Python SQLite Music Library Relational Database

Contents

SQL Tutorial

Go through these tutorials.

- https://www.w3schools.com/sql/sql_intro.asp
- https://www.w3schools.com/sql/sql_syntax.asp
- https://www.w3schools.com/sql/sql_create_db.asp
- https://www.w3schools.com/sql/sql_create_table.asp
- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_insert.asp
- https://www.w3schools.com/sql/sql_update.asp
- https://www.w3schools.com/sql/sql_delete.asp
- https://www.w3schools.com/sql/sql_select.asp
- https://www.w3schools.com/sql/sql_in.asp
- https://www.w3schools.com/sql/sql_wildcards.asp
- https://www.w3schools.com/sql/sql_join_inner.asp

Entity Relationship Diagram Tutorials

- https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm
- https://www.tutorialspoint.com/dbms/er_diagram_representation.htm
- https://www.lucidchart.com/pages/videos/entity-relationship-diagram-erd-tutorial-part-1

SQLite and Python

SQLite is a self-contained, file-based SQL database. SQLite comes bundled with Python and can be used in any of your Python applications without having to install any additional software.

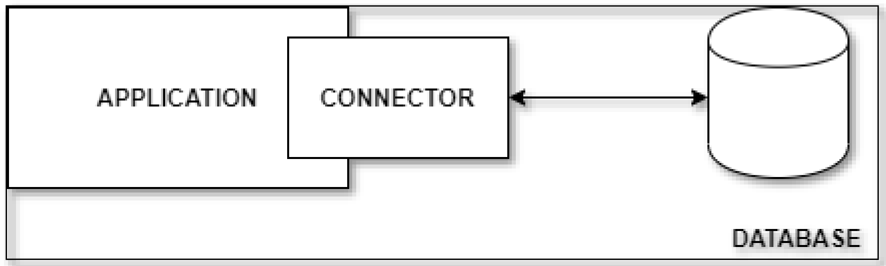
Databases can manage large amounts of information easily over the web and high-volume data input and output. SQL is a query language and is very popular in databases. Many websites use SQL. SQLite is a "light" version that works over syntax very similar to SQL.

SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine on the world wide web. Python has a library to access SQLite databases, called sqlite3, intended for working with SQLite.

SQLite has the following features.

1. Serverless
2. Self-Contained
3. Zero-Configuration
4. Transactional (ACID)
5. Single Database file

SQLite does not require a server to run. The SQLite database is joined with the application that accesses the database. SQLite database read and write directly from the database files stored on disk and applications interact with that SQLite database.



Self-Contained

SQLite does not need any external dependencies like an operating system or external library. This feature of SQLite helps especially in embedded devices like iPhones, Android phones, game consoles, handheld media players, etc.

Zero-Configuration

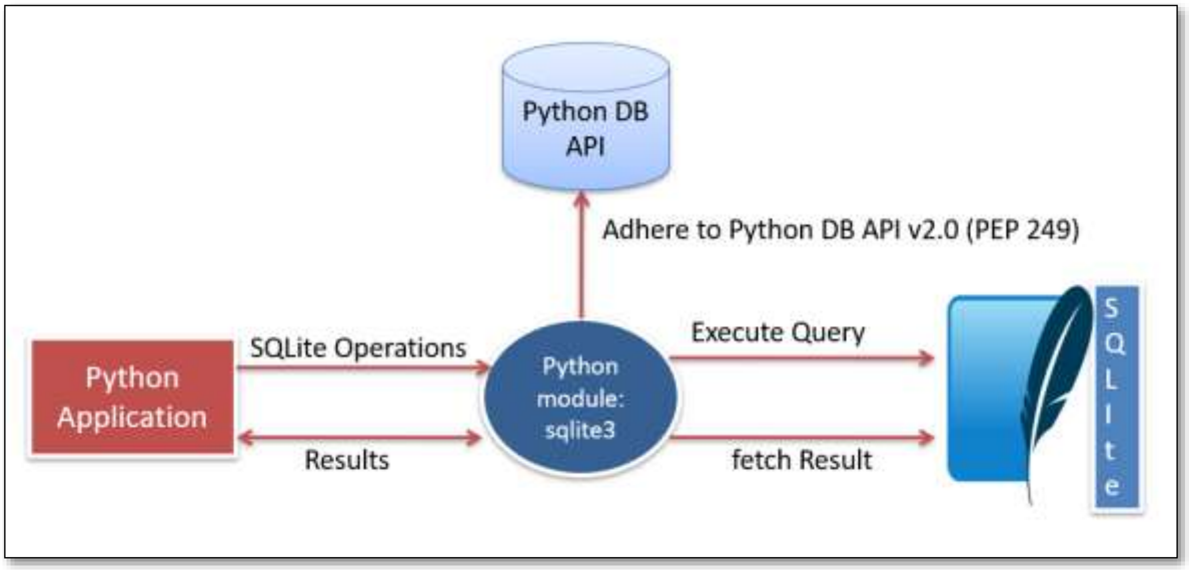
Zero-configuration means no setup or administration needed. Because of the serverless architecture, you don’t need to “install” SQLite before using it. There is no server process that needs to be configured, started, and stopped.

Transactional (ACID)

Transactional means they are atomic, consistent, isolated, and durable (ACID). All transactions in SQLite are fully ACID-compliant. In other words, all changes within a transaction take place completely or not at all even when an unexpected situation like application crash, power failure, or operating system crash occurs.

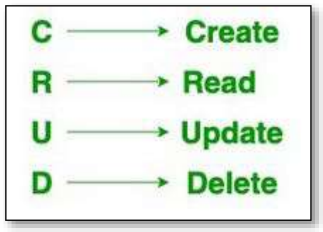
Single Database File

SQLite is a single database that means it allows a single database connection to access multiple database files simultaneously. These features bring many nice features like joining tables in different databases or copying data between databases in a single command. SQLite also uses dynamic types for tables. It means you can store any value in any column, regardless of the data type.



CRUD

A database provides four major operations, usually referred to as CRUD.



The corresponding SQL commands. Notice that SQL commands are typically in UPPER CASE. This is a convention; SQL does not require UPPER CASE. It makes the SQL commands easier to pick out of an SQL statement

CREATE: Create new tables and records

READ (SELECT): Select all or selected fields and records

UPDATE: Modify existing tables and records

DELETE: Delete existing tables and records

SQLite Relational Database

SQLite is a relational database. We can create tables related by primary keys. We will design our databases using an ERD (Entity Relationship Diagram). www.lucidchart.com is free web-based diagram site used in these SQLite tutorials.

In this tutorial, we will create a relational database with two related tables as an example for you to create your own relational database.

SQLite DataTypes

SQLite has very simple data requirements.

- **NULL:** The value is a NULL value.
- **INTEGER:** Store a whole number.
- **REAL:** Floating-point value, for example, 3.14, the value of PI.
- **TEXT:** A text string. TEXT value stored using UTF-8, UTF-16BE or UTF-16LE encoding.
- **BLOB:** The value is a blob of data, i.e., binary data. It is used to store images and files.

The following Python types convert to SQLite types.

Python Types	SQLite Types
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Normalize Redundant Data

To eliminate redundancy in a table and divide it into two related tables, the process is called **normalization**. This involves restructuring the data to minimize duplication and dependencies. We'll go through the steps with an example table.

Scenario:

We have a single table called Employees with the following data:

Employee_ID	Employee_Name	Department_ID	Department_Name
1	John Doe	101	Sales
2	Jane Smith	102	Marketing
3	John Doe	101	Sales
4	Alex Brown	103	IT

Problem:

In this table, the department information is redundant. The Department_Name is repeated for every employee in the same department, which causes redundancy.

1. Identify Redundant Data:

In the Employees table, both **Department_ID** and **Department_Name** are repeated multiple times for employees in the same department. The goal is to break this into two tables: one for employees and one for departments.

2. Create Two Related Tables:

- **Employees Table:** Contains employee-specific data.

- **Departments Table:** Contains department-specific data.

Step-by-Step Example:

Original Table: Employees

Employee_ID	Employee_Name	Department_ID	Department_Name
1	John Doe	101	Sales
2	Jane Smith	102	Marketing
3	John Doe	101	Sales
4	Alex Brown	103	IT

Step 1: Create the Departments Table

Extract department-related data and create a separate Departments table. Each department should only appear once.

dpt_id PK	dpt_name
101	Sales
102	Marketing
103	IT

Step 2: Modify the Employees Table

Remove the redundant dpt_name from the original Employees table and replace it with the dpt_id as a foreign key.

emp_id PK	emp_fname	emp_lname	dpt_id FK
1	John	Doe	101
2	Jane	Smith	102
3	John	Doe	101

4	Alex	Brown	103

3. Establish Relationship:

The Employees table now references the Departments table using the dpt_id column, which acts as the foreign key linking these two tables.

Business Rules

Business rules describe the table relationships in words.

- A department can have many employees.
- An employee can be in one department.

What is an ERD?

An Entity Relationship Diagram, also known as an ERD, is a diagram that displays the relationship of entity sets stored in a database. ER diagrams help to explain the logical structure of databases.

ER diagrams are created based on three basic concepts: entities (tables), attributes (fields), and relationships.

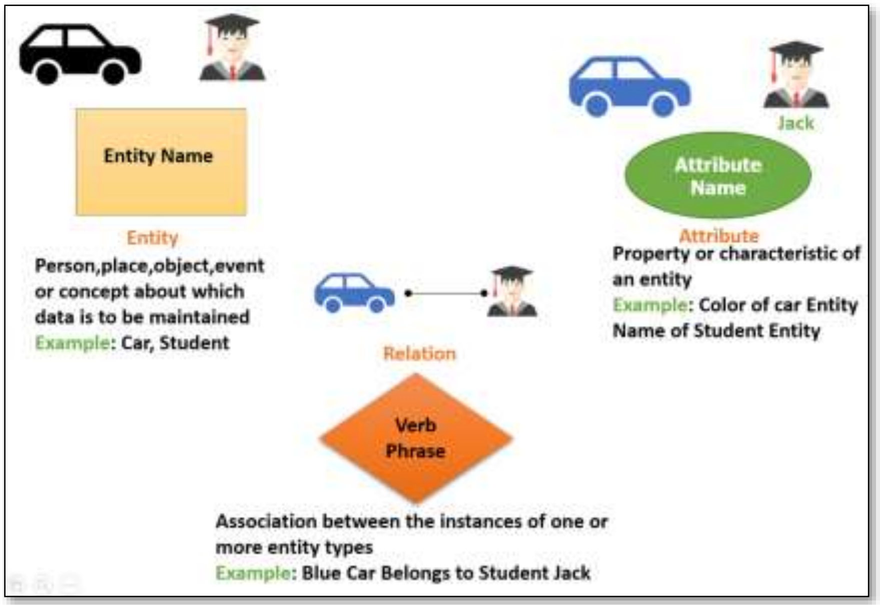
Components of the ER Diagram

This model is based on three basic concepts:

- Entities (Objects)
- Attributes (Properties)
- Relationships

ER Diagram Example

For example, in a University database, we might have entities for Students, Courses, and Professors. The Student entity can have attributes like StudentID, Name, and DeptID. They might have relationships with Courses and Professors.


Entity Name
Entity
Person,place,object,event or concept about which data is to be maintained
Example: Car, Student
Relation
Verb Phrase
Association between the instances of one or more entity types
Example: Blue Car Belongs to Student Jack
Jack
Attribute Name
Attribute
Property or characteristic of an entity
Example: Color of car Entity
Name of Student Entity

Music Library Database ERD

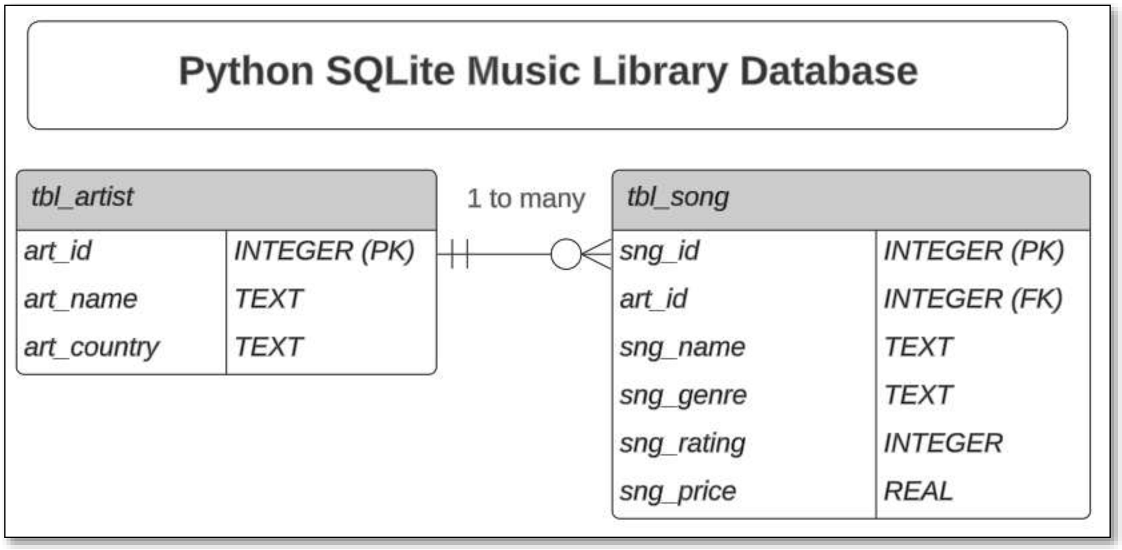
An Entity Relationship Diagram, also known as ERD, is a diagram that displays the relationship of entities stored in a database. ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities (tables), attributes (fields), and relationships.

An ERD of our music library database is based on the following business rules:

- An artist can produce many songs.
- A song can only have one artist.

This is a crow's foot diagram illustrating a one to many relationship between the entities.

I used www.lucid.app to draw the following ERD diagram.



Reference: https://vertabelo.com/blog/crow-s-foot-notation/

PK (Primary Key): A primary key is a column or a set of columns in a table whose values uniquely identify a row in the table. A primary key typically has no meaning other than to uniquely identify each record.

FK (Foreign Key): A foreign key is a column or a set of columns in a table whose values correspond to the values of the primary key in another table.

These two tables are related through a primary key in the **tbl_artist** table, **art_id**. A foreign key **art_id** is in the **tbl_song** table. This key connects the two tables. This is an example of a one-to-many relationship.

Part 1: Business Rules and ERD

Use your own database design.

Assignment 1: Normalize the Data

1. Put some sample data in a table like an Excel Spreadsheet.

Example:

Artist	**Country**	**Song Name**	**Genre**	**Rating**	**Price**
ZZ Top	US	Tush	Rock	1	2.99
Journey	US	Don't Stop Believing	Rock	2	2.99

2. Determine the Entities (tables).

artist: Artist specific data

song: Song specific data

Your Database Assignment:

1. Create a Word document or use an earlier planning document.
2. Add your sample data in a table and determine the entities.

Assignment 2: Business Rules

Business rules establish how the entities interact and their relationship with each other. For example:

- An artist can produce many songs.
- A song can only have one artist.

Add the business rules for your database to your planning document.

Assignment 3: Draw ERD

1. Draw a Crow's Foot ERD for a relational version of your project.
2. Place a screenshot of your ERD in your planning document.

Assignment Submission

1. Create business rules and an ERD for your database in your planning document.
2. Submit in Blackboard.

Part 2: Create Data Dictionary

Let's create a data dictionary for your database design

This is the structure of our sample database. This is called a data dictionary.

Table Name	Field Name	Field Data Type	Description
tbl_artist	art_id	INTEGER – Primary Key	Artist unique identifier
tbl_artist	art_name	TEXT	Artist name
tbl_artist	art_country	TEXT	Artist country
tbl_song	sng_id	INTEGER – Primary Key	Song unique identifier

tbl_song	art_id	INTEGER – Foreign Key	Foreign key referencing the artist associated with the song
tbl_song	sng_name	TEXT	Song title
tbl_song	sng_genre	TEXT	Song musical genre
tbl_song	sng_rating	INTEGER	Song rating
tbl_song	sng_price	REAL	Song price

Assignment Submission

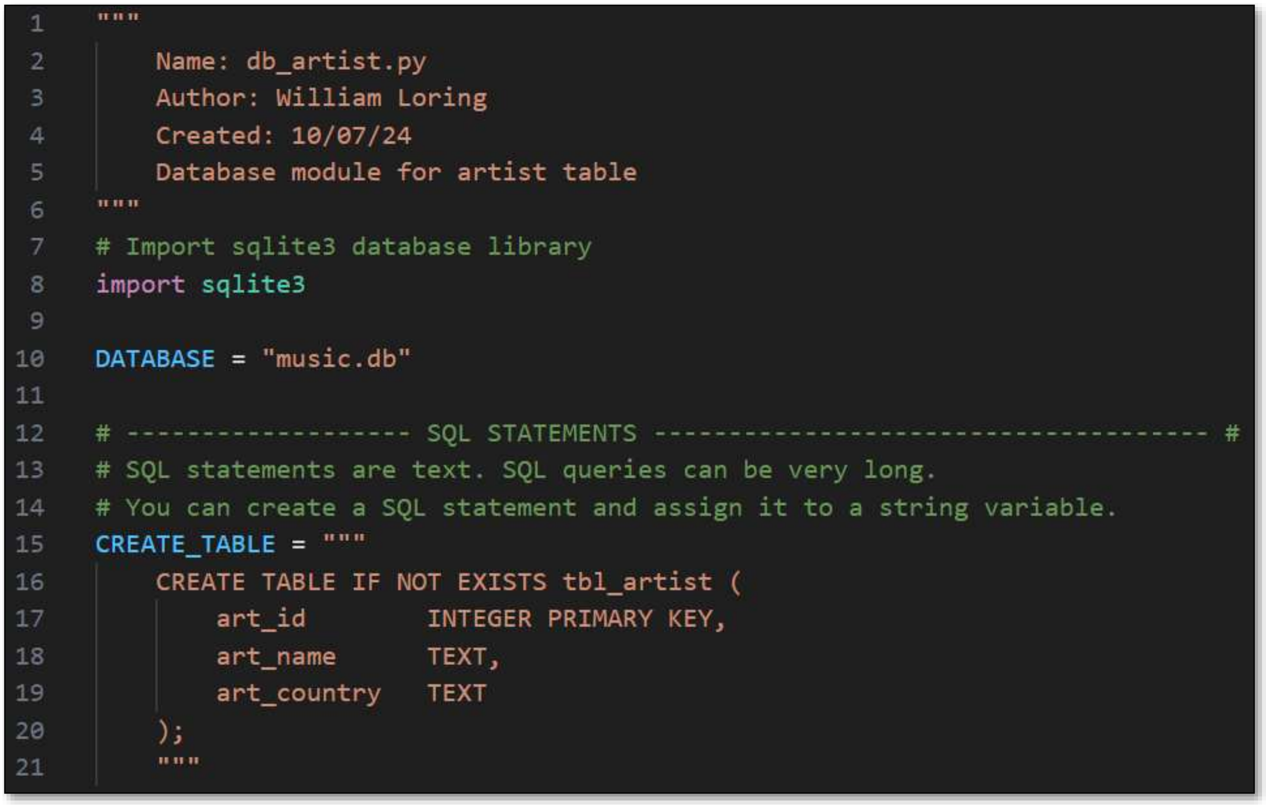
1. Create a data dictionary for your database in your planning document.
2. Submit in Blackboard.

Part 3: Create the Database in SQLite

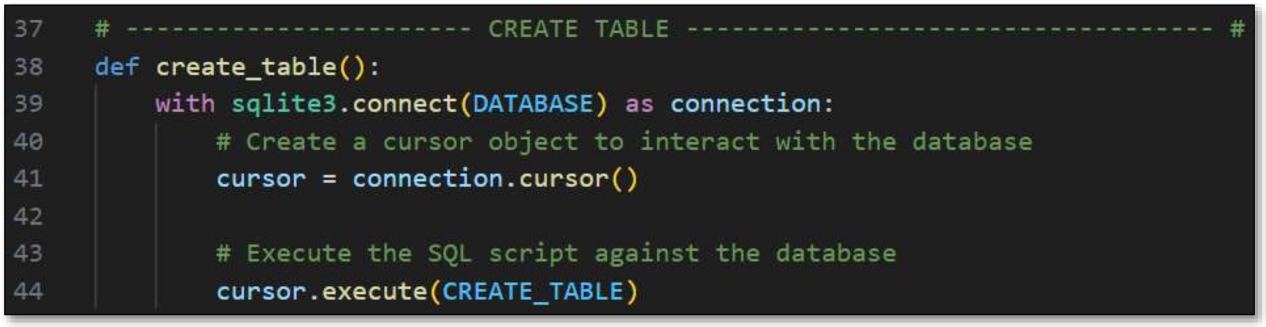
Use your own database design.

We want to create our artist table first. We need to create the table with the primary key before it can be used as a foreign key in another table.

db_artist.py

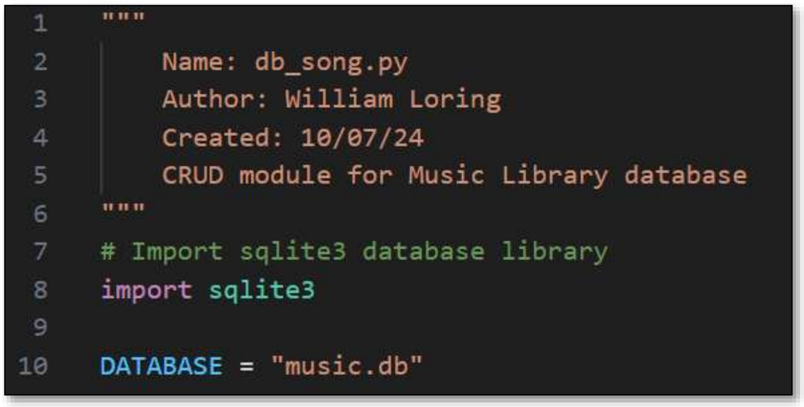


```python
"""
    Name: db_artist.py
    Author: William Loring
    Created: 10/07/24
    Database module for artist table
"""
# Import sqlite3 database library
import sqlite3

DATABASE = "music.db"

# ------------------- SQL STATEMENTS ------------------------------------- #
# SQL statements are text. SQL queries can be very long.
# You can create a SQL statement and assign it to a string variable.
CREATE_TABLE = """
    CREATE TABLE IF NOT EXISTS tbl_artist (
        art_id        INTEGER PRIMARY KEY,
        art_name      TEXT,
        art_country   TEXT
    );
    """
```



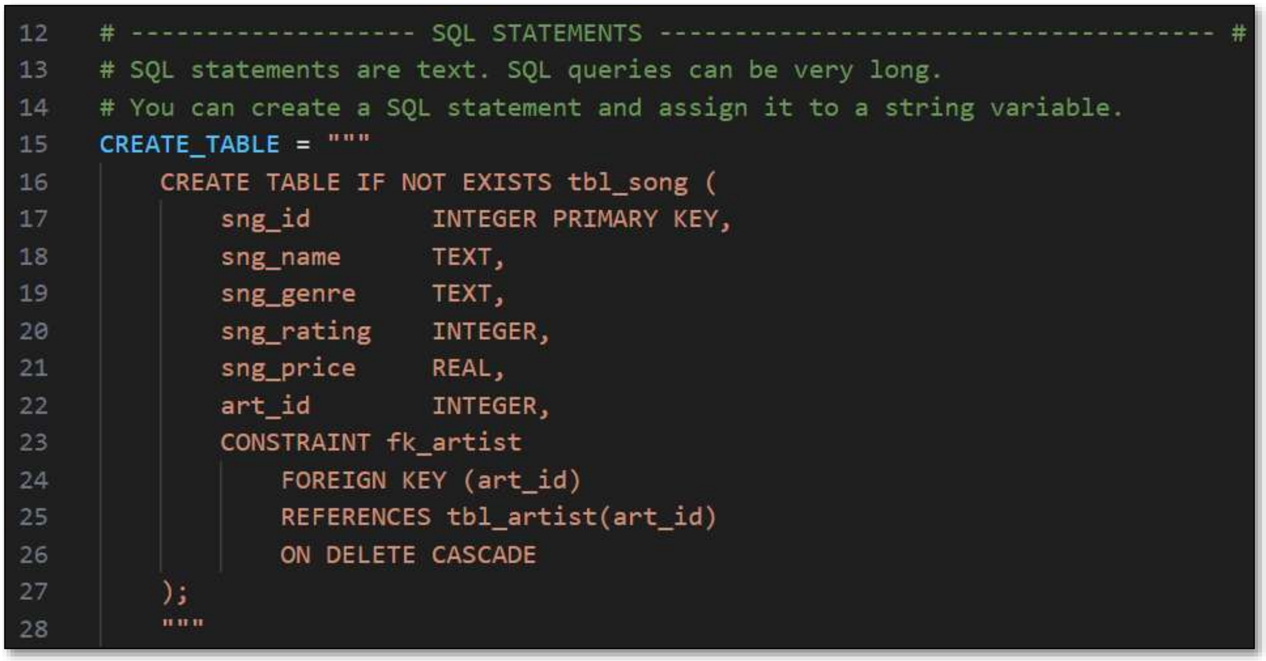
```python
# ----------------------- CREATE TABLE ----------------------------------- #
def create_table():
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Execute the SQL script against the database
        cursor.execute(CREATE_TABLE)
```

db_song.py



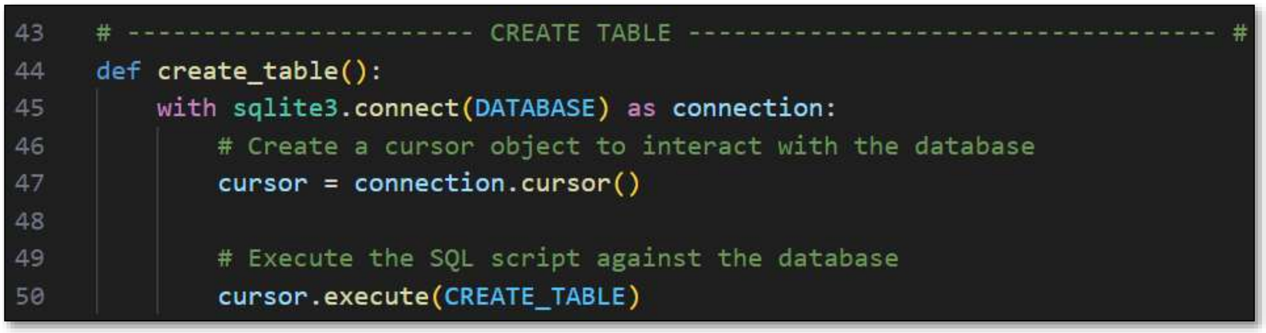
```python
"""
    Name: db_song.py
    Author: William Loring
    Created: 10/07/24
    CRUD module for Music Library database
"""
# Import sqlite3 database library
import sqlite3

DATABASE = "music.db"
```

All SQL statements will be stored at the top of the program for easy access.



```python
# ------------------- SQL STATEMENTS ------------------------------------ #
# SQL statements are text. SQL queries can be very long.
# You can create a SQL statement and assign it to a string variable.
CREATE_TABLE = """
    CREATE TABLE IF NOT EXISTS tbl_song (
        sng_id        INTEGER PRIMARY KEY,
        sng_name      TEXT,
        sng_genre     TEXT,
        sng_rating    INTEGER,
        sng_price     REAL,
        art_id        INTEGER,
        CONSTRAINT fk_artist
            FOREIGN KEY (art_id)
            REFERENCES tbl_artist(art_id)
            ON DELETE CASCADE
    );
    """
```

The functions go below the SQL statements. This function uses the **CREATE_TABLE SQL** script to create **tbl_song**.



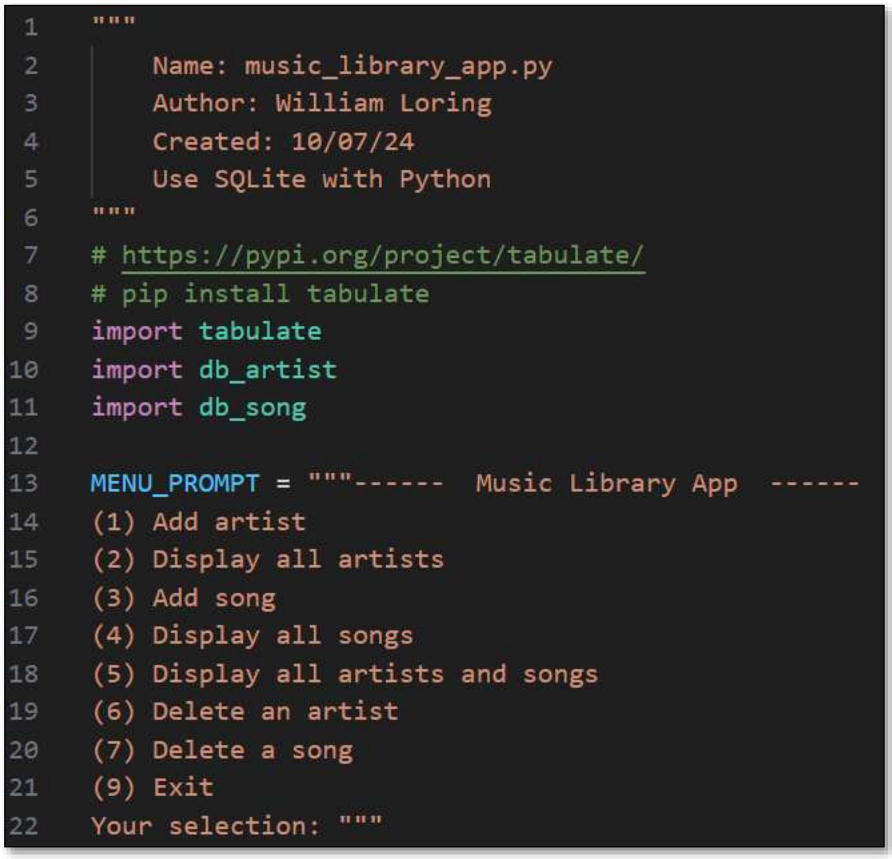
```python
# ---------------------- CREATE TABLE --------------------------------- #
def create_table():
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Execute the SQL script against the database
        cursor.execute(CREATE_TABLE)
```

The application is next.

music_library_app.py

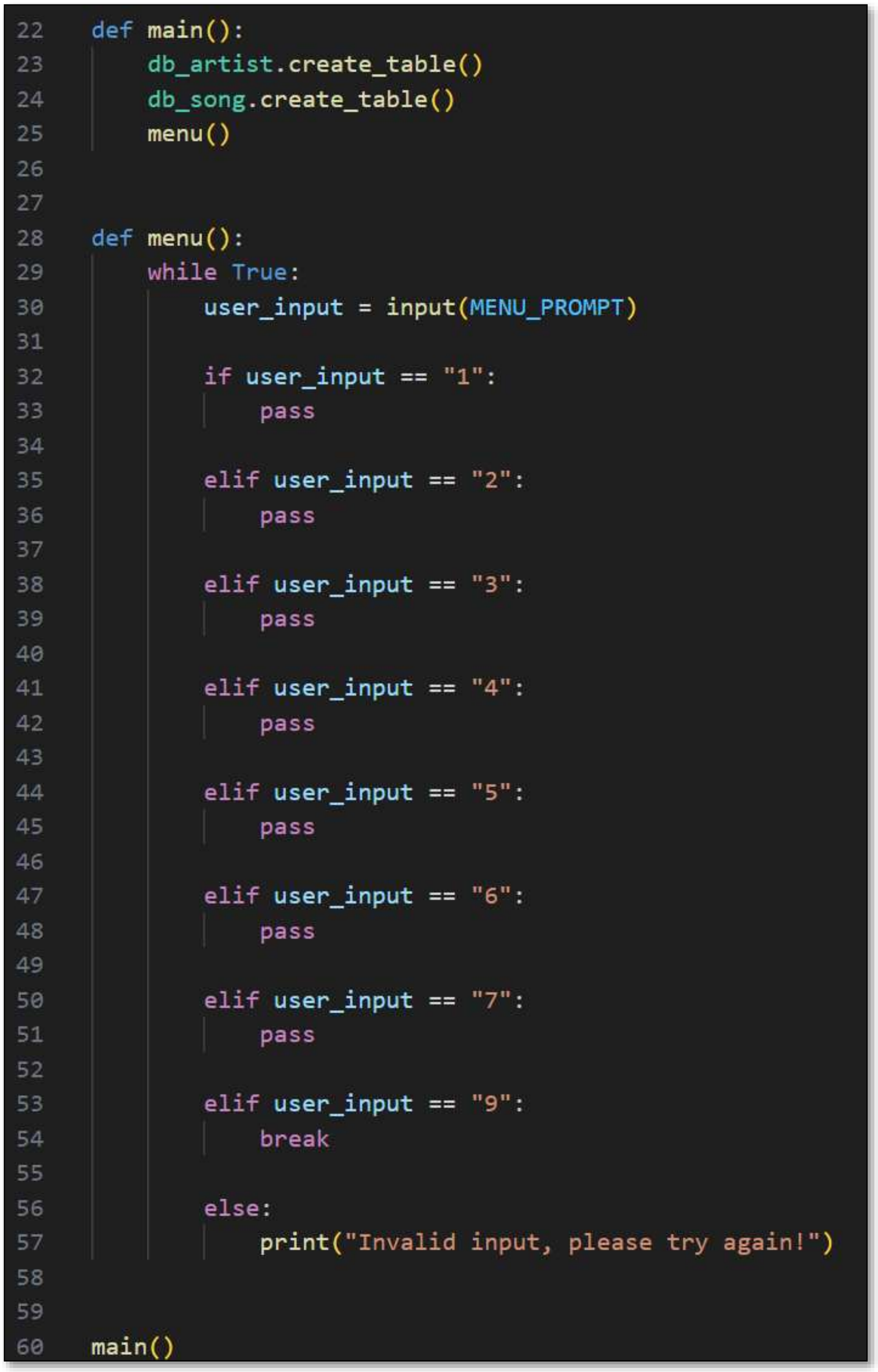
Create a menu prompt and store it as a multi line string.

Install the tabulate library. This allows us to format our displays in a nice table format.



```python
"""
    Name: music_library_app.py
    Author: William Loring
    Created: 10/07/24
    Use SQLite with Python
"""
# https://pypi.org/project/tabulate/
# pip install tabulate
import tabulate
import db_artist
import db_song

MENU_PROMPT = """------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: """
```

This is the outline of the menu system for the program.



```
def main():
    db_artist.create_table()
    db_song.create_table()
    menu()


def menu():
    while True:
        user_input = input(MENU_PROMPT)

        if user_input == "1":
            pass

        elif user_input == "2":
            pass

        elif user_input == "3":
            pass

        elif user_input == "4":
            pass

        elif user_input == "5":
            pass

        elif user_input == "6":
            pass

        elif user_input == "7":
            pass

        elif user_input == "9":
            break

        else:
            print("Invalid input, please try again!")


main()
```

Assignment Submission

1. Attach the program files.

2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.

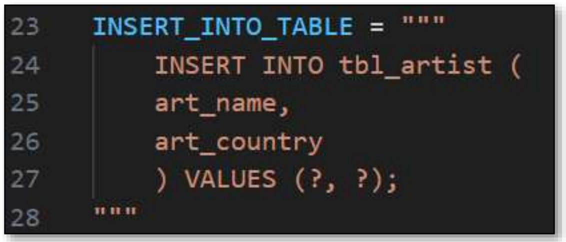
Part 4: Insert Artist

Use your own database design.

Our 1 to many relationship requires that the artist primary key exists before it can be used in the song table.

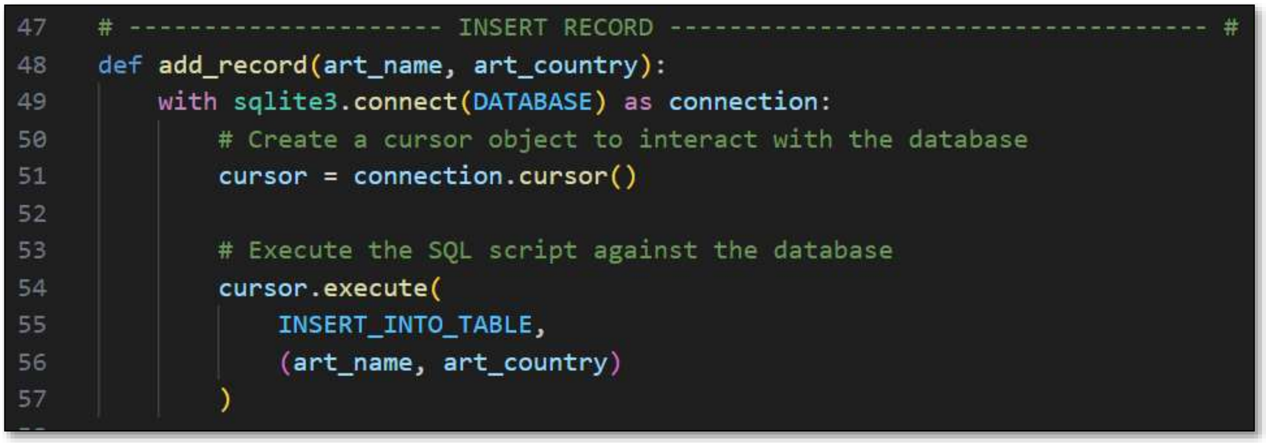
When a new record is added to the artist table, the **art_id** PK is automatically incremented starting at 1. This unique value is the primary key of the artist table. To connect the tables in a 1 to many relationship, **art_id** is used in the song table to determine which artist created the song. The artist table **art_id** primary key must exist first.

db_artist.py



```python
INSERT_INTO_TABLE = """
    INSERT INTO tbl_artist (
    art_name,
    art_country
    ) VALUES (?, ?);
"""
```

art_name and **art_country** are passed as parameters to the insert record function.



```python
# -------------------- INSERT RECORD ---------------------------------- #
def add_record(art_name, art_country):
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Execute the SQL script against the database
        cursor.execute(
            INSERT_INTO_TABLE,
            (art_name, art_country)
        )
```

To see if we have actually inserted a record, we want to add a query to show our records. Add this line right below INSERT_INTO_TABLE. This query will select all records from the artist table.

```python
FETCH_ALL_RECORDS = "SELECT * FROM tbl_artist;"
```

Let’s add a function to return our records to the application. The records are returned as a list, with each record being a tuple in that list.

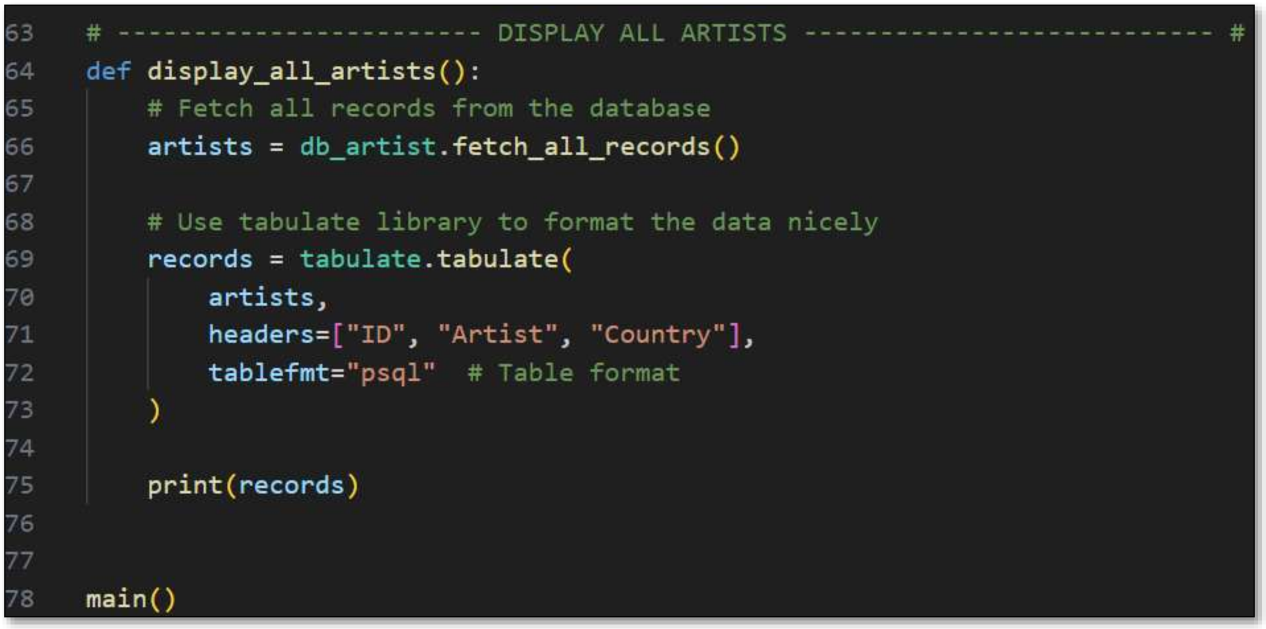
```python
# ------------------- FETCH ALL RECORDS ---------------------------------- #
def fetch_all_records():
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # A list of tuples. Each tuple is a record/row in the database
        records = cursor.execute(FETCH_ALL_RECORDS).fetchall()

        return records
```

In **music_library_app.py** we must add the artist first. We can’t add a song until we know the artist’s primary key. Let’s setup the menu and code to insert records, and show the artists table.

music_library_app.py

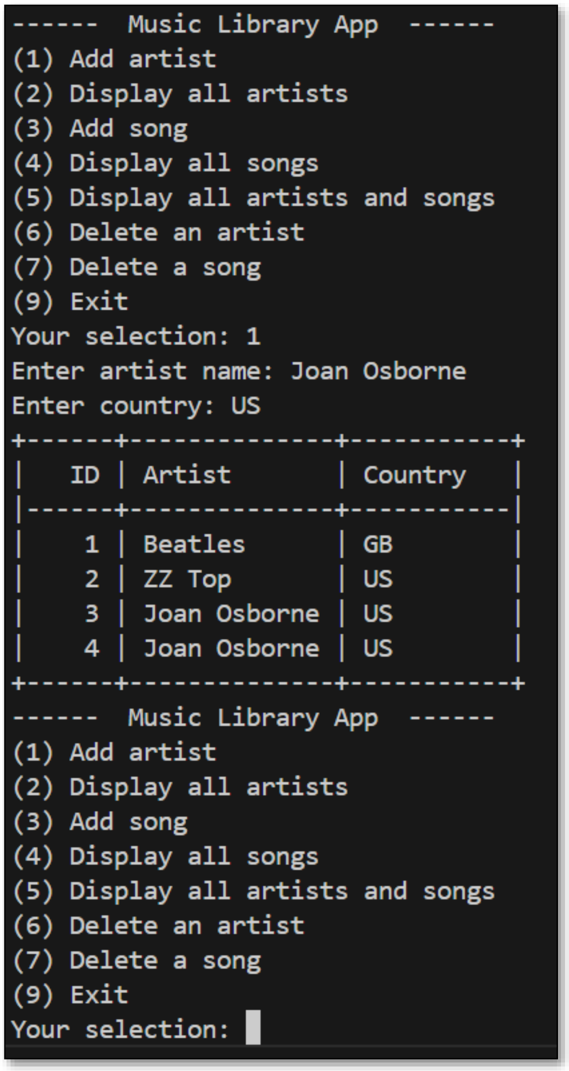
```python
def menu():

    while (True):
        user_input = input(MENU_PROMPT)

    # ---------------------- INSERT ARTIST ------------------------------- #
        if user_input == "1":
            art_name = input("Enter artist name: ")
            art_country = input("Enter country: ")

            db_artist.add_record(
                art_name,
                art_country
            )

            display_all_artists()

    # -------------------- DISPLAY ALL ARTISTS --------------------------- #
        elif user_input == "2":
            # Display the returned records from the SQL query
            display_all_artists()
```

The tabulate library creates a nice table to display our data.



```python
# ------------------------ DISPLAY ALL ARTISTS --------------------------- #
def display_all_artists():
    # Fetch all records from the database
    artists = db_artist.fetch_all_records()

    # Use tabulate library to format the data nicely
    records = tabulate.tabulate(
        artists,
        headers=["ID", "Artist", "Country"],
        tablefmt="psql"  # Table format
    )

    print(records)


main()
```

Example run:



```
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 1
Enter artist name: Joan Osborne
Enter country: US
+------+-------------+-----------+
|   ID | Artist      | Country   |
|------+-------------+-----------|
|    1 | Beatles     | GB        |
|    2 | ZZ Top      | US        |
|    3 | Joan Osborne | US       |
|    4 | Joan Osborne | US       |
+------+-------------+-----------+
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection:
```

Assignment Submission

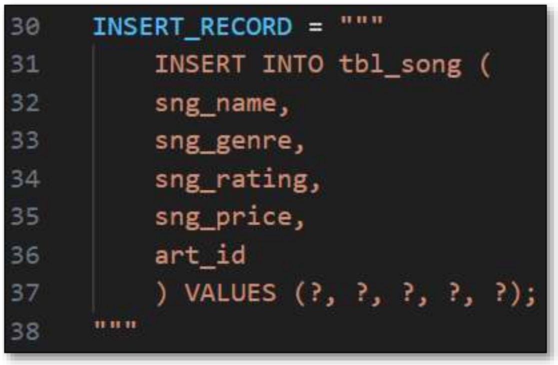
1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.

Part 5: Insert Song

Use your own database design.

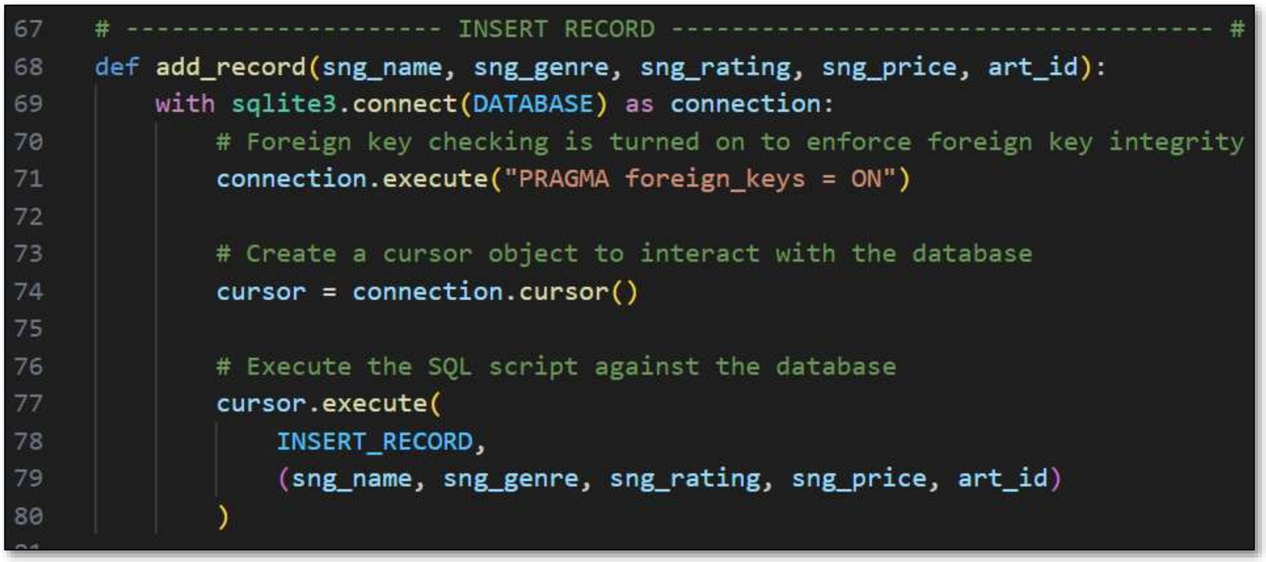
db_song.py

Insert the following SQL code at the top of the program file.

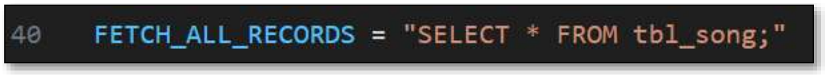

```sql
INSERT_RECORD = """
    INSERT INTO tbl_song (
    sng_name,
    sng_genre,
    sng_rating,
    sng_price,
    art_id
    ) VALUES (?, ?, ?, ?, ?);
"""
```

By default, foreign_key restriction is turned off for safety. We want it turned on. When we delete an artist, we want to delete their songs. If we don't do that, we will end up with orphaned records.

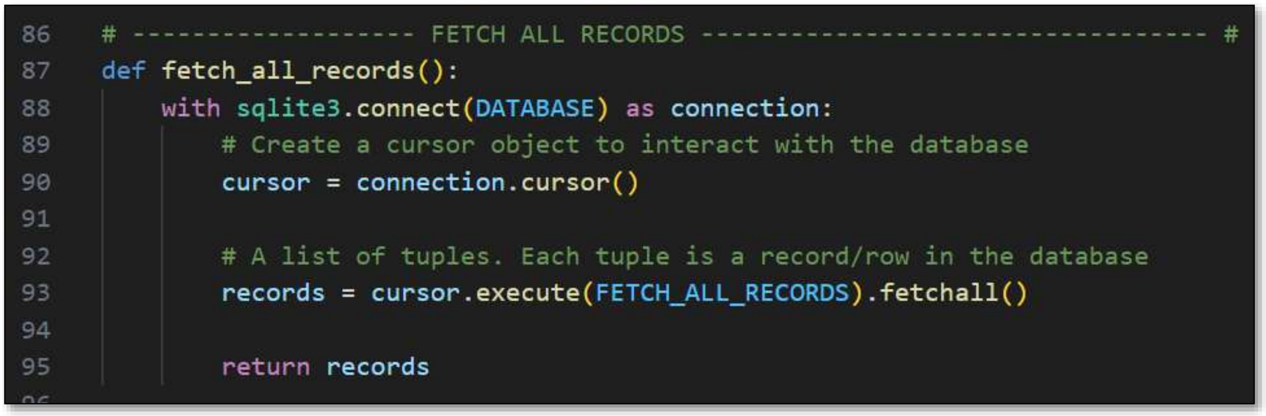
Foreign key checking is turned on to enforce foreign key integrity.


```python
# --------------------- INSERT RECORD ----------------------------------- #
def add_record(sng_name, sng_genre, sng_rating, sng_price, art_id):
    with sqlite3.connect(DATABASE) as connection:
        # Foreign key checking is turned on to enforce foreign key integrity
        connection.execute("PRAGMA foreign_keys = ON")

        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Execute the SQL script against the database
        cursor.execute(
            INSERT_RECORD,
            (sng_name, sng_genre, sng_rating, sng_price, art_id)
        )
```

We need an SQL query to select our song records.


```python
FETCH_ALL_RECORDS = "SELECT * FROM tbl_song;"
```

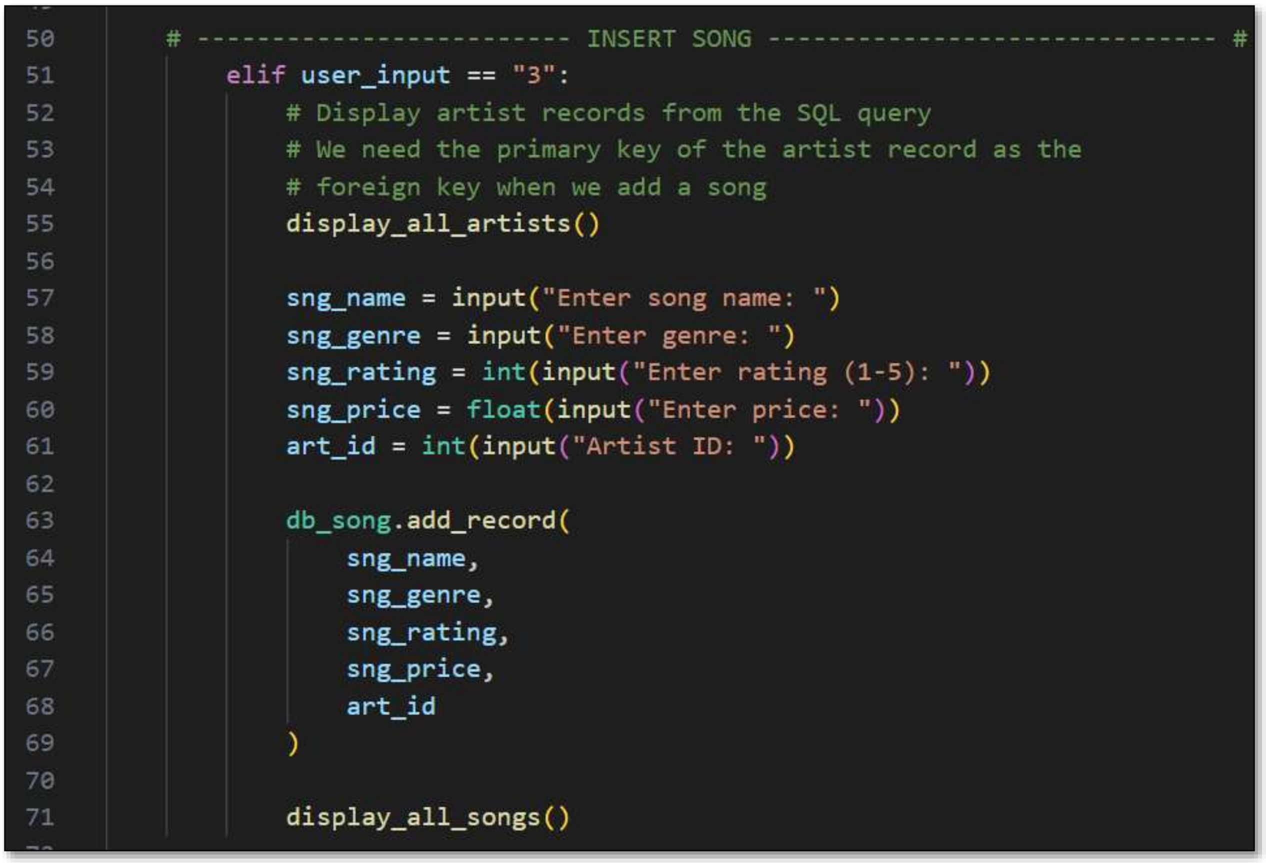
And a corresponding function to return those records to the application.



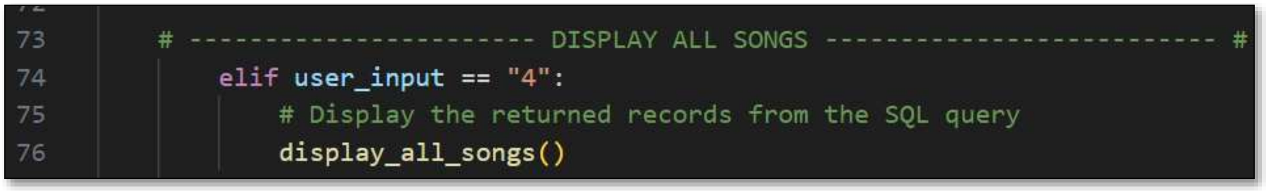
```python
# ------------------- FETCH ALL RECORDS ---------------------------------- #
def fetch_all_records():
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # A list of tuples. Each tuple is a record/row in the database
        records = cursor.execute(FETCH_ALL_RECORDS).fetchall()

        return records
```

music_library_app.py

When we insert a song, we need the primary key of the artist record for the foreign key in the song table.

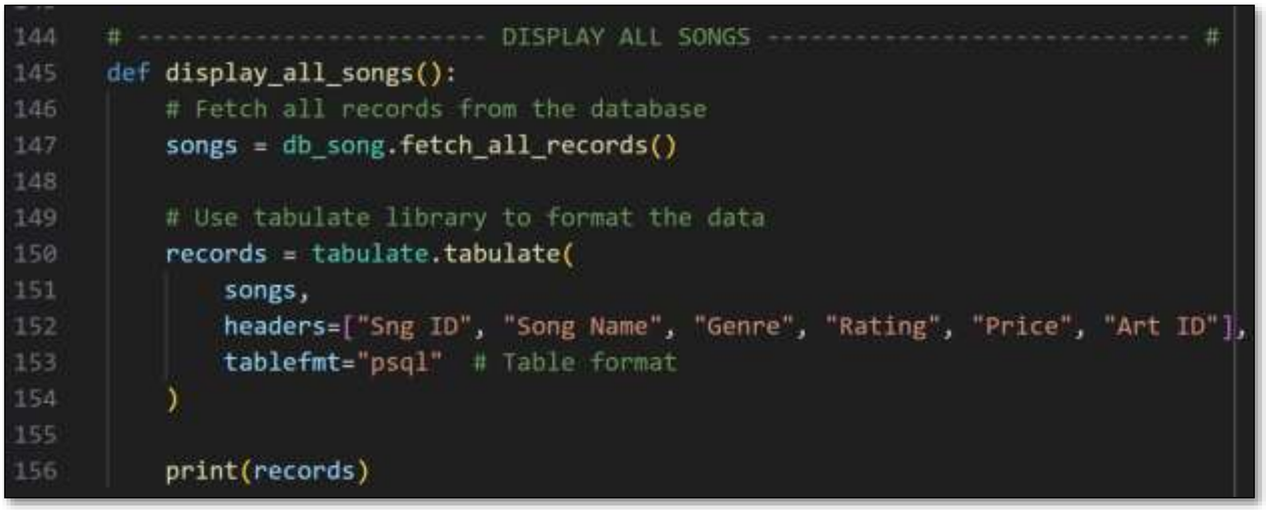


```python
    # ------------------------- INSERT SONG ----------------------------- #
        elif user_input == "3":
            # Display artist records from the SQL query
            # We need the primary key of the artist record as the
            # foreign key when we add a song
            display_all_artists()

            sng_name = input("Enter song name: ")
            sng_genre = input("Enter genre: ")
            sng_rating = int(input("Enter rating (1-5): "))
            sng_price = float(input("Enter price: "))
            art_id = int(input("Artist ID: "))

            db_song.add_record(
                sng_name,
                sng_genre,
                sng_rating,
                sng_price,
                art_id
            )

            display_all_songs()
```



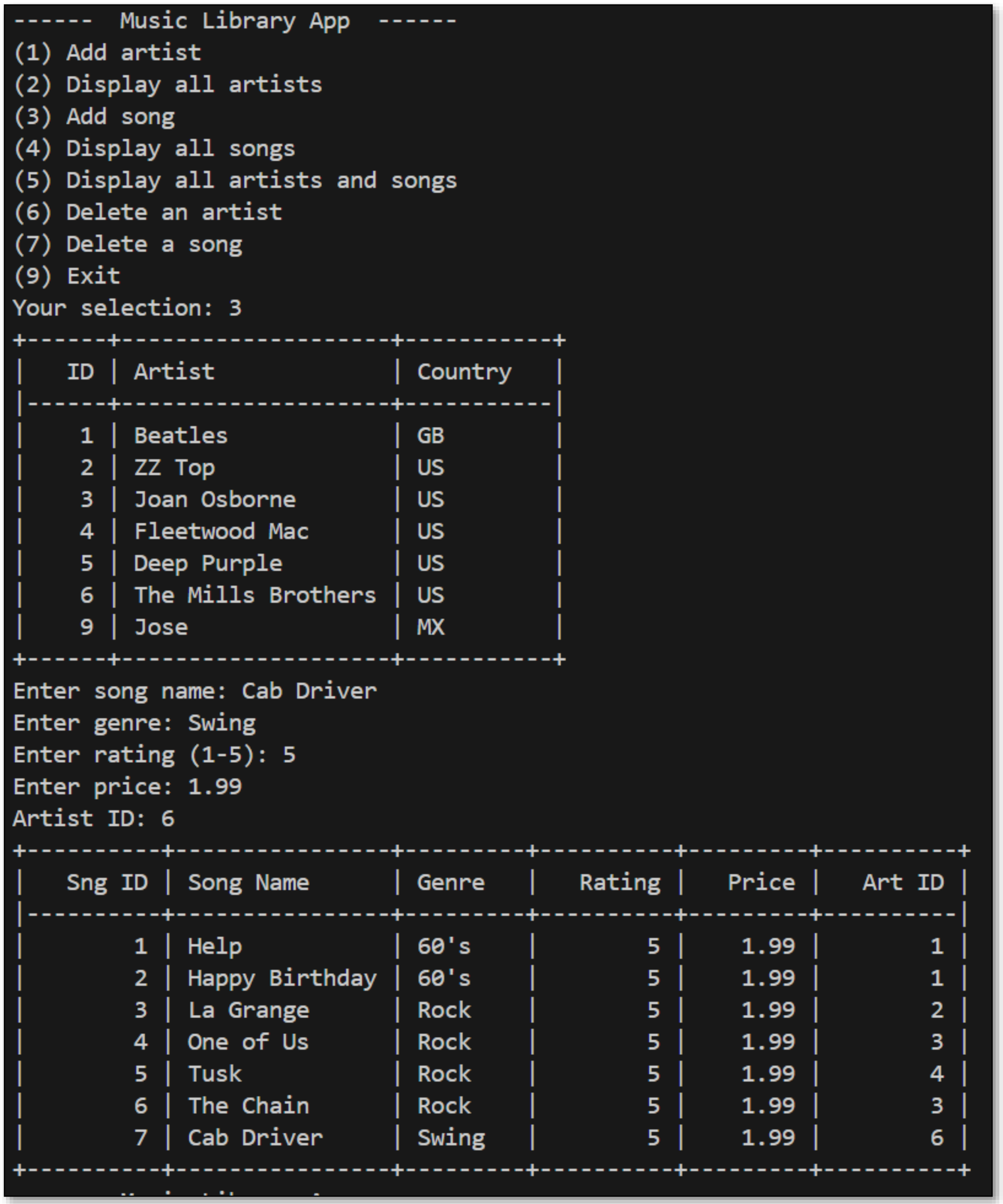
```python
    # ---------------------- DISPLAY ALL SONGS ------------------------ #
        elif user_input == "4":
            # Display the returned records from the SQL query
            display_all_songs()
```

Add this function at the bottom of the program before the call to main().



```python
# ----------------------- DISPLAY ALL SONGS ---------------------------- #
def display_all_songs():
    # Fetch all records from the database
    songs = db_song.fetch_all_records()

    # Use tabulate library to format the data
    records = tabulate.tabulate(
        songs,
        headers=["Sng ID", "Song Name", "Genre", "Rating", "Price", "Art ID"],
        tablefmt="psql"  # Table format
    )

    print(records)
```

Example run:



```
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 3
+------+-------------------+-----------+
|   ID | Artist            | Country   |
|------+-------------------+-----------|
|    1 | Beatles           | GB        |
|    2 | ZZ Top            | US        |
|    3 | Joan Osborne      | US        |
|    4 | Fleetwood Mac     | US        |
|    5 | Deep Purple       | US        |
|    6 | The Mills Brothers | US       |
|    9 | Jose              | MX        |
+------+-------------------+-----------+
Enter song name: Cab Driver
Enter genre: Swing
Enter rating (1-5): 5
Enter price: 1.99
Artist ID: 6
+----------+----------------+---------+----------+---------+----------+
|   Sng ID | Song Name      | Genre   |   Rating |   Price |   Art ID |
|----------+----------------+---------+----------+---------+----------|
|        1 | Help           | 60's    |        5 |    1.99 |        1 |
|        2 | Happy Birthday | 60's    |        5 |    1.99 |        1 |
|        3 | La Grange      | Rock    |        5 |    1.99 |        2 |
|        4 | One of Us      | Rock    |        5 |    1.99 |        3 |
|        5 | Tusk           | Rock    |        5 |    1.99 |        4 |
|        6 | The Chain      | Rock    |        5 |    1.99 |        3 |
|        7 | Cab Driver     | Swing   |        5 |    1.99 |        6 |
+----------+----------------+---------+----------+---------+----------+
```

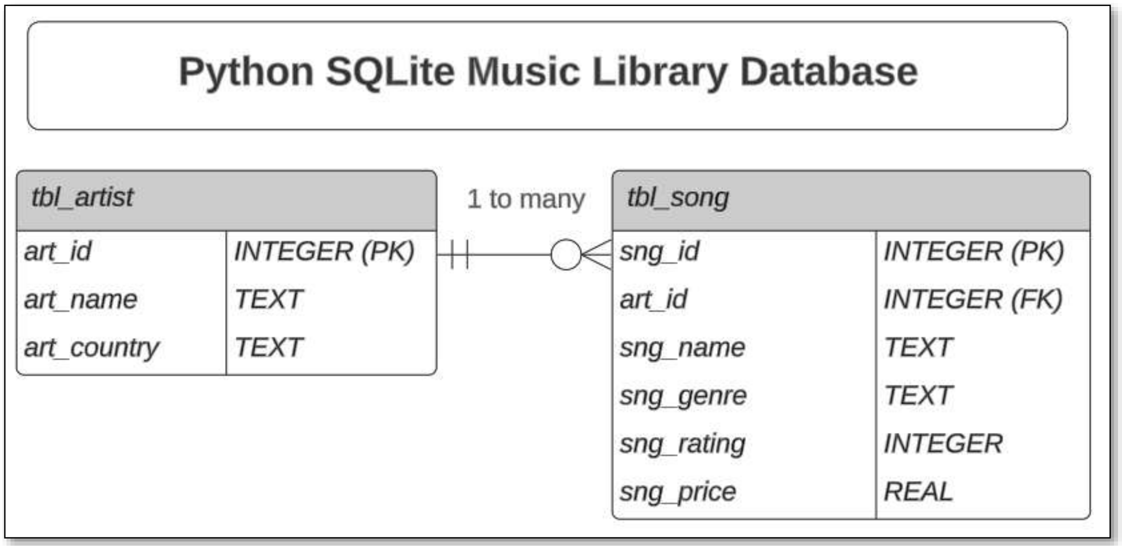
Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.

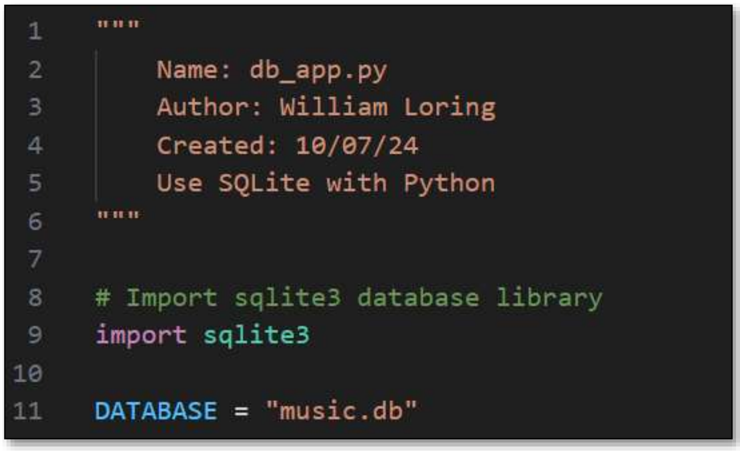
Part 6: Fetch Related Data

This code retrieves song and artist information a SQL JOIN operation.

For every artist, we are going to select multiple songs.

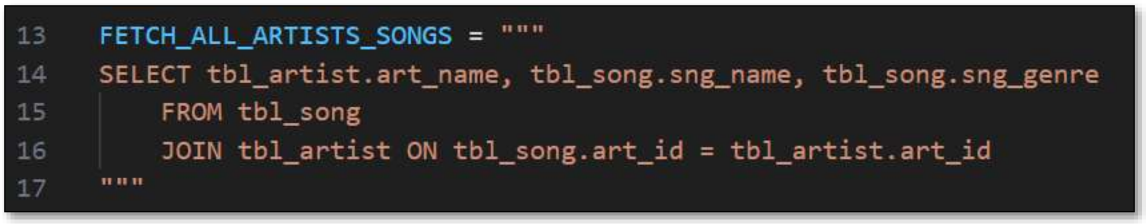


Create a new file named **db_app.py**

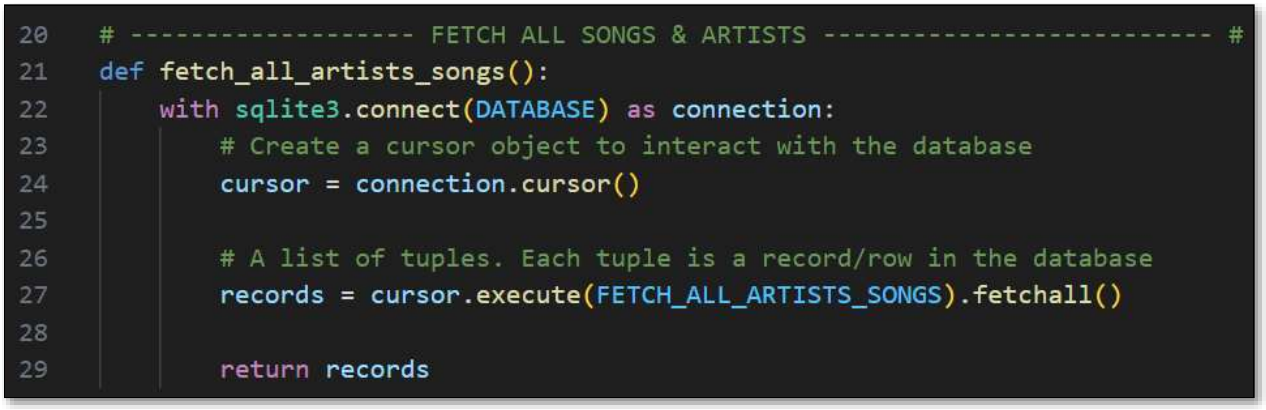


```python
"""
    Name: db_app.py
    Author: William Loring
    Created: 10/07/24
    Use SQLite with Python
"""

# Import sqlite3 database library
import sqlite3

DATABASE = "music.db"
```

SQL code to fetch related song information for an artist.



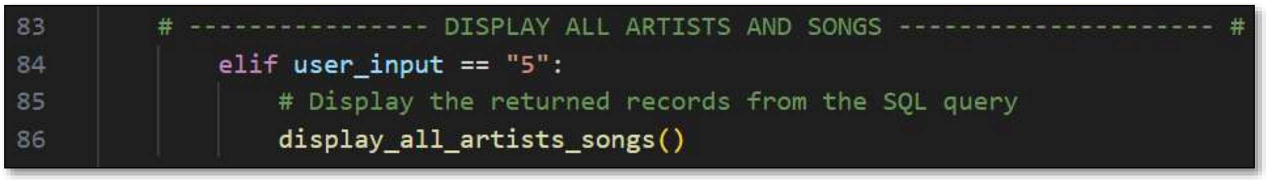
```python
FETCH_ALL_ARTISTS_SONGS = """
SELECT tbl_artist.art_name, tbl_song.sng_name, tbl_song.sng_genre
    FROM tbl_song
    JOIN tbl_artist ON tbl_song.art_id = tbl_artist.art_id
"""
```



```python
# ------------------- FETCH ALL SONGS & ARTISTS ------------------------- #
def fetch_all_artists_songs():
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # A list of tuples. Each tuple is a record/row in the database
        records = cursor.execute(FETCH_ALL_ARTISTS_SONGS).fetchall()

        return records
```

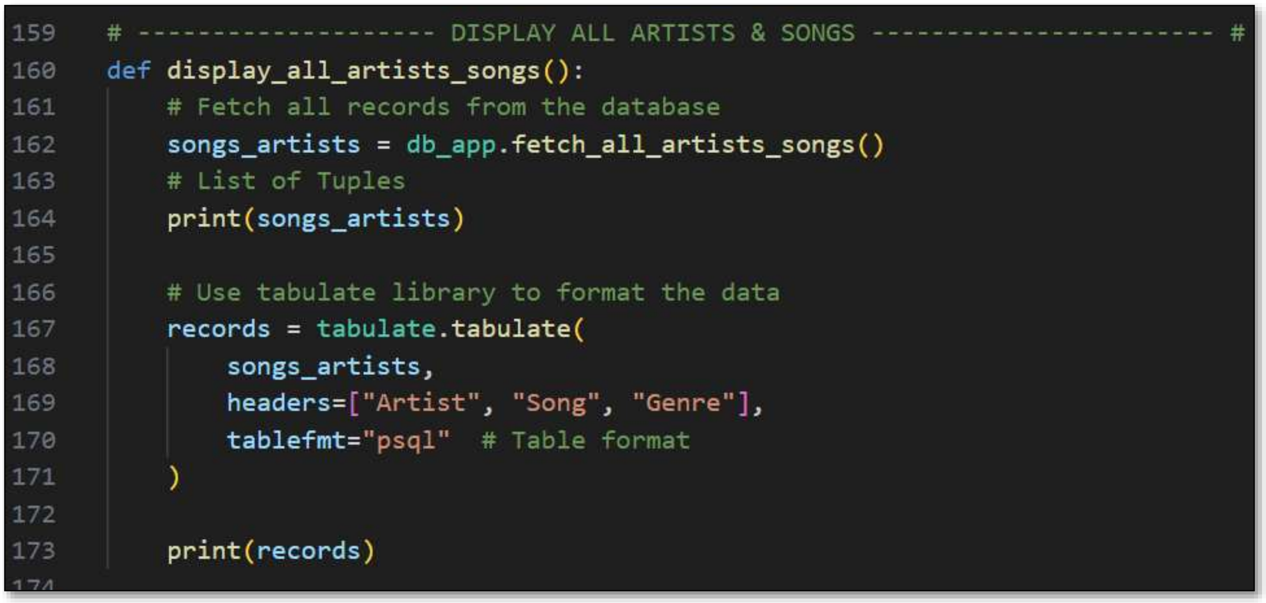
music_library_app.py

Add this code to the menu.



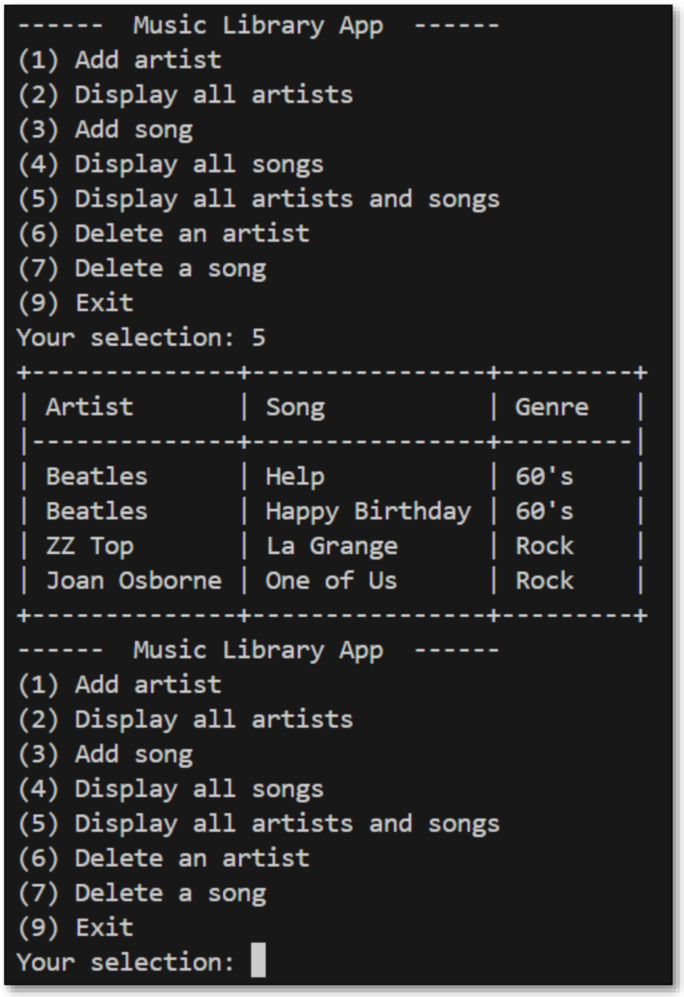
```python
    # ---------------- DISPLAY ALL ARTISTS AND SONGS --------------------- #
        elif user_input == "5":
            # Display the returned records from the SQL query
            display_all_artists_songs()
```

This function goes at the bottom of the file before the main() call.



```python
# ------------------- DISPLAY ALL ARTISTS & SONGS ---------------------- #
def display_all_artists_songs():
    # Fetch all records from the database
    songs_artists = db_app.fetch_all_artists_songs()
    # List of Tuples
    print(songs_artists)

    # Use tabulate library to format the data
    records = tabulate.tabulate(
        songs_artists,
        headers=["Artist", "Song", "Genre"],
        tablefmt="psql"  # Table format
    )

    print(records)
```

Example run:



```
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 5
+-------------+---------------+---------+
| Artist      | Song          | Genre   |
|-------------+---------------+---------|
| Beatles     | Help          | 60's    |
| Beatles     | Happy Birthday| 60's    |
| ZZ Top      | La Grange     | Rock    |
| Joan Osborne| One of Us     | Rock    |
+-------------+---------------+---------+
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 
```

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.

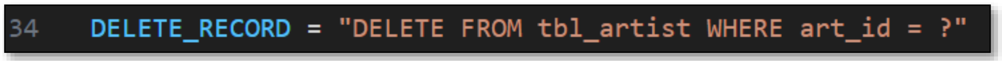
Part 7: Delete Data

Use your own database design.

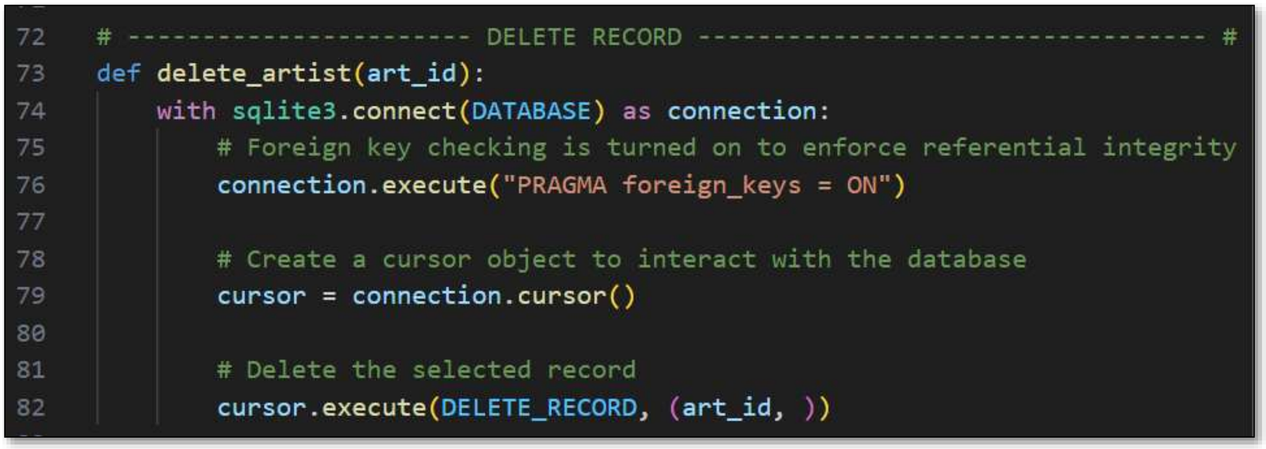
The relationship between the tables restricts the order in which we can delete records. An artist can have many songs. A song must have an artist.

We can delete a song with no trouble. Deleting an artist is different as songs are tied by a foreign key to the artist. If we delete the artist without deleting the songs, we are left with orphaned records. We do what is called a cascade delete. When an artist is deleted, all the artist’s songs are deleted.

db_artist.py

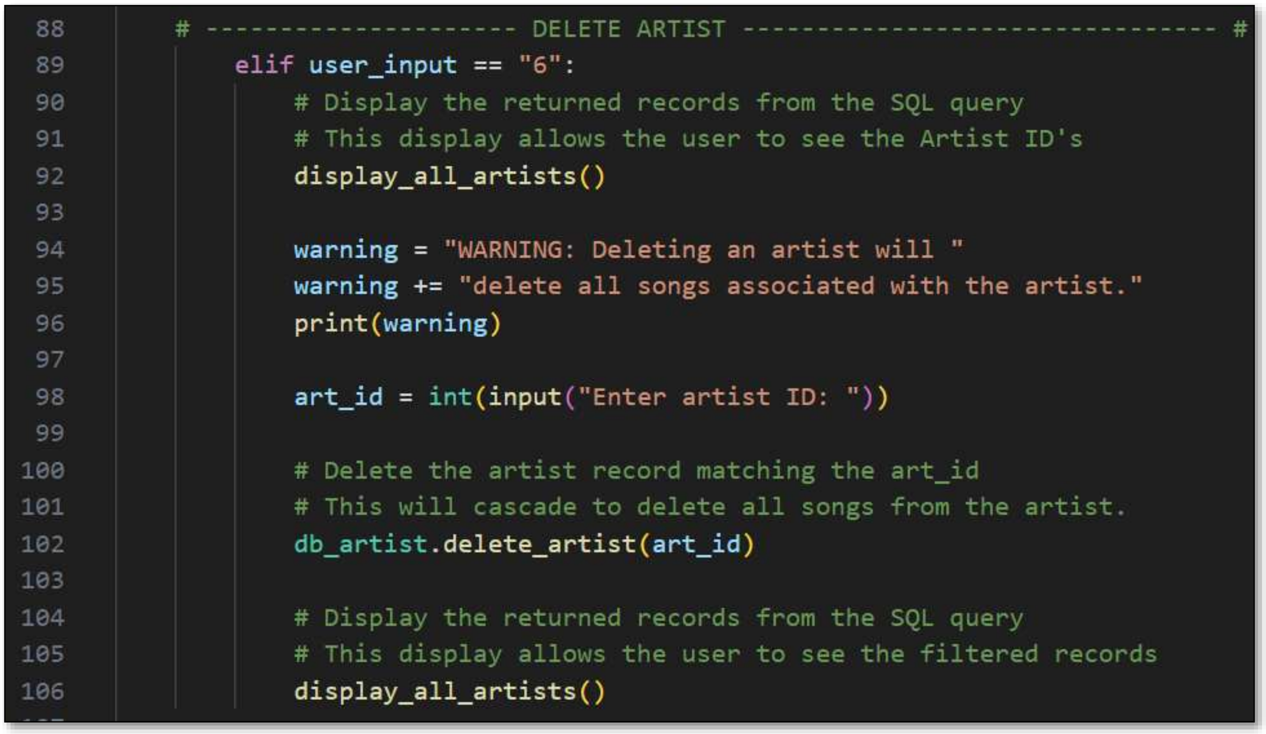


```python
DELETE_RECORD = "DELETE FROM tbl_artist WHERE art_id = ?"
```



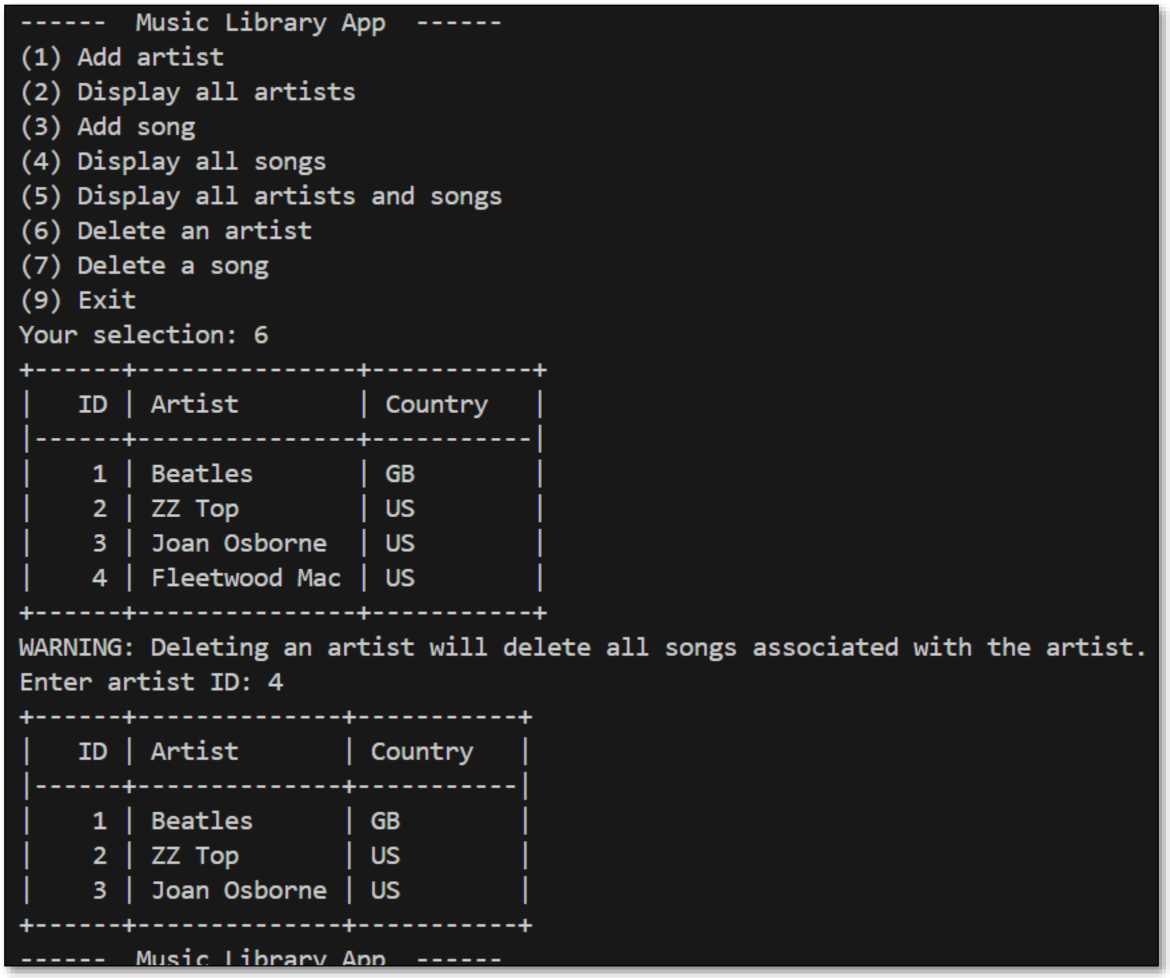
```python
# ----------------------- DELETE RECORD ---------------------------------- #
def delete_artist(art_id):
    with sqlite3.connect(DATABASE) as connection:
        # Foreign key checking is turned on to enforce referential integrity
        connection.execute("PRAGMA foreign_keys = ON")

        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Delete the selected record
        cursor.execute(DELETE_RECORD, (art_id, ))
```

music_library_app.py



```python
    # -------------------- DELETE ARTIST ------------------------------ #
        elif user_input == "6":
            # Display the returned records from the SQL query
            # This display allows the user to see the Artist ID's
            display_all_artists()

            warning = "WARNING: Deleting an artist will "
            warning += "delete all songs associated with the artist."
            print(warning)

            art_id = int(input("Enter artist ID: "))

            # Delete the artist record matching the art_id
            # This will cascade to delete all songs from the artist.
            db_artist.delete_artist(art_id)

            # Display the returned records from the SQL query
            # This display allows the user to see the filtered records
            display_all_artists()
```

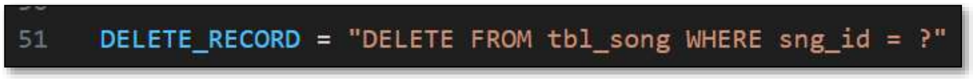
Example run:



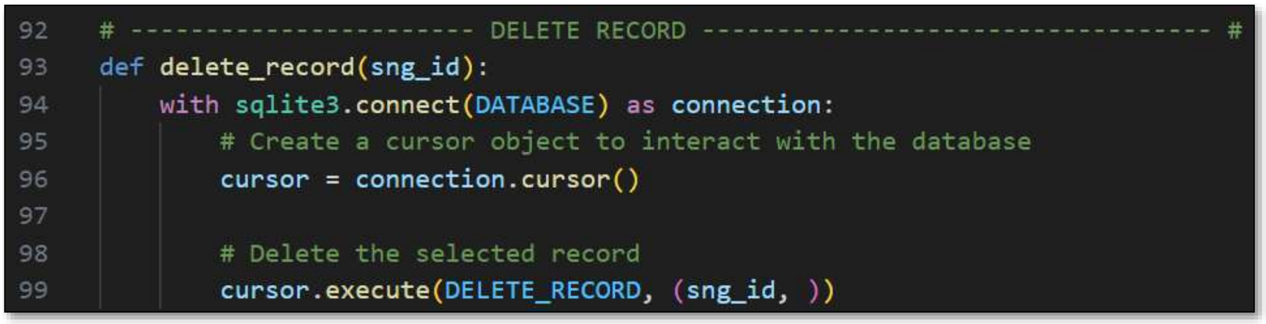
```
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 6
+------+--------------+-----------+
|   ID | Artist       | Country   |
|------+--------------+-----------|
|    1 | Beatles      | GB        |
|    2 | ZZ Top       | US        |
|    3 | Joan Osborne | US        |
|    4 | Fleetwood Mac | US       |
+------+--------------+-----------+
WARNING: Deleting an artist will delete all songs associated with the artist.
Enter artist ID: 4
+------+--------------+-----------+
|   ID | Artist       | Country   |
|------+--------------+-----------|
|    1 | Beatles      | GB        |
|    2 | ZZ Top       | US        |
|    3 | Joan Osborne | US        |
+------+--------------+-----------+
------  Music Library App  ------
```

Deleting a song does not require foreign key checking.

db_song.py

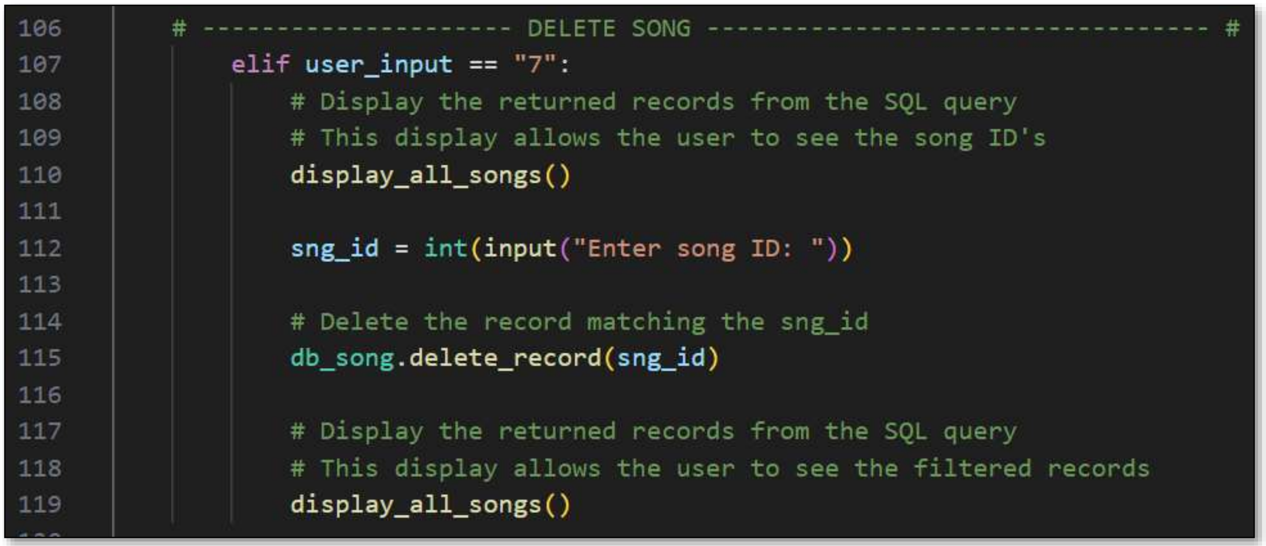


```python
DELETE_RECORD = "DELETE FROM tbl_song WHERE sng_id = ?"
```



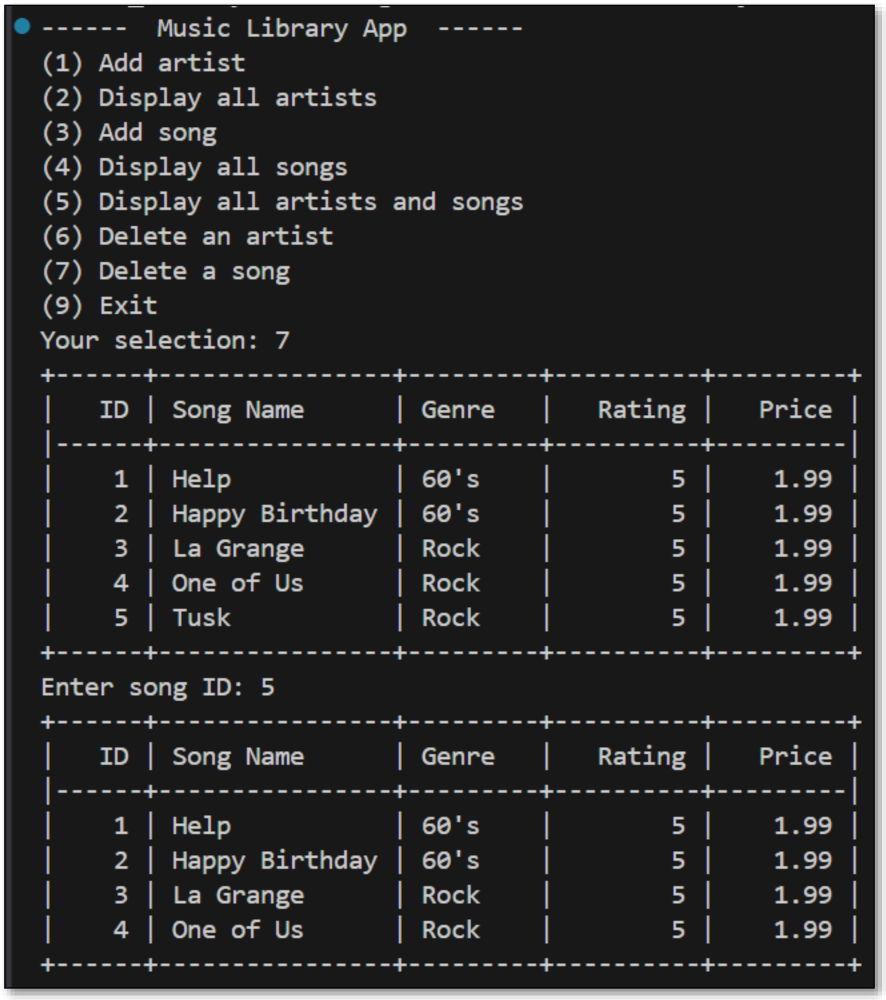
```python
# ---------------------- DELETE RECORD -------------------------------- #
def delete_record(sng_id):
    with sqlite3.connect(DATABASE) as connection:
        # Create a cursor object to interact with the database
        cursor = connection.cursor()

        # Delete the selected record
        cursor.execute(DELETE_RECORD, (sng_id, ))
```

music_library_app.py



```python
    # -------------------- DELETE SONG -------------------------------- #
        elif user_input == "7":
            # Display the returned records from the SQL query
            # This display allows the user to see the song ID's
            display_all_songs()

            sng_id = int(input("Enter song ID: "))

            # Delete the record matching the sng_id
            db_song.delete_record(sng_id)

            # Display the returned records from the SQL query
            # This display allows the user to see the filtered records
            display_all_songs()
```

Example run:



```
------  Music Library App  ------
(1) Add artist
(2) Display all artists
(3) Add song
(4) Display all songs
(5) Display all artists and songs
(6) Delete an artist
(7) Delete a song
(9) Exit
Your selection: 7
+------+----------------+---------+----------+---------+
|   ID | Song Name      | Genre   |   Rating |   Price |
|------+----------------+---------+----------+---------|
|    1 | Help           | 60's    |        5 |    1.99 |
|    2 | Happy Birthday | 60's    |        5 |    1.99 |
|    3 | La Grange      | Rock    |        5 |    1.99 |
|    4 | One of Us      | Rock    |        5 |    1.99 |
|    5 | Tusk           | Rock    |        5 |    1.99 |
+------+----------------+---------+----------+---------+
Enter song ID: 5
+------+----------------+---------+----------+---------+
|   ID | Song Name      | Genre   |   Rating |   Price |
|------+----------------+---------+----------+---------|
|    1 | Help           | 60's    |        5 |    1.99 |
|    2 | Happy Birthday | 60's    |        5 |    1.99 |
|    3 | La Grange      | Rock    |        5 |    1.99 |
|    4 | One of Us      | Rock    |        5 |    1.99 |
+------+----------------+---------+----------+---------+
```

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.