

Chapter 12: Python Database with SQLite

Contents

Chapter 12: Python Database with SQLite	1
SQL Online Tutorial	2
SQLite with Python Tutorials	2
SQLite and Python	2
Self-Contained	3
Zero-Configuration	3
Transactional (ACID)	3
Single Database File	4
CRUD	4
Tutorial 1: DB Browser for SQLite	5
Install SQLite DB Browser for SQLite	5
Create Database	5
Model View Controller (MVC)	7
Assignment 1: Create Your Own Database Planning Document	8
Tutorial 2: Data Dictionary	9
Data Dictionary	9
Assignment 2: Data Dictionary	9
Assignment 3: Create Table	10
Explanation of SQL Code	10
with Statement	11
Cursor Object	11
Assignment 4: Database Application	12
Assignment 5: Insert Records	13
Assignment 6: Select Records	16
Assignment 7: Display All Records Function	17
Assignment 8: Display a Specific Record	19
Assignment 9: Delete a Specific Record	21
MVC Final Note	23



No AI use.

Time required: 120 minutes

- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

SQL Online Tutorial

Please go through these tutorials to familiarize yourself with SQL scripting.

- https://www.w3schools.com/sql/sql_intro.asp
- https://www.w3schools.com/sql/sql_syntax.asp
- https://www.w3schools.com/sql/sql_select.asp
- https://www.w3schools.com/sql/sql_create_db.asp

SQLite with Python Tutorials

These are optional tutorials for deeper study of SQL.

- [SQLite Databases with Python - Full Course](#) – FreeCodeCamp.org
- <https://www.sqlitetutorial.net>

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.

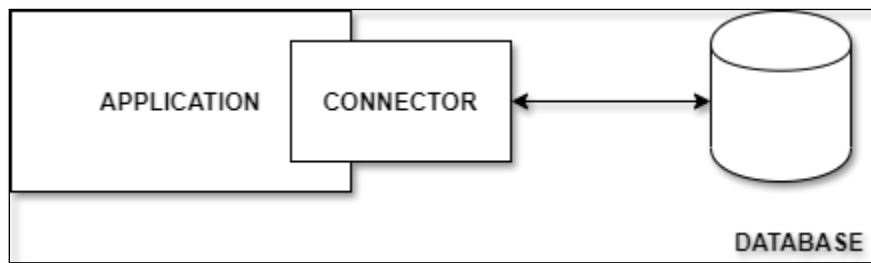
Databases offer numerous functionalities by which one can manage large amounts of information easily over the web and high-volume data input and output over a typical file such as a text file. SQL is a query language and is very popular in databases. Many websites use MySQL. SQLite is a “light” version 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

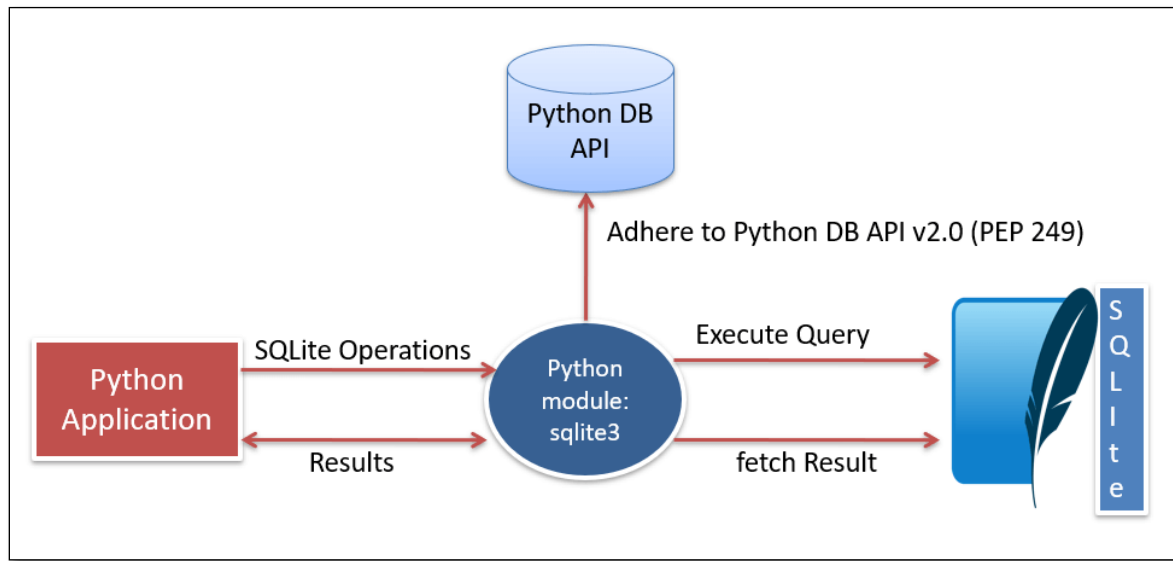
Zero-configuration means no setup or administration needed. Because of the serverless architecture, you don't need to "install" SQLite before using it. There is no server process that needs to be configured, started, and stopped.

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.

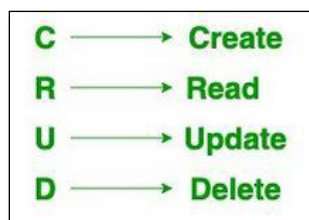
Single Database File

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

Tutorial 1: DB Browser for SQLite

Install SQLite DB Browser for SQLite

This is a handy tool to look at, troubleshoot, and manipulate your database. We will use it to learn the basics of SQL.

1. Go to <https://sqlitebrowser.org>
2. Go to the **Download** tab.
3. Download the **Windows PortableApp → DB Browser for SQLite - PortableApp**
4. Double Click the installation file. Click **Next**.
5. Click **Install**. Click **Finish**.
6. You will find a new folder: **SQLiteDatabaseBrowserPortable**
7. This folder can be moved anywhere, the program will work just fine.
8. In the folder you will find **SQLiteDatabaseBrowserPortable.exe**
9. Double Click the file. Click **OK** on the warning if there is one.

Create Database

1. Double Click **SQLiteDatabaseBrowserPortable.exe**
2. Click **New Database**.
3. Filename: **game_shop.db** → Click **Save**.
4. Edit table definition
 - a. Table: **products**
5. Fields:
 - a. Click **Add**: Name: **prod_id**
 - b. Click **Add**: Name: **prod_name**
 - c. Click **Add**: Name: **prod_price**
 - d. Click **Add**: Name: **prod_qty**

6. Make the setting changes as shown below. Notice the SQL code is created automatically. This is same code you would use to create this database and table in Python. SQL is SQL wherever it is used.

The screenshot shows the 'Edit table definition' dialog box for a table named 'products'. The 'Table' field contains 'products'. Below it is an 'Advanced' button. The 'Fields' tab is selected, showing a table with columns: Name, Type, NN, PK, AI, U, Default, and Check. The columns are: prod_id (INTEGER, NN, PK, AI, U, Default, Check), prod_name (TEXT, NN, PK, AI, U, Default, Check), prod_price (REAL, NN, PK, AI, U, Default, Check), and prod_qty (INTEGER, NN, PK, AI, U, Default, Check). The 'prod_price' row is highlighted. Below the table is a scroll bar. At the bottom, the SQL code is displayed:

```
1 CREATE TABLE "products" (  
2     "prod_id"  INTEGER,  
3     "prod_name" TEXT,  
4     "prod_price" REAL,  
5     "prod_qty" INTEGER,  
6     PRIMARY KEY("prod_id")  
7 );
```

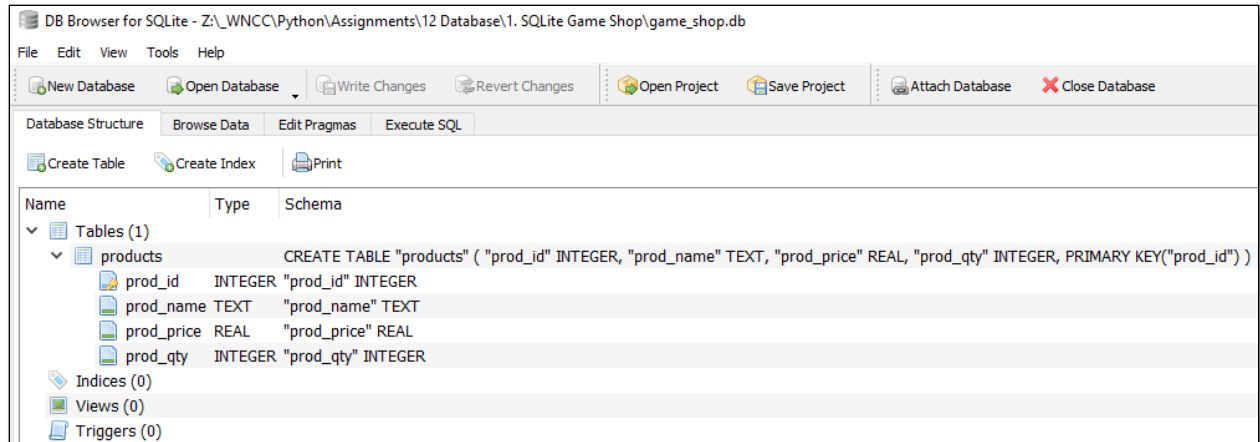
 The 'OK' and 'Cancel' buttons are at the bottom right.

Name	Type	NN	PK	AI	U	Default	Check
prod_id	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
prod_name	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
prod_price	REAL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
prod_qty	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

```
1 CREATE TABLE "products" (  
2     "prod_id"  INTEGER,  
3     "prod_name" TEXT,  
4     "prod_price" REAL,  
5     "prod_qty" INTEGER,  
6     PRIMARY KEY("prod_id")  
7 );
```

7. Click **OK** to create your table.

You should see the following result.



8. The changes you made are only in memory. To commit the changes to the database file, Click **Write Changes**.
9. Include a screenshot with the assignment.
10. Click **Close Database** to disconnect from the database.
11. Close DB Browser for SQLite.

Model View Controller (MVC)

This tutorial breaks the program into MVC (Model View Controller)

Model-view-controller (MVC) is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

Model

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

View

Any representation of information such as a chart, diagram, or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

Controller

Accepts input and converts it to commands for the model or view.[15]

In addition to dividing the application into these components, the model-view-controller design defines the interactions between them.

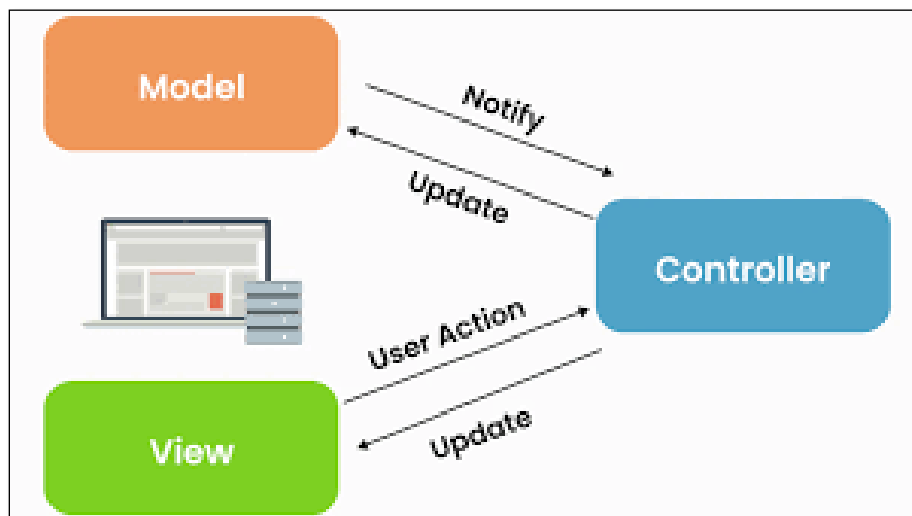
- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view renders presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

From Wikipedia.

Model: music.db (Database)

View: music_library_app.py (User interface)

Controller: database.py (This contains the SQL code that works with the database.)



Assignment 1: Create Your Own Database Planning Document

You are going to use this tutorial as a guide to create your own database.

Tables are like nouns. In OOP, we call it a class. In database speak, it is called an entity.

Car, Cupcake, Spaceship, Video Game, Team, etc.

Create a planning document for your database. Add the following items to this document.

1. What information do you want to store in the database, what is it's purpose?
2. Identify a single entity for a table.

Tutorial 2: Data Dictionary

In an SQL database, data is stored in tables. Tables define a set of columns and fields, much like a spreadsheet. They contain 0 or more rows with data for each of the defined fields.

Database design is similar to OOP class design. You have an entity and attributes. Attributes describe the class or table.

You have identified your entity/table. We want to find what describes the table, the attributes of the entity.

Car – Speed, color, doors, type, manufacturer, etc.

Data Dictionary

Let's look at an example table named **song** that stores the following data about a song. This is a data dictionary. We will use data dictionaries to design our future databases.

Field Name	Data Type	Description
sng_id (primary key)	INTEGER	Autonumber unique song identifier
sng_name	TEXT	Song name
sng_genre	TEXT	Song genre
sng_artist	TEXT	Song artist
sng_rating	INTEGER	Song rating 1-5
sng_price	REAL	Song price

Assignment 2: Data Dictionary

Create your own database that you will use for this assignment. The code and examples are here as a guide for you to create your own project.

Create a data dictionary for your database using the above example as a guide.

Add the following to your planning document.

1. Create a data dictionary with a minimum of 4 fields, not including the primary key.

Assignment 3: Create Table

NOTE: Use your database design.

Create a Python program file named **database.py**

This file will contain all the logic for working with the database. We will have a separate file for user interaction.

We will create a table using SQLite3 in Python. Use the data dictionary you created earlier. Use the example code as a guide for your database.

```
1  """
2      Name: database.py
3      Author: William Loring
4      Created: 10/07/24
5      CRUD module for Music Library database
6  """
7  # Import sqlite3 database library
8  import sqlite3
9
10 DATABASE = "music.db"
11
12 # ----- SQL STATEMENTS ----- #
13 # SQL statements are text. SQL queries can be very long.
14 # A SQL statement can be assigned to a string variable.
15 CREATE_TABLE = """
16     CREATE TABLE IF NOT EXISTS song (
17         sng_id        INTEGER PRIMARY KEY,
18         sng_name       TEXT,
19         sng_genre      TEXT,
20         sng_artist     TEXT,
21         sng_rating     INTEGER,
22         sng_price      REAL
23     );
24 """
```

Explanation of SQL Code

SQL is a scripting language like Python. You can assign SQL code to a String variable using a single line or multiline string. The formatting of the text in the following SQL code is for readability. We will keep all SQL code at the top of the database.py program.

- The **CREATE_SONG_TABLE** constant is a multiline string is a SQL statement that creates a table named **song** with the columns described earlier:

- **sng_id** of type INTEGER PRIMARY KEY
 - **sng_name** of type TEXT
 - **sng_genre** of type TEXT
 - **sng_artist** of type TEXT
 - **sng_rating** of type INTEGER
- A **primary key** is a unique value that provides a unique identifier for each record. A primary key is typically a sequence of whole numbers (int). We are using the built-in SQLite autonumber function. This will give us a unique PK for each record.

with Statement

Notice the **with** statement in the below code. The **with** statement in Python automatically manages resources like files and databases. It ensures that resources are cleaned up (like closing a database connection) after you're done, even if errors occur. The cursor object and all connections to the database are closed when the with statement exits.

Cursor Object

Let's discuss the cursor object.

- The cursor object is used to make the connection to the database for executing text-based SQL queries.
- It acts as middleware or controller between the SQLite database connection and the SQL query. It is created after creating a connection to the SQLite database.
- The cursor is a control structure used to traverse the records of the database.
- All SQL commands are be executed using the cursor object only.

The following code is inserted after the existing code. Read the comments to understand the code.

```

43 # ----- CREATE TABLE ----- #
44 def create_table():
45     with sqlite3.connect(DATABASE) as connection:
46         # Create a cursor object to interact with the database
47         cursor = connection.cursor()
48
49         # Execute the SQL script against the database
50         cursor.execute(CREATE_TABLE)

```

Assignment 4: Database Application

NOTE: Use your database design.

Let's create the application that will use **database.py** to allow user interaction.

For our example database uses a file named **music_library_app.py** Create a name for your database app file.

Our first step is to import the database module created earlier.

```

1  """
2      Name: music_library_app.py
3      Author: William Loring
4      Created: 10/07/24
5      Use SQLite with Python
6  """
7  import database

```

We will need a menu prompt to guide the user through the program. Modify this menu prompt for your database.

```

10 MENU_PROMPT = """----- Music Library App -----
11 (1) Add new song
12 (2) Display all songs
13 (3) Find song by ID
14 (4) Delete song
15 (5) Exit
16 Your selection: """

```

This is the skeleton of the rest of the application.

```

19  def main():
20      database.create_table()
21      menu()
22
23
24  def menu():
25      while True:
26          user_input = input(MENU_PROMPT)
27
28          if user_input == "1":
29              pass
30
31          elif user_input == "2":
32              pass
33
34          elif user_input == "3":
35              pass
36
37          elif user_input == "4":
38              pass
39
40          elif user_input == "5":
41              break
42
43          else:
44              print("Invalid input, please try again!")
45
46
47  main()

```

Assignment 5: Insert Records

NOTE: Use your database design.

In SQL, the **INSERT INTO** statement is used to add new records to a table.

```

INSERT INTO table_name
VALUES (value1, value2, ...);

```

In **database.py**, add the following SQL statement below the previous SQL statement.

The ? marks are placeholders to prevent SQL injection.

```

26  INSERT_RECORD = """
27      INSERT INTO song (
28          sng_name,
29          sng_genre,
30          sng_artist,
31          sng_rating,
32          sng_price
33      ) VALUES (?, ?, ?, ?, ?);
34  """

```

Notice in the code above we are not setting a value for the `trk_id`. When the record is created, SQLite autonumbers the primary key. A primary key serves as a unique identifier for a record. It typically has no meaning to the database.

Add the following code below the `create_table()` method.

```

53  # ----- INSERT RECORD ----- #
54  def insert_record(sng_name, sng_genre, sng_artist, sng_rating, sng_price):
55      with sqlite3.connect(DATABASE) as connection:
56          # Create a cursor object to interact with the database
57          cursor = connection.cursor()
58
59          # Execute the SQL script against the database
60          cursor.execute(
61              INSERT_RECORD,
62              (sng_name, sng_genre, sng_artist, sng_rating, sng_price)
63          )
64

```

To insert a record into our table, we will get input from the user for each field and add the record to the **song** table.

In **music_library_app.py**

```

24 def menu():
25     while True:
26         user_input = input(MENU_PROMPT)
27
28         # ----- ADD RECORD TO TABLE ----- #
29         if user_input == "1":
30             # Get data from user
31             sng_name = input("Enter song name: ")
32             sng_genre = input("Enter genre: ")
33             sng_artist = input("Enter artist: ")
34             sng_rating = int(input("Enter rating (1-5): "))
35             sng_price = float(input("Enter price: "))
36
37             database.insert_record(
38                 sng_name,
39                 sng_genre,
40                 sng_artist,
41                 sng_rating,
42                 sng_price
43             )

```

Start your app file. Add a record. There shouldn't be any errors.

Example run:

```

----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 1
Enter song name: Proud Mary
Enter genre: Rock
Enter artist: Credence Clearwater Revival
Enter rating (1-5): 5
Enter price: 1.99
----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 5

```

Did it work? We won't know until the next assignment when we can see the records.

Assignment 6: Select Records

NOTE: Use your database design.

The SELECT statement in SQL is used to retrieve data from a database. The following statement will select all records from the specified table.

```
SELECT * from table
```

In **database.py** add the following below the last SQL statement.

```
36  FETCH_ALL_RECORDS = "SELECT * FROM song;"
```

Add the following function below the last function.

```
66  # ----- FETCH ALL RECORDS ----- #
67  def fetch_all_records():
68      with sqlite3.connect(DATABASE) as connection:
69          # Create a cursor object to interact with the database
70          cursor = connection.cursor()
71
72          # A list of tuples. Each tuple is a record/row in the database
73          records = cursor.execute(FETCH_ALL_RECORDS).fetchall()
74
75          return records
```

In **music_library_app.py** add this code to our menu.

Use your database design.

```
45  # ----- DISPLAY ALL RECORDS ----- #
46  elif user_input == "2":
47      songs = database.fetch_all_records()
48
49      # Iterate through the list of tuples returned
50      # from the database query
51      for song in songs:
52          # Print each item in the tuple using the [] bracket operator
53          # to retrieve each item in the tuple
54          record = f"ID:({song[0]}) {song[1]} "
55          record += f"{song[2]} {song[3]} {song[4]} {song[5]}"
56          print(record)
```

Example run:


```

----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 2
ID:(1) Bohemian Rhapsody Rock Queen 5 2.59
ID:(2) Hotel California Rock Eagles 5 2.3
ID:(3) Imagine Pop Beatles 5 3.19
ID:(4) Thriller Pop Michael Jackson 5 4.57
ID:(5) Purple Haze Rock Jimi Hendrix 4 2.59
ID:(7) La Grange Rock ZZ Top 5 2.99
ID:(9) One of Us Rock Joan Osborne 5 1.99
----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 5

```

Assignment 7: Display All Records Function

NOTE: Use your database design.

We want to be able to display all the records from a function, rather than repeating the code. At the top of the program, copy the previous select code and create this function. We can use this function any time we want to display our records.

In **music_library_app.py** add this function.

```

73  # ----- DISPLAY ALL RECORDS ----- #
74  def display_all_songs():
75      # Fetch all records from the database
76      songs = database.fetch_all_records()
77
78      # Iterate through the list of tuples returned from the database query
79      for song in songs:
80          # Print each item in the tuple using the [] bracket operator
81          # to retrieve each item in the tuple
82          record = f"ID:({song[0]}) {song[1]} "
83          record += f"{song[2]} {song[3]} {song[4]} {song[5]}"
84          print(record)

```

Change the following code in the menu. We will call the **display_all_songs()** function whenever we want to display any returned records.

```
24 def menu():
25     while True:
26         user_input = input(MENU_PROMPT)
27
28         # ----- ADD RECORD TO TABLE ----- #
29         if user_input == "1":
30             # Get data from user
31             sng_name = input("Enter song name: ")
32             sng_genre = input("Enter genre: ")
33             sng_artist = input("Enter artist: ")
34             sng_rating = int(input("Enter rating (1-5): "))
35             sng_price = float(input("Enter price: "))
36
37             database.insert_record(
38                 sng_name,
39                 sng_genre,
40                 sng_artist,
41                 sng_rating,
42                 sng_price
43             )
44
45             display_all_songs()
46
47         # ----- DISPLAY ALL RECORDS ----- #
48         elif user_input == "2":
49             # Display the returned records from the SQL query
50             display_all_songs()
```

Example run:

```

----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 2
ID:(1) Bohemian Rhapsody Rock Queen 5 2.59
ID:(2) Hotel California Rock Eagles 5 2.3
ID:(3) Imagine Pop Beatles 5 3.19
ID:(4) Thriller Pop Michael Jackson 5 4.57
ID:(5) Purple Haze Rock Jimi Hendrix 4 2.59
ID:(7) La Grange Rock ZZ Top 5 2.99
ID:(9) One of Us Rock Joan Osborne 5 1.99
----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 5

```

Assignment 8: Display a Specific Record

NOTE: Use your database design.

Add this SQL statement to your **database.py**

```

38  FETCH_RECORD = "SELECT * FROM song WHERE sng_id = ?;"

```

Add this function.

```

78  # ----- FETCH RECORD ----- #
79  def fetch_record(sng_id: int):
80      with sqlite3.connect(DATABASE) as connection:
81          # Create a cursor object to interact with the database
82          cursor = connection.cursor()
83
84          # A list of tuples. Each tuple is a record/row in the database
85          records = cursor.execute(FETCH_RECORD, (sng_id, )).fetchall()
86
87          return records

```

In your application, add the following.

```

52      # ----- FIND RECORD ----- #
53      elif user_input == "3":
54          # This display allows the user to see the track ID's
55          display_all_songs()
56
57          sng_id = int(input("Enter song ID: "))
58
59          # Display the returned records from the SQL query
60          # This display allows the user to see the filtered records
61          display_song(sng_id)

```

```

100     # ----- DISPLAY SPECIFIC RECORD ----- #
101     def display_song(sng_id: int):
102         # Fetch all records from the database
103         songs = database.fetch_record(sng_id)
104
105         # Iterate through the list of tuples returned from the database query
106         for song in songs:
107             # Print each item in the tuple using the [] bracket operator
108             # to retrieve each item in the tuple
109             record = f"ID:({song[0]}) {song[1]} "
110             record += f"{song[2]} {song[3]} {song[4]} {song[5]}"
111             print(record)

```

Example run:

```

----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 3
ID:(1) Bohemian Rhapsody Rock Queen 5 2.59
ID:(2) Hotel California Rock Eagles 5 2.3
ID:(3) Imagine Pop Beatles 5 3.19
ID:(4) Thriller Pop Michael Jackson 5 4.57
ID:(5) Purple Haze Rock Jimi Hendrix 4 2.59
ID:(7) La Grange Rock ZZ Top 5 2.99
ID:(9) One of Us Rock Joan Osborne 5 1.99
Enter song ID: 2
ID:(2) Hotel California Rock Eagles 5 2.3
----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 5

```

Assignment 9: Delete a Specific Record

NOTE: Use your database design.

Add this SQL statement to your **database.py**

```

40 DELETE_RECORD = "DELETE FROM song WHERE sng_id = ?"

```

Add this function.

```

90 # ----- DELETE RECORD ----- #
91 def delete_record(sng_id: int):
92     with sqlite3.connect(DATABASE) as connection:
93         # Create a cursor object to interact with the database
94         cursor = connection.cursor()
95
96         # Delete the selected record
97         cursor.execute(DELETE_RECORD, (sng_id, ))

```

In your application, add the following.

```

64         # ----- DELETE RECORD ----- #
65         elif user_input == "4":
66             # Display the returned records from the SQL query
67             # This display allows the user to see the track ID's
68             display_all_songs()
69
70             sng_id = int(input("Enter song ID: "))
71
72             # Fetch only the record matching the sng_id
73             database.delete_record(sng_id)
74
75             # Display the returned records from the SQL query
76             # This display allows the user to see the filtered records
77             display_all_songs()

```

Example run: Use your database design.

```

----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: 4
ID:(1) Bohemian Rhapsody Rock Queen 5 2.59
ID:(2) Hotel California Rock Eagles 5 2.3
ID:(3) Imagine Pop Beatles 5 3.19
ID:(4) Thriller Pop Michael Jackson 5 4.57
ID:(5) Purple Haze Rock Jimi Hendrix 4 2.59
ID:(7) La Grange Rock ZZ Top 5 2.99
ID:(9) One of Us Rock Joan Osborne 5 1.99
Enter song ID: 3
ID:(1) Bohemian Rhapsody Rock Queen 5 2.59
ID:(2) Hotel California Rock Eagles 5 2.3
ID:(4) Thriller Pop Michael Jackson 5 4.57
ID:(5) Purple Haze Rock Jimi Hendrix 4 2.59
ID:(7) La Grange Rock ZZ Top 5 2.99
ID:(9) One of Us Rock Joan Osborne 5 1.99
----- Music Library App -----
(1) Add new song
(2) Display all songs
(3) Find song by ID
(4) Delete song
(5) Exit
Your selection: █

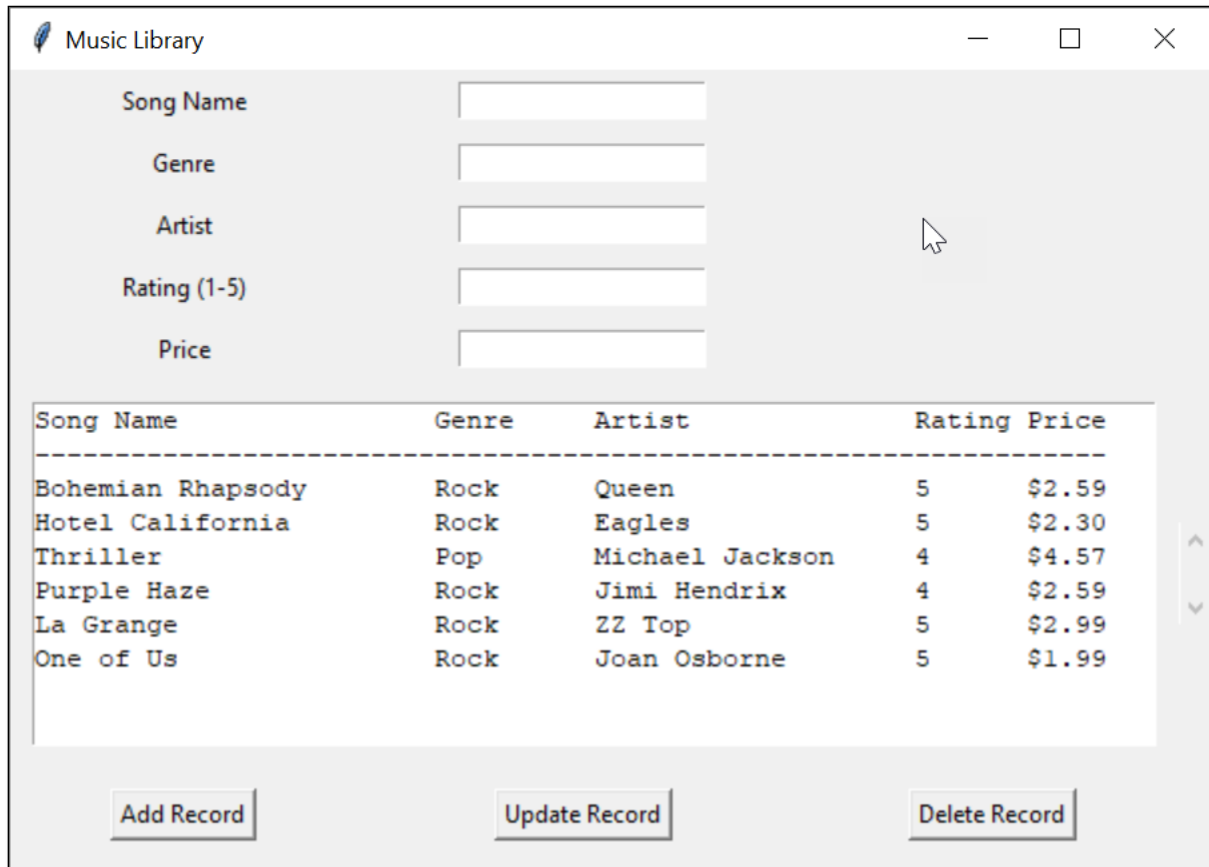
```

MVC Final Note

Separating the view/user interface from the data controller, it is fairly easy to change the user interface to a GUI or a Web page.

NOTE: Populating the listbox required another SQL statement and code.

Here is a Tkinter example using the same model and controller.



Song Name	Genre	Artist	Rating	Price
Bohemian Rhapsody	Rock	Queen	5	\$2.59
Hotel California	Rock	Eagles	5	\$2.30
Thriller	Pop	Michael Jackson	4	\$4.57
Purple Haze	Rock	Jimi Hendrix	4	\$2.59
La Grange	Rock	ZZ Top	5	\$2.99
One of Us	Rock	Joan Osborne	5	\$1.99

Assignment Submission

1. Attach your database design document.
2. Attach all files and assignments.
3. Attach screenshots showing the successful operation of the final program.
4. Submit in Blackboard.