Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

Learning Objectives:

Database operations on SQLite

– Session 9 , 20 May 2023

Dr N V Ganapathi Raju
Professor and HOD of IT
Gokaraju Rangaraju Institute of Eng and Tech

© Replication or other unauthorized use of this material is prohibited

SQLite database

- SQLite is the most widely deployed SQL database engine in the world.
    - SQLite does not require a separate server process or system to operate (serverless).
    - SQLite comes with zero-configuration, which means no setup or administration needed.
    - A complete SQLite database is stored in a single cross-platform disk file.
    - SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
    - SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
    - Supports in all O.S, written in ANSI-C and provides simple and easy-to-use API.

© Replication or other unauthorized use of this material is prohibited

How to get start with databases

- Databases require more defined structure than Python lists or dictionaries.

- When we create a database *table,* we must tell the database in advance the names of each of the *columns* in the table and the type of data which we are planning to store in each *column*.

- When the database software knows the type of data in each column, it can choose the most efficient way to store and look up the data based on the type of data.

© Replication or other unauthorized use of this material is prohibited

How to work with SQLite

- **SQLite database is in-built in Python** using **sqlite3 module**
- When we connect to an SQLite database file that does not exist, SQLite automatically creates the new database for us.
- To create a database, first, we must create a Connection object that represents the database using the connect() function of the sqlite3 module.

```
import sqlite3
conn = sqlite3.connect('emp.db')
print ("Opened database successfully")
```

© Replication or other unauthorized use of this material is prohibited

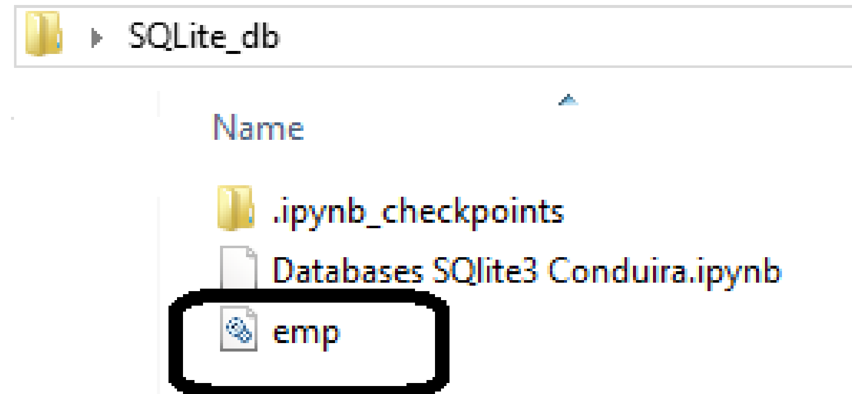
Steps to work with SQLite

Note: We must create a folder first,

before we execute the program. or

```
import sqlite3
conn = sqlite3.connect('emp.db')
print ("Opened database successfully")
```

- **we can place the database file a folder of our choice.**
- **The connect() function opens a connection to an SQLite database. It returns a Connection object that represents the database. By using the Connection object, you can perform various database operations.**



© Replication or other unauthorized use of this material is prohibited

Steps to work with SQLite

Note:

- If we skip the folder path C:\Users\PERSONAL\Desktop\SQLite_db, the program will create the database file in the current working directory (CWD).
- If we pass the file name as **:memory:** to the connect() function of the sqlite3 module, it will create a new database that resides in the memory (RAM) instead of a database file on disk.

```
import sqlite3

conn = sqlite3.connect(':memory:')

print ("Opened database successfully")
```

© Replication or other unauthorized use of this material is prohibited

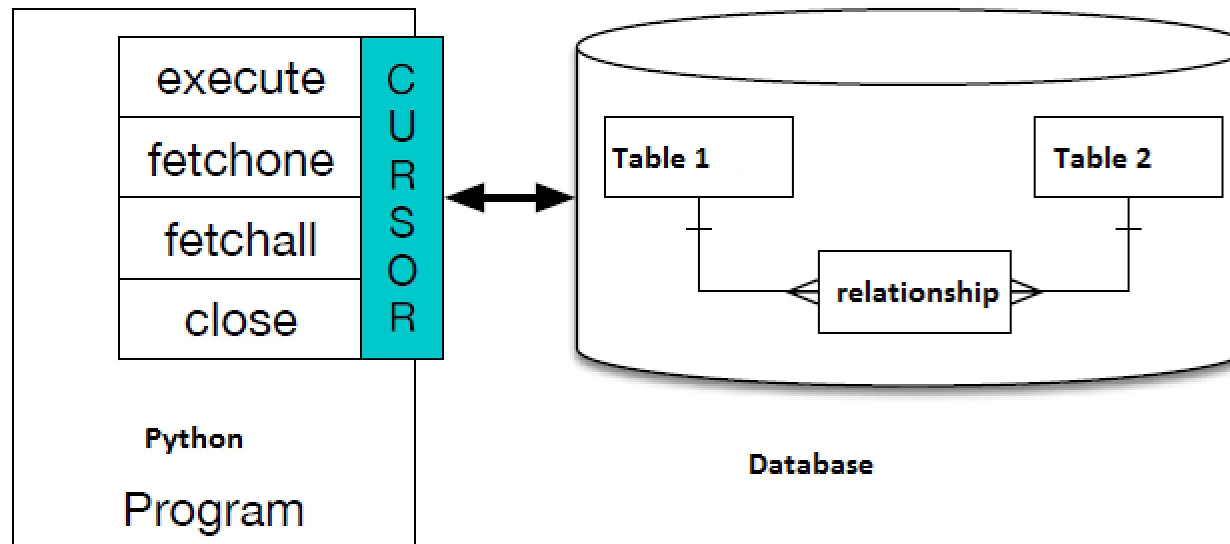
Creating Tables in SQLite

- To create a new table in an SQLite database from a Python program, you use the following steps:
  - First, create a **Connection** object using the **connect()** method of the sqlite3 module.
  - Second, create a **Cursor object** by calling the **cursor()** method of the Connection object.
  - Third, pass the **CREATE TABLE** statement to the **execute**() method of the Cursor object and execute this method.

© Replication or other unauthorized use of this material is prohibited

Database cursors

- A *cursor* is like a file handle that we can use to perform operations on the data stored in the database.
- Calling cursor() is very similar conceptually to calling open() when dealing with text files.



- Once we have the cursor, we can begin to execute commands on the contents of the database using the execute() method.

© Replication or other unauthorized use of this material is prohibited

Creating Tables in SQLite

```
import sqlite3

conn = sqlite3.connect('emp.db')
print ("Opened database successfully")

conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print ("Table created successfully")

conn.close()
```

```
import sqlite3

conn = sqlite3.connect('emp.db')
print ("Opened database successfully")

conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print ("Table created successfully")

conn.close()
```

```
Opened database successfully
Table created successfully
```

© Replication or other unauthorized use of this material is prohibited

CURD operations using SQLite

- CURD stands for CREATE, UPDATE, RETRIEVE, DELETE operations.
    - Create – Insert operation
    - Update- Update operation
    - Retrieve – Select operation
    - Delete – Delete operation

© Replication or other unauthorized use of this material is prohibited

Insert data into SQLite

```
import sqlite3

conn = sqlite3.connect('emp.db')
print ("Opened database successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print ("Records created successfully")
conn.close()
```

```
Opened database successfully
Records created successfully
```

© Replication or other unauthorized use of this material is prohibited

Select data from SQLite

```
import sqlite3

conn = sqlite3.connect('emp.db')
print ("Opened database successfully")

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")
conn.close()
```

```
Opened database successfully
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0
..
..

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

© Replication or other unauthorized use of this material is prohibited

Cursor methods

- cursor.fetchall() fetches all the rows of a query result.
    - It returns all the rows as a list of tuples.
    - An empty list is returned if there is no record to fetch.
- cursor.fetchmany(size) returns the number of rows specified by size argument.
    - When called repeatedly this method fetches the next set of rows of a query result and returns a list of tuples.
    - If no more rows are available, it returns an empty list.
- cursor.fetchone() method returns a single record or None if no more rows are available.

© Replication or other unauthorized use of this material is prohibited

Select data from SQLite

```
import sqlite3 as lite

con = lite.connect('emp.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
import sqlite3 as lite

con = lite.connect('emp.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(1, 'Paul', 32, 'California', 25000.0)
(2, 'Allen', 25, 'Texas', 15000.0)
(3, 'Teddy', 23, 'Norway', 20000.0)
(4, 'Mark', 25, 'Rich-Mond ', 65000.0)
```

© Replication or other unauthorized use of this material is prohibited

Retrieving one record at a time

```
import sqlite3 as lite

con = lite.connect('emp.db')

with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY")

    while True:
        row = cur.fetchone()

        if row == None:
            break

        print (row[0], row[1], row[2])
```

```
import sqlite3 as lite

con = lite.connect('emp.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY")

    while True:

        row = cur.fetchone()

        if row == None:
            break

        print (row[0], row[1], row[2])
```

```
1 Paul 32
2 Allen 25
3 Teddy 23
4 Mark 25
```

© Replication or other unauthorized use of this material is prohibited

Update SQLite data

```
import sqlite3

conn = sqlite3.connect('emp.db')

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit()
print ("Total number of rows updated :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

conn.close()
```

```
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0
```

© Replication or other unauthorized use of this material is prohibited

Deleting data from SQLite

```
import sqlite3

conn = sqlite3.connect('emp.db')
print ("Opened database successfully")

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print ("Total number of rows deleted :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")
conn.close()
```

```
Opened database successfully
Total number of rows deleted : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

© Replication or other unauthorized use of this material is prohibited

executescript() - executing multiple SQL statements

- This is a nonstandard convenience method for executing multiple SQL statements at once. It issues a COMMIT statement first, then executes the SQL script it gets as a parameter.

```
import sqlite3
con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.executescript("""
    create table samples(id,value);
    insert into samples(id, value)  values ('123','abcdef');
    """)
cur.execute("SELECT * from samples")
print (cur.fetchone())
```

```
import sqlite3

con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.executescript("""
    create table samples(id,value);
    insert into samples(id, value)
    values ('123','abcdef');
    """)
cur.execute("SELECT * from samples")
print (cur.fetchone())
```

```
('123', 'abcdef')
```

© Replication or other unauthorized use of this material is prohibited

Parameterized queries

- A parameterized query is a **query in which placeholders used for parameters and the parameter values supplied at execution time**.

- That means parameterized query gets compiled only once.

- **There are the main 4 reasons to use**

  - Improves Speed: If you want to execute SQL statement/query many times, it usually reduces execution time

  - Compile Once: The main advantage of using a parameterized query is that parameterized query compiled only once

  - Same Operation with Different Data: if you want to execute the same query multiple times with different data.

  - Preventing SQL injection attacks.

© Replication or other unauthorized use of this material is prohibited

Parameterized queries

```
import sqlite3
conn = sqlite3.connect('LanguageDB')
cur = conn.cursor()
cur.execute('DROP TABLE IF EXISTS languages')
cur.execute('CREATE TABLE languages (subject TEXT, marks INTEGER)')

cur.execute('INSERT INTO languages (subject, marks) VALUES (?, ?)',  ('C', 100))
cur.execute('INSERT INTO languages (subject, marks) VALUES (?, ?)',  ('Java', 200))
cur.execute('INSERT INTO languages (subject, marks) VALUES (?, ?)',  ('Python', 300))
conn.commit()

print('Languages:')
cur.execute('SELECT subject, marks FROM languages')
for row in cur:
    print(row)
cur.close()
```

```
Languages:
('C', 100)
('Java', 200)
('Python', 300)
```

© Replication or other unauthorized use of this material is prohibited

Consider database as emp.db

emp10.db

(1, 'Paul', 32, 'California', 20000.0)

(2, 'Allen', 25, 'Texas', 15000.0)

(3, 'Teddy', 23, 'Norway', 20000.0)

(4, 'Mark', 25, 'Rich-Mond ', 65000.0)

© Replication or other unauthorized use of this material is prohibited

select with where clause

```
import sqlite3 as lite

con = lite.connect('emp10.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY where ID=1")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(1, 'Paul', 32, 'California', 20000.0)
```

© Replication or other unauthorized use of this material is prohibited

select with where clause

```
import sqlite3 as lite

con = lite.connect('emp0.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY where NAME='Paul'")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(1, 'Paul', 32, 'California', 25000.0)
```

© Replication or other unauthorized use of this material is prohibited

select with where clause

```
import sqlite3 as lite

con = lite.connect('emp10.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY where SALARY >=30000")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(4, 'Mark', 25, 'Rich-Mond ', 65000.0)
```

© Replication or other unauthorized use of this material is prohibited

select with *order by* clause

```
import sqlite3 as lite

con = lite.connect('emp10.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY ORDER BY  SALARY")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(2, 'Allen', 25, 'Texas', 15000.0)
(1, 'Paul', 32, 'California', 20000.0)
(3, 'Teddy', 23, 'Norway', 20000.0)
(4, 'Mark', 25, 'Rich-Mond ', 65000.0)
```

© Replication or other unauthorized use of this material is prohibited

select with where clause (like operator)

```
import sqlite3 as lite

con = lite.connect('emp10.db')

with con:


    cur = con.cursor()
    cur.execute("SELECT * FROM COMPANY WHERE NAME LIKE 'a%'")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(2, 'Allen', 25, 'Texas', 15000.0)
```

© Replication or other unauthorized use of this material is prohibited

select with where clause (max())

```
import sqlite3 as lite

con = lite.connect('emp10.db')

with con:


    cur = con.cursor()
    cur.execute("SELECT MAX(salary) FROM COMPANY")

    rows = cur.fetchall()

    for row in rows:
        print (row)
```

```
(65000.0,)
```

©Replication or other unauthorized use of this material is prohibited