as the data was manually checked and created at each iteration.

Application Description

The graphical user interface (GUI) within the application has been made using the Tkinter library, a robust Python framework dedicated to facilitating GUI development. Tkinter, known for its flexibility, offers a high degree of extensibility, providing users with the capability to both custom widgets and modify existing ones to precisely align with the specific design requisites of the application. This flexibility empowers us to make visually appealing interfaces.

Furthermore, Tkinter seamlessly integrates with the MySQL database with the help of the mysql-connector package. This establishes a highly efficient channel for communication between the front-end interface constructed with Tkinter and the MySQL database. The result is a seamless and streamlined process for data retrieval, storage, and manipulation.

Libraries Used:

- **Tkinter** (Used for the development of the Graphical User Interface (GUI)).
- **MySQLConnector** (To establish a channel for communication between the front end (Tkinter window) and the backend (MySQL database)).
- **datetime** module has been employed to automatically generate default dates during the ticket booking process initiated by the user.
- **random** module has been utilized to generate random values for ticket prices between a certain range, when a user books a ticket.

Pages:

Login/Signup Page:

- The first page features options for user login and signup.
- Upon selecting the login option users are prompted to enter their user ID and password.
- New users are required to complete the signup process, and upon successful data storage in the database, they gain access to the login page. Upon entering the correct credentials on the login page, users are redirected to the main page.

Main page:

- Upon successful authentication, users are redirected to the main page, where they gain access to a suite of features. Notably, one of these features involves the process of ticket purchasing. Upon selecting the desired concert and submitting the transaction, the system automatically initiates the closure of the window in the event of a successful transaction. Furthermore, each successful purchase triggers the immediate and accurate updating of information within both the tickets and seats tables, in the backend.

- The "View Concert Information" window is designed with a dropdown menu containing a comprehensive list of concert names. Upon the user's selection of a specific concert name and subsequent submission, the corresponding venue name and address are dynamically displayed. This functionality is accomplished through the integration of three backend tables, specifically the "concert," "venue," and "address" tables.

- The "Post your Reviews" page allows users to share their feedback. To contribute a review, users are prompted to input a numerical rating on a scale of 1 through 10 and provide a detailed commentary within the designated textbox. Upon submitting their feedback, the system records the review, rating, and the associated user ID in the review table.

- In the event of a user opting to delete a review, they can click the "Delete Your Reviews" button, enter the relevant review details, and, if the review exists, it will be removed from the database; otherwise, an error message stating "No such review exists" will be displayed.

In summary, this application provides users with the capability to execute queries on the tables and perform operations such as inserting, updating, deleting, and joining data within the tables.

Conclusion/Future Directions

This project allowed for a deeper understanding of database design techniques through effective implementation. The various tasks such as data normalization in 3NF, implementation of the Faker Library in python, using mysql-connector to connect Python to the database, and working with triggers, stored procedures, views, and more all helped solidify the usefulness of MySQL.

In terms of future directions, there are various ways to expand and improve this project. One simpler way could be to connect the reviews and concert table using a foreign key. In this case, user reviews could be connected to a specific concert. As the project stands now, users are unable to see the specific concerts each review references and instead see a general review of all concerts available. This functionality would improve the user experience and allow them to make better decisions when purchasing tickets. This would also be needed due to how the opinion on one concert may be different from others.

Additionally, this project would have also benefited from a way to access different views of the database. There is currently no way to access the user or staff views. To implement this, the staff would have to be given their own specific login credentials that was give them access to the staff view. Without these specific credentials, logging in as normal would direct them to see the user's view.

The best advice to give future students, in relation to the project, would be to complete data creation/pre-processing as early as possible. The amount of time to complete this step was initially underestimated, and while implementing Faker was a great learning experience, the time allocated to this task was unexpected.