PYTHON – MySQL CONNECTIVITY

CBSE Class XII – Informatics Practices (Code 065)

As per Syllabus 2026-27

1. Introduction

Database connectivity means connecting a front-end application (Python) with a back-end database (MySQL) so that data stored in the database can be accessed, inserted, updated, or deleted through the Python program.

- Front End – The application through which the user interacts (here, Python).
- Back End – The database that stores data permanently (here, MySQL).

Why connect Python with MySQL?

- Python provides powerful data-handling and logic-building capability but does not store data permanently.
- MySQL stores data permanently (persistent storage) but has limited processing/logic capability.
- Combining both allows us to build real-world applications (e.g., Student Management System, Library System, Billing System).

2. The Connector Module — mysql.connector

To connect Python with MySQL, we use the mysql.connector module (a Python Database API / DB-API driver).

2.1 Installing the Connector

Run this command on the command prompt/terminal:

```
pip install mysql-connector-python
```

2.2 Importing the Connector

```
import mysql.connector as sqltor
# or
import mysql.connector
```

3. Steps for Python-MySQL Connectivity (Step by Step)

Step 1: Import the module

```
import mysql.connector
```

Step 2: Create a Connection Object

The connect() function establishes a connection with the MySQL server.

```
mycon = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="school"
)
```

Parameter	Meaning
host	Name/IP of the server where MySQL is running (usually "localhost")
user	MySQL username (default: "root")
password	Password set for the MySQL user

database	Name of the database to connect to (optional at this stage)

Check the connection:

```
if mycon.is_connected():
    print("Connection Successful")
else:
    print("Connection could not be established")
```

Step 3: Create a Cursor Object

A cursor is a control structure that enables Python to execute SQL commands and traverse/retrieve records from the result set.

```
mycursor = mycon.cursor()
```

Step 4: Execute SQL Query using execute()

```
mycursor.execute("SELECT * FROM student")
```

Step 5: Fetch the Results (for SELECT queries)

```
data = mycursor.fetchall()
for row in data:
    print(row)
```

Step 6: Commit the Changes (for INSERT/UPDATE/DELETE)

```
mycon.commit()
```

Step 7: Close the Cursor and Connection

```
mycursor.close()
mycon.close()
```

4. Important Cursor Methods

Method	Description
execute(query)	Executes a single SQL statement
executemany(query, data)	Executes the same SQL statement for multiple sets of data (e.g., multiple INSERTs)
fetchone()	Returns the next single row of the result set as a tuple
fetchall()	Returns all remaining rows as a list of tuples
fetchmany(n)	Returns the next n rows as a list of tuples
rowcount	Returns the number of rows affected/retrieved by the last query

5. Connection Object Methods

Method	Description
commit()	Permanently saves the changes made (INSERT/UPDATE/DELETE) to the database

rollback()	Undoes the changes made since the last commit (used in case of error)
close()	Closes the connection with the database
is_connected()	Returns True if connection is active

Note on Commit: MySQL Connector/Python works in auto-commit = False mode by default, i.e., changes made through INSERT, UPDATE, or DELETE are NOT saved permanently unless commit() is called explicitly.

6. Performing CRUD Operations (with Code)

6.1 CREATE (Insert Record)

```
import mysql.connector
mycon = mysql.connector.connect(host="localhost", user="root",
                                password="1234", database="school")
mycursor = mycon.cursor()

sql = "INSERT INTO student (rollno, name, marks) VALUES (%s, %s, %s)"
val = (1, "Amit", 89)
mycursor.execute(sql, val)

mycon.commit()
print(mycursor.rowcount, "record inserted")
mycon.close()
```

6.2 READ (Retrieve Records)

```
mycursor.execute("SELECT * FROM student")

# Using fetchone()
row = mycursor.fetchone()
print(row)

# Using fetchall()
rows = mycursor.fetchall()
for r in rows:
    print(r)

# Using fetchmany()
rows = mycursor.fetchmany(3)
for r in rows:
    print(r)
```

6.3 UPDATE (Modify Record)

```
sql = "UPDATE student SET marks = %s WHERE rollno = %s"
val = (95, 1)
mycursor.execute(sql, val)
mycon.commit()
print(mycursor.rowcount, "record updated")
```

6.4 DELETE (Remove Record)

```
sql = "DELETE FROM student WHERE rollno = %s"
val = (1,)
mycursor.execute(sql, val)
mycon.commit()
print(mycursor.rowcount, "record deleted")
```

6.5 Inserting Multiple Records — executemany()

```
sql = "INSERT INTO student (rollno, name, marks) VALUES (%s, %s, %s)"
val = [(2, "Riya", 78), (3, "Karan", 65), (4, "Neha", 91)]
mycursor.executemany(sql, val)
mycon.commit()
print(mycursor.rowcount, "records inserted")
```

6.6 Taking Input from User and Inserting into Table

```
import mysql.connector
mycon = mysql.connector.connect(host="localhost", user="root",
                                password="1234", database="school")
mycursor = mycon.cursor()

rno = int(input("Enter Roll No: "))
name = input("Enter Name: ")
marks = float(input("Enter Marks: "))

sql = "INSERT INTO student VALUES (%s, %s, %s)"
mycursor.execute(sql, (rno, name, marks))
mycon.commit()
print("Record Inserted Successfully")
mycon.close()
```

7. Fetching Data into a Pandas DataFrame (Integration with Data Handling)

Since the IP syllabus links Pandas with SQL, this is an important extension:

```
import mysql.connector
import pandas as pd

mycon = mysql.connector.connect(host="localhost", user="root",
                                password="1234", database="school")

df = pd.read_sql("SELECT * FROM student", mycon)
print(df)
mycon.close()
```

(Note: read_sql() may show a warning with mysql.connector but works correctly; SQLAlchemy is the officially recommended engine, though mysql.connector is used at school level.)

8. Other Python Libraries for MySQL Connectivity

Apart from mysql.connector, two other important libraries can be used to connect Python with MySQL. Awareness of these is useful for board exams and projects, though mysql.connector remains the primary CBSE-recommended module.

8.1 PyMySQL

PyMySQL is a pure-Python MySQL client library. It works almost identically to mysql.connector in terms of DB-API structure (connection object, cursor object, execute(), fetchall(), commit()), which makes switching between the two very easy.

Installation:

```
pip install pymysql
```

Basic Connectivity using PyMySQL:

```
import pymysql

# Step 1: Create connection
mycon = pymysql.connect(
    host="localhost",
    user="root",
    password="1234",
    database="school"
)

# Step 2: Create cursor
mycursor = mycon.cursor()

# Step 3: Execute query
mycursor.execute("SELECT * FROM student")

# Step 4: Fetch data
data = mycursor.fetchall()
for row in data:
    print(row)

# Step 5: Close connection
mycon.close()
```

Insert Record using PyMySQL:

```
import pymysql

mycon = pymysql.connect(host="localhost", user="root",
                        password="1234", database="school")
mycursor = mycon.cursor()

sql = "INSERT INTO student (rollno, name, marks) VALUES (%s, %s, %s)"
val = (5, "Sanya", 88)
mycursor.execute(sql, val)

mycon.commit()
print(mycursor.rowcount, "record inserted")
mycon.close()
```

Key Points on PyMySQL:

- Written entirely in Python (no C extension dependency), so it is easy to install across platforms.
- Follows the Python DB-API 2.0 specification, same as mysql.connector.
- Method names are the same: connect(), cursor(), execute(), executemany(), fetchone(), fetchall(), fetchmany(), commit(), rollback(), close().

- Commonly used with Django ORM and other Python web frameworks as an alternative connector.

Difference between mysql.connector and PyMySQL:

Basis	mysql.connector	PyMySQL
Developed by	Oracle (official driver)	Community-driven, pure-Python
Installation	pip install mysql-connector-python	pip install pymysql
Dependency	Can use C extension for speed	Pure Python, no C dependency
CBSE Recommendation	Primary/most common in CBSE textbooks	Alternative, functionally similar
Syntax	mysql.connector.connect()	pymysql.connect()

8.2 SQLAlchemy

SQLAlchemy is a powerful Python SQL toolkit and Object Relational Mapper (ORM). Unlike mysql.connector or pymysql, which let you write raw SQL queries directly, SQLAlchemy allows two ways of working with databases:

1. Core (Engine-based) — write SQL through a Python expression layer or raw SQL strings via an "engine".
2. ORM (Object Relational Mapping) — represent database tables as Python classes and rows as Python objects.

Why use SQLAlchemy?

- Provides a database-agnostic interface — the same code can work with MySQL, SQLite, PostgreSQL, etc., with minimal changes.
- Officially recommended for use with pandas.read_sql() (avoids the warning shown when using mysql.connector directly with Pandas).
- Useful for larger, more maintainable applications and projects.

Installation:

```
pip install sqlalchemy
pip install pymysql
```

(SQLAlchemy needs an underlying driver like pymysql or mysql-connector-python to actually talk to MySQL.)

Basic Connectivity using SQLAlchemy (Core/Engine approach):

```
from sqlalchemy import create_engine, text

# Step 1: Create an engine
# format: dialect+driver://user:password@host/database
engine = create_engine("mysql+pymysql://root:1234@localhost/school")

# Step 2: Connect and execute a query
with engine.connect() as conn:
    result = conn.execute(text("SELECT * FROM student"))
    for row in result:
        print(row)
```

Insert Record using SQLAlchemy:

```
from sqlalchemy import create_engine, text

engine = create_engine("mysql+pymysql://root:1234@localhost/school")

with engine.connect() as conn:
    conn.execute(
        text("INSERT INTO student (rollno, name, marks) VALUES
(:rno, :nm, :mk)"),
```

```
        {"rno": 6, "nm": "Vivek", "mk": 74}
    )
    conn.commit()
    print("Record Inserted Successfully")
```

Using SQLAlchemy with Pandas (very useful for IP – Data Handling using Pandas):

```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("mysql+pymysql://root:1234@localhost/school")

df = pd.read_sql("SELECT * FROM student", engine)
print(df)
```

Connection String Format:

```
dialect+driver://username:password@host:port/database
```

Example: mysql+pymysql://root:1234@localhost:3306/school

Key Points on SQLAlchemy:

- Uses an Engine object instead of a plain connection object.
- Query results are fetched using text() for raw SQL strings.
- Requires calling conn.commit() explicitly for DML operations, similar to other connectors.
- Provides an ORM layer for defining tables as Python classes (advanced use, generally beyond basic school-level scope but good for awareness).
- Works seamlessly with pandas.read_sql() and DataFrame.to_sql() for reading/writing data between MySQL and Pandas DataFrames.

Comparison: mysql.connector vs PyMySQL vs SQLAlchemy

Basis	mysql.connector	PyMySQL	SQLAlchemy
Type	Official MySQL driver	Pure-Python driver	SQL Toolkit / ORM
Query Style	Raw SQL strings	Raw SQL strings	Raw SQL (text()) or ORM classes
Main Object	Connection + Cursor	Connection + Cursor	Engine (+ optional ORM Session)
Placeholder Style	%s	%s	:name (named) or %s
DB Independence	MySQL only	MySQL only	MySQL, SQLite, PostgreSQL, Oracle, etc.
Best Suited For	Direct/simple school-level programs	Lightweight alternative	Larger projects, Pandas integration
Installation	pip install mysql-connector-python	pip install pymysql	pip install sqlalchemy (+ a driver)

9. Handling Errors — try/except with Rollback

```
try:
    mycursor.execute("UPDATE student SET marks = 100 WHERE rollno = 5")
    mycon.commit()
except mysql.connector.Error as e:
    mycon.rollback()
    print("Error occurred:", e)
```

10. Complete Menu-Driven Program (Sample Project Style)

```
import mysql.connector as sql

def connect():
    return sql.connect(host="localhost", user="root",
                       password="1234", database="school")

def insert():
    con = connect()
    cur = con.cursor()
    rno = int(input("Roll No: "))
    name = input("Name: ")
    marks = float(input("Marks: "))
    cur.execute("INSERT INTO student VALUES (%s,%s,%s)", (rno, name, marks))
    con.commit()
    print("Inserted Successfully")
    con.close()

def display():
    con = connect()
    cur = con.cursor()
    cur.execute("SELECT * FROM student")
    for row in cur.fetchall():
        print(row)
    con.close()

def update():
    con = connect()
    cur = con.cursor()
    rno = int(input("Enter Roll No to update: "))
    marks = float(input("Enter new marks: "))
    cur.execute("UPDATE student SET marks=%s WHERE rollno=%s", (marks, rno))
    con.commit()
    print(cur.rowcount, "record(s) updated")
    con.close()

def delete():
    con = connect()
    cur = con.cursor()
    rno = int(input("Enter Roll No to delete: "))
    cur.execute("DELETE FROM student WHERE rollno=%s", (rno,))
    con.commit()
    print(cur.rowcount, "record(s) deleted")
    con.close()

while True:
    print("\n1.Insert 2.Display 3.Update 4.Delete 5.Exit")
    ch = int(input("Enter choice: "))
    if ch == 1: insert()
    elif ch == 2: display()
    elif ch == 3: update()
    elif ch == 4: delete()
    elif ch == 5: break
    else: print("Invalid Choice")
```

11. Key Points / Quick Revision (Very Important for Boards)

3. Module used: mysql.connector
4. Function to connect: connect()
5. Object created after connecting: Connection object
6. Object used to execute queries: Cursor object, created using cursor()

7. To run SQL command: execute()
8. To permanently save changes: commit()
9. To undo changes: rollback()
10. To fetch one row: fetchone()
11. To fetch all rows: fetchall()
12. To fetch n rows: fetchmany(n)
13. To know number of affected rows: rowcount
14. To close connection: close()
15. %s is used as a placeholder for parameterized queries (prevents SQL injection and works for all data types).
16. Default auto-commit is OFF — must call commit() explicitly for DML operations.
17. executemany() is used to insert/update multiple records in a single call.

12. Previous Years' CBSE Board Exam Questions (IP / Computer Science)

Note: Exact figures/table data used in some board questions vary by year; the questions below reflect the type and pattern of questions asked in CBSE board examinations on this topic. Please verify the latest official CBSE sample papers/marking schemes on cbseacademic.nic.in for exact repeated wording.

Q1. What do you understand by the term "Database Connectivity"? Why is it required?
(2 Marks – Short Answer)

Q2. Name the package/module that needs to be imported in Python to connect with a MySQL database.
(1 Mark) — Answer: mysql.connector

Q3. Write the Python statement to establish connectivity between Python and MySQL, using the following details: Host: localhost, User: root, Password: tiger, Database: company
(2 Marks)

Q4. Differentiate between fetchone() and fetchall() functions with the help of an example.
(2 Marks)

Q5. What is the significance of a cursor object in Python-MySQL connectivity?
(2 Marks)

Q6. What is the purpose of the commit() method? What will happen if a programmer forgets to use commit() after an INSERT statement?
(2 Marks)

Q7. Rewrite the following code after removing errors, if any. Underline each correction made — this type of "Find and Correct Errors" question on Python-MySQL code is a recurring pattern in CBSE board papers.
(2-3 Marks)

Q8. Write a Python program/function to insert a new record into the table Employee (Eno, Ename, Salary) in the database Office using the mysql.connector module. Also display "Record Inserted" after successful insertion, and use exception handling to display an appropriate error message otherwise.
(4 Marks)

Q9. Write a Python program to display all records from table Student where marks are greater than 80, using MySQL connectivity.
(4 Marks)

Q10. Give any two differences between execute() and executemany() methods of a cursor object.
(2 Marks)

Q11. What is rowcount? Write a statement to display the number of records affected after a DELETE query.
(2 Marks)

Q12. Write the outputs of the following, when executed with the given table (typical table given in board paper), for queries using fetchone(), fetchmany(2) and fetchall() — showing understanding of cursor position/pointer movement.
(3 Marks)

Q13. WAP (Write a Program) in Python to update the salary of an employee whose Eno is entered by the user, in the table Employee stored in MySQL database Company.
(4 Marks)

Q14. What are the advantages of using a parameterized query (using %s placeholders) over directly embedding values in an SQL string?
(2 Marks)

Q15A. Name two Python libraries, other than mysql.connector, that can be used to connect Python with a MySQL database. Answer: pymysql and sqlalchemy.

(1-2 Marks)

Q15. Case Study Based Question: Ravi is developing a Student Management System using Python and MySQL. (a) Which module will Ravi import? (b) Which function will he use to create the cursor object? (c) Write the statement to fetch all records from the "Student" table. (d) Write the statement to save changes permanently after updating a record.

(4 Marks – common pattern in recent papers)

Sample buggy code referred to in Q7:

```
import mysql.connector as mycon
con = mycon.connect(host = localhost, user = "root", passwd = "1234")
cursor = con.Cursor()
cursor.execute(Select * From Employee)
data = cursor.fetchone
print(data)
```

13. Common Mistakes to Avoid (Examiner's Observations)

- Writing mysql.Connector instead of mysql.connector (case-sensitive).
- Forgetting con.commit() after INSERT/UPDATE/DELETE — most common mistake, leads to loss of marks even if logic is correct.
- Using cursor.Execute() instead of cursor.execute().
- Forgetting quotes around string values in a query written without parameterization.
- Not closing the connection (close()) at the end of the program.
- Confusing fetchall() with fetchmany() — students often forget fetchmany() requires an argument n.
