Database writeup v.3

Alexander Archer

Files

db.py, schema.sql, *.db

Use Case

Store and manage user and song metadata locally and later, remotely

Use Case Explained

Handles the creation and maintenance of a local database. It will contain all of the associated song metadata; that is, any data associated with a song that is not the actual music file

Introduction

- Databases contain tables
- Tables are named, and are two dimensional: they have columns ("x-axis") and records ("y-axis")
- The columns define the type of data, records are rows of data
- Schemas are the "blueprints" for tables, they contain the definitions for the table column names and column data types
- Every database implements CRUD: create, read, update, and delete operations; these are what one uses to manipulate data inside databases
- SQL is the main language for relational databases
- SQLite is a particular implementation of SQL that is simple and well-documented
- Accessing correct records requires a series of SQL statements which are relational comparisons (hence relational databases)
- SQL statements are called queries; the important SQL keywords to know are SELECT, INSERT, UPDATE, WHERE, IN, CREATE, DELETE, I suggest reading the documentation for these

sqlite3 Installation

- Install "sqlite3" program (and command line interface) via package manager or via binary from https://www.sqlite.org/download.html
- test for "sqlite3" command:
    - sqlite3 test.db
    - > create table test (id);
    - > .quit
    - Check to see if test.db file exists in filesystem

sqlite3 Usage

- The "sqlite3" command and command line will be useful for testing your implementations quickly
- The schema of the database should be a .sql file, a database is generated from running a .sql file that contains SQL statements which form the database schema
- To create a database from a given .sql file use:
    - sqlite3 [db filename].db < [schema filename].sql
- Multiple tables can be created from a single .sql file

- Another .sql file can be used to fill the table with data or test operations:
    - sqlite3 [db filename].db < [sql data insertion filename].sql
- When testing, it will be helpful to add multiple "DROP TABLE IF EXISTS [table name]" commands to the top of your files so you do not have to continuously remove the .db file
- sqlite3 -echo argument added to sqlite3 will print out what sqlite3 is doing
- sqlite3 -header -column -echo arguments added to sqlite3 will pretty print SELECT results

Notes

- SQLite3 is bundled with Python and would be an ideal choice to use in this case
- SQLite documentation can be found here: https://www.sqlite.org/lang.html

Intended File Behavior

db.py: The python file that handles all database operations

- Load the schema file and generate the database and tables if they do not exist
- Generate the correct SQL queries to respond to the various events that happen over the lifetime of the program (e.g. load a playlist)

schema.sql: The SQL file that contains the definition of the database

- Below, you will find the relational table called "PLAYLIST_SONGS" that maps song records to playlist records

***.db:** The database file itself which contains tables and records (data)

- This database will be accessed or updated every time the program or user makes an update to the state (such as updating a mood or playing a song)

Example Table Schemas

SONG table

(Primary key) Index Name Artist Album Genre Year Track # Play Count Rating Mood

PLAYLIST table

(Primary key) Index Name

PLAYLIST_SONGS table

Playlist_index Song_index

Database Event Handling

On program start:

- If db doesn't exist
    - load schema and create database and appropriate tables (paths defined in config)
- else
    - respond to initialization queries

On program close:

- Close all database connections

On song add:

- Create correct record in “songs” table via SELECT and INSERT

On song load:

- Read correct record in “songs” table via SELECT

On song metadata change:

- Update the correct record in “songs” table via SELECT and UPDATE

On song remove:

- Delete correct record in “songs” table via SELECT and DELETE

On playlist add:

- Create correct record in “playlist” table via SELECT and INSERT

On playlist load:

- Read correct record in “playlist” table and do a join with the “playlist_songs” and “song” tables to select the correct songs via SELECT

On playlist metadata change:

- Update the correct records in “playlist” table via a join with “playlist_songs” and “song” tables to update a playlist via SELECT and UPDATE

On playlist remove:

- Delete correct records in the “playlist” and “playlist_songs” tables via SELECT and DELETE

Future

Handle a centralized database that merges different user’s ratings