Library Database in SQL using SQLite

Max Biundo, Andrew Vanderwerf, Mario Simental, Samuel Buehler, Karsten Wolter, and Humza Qureshi

University of Kansas

Abstract

This project builds a Library database using SQLite, chosen for its ease of use with Python. The database is managed through several Python scripts that handle creation, data population, and queries. Random data is generated using predefined lists (e.g., names, genres) and inserted into the database via an automated script. Functionality is tested by running simple queries for individual tables, followed by more complex queries to check data relationships. The project is documented and organized to ensure smooth development and testing, resulting in a functional and scalable library system.

Platform Choice

This project uses SQLite to contain the Library database. Initially, the MariaDB database management system was considered, but SQLite was chosen instead. SQLite was chosen for its accessibility to the project developers and its compatibility with the Python language. To initially access and test MariaDB using the University of Kansas' School of Engineering cycle servers, which required the developers to connect from their homes via SSH. MariaDB also proved difficult to have multiple developers collaborate on simultaneously. SQLite was chosen

Database

This database was created with Python using the database's DDL scripts as reference for the tables. The database utilizes several python files such as constants.py, createDatabase.py, populateDatabase.py, queryCommands.py, and main.py, these are required to construct the database tables, populate the tables, and run custom queries.

Data Population

The database was populated using an automated script written in the Python language (/sqliteDB/populateDatabase.py) that generated pseudo-random data for each attribute in each table. These were generated using a variety lists of constants (/sqliteDB/constants.py) such as first names, last names, genres, equipment products, adjectives and base words to generate random media names. The population process was activated by including additional commands in our terminal interface.

Functionality Testing

Functionality was tested using simple sample queries. These queries were specific to each table and initially involved no joins in order to test each specific table to make sure it was populated properly. After it was determined that every individual table

worked as intended, more complex queries were implemented to test how the data relates to other data in the schema.

Testing and Validation

To test and validate the project, simple queries were used to validate changes being reflected in the .db file. Custom commands were also tested that come within the program using the same strategy if applicable, or their results were validated to be true when cross-checking with the data in our database (such as checking if get_overdue_members actually returns only members with overdue items). Basic functions, such as getting a full table, were also validated to be a reflection of the database.

Documentation Process

The group has made sure to document whenever a meeting takes place. In these meeting notes, there is a layout of what was discussed in the meeting along with a continuation of the meeting in the footnotes. Every group member that attended can be found along with the meeting objective and the date and time that the meeting took place at.

Code and Repository Organization

The repository is organized to have all the initial planning immediately available to the viewer. The code to the database exists within a folder from the homepage with all the code inside of that folder. There is also a separate folder from the homepage containing all the meeting notes from each time the team met to talk about and plan the project.

Conclusion

This project successfully built a Library database using SQLite, chosen for its simplicity and compatibility with Python. Python scripts were used to create, populate, and query the database. Testing occurred at each step to ensure everything worked as expected. Random data was generated to fill the database, and queries both basic and complex were used to confirm functionality. By using GitHub for version control and collaboration, group members kept the project organized and easy to manage. In the end, a reliable, scalable system with clear documentation and a smooth development process was created.