**Module 12: Python SQL Database Access**

* **Introduction**

Python provides several libraries to connect and interact with databases. One of the most widely used for SQL Server is pyodbc, while sqlite3 is built-in for working with SQLite databases.

Using these libraries, Python can:

* Connect to a database (SQL Server, MySQL, SQLite, PostgreSQL, etc.)
* Create and manage tables
* Insert, update, and delete data
* Run SQL queries (SELECT)
* Fetch and display results
* **Install pyodbc**

Run in your terminal/command prompt:

pip install pyodbc

* **DB Connection**

**🔹 Step 2: Connection String**

To connect Python with SQL Server, you need:

* Server Name (e.g., localhost\SQLEXPRESS or just localhost)
* Database Name
* Authentication Method (Windows Authentication or SQL Authentication)
* **Create database**

import pyodbc

# Connect to SQL Server with autocommit ON

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'Trusted\_Connection=yes;',

    autocommit=True   # <-- IMPORTANT

)

cursor = conn.cursor()

# Create the database

cursor.execute("CREATE DATABASE Learning\_python")

print("Database created successfully!")

conn.close()

Example :-

* **Create a Table**

import pyodbc

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'DATABASE=PermanentEmployees;'

    'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Create Table

cursor.execute('''

    CREATE TABLE Infosys\_Company (

        ID INT PRIMARY KEY,

        Name NVARCHAR(50),

        Age INT,

        Salary INT,

               Company NVARCHAR(50),

               Department NVARCHAR(50)

    )

''')

conn.commit()

* **INSERT**

import pyodbc

# Connect to SQL Server

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'DATABASE=PermanentEmployees;'

    'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Insert single record

cursor.execute('''

    INSERT INTO Infosys\_Company (ID, Name, Age, Salary, Company, Department)

    VALUES (?, ?, ?, ?, ?, ?)

''', (1, 'Amit Kumar', 30, 60000, 'Infosys', 'IT'))

# Insert multiple records

records = [

    (2, 'Ravi Sharma', 28, 55000, 'Infosys', 'HR'),

    (3, 'Neha Gupta', 32, 75000, 'Infosys', 'Finance'),

    (4, 'Sunil Verma', 27, 50000, 'Infosys', 'Marketing')

]

cursor.executemany('''

    INSERT INTO Infosys\_Company (ID, Name, Age, Salary, Company, Department)

    VALUES (?, ?, ?, ?, ?, ?)

''', records)

# Commit the changes

conn.commit()

print("Data inserted successfully!")

# Close connection

cursor.close()

conn.close()

* **READ**

import pyodbc

# Connect to SQL Server

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'DATABASE=PermanentEmployees;'

    'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Fetch all data

cursor.execute("SELECT \* FROM Infosys\_Company")

rows = cursor.fetchall()

# Print results

print("ID | Name        | Age | Salary | Company | Department")

print("-" \* 60)

for row in rows:

    print(f"{row.ID} | {row.Name} | {row.Age} | {row.Salary} | {row.Company} | {row.Department}")

# Close connection

cursor.close()

conn.close()

* **UPDATE**

import pyodbc

# Connect to SQL Server

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'DATABASE=PermanentEmployees;'

    'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Example: Update Salary of employee with ID = 2

cursor.execute('''

    UPDATE Infosys\_Company

    SET Salary = ?

    WHERE ID = ?

''', (65000, 2))

# Example: Update Department of employee with Name = 'Neha Gupta'

cursor.execute('''

    UPDATE Infosys\_Company

    SET Department = ?

    WHERE Name = ?

''', ('Accounts', 'Neha Gupta'))

# Commit changes

conn.commit()

print("Data updated successfully!")

# Close connection

cursor.close()

conn.close()

* **DELETE**

import pyodbc

# Connect to SQL Server

conn = pyodbc.connect(

    'DRIVER={ODBC Driver 17 for SQL Server};'

    'SERVER=DESKTOP-VH02L8Q;'

    'DATABASE=PermanentEmployees;'

    'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Example: Delete a record where ID = 3

cursor.execute('''

    DELETE FROM Infosys\_Company

    WHERE ID = ?

''', (3,))

# Example: Delete a record by Name

cursor.execute('''

    DELETE FROM Infosys\_Company

    WHERE Name = ?

''', ('Sunil Verma',))

# Commit changes

conn.commit()

print("Record(s) deleted successfully!")

# Close connection

cursor.close()

conn.close()

* **COMMIT**

In SQL Server with pyodbc, commit() is needed whenever you make a change (INSERT, UPDATE, DELETE, CREATE, DROP).

import pyodbc

# Connect

conn = pyodbc.connect(

'DRIVER={ODBC Driver 17 for SQL Server};'

'SERVER=DESKTOP-VH02L8Q;'

'DATABASE=PermanentEmployees;'

'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Insert data

cursor.execute('''

INSERT INTO Infosys\_Company (ID, Name, Age, Salary, Company, Department)

VALUES (?, ?, ?, ?, ?, ?)

''', (5, 'Pooja Singh', 29, 70000, 'Infosys', 'Operations'))

# ✅ Save changes

conn.commit()

print("Data committed successfully!")

# Close

cursor.close()

conn.close()

**When to use commit():**

* **Required** → INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE.
* **Not required** → SELECT (since it doesn’t change data).

If you don’t call commit(), your changes will not be saved in SQL Server (they remain in a temporary transaction).

* **ROLLBACK**

you can use rollback() to undo uncommitted changes if something goes wrong.

Here’s an example:

import pyodbc

# Connect

conn = pyodbc.connect(

'DRIVER={ODBC Driver 17 for SQL Server};'

'SERVER=DESKTOP-VH02L8Q;'

'DATABASE=PermanentEmployees;'

'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

try:

# Start transaction (implicitly started when connection is made)

cursor.execute('''

INSERT INTO Infosys\_Company (ID, Name, Age, Salary, Company, Department)

VALUES (?, ?, ?, ?, ?, ?)

''', (6, 'Rajesh Kumar', 35, 80000, 'Infosys', 'R&D'))

# ❌ Force an error (duplicate ID to trigger rollback)

cursor.execute('''

INSERT INTO Infosys\_Company (ID, Name, Age, Salary, Company, Department)

VALUES (?, ?, ?, ?, ?, ?)

''', (6, 'Duplicate User', 40, 90000, 'Infosys', 'Finance'))

# If all good → commit

conn.commit()

print("Transaction committed!")

except Exception as e:

# If error → rollback

print("Error occurred, rolling back transaction:", e)

conn.rollback()

finally:

cursor.close()

conn.close()

🔑 Key Points about rollback():

SQL Server starts a transaction automatically when you run INSERT, UPDATE, or DELETE.

If you don’t call commit(), you can call rollback() to cancel all uncommitted changes.

After rollback → database returns to the state before the transaction started.

* **handling Errors**

When using pyodbc (or any DB API), you should use try-except-finally blocks to catch errors, rollback transactions if needed, and close connections cleanly.

Here’s a good structure for Python SQL Server database access with error handling:

import pyodbc

try:

# Connect to SQL Server

conn = pyodbc.connect(

'DRIVER={ODBC Driver 17 for SQL Server};'

'SERVER=DESKTOP-VH02L8Q;'

'DATABASE=PermanentEmployees;'

'Trusted\_Connection=yes;'

)

cursor = conn.cursor()

# Example: Create Table

try:

cursor.execute('''

CREATE TABLE Employees (

ID INT PRIMARY KEY,

Name NVARCHAR(50),

Age INT,

Salary INT

)

''')

conn.commit()

print("Table created successfully.")

except pyodbc.Error as e:

print("Error creating table:", e)

conn.rollback() # rollback if table creation failed

# Example: Insert Data

try:

cursor.execute("INSERT INTO Employees (ID, Name, Age, Salary) VALUES (?, ?, ?, ?)",

(1, 'Amit', 30, 50000))

conn.commit()

print("Data inserted successfully.")

except pyodbc.IntegrityError as e: # e.g., duplicate primary key

print("Integrity Error:", e)

conn.rollback()

except pyodbc.Error as e:

print("Error inserting data:", e)

conn.rollback()

# Example: Fetch Data

try:

cursor.execute("SELECT \* FROM Employees")

rows = cursor.fetchall()

for row in rows:

print(row)

except pyodbc.Error as e:

print("Error fetching data:", e)

except pyodbc.Error as conn\_error:

print("Database connection failed:", conn\_error)

finally:

# Ensure resources are closed

if 'cursor' in locals():

cursor.close()

if 'conn' in locals():

conn.close()

print("Database connection closed.")

**🔑 Key Points:**

1. **try-except** → Catch SQL errors like syntax errors, primary key violations, etc.
2. **rollback()** → Undo changes if something fails inside a transaction.
3. **commit()** → Save successful changes.
4. **finally** → Always close cursor and conn to free resources.