# 🗃️ Database Connectivity in Python (SQLite3 & PyMySQL)

## 🔹 Introduction

Database connectivity allows Python applications to **store, manage, and retrieve data** efficiently using structured query language (SQL).
Python provides powerful libraries like **SQLite3** and **PyMySQL** that make database operations simple and consistent, regardless of whether you're working with a **local file-based database** (SQLite) or a **remote server-based database** (MySQL).

---

## 🧩 SQLite3 — Lightweight Embedded Database

**SQLite3** is a built-in Python module used for working with **SQLite databases**, which are **serverless, self-contained, and zero-configuration**.
It is ideal for smaller applications, local storage, or development testing because it stores data in a **single .db file** on disk.

**Key Features:**

- No installation or server setup required (built into Python).
- Data stored locally in a file — perfect for testing or standalone applications.
- Supports all standard SQL commands like CREATE, INSERT, UPDATE, DELETE, and SELECT.
- Fast and secure with minimal resource usage.

**Example Workflow:**

1. Import the sqlite3 module.
2. Connect to a database (creates the file if it doesn't exist).
3. Create a cursor object to execute SQL queries.
4. Use SQL statements to create tables or manipulate data.
5. Commit changes and close the connection.

---

## 🧩 PyMySQL — MySQL Database Connector for Python

**PyMySQL** is a third-party Python library that allows connection to a **MySQL database server** using Python code.
It is used for projects where a **multi-user environment** or **large-scale data storage** is required.

**Key Features:**

- Connects to remote or local MySQL servers using credentials (host, user, password, database name).
- Fully supports SQL operations (DDL, DML, and DQL).
- Enables Python to interact with enterprise-level MySQL databases used in web apps or data systems.
- Provides exception handling and secure transactions.

**Example Workflow:**

1. Install the library using pip install pymysql.
2. Import the module and connect to the MySQL server using credentials.
3. Create a cursor object to execute SQL statements.
4. Execute queries (e.g., INSERT, UPDATE, SELECT).
5. Commit changes and close the connection to prevent data loss.

---

## ⚙️ Executing SQL Queries in Python

Python acts as a **bridge** between the user and the database through these connectors.
After establishing a connection:

- SQL queries are written as **strings** inside Python code.
- These queries are executed using a **cursor object**.
- The results are fetched, displayed, or stored in Python variables for further use.

**Example of Basic SQL Operations:**

| **Operation** | **SQL Command** | **Python Method** |
|---|---|---|
| Create Table | CREATE TABLE students(...) | cursor.execute() |
| Insert Data | INSERT INTO students VALUES(...) | cursor.execute() |
| Fetch Data | SELECT * FROM students | cursor.fetchall() |
| Commit Changes | - | connection.commit() |
| Close Connection | - | connection.close() |

---

## 🔐 **Importance of Using Connectors**

Using SQLite3 or PyMySQL allows Python developers to:

- Automate database operations.
- Reduce manual SQL work.
- Integrate back-end databases into real-world applications.
- Maintain data integrity and consistency.
- Support scalability — from small local apps to enterprise-level solutions.