

Practical No 11

Aim: To install and configure a database connector in Python and establish a connection to a relational database system (e.g. SQLite).

Theory:

Database connectivity allows Python programs to interact with databases — to store, retrieve, update, and manage data efficiently.

Through database connectors (like SQLite or MySQL Connector), Python can execute SQL commands directly within a program.

Python provides a built-in module called sqlite3 that enables smooth interaction with SQLite, a lightweight, serverless, and zero-configuration relational database management system (RDBMS).

Database connectivity refers to the process of connecting an application (Python program) to a database so that the program can:

- Create and manage tables
- Insert, modify, and delete data
- Retrieve and display records

This connection acts as a **bridge** between the Python application and the database engine.

Installing and Configuring the Database Connector

- SQLite comes pre-installed with Python.
- To use it, simply import the module:

```
import sqlite3
```

Establish connection

```
conn=sqlite3.connect("college.db")
```

- If the database file doesn't exist, SQLite automatically creates it.

Creating a Cursor Object:

- A cursor object is used to execute SQL commands and fetch results. You create one from the connection object:

```
cursor = conn.cursor()
```

Executing SQL Commands:

- Execute SQL commands using the cursor.execute() method. This can be for creating tables, inserting data, updating, deleting, or querying.

```
# Create a table
```

```
cursor.execute("")
```

```

CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER
)
"")

# Insert data

cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Alice", 30))

```

Retrieving Data:

```

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

```

Closing the Connection:

- It's crucial to close the database connection when you're finished to release resources:

```
conn.close()
```



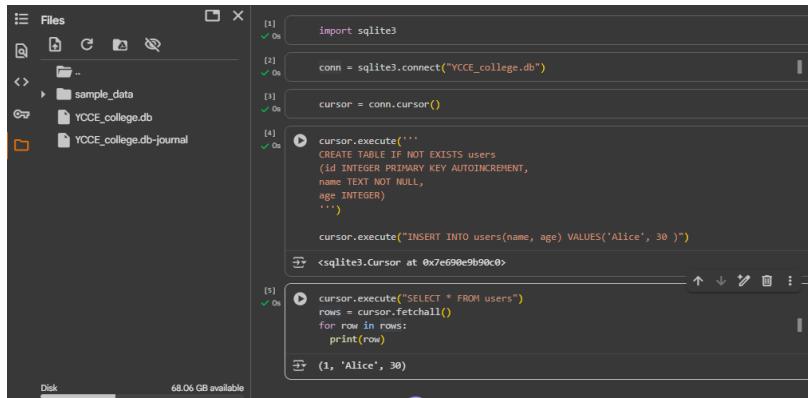
The screenshot shows a code editor with a sidebar displaying file navigation. The main area contains Python code for interacting with a SQLite database. The code includes creating a table named 'users' with columns 'id', 'name', and 'age', inserting a row for 'Alice' at age 30, and then selecting all rows from the table to print them. The code is numbered from [35] to [22]. A status bar at the bottom indicates the cursor is at address 0x789607376ac0.

```

import sqlite3
conn=sqlite3.connect("YCCE_college.db")
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER
)
''')
cursor.execute("INSERT INTO users (name, age) VALUES('Alice', 30)")
<sqlite3.Cursor at 0x789607376ac0>
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
for row in rows:
    print(row)
(1, 'Alice', 30)
conn.close()

```

Output:



The screenshot shows a terminal window with a file browser on the left. The terminal has five numbered lines of code:

```
[1] import sqlite3
[2] conn = sqlite3.connect("YCCE_college.db")
[3] cursor = conn.cursor()
[4]
[5]
```

Line 4 contains a multi-line SQL command to create a table named 'users' if it does not exist, with columns 'id' (INTEGER PRIMARY KEY AUTOINCREMENT), 'name' (TEXT NOT NULL), and 'age' (INTEGER). Line 5 shows the insertion of a row ('Alice', 30) into the 'users' table. Line 6 shows the execution of a SELECT query to retrieve all rows from the 'users' table. The output of the SELECT query is shown in line 7, displaying the row (1, 'Alice', 30).

```
CREATE TABLE IF NOT EXISTS users
(id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL,
age INTEGER)
```
cursor.execute("INSERT INTO users(name, age) VALUES('Alice', 30)")
<sqlite3.Cursor at 0x7e690e9b90c0>
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
for row in rows:
 print(row)
(1, 'Alice', 30)
```