📘 PYTHON DATABASE CONNECTIVITY (PDBC) — NOTES

⭐ 1. What is PDBC?

PDBC stands for **Python Database Connectivity**.
It is the process of connecting Python programs with a database so we can perform:

- **C**reate → Insert data
- **R**ead → Fetch data
- **U**pdate → Modify data
- **D**elete → Remove data

With PDBC, Python can communicate with databases like **MySQL, MongoDB, PostgreSQL, SQLite**, etc.

⭐ 2. What is PyMySQL?

PyMySQL is a **Python library** used to connect Python with **MySQL**.

It allows Python to:

- Connect to MySQL server
- Run SQL queries
- Insert / Update / Delete / Fetch records
- Manage databases and tables

Install once using:

pip install pymysql

⭐ 3. Steps in Python Database Connectivity

PDBC follows **5 major steps**:

STEP 1: Import the Connector

Load PyMySQL into the program.

STEP 2: Connect to Database Server

Create a connection object (conn) using:

- host
- username
- password

STEP 3: Create Cursor

A cursor is an object used to run SQL commands.

STEP 4: Execute SQL Queries

Use SQL to:

- Create databases
- Create tables
- Insert data
- Update data
- Fetch data
- Delete data

STEP 5: Close the Connection

Always close:

- cursor
- connection

To free resources and avoid errors.

⭐ 4. Creating a Database

A database is created using SQL:

CREATE DATABASE IF NOT EXISTS db_name;

This is typically the first step after connecting to MySQL.

⭐ 5. Creating a Table

A table is a structured format inside a database.
It contains:

- columns (name, age, gender…)
- data types (int, varchar…)
- primary key (unique identifier)

For example, a students table often includes:

- id
- name
- age
- gender
- email

Tables must be created **before** inserting data.

⭐ 6. INSERTING DATA

This is the **Create** operation from CRUD.

There are two methods:

✓ Single row insertion

Insert one record at a time.

✓ Multiple row insertion

Insert many records at once using executemany() → faster and efficient.

💡 Parameterized Queries

To avoid SQL injection and errors, always insert using **placeholders** %s.
Python automatically handles data types and quoting.

⭐ 7. FETCHING DATA (READ Operation)

To display or retrieve data from a table, we use:

- SELECT query
- fetchall() or fetchone()

This prints or returns all rows stored in the table.

Fetching is used for:

- displaying all students
- checking inserted records
- showing updated values

⭐ 8. UPDATING RECORDS

Update modifies specific fields of existing rows.

Common update patterns:

- Update email by ID
- Update name by ID
- Update age by ID

Updating needs:

- a WHERE condition
- new value(s) to update

This is the **U** in CRUD.

⭐ 9. DELETING RECORDS

Deleting removes a row permanently.

Usually done based on:

- ID
- email
- any unique field

DELETE query also needs a WHERE condition to avoid deleting all records.

This is the **D** in CRUD.

⭐ 10. SQL Queries Covered Today

✓ CREATE DATABASE

✓ CREATE TABLE

✓ INSERT DATA (single & multiple)

✓ SELECT (fetch)

✓ UPDATE

✓ DELETE

These six SQL commands complete full **CRUD** workflow.

⭐ 11. Using USE database

Before working on tables, we must activate the database using:

USE db_name;

This tells MySQL where to run the upcoming SQL commands.

⭐ 12. Commit & Close

✓ commit()

Saves all changes permanently inside the database.
Required after:

- INSERT
- UPDATE
- DELETE
- CREATE queries

Without commit → changes are not saved.

✓ **cursor.close()**

Closes the SQL executor.
Prevents memory leaks and errors.

✓ **conn.close()**

Closes the MySQL connection.
Always done at the end of the program.

⭐ **13. Menu-Driven Program**

A menu-driven system allows users to choose operations:

1 → Create table

2 → Insert data

3 → Fetch data

4 → Insert multiple data

5 → Update data

6 → Delete data

This is commonly used for:

- student management
- employee management
- product management

It makes the program interactive and modular.

⭐ **14. What Students Should Understand**

By the end of this module, students should understand:

✓ **How to connect Python with MySQL**

✓ **How to create databases and tables**

✓ **How to insert one or many records**

✓ **How to fetch/read table content**

✓ **How to update records using WHERE clause**

✓ **How to delete records safely**

✓ **How CRUD works in real applications**

✓ **How to combine Python + SQL for real backend work**

✓ **Why commit(), close(), use db, cursor, etc. are important**

✓ **How menu-driven programs structure CRUD operations**

⭐ 15. Real-World Use Cases

These concepts are used in:

- Django backend
- Flask backend
- User registration systems
- E-commerce product storage
- Student/employee management apps
- Banking or credit card apps
- Any CRUD-based backend system

Learning PyMySQL CRUD is the foundation for **full-stack development**.

🎯 Final Takeaway

Today's class covered the **entire CRUD lifecycle** using Python + MySQL:

➞ Connecting Python to DB
➞ Creating DB & tables
➞ Inserting single/multiple records
➞ Fetching (reading) records
➞ Updating records

➞ Deleting records

➞ Building menu-driven programs

➞ Closing and committing safely

These are essential backend fundamentals every developer must know.
