

INDUSTRIAL TRAINING DAILY DIARY

DAY 10

05 July, 2025

Topic : Integrating SQL with Python: Executing Database Operations Programmatically

Objectives :

- Learned how to integrate SQL with Python using basic SQL commands such as creating tables, inserting records, and modifying data.
 - Solved basic exercises involving SQL operations through Python scripts.
 - Implemented a small project titled “**Inventory Summary**” to apply the learned concepts in a real-world scenario.
-

- **SQLite3 in Python**

[SQLite](#) is the world’s most-used database and allows you to create an SQL-based database which can be integrated into all sorts of different applications. Python also supports SQLite3 by way of a module.

You can connect SQLite to Python by using a module. To make your work more efficient with the database system in Python, we recommend you have some background knowledge of both. This makes the setup and, later on, use of the database much easier.

- **How to set up a database – step by step**

You can set up your own SQLite3 database in Python in a couple of steps. Once you’ve finished setting it up, you will be able to use the database using **standard SQL commands**.

Step 1: Import SQLite3 into Python

So that you have access to the functions that are specific to SQLite3 in Python you will need to import the corresponding module first:

```
python
import sqlite3
```

Step 2: Create the database

The next step is to create your own database by using the SQLite function "connect()". It will give you a connection object, from which you can connect to your database. To open connect and create a connection to the database "test", you will need the following:

python

```
connection = sqlite3.connect("test.db")
```

The transfer parameter of the function is a database file. If you still haven't created a database file called "test.db" this will be done automatically by using the connect function.

Step 3: Check whether the database was successfully created

At this point you can check whether your SQLite3 database has been successfully created. To do so you can call up the "total_changes" function on the connection object you just created. It will show you how many table rows in your relational database have been changed since you connected to the database.

python

```
print(connection.total_changes)
```

In this case the value would be "0" since we've not yet used any SQL commands and, therefore, have an empty database. In case you've already got data in your database, the value you see may be different.

Step 4: Create the basic framework of your database

To add data to your SQLite3 database in Python, you will now need to create a table to store your data, as is the norm with [relational databases](#).

To do this you will first need to create a cursor object with the pre-defined "cursor" function using SQLite3 in Python. This will allow you to send SQL commands to your database. The code you need to do this looks as follows:

python

```
cursor = connection.cursor()
```

You can then use the "execute" function from the SQLite3 module to send SQL commands to your database. The function takes your commands, which follow standard SQL syntax, as strings. So, if you want to create a database with the name "example" and the columns "ID", "name", and "age", then the code you use in Python will look as follows:

python

```
cursor.execute("CREATE TABLE IF NOT EXISTS example (id INTEGER, name TEXT, age IN"
```

Step 5: Add data

You will need the same syntax as you used to create a table to add data to a table:

python

```
cursor.execute("INSERT INTO example VALUES (1, alice, 20)")
cursor.execute("INSERT INTO example VALUES (2, 'bob', 30)")
cursor.execute("INSERT INTO example VALUES (3, 'eve', 40)")
```

Using the code above you will have added three entries to your database table "example".

To save the changes to your database you need to use the "commit" function.

python

```
connection.commit()
```

- **Read the data**

Of course, you can also **read and output data from your databases** instead of just using SQLite3 in Python to create them. To do this, you need to first connect to the database. From there you can create a connection and cursor object as explained in the step-by-step guide above. Finally you can formulate your SQL request, send it to the database using the "execute" function and use the "fetch_all" function to display the results:

python

```
cursor.execute("SELECT * FROM example")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

The "fetch_all" function will give you a list of rows which correspond to your request. To output these rows on the console, you can use a [Python for loop](#) alongside a print statement.

- **Modify existing data**

Since SQLite3 supports SQL commands, you can also **modify or delete** data within your database. To do so you just need to use the corresponding SQL command in your database. To start, this step also requires you to create a connection to your database followed by a connection and cursor object.

- **Delete data**

To delete the row with the ID 1, you can use the following code:

python

```
cursor.execute("DELETE FROM example WHERE id = 1")
```

- **Use a place holder**

If you want to use dynamic SQL requests in your Python program, you should never use [Python string](#) operations. Hackers can use them to attack your program with [SQL injections](#). Instead you should use the **SQL place holder** "?" which you can use with the SQL request you sent using "execute" instead of a specific value. You can also enter a second parameter as a [Python tuple](#) to the "execute" function, which will be replaced by the question mark.

The following requests are the same:

python

```
# Direct SQL requests
cursor.execute("UPDATE example SET age = 31 WHERE id = 2")
# SQL requests with place holders
age_var = 31
id_var = 2
cursor.execute("UPDATE example SET age = ? WHERE id = ?", (age_var, id_var))
```

Be aware that the question marks in the SQL request will be replaced in the order in which you set the variables in the tuple.

- **End the connection to your database**

After you have carried out all your tasks in your database, you will now need to end your connection. You can again use the SQLite3 module in Python to do so by opening it directly in your connection object:

python

```
connection.close()
```

- **Exercise:**

1. **Create and Insert into Employee Table:**

- Write a Python script that creates a table called `employees` with the following columns: `id` (INTEGER, primary key), `name` (TEXT), and `salary` (REAL). Insert 3 records into this table. Print the records after insertion.
- **Desired Output Example:**



A terminal window with a dark background. The title bar says 'bash' and there is a 'Copy code' button. The output shows a table with three columns: 'id', 'name', and 'salary'. The data rows are: 1 Alice 50000, 2 Bob 60000, and 3 Charlie 55000.

id	name	salary
1	Alice	50000
2	Bob	60000
3	Charlie	55000

```
import sqlite3

connection = sqlite3.connect("test.db")

cursor.execute("CREATE TABLE employees(id Integer , name text , salary text)")

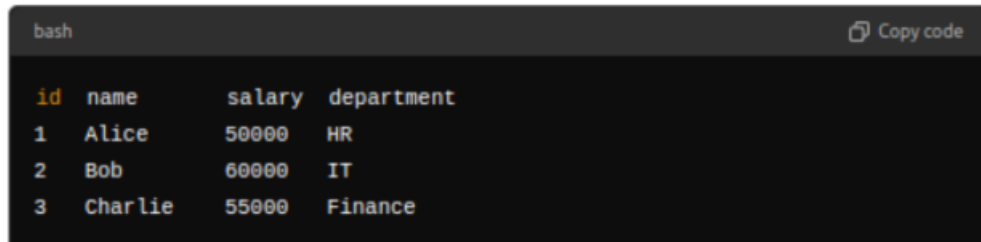
cursor.execute("INSERT INTO employees VALUES (1,'Alice','50000')")
cursor.execute("INSERT INTO employees VALUES (2,'Bob','60000')")
cursor.execute("INSERT INTO employees VALUES (3,'Charlie','55000')")

connection.commit()

cursor.execute("SELECT * FROM employees")
rows = cursor.fetchall()
for row in rows :
    print(row)
```

2. Add a Column to an Existing Table:

- Extend the previous script to add a new column called `department` (TEXT) to the `employees` table. Update each record to include a department name and then print the updated records.
- **Desired Output Example:**



id	name	salary	department
1	Alice	50000	HR
2	Bob	60000	IT
3	Charlie	55000	Finance

```
cursor.execute("ALTER TABLE employees ADD department")
cursor.execute("UPDATE employees SET department = 'HR' WHERE id=1")
cursor.execute("UPDATE employees SET department = 'IT' WHERE id=2")
cursor.execute("UPDATE employees SET department = 'Finance' WHERE id=3")
```

3. Drop and Recreate a Table:

- Write a Python script that checks if a table called `projects` exists. If it does, drop the table. Then create a new `projects` table with columns `project_id` (INTEGER, primary key) and `project_name` (TEXT). Insert 2 sample records and print them.
- **Desired Output Example:**



project_id	project_name
1	Project A
2	Project B

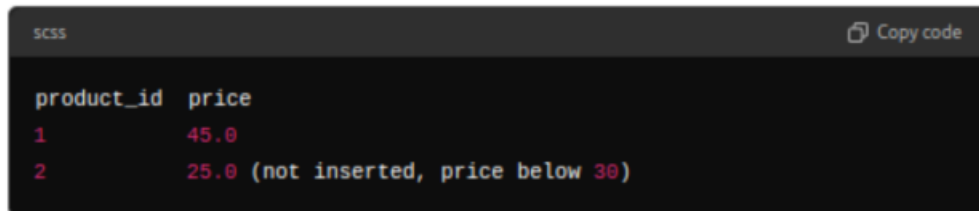
```
cursor.execute("DROP TABLE IF EXISTS projects")

cursor.execute("CREATE TABLE projects(project_id integer, project_name text)")
cursor.execute("INSERT INTO projects VALUES (1,'Project A')")
cursor.execute("INSERT INTO projects VALUES (2,'Project B')")
connection.commit()

cursor.execute("SELECT * FROM projects")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

4. Conditional Insertion into a Table:

- Create a table named `products` with `product_id` (INTEGER, primary key) and `price` (REAL). Write a Python script that inserts a record only if the `price` is above 30. Print the table contents after insertion.
- **Desired Output Example:**



product_id	price
1	45.0
2	25.0 (not inserted, price below 30)

```
cursor.execute("DROP TABLE IF EXISTS products")
cursor.execute("DELETE FROM products")

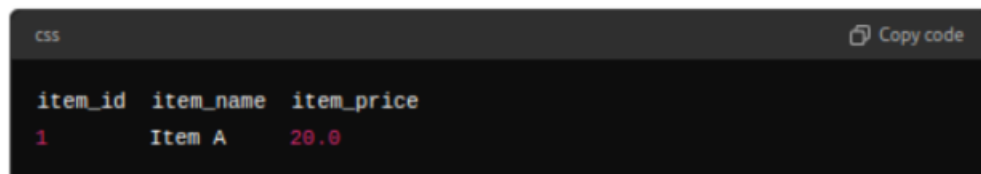
cursor.execute("CREATE TABLE products (product_id integer , price real )")

entries = [[1,30],[2,25],[3,35],[4,40],[5,20],[6,50]]
for [i,j] in entries:

    if j > 30 :
        cursor.execute("INSERT INTO products VALUES (?,?) ",(i,j))
connection.commit()
cursor.execute("SELECT * FROM products")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

5. Dynamic Table Creation:

- Write a Python script that creates a table named `items` with `item_id` (INTEGER, primary key) and `item_name` (TEXT). Allow the user to add a new column named `item_price` (REAL) and then insert a sample record. Print the table.
- **Desired Output Example:**



item_id	item_name	item_price
1	Item A	20.0

```
cursor.execute("CREATE TABLE items(item_id INTEGER , item_name TEXT)")
col = input("Enter column name : ")
typ = input("Enter data type : ")

cursor.execute(f"ALTER TABLE items ADD COLUMN {col} {typ}")
```

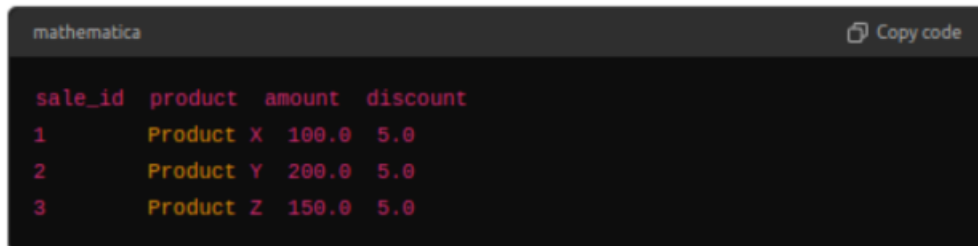
```
connection.commit()

cursor.execute("INSERT INTO items VALUES (1,'ITEM A',20)")
connection.commit()

cursor.execute("SELECT * FROM items")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

6. Update Existing Records:

- Create a table `sales` with columns `sale_id` (INTEGER, primary key), `product` (TEXT), and `amount` (REAL). After inserting 3 records, add a new column `discount` (REAL). Update all records to include a discount of 5.0 and print the table.
- **Desired Output Example:**

A screenshot of a terminal window with a dark background. The title bar says "mathematica" and there is a "Copy code" button. The terminal displays a table with four columns: sale_id, product, amount, and discount. There are three rows of data.

sale_id	product	amount	discount
1	Product X	100.0	5.0
2	Product Y	200.0	5.0
3	Product Z	150.0	5.0

```
cursor.execute("CREATE TABLE sales (sale_id INTEGER, product TEXT, amount REAL)")
cursor.execute("INSERT INTO sales VALUES (1,'product x',100)")
cursor.execute("INSERT INTO sales VALUES (2,'product y',200)")
cursor.execute("INSERT INTO sales VALUES (3,'product z',150)")

connection.commit()

cursor.execute("ALTER TABLE sales ADD discount")
connection.commit()

cursor.execute("UPDATE sales SET discount = '5'")
connection.commit()

cursor.execute("SELECT * FROM sales")
rows = cursor.fetchall()
for row in rows:
    print(row)
```


10. Basic Data Insertion and Query:

- Create a table `users` with `user_id` (INTEGER, primary key) and `username` (TEXT). Insert 2 records and write a query to display all records.
- Desired Output Example:



user_id	username
1	User1
2	User2

```
cursor.execute("CREATE TABLE users (user_id INTEGER, username TEXT)")
cursor.execute("INSERT INTO users VALUES (1,'user1')")
cursor.execute("INSERT INTO users VALUES (2,'user2')")

connection.commit()

cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
for row in rows :
    print(row)
```

• Creating GUI for an Inventory Summary using SQL and Database

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import font as tkfont
import sqlite3

connection = sqlite3.connect("Inventory_Table.db")

cursor = connection.cursor()

screen = Tk()

h = screen.winfo_screenheight() # 864
w = screen.winfo_screenwidth() # 1536

global count
count = 1

global sum
sum = 0.00

# cursor.execute("CREATE TABLE database(ID INTEGER , item_name TEXT , quantity INTEGER , price REAL)")
# connection.commit()

def delete_record():
    new2 = Tk()
    new2.geometry("300x300")
```

```
def delete():
    id_d = e7.get()
    cursor.execute("DELETE FROM database WHERE ID = (?)", (id_d))
    connection.commit()
```

```
l11 = Label(new2, text=" item ID ", font=lb_font)
l11.place(x=20, y=20)
```

```
e7 = Entry(new2, font=en_font)
e7.place(x=150, y=20)
```

```
b7 = Button(new2, font = bt_font, command = delete)
b7.place(x=100, y=250)
```

```
def update_record():
    new = Tk()
    new.geometry("400x400")
```

```
def update():
    id_u = e6.get()
    price_u = e7.get()
    cursor.execute("UPDATE database SET price = (?) WHERE ID = (?)", (price_u, id_u))
    connection.commit()
```

```
l10 = Label(new, text = " item ID ", font=lb_font)
l10.place(x=20, y=20)
```

```
e6 = Entry(new, font=en_font)
e6.place(x=150, y=20)
```

```
l11 = Label(new, text=" New Price ", font=lb_font)
l11.place(x=20, y=70)
```

```
e7 = Entry(new, font = en_font)
e7.place(x=150, y=70)
```

```
b7 = Button(new, text = "Update", font = bt_font, command=update)
b7.place(x=100, y=150)
```

```
def clear_output():
    global sum
    sum = 0.00
    l9.config(text = str(sum))
```

```
def price_sum():
    id = e4.get()
    quant = e5.get()

    global sum

    cursor.execute("SELECT * FROM database WHERE ID = (?)", (id))
    entry = cursor.fetchall()
    sum = sum + int(quant) * int(entry[0][3])
```

```
# l9 = Label(screen, text = str(sum), font = lb_font)
# l9.place(x=500, y=550)
```

```

l9.config(text=str(sum))

def add_to_database():
    name = e1.get()
    quantity = e2.get()
    price = e3.get()
    global count
    cursor.execute("INSERT INTO database VALUES (?,?,?)",(count,name,quantity,price))
    count+=1

def show_records():
    cursor.execute("SELECT * FROM database")
    rows = cursor.fetchall()
    tmp = ""
    for data in rows :
        tmp = tmp + str(data[1]) + " , " + str(data[2]) + " items " + ' , ' + " $$ " + str(data[3]) + " , " + "ID : " +
str(data[0]) + "\n"
    l8 = Label(screen, text=tmp, font=lb_font)
    l8.place(x=500,y=300)

screen.geometry(f"{h}x{h}")

img = Image.open(r"E:\google_images\inventory_image")
img = img.resize((h,h))
img_t = ImageTk.PhotoImage(img)

custom_f = tkfont.Font(family="",size = 20)

lb_font = ("Arial",16)
en_font=("Arial",16)
bt_font = ("Arial",12)

cursor.execute("DELETE FROM database")
connection.commit()

l1 = Label(screen,image = img_t)
l1.place(x=0,y=0)

l2_font = ("Arial",18,"bold")
l2 = Label(screen,text = " INVENTORY SUMMARY ",font = l2_font,background = "pink")
l2.place(x=280,y=20)

l3 = Label(screen,text = " Name ",font = lb_font)
l3.place(x=200,y=100)

e1 = Entry(screen,font = en_font)
e1.place(x=500,y=100)

l4 = Label(screen,text = " Quantity ",font = lb_font)
l4.place(x=200,y=150)

e2 = Entry(screen,font = en_font)
e2.place(x=500,y=150)

l5 = Label(screen,text = " Price($) ",font = lb_font)
l5.place(x=200,y=200)

e3 = Entry(screen,font = en_font)
e3.place(x=500,y=200)

```

```

b1 = Button(screen,text = "Add Record To Database", background = "light blue",font = bt_font, command =
add_to_database )
b1.place(x=200,y=250)

b2 = Button(screen,text = " Show Records ", background = "light blue",font = bt_font,command = show_records)
b2.place(x=200,y=300)

l6 = Label(screen,text = 'Select ID' , font = lb_font)
l6.place(x = 200,y = 450)

e4 = Entry(screen,font = en_font)
e4.place(x=500,y=450)

l7 = Label(screen,text = ' Quantity for price sum ' , font = lb_font)
l7.place(x = 200,y = 500)

e5 = Entry(screen,font = en_font)
e5.place(x=500,y=500)

b3 = Button(screen,text = " Calculate Price Sum " , font = bt_font,background="light blue",command= price_sum)
b3.place(x=200,y=550)

b4 = Button(screen,text = " Clear output " , font = bt_font,background="light blue",command = clear_output )
b4.place(x=200,y=600)

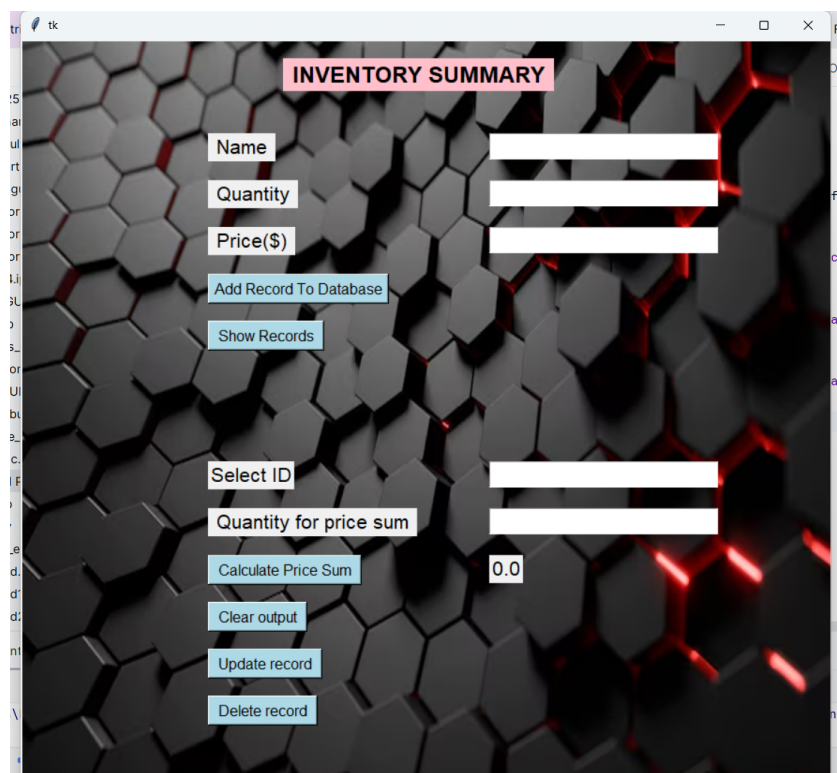
b5 = Button(screen,text = " Update record " , font = bt_font,background="light blue",command = update_record )
b5.place(x=200,y=650)

b6 = Button(screen,text = " Delete record " , font = bt_font,background="light blue",command = delete_record)
b6.place(x=200,y=700)

l9 = Label(screen, text=str(sum), font=lb_font)
l9.place(x=500, y=550)

screen.mainloop()

```



tk

INVENTORY SUMMARY

Name	shoes
Quantity	100
Price(\$)	500

Add Record To Database

Show Records

bags , 400 items , \$\$ 200.0 , ID : 1
shoes , 100 items , \$\$ 500.0 , ID : 2

Select ID

2

Quantity for price sum

4

Calculate Price Sum

2000.0

Clear output

Update record

Delete record