4. Python SQLite Game Shop Tutorial – Insert Records

Contents

Time required: 30 minutes

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

SQL Tutorial

- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_insert.asp

SQLite with Python Tutorials

- SQLite Databases with Python - Full Course – FreeCodeCamp.org
- https://www.sqlitetutorial.net

Insert Records

Examples of sample records we will be inserting. PK stands for **P**rimary **K**ey.

prod_id (PK)	prod_name	prod_desc	prod_rank	prod_price	prod_qty
1	Minecraft		4	15.99	1
2	Tetris		10	12.99	7
3	Pong		1	2.99	6
4	Guessing Game		1	.99	200

The product table will track a value for id, name, description, rank, price and quantity. The id is the primary key for the table. A primary key is a special relational database table column (or combination of columns) designated to uniquely identify each table record. You cannot have two id's with the same value

Tutorial 1: Refactor Program

You might have noticed we are starting to accumulate quite a bit of repeated and unorganized code. DRY time. Refactoring means to reorganize and improve your code.

Each time we execute an SQL operation on a database, we do the following. We do this to not leave the database in an unstable condition. This allows other users to work with the database and not have data integrity issues. SQLite supports one connection at a time.

1. Connect
2. Create a cursor
3. Commit
4. Close

Context Manager

A context manager in SQLite is typically used to connect to/create and close a database. A database connection object creates a transaction when used as a context manager. If you want to auto-connect or auto-close the connection, you can do that using the **with** keyword. A context manager reduces the amount of repeated code, it is a DRY solution.

with an object as stuff, do stuff, automatically close when the **with** statement exits.

```
with object as stuff:
    stuff.open_resources()
    stuff.do()
# All resources are closed
```

Drop Table

You use the **DROP** command to delete the table so we can rerun the program each time we modify the table structure. **DROP** deletes the records along with the table structure.

```
DROP TABLE IF EXISTS table_name
```

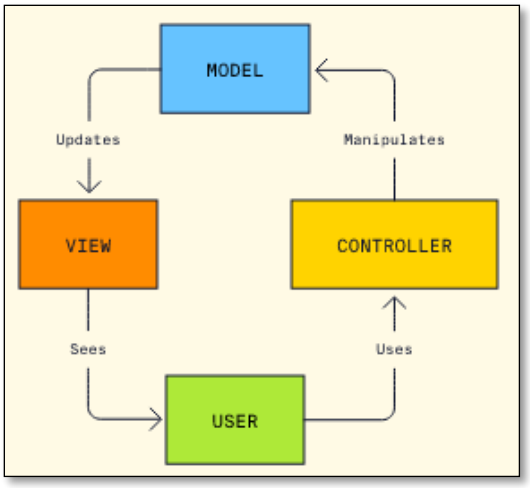
MVC (Model View Controller)

MVC is short for Model, View, and Controller. MVC is a popular way of organizing your code. The big idea behind MVC is that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

Model: Model code typically reflects real-world things. This code can hold raw data, or it will define the essential components of your app. For instance, if you were building a To-do app, the model code would define what a "task" is and what a "list" is – since those are the main components of a todo app.

View: View code is made up of all the functions that directly interact with the user. This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.

Controller: Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It's the brains of the application, and ties together the model and the view.



Tutorial 2: db_controller.py

We are going to move all the database specific functions into a separate file using the MVC paradigm.

Model: game_shop_3.db

View: sqlite_tutorial_3_delete.py

Controller: db_operations.py

Our program will grow increasingly cumbersome as we add more code. We are going to move all the controller code that works with the database into a separate file.

1. Create a Python file named **db_controller.py**
2. Start by creating a GameShop class.

```python
"""
    Name: db_operations.py
    Author: William Loring
    Created: 07/07/24
    CRUD module for GameShop database
"""

# Import sqlite3 database library
import sqlite3


Codiumate: Options | Test this class
class GameShop():
    Codiumate: Options | Test this method
    def __init__(self, database: str):
        # Initialize the database file with name passed as an argument
        self.database = database
```

3. Create a create table method. This does the same thing as we did earlier, but in a method form using the with context manager. Notice the **DROP TABLE** SQL command. This command would be removed once the development process is complete.

```
# ------------------------ CREATE TABLE ---------------------------------#
    def create_table(self):
        """Create database and table if not exists."""
        # If everything inside the with sqlite3.connect is successful,
        # connect.commit() and connect.close() are automatically called
        # when the with statement exits
        # Establish a connection to the database file using a
        # with context manager
        with sqlite3.connect(self.database) as connection:
            # Create a cursor object to interact with the database
            cursor = connection.cursor()

            # SQL statement to drop 'products' table if it exists
            # SQL statement to create the 'products' table
            SQL = """
            DROP TABLE IF EXISTS products;
            CREATE TABLE products (
                prod_id       INTEGER PRIMARY KEY,
                prod_name     TEXT,
                prod_desc     TEXT,
                prod_price    REAL,
                prod_rank     INTEGER,
                prod_qty      INTEGER
            );
            """

            # Execute the SQL sscript
            cursor.executescript(SQL)

        # Changes are committed automatically after the with handler exits
        # All connections are closed
```

We are hard coding the records for this tutorial for simplicity. Normally, we would pass the records from the user interface to the db_operations.py file.

- **INSERT INTO products** specifies that we are inserting into the products table.
- **VALUES** says that the values are coming up next.
- The values must match the datatypes of the table fields.

Insert the following changes into the program. You can insert any games you wish. Where it says: **insert your own description**, describe the game you are inserting.

```python
# ----------------------- INSERT RECORDS --------------------------------#
    def insert_records(self):
        """Insert predefined product records into the 'products' table."""
        # Establish a connection to the database file using the
        # with context manager
        with sqlite3.connect(self.database) as connection:
            # Create a cursor object to interact with the database
            cursor = connection.cursor()

            # SQL statements to insert predefined records
            # into the 'products' table
            SQL = """
            INSERT INTO products VALUES (
                1,
                'Minecraft',
                'Insert your own description',
                15.99,
                1,
                1);
            INSERT INTO products VALUES (
                2,
                'Tetris',
                'Insert your own description',
                12.99,
                3,
                7);
            """

            # Execute the SQL script
            cursor.executescript(SQL)
```

Tutorial 3: User Interface

This file is our user interface. There won't be much here as we are hard coding the database SQL.

Create a Python file named **sql_3_tutorial_insert.py**

```python
"""
    Name: sql_3_insert.py
    Author: William Loring
    Created: 07/06/24
    Use SQLite with Python
"""
from db_operations import GameShop

# Create a GameShop instance with the specified database file
game_shop = GameShop("game_shop_3.db")

# Create the 'products' table in the database
game_shop.create_table()
print("Table dropped if it exists")
print("Table created")

# Insert predefined records into the 'products' table
game_shop.insert_records()
print("Records inserted")
```

Notice how much simpler our user interface file is.

Example run:

```
Table dropped if it exists
Table created
Records inserted
```

Assignment: Add Records

1. Add two more game records to the **insert_record()** method.

You will have 4 game records in total, each with a product rank.

Assignment Submission

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