**Connect To Postgres With Python**

**Part 1: Installing PostgreSQL on Windows (Step-by-Step)**

**1️⃣ Go to the official download page:**

👉 <https://www.postgresql.org/download/windows/>

You’ll be redirected to **EnterpriseDB's Windows installer**.

**2️⃣ Download & Run Installer**

* Select your version (e.g., **PostgreSQL 14**)
* Choose **64-bit** (if your Windows is 64-bit)
* Download and double-click the .exe to launch installer

**3️⃣ Follow the Installation Wizard:**

| **Step** | **What to Do** |
| --- | --- |
| Welcome | Click **Next** |
| Choose Directory | Leave default (C:\Program Files\PostgreSQL\14) or change |
| Select Components | Keep all default ✅ |
| Data Directory | This is where PostgreSQL stores all data. Leave as-is |
| Password | Set the password for the postgres superuser (example: admin123) |
| Port | Default is 5432 – leave it unless there's a conflict |
| Locale | Leave default (matches your OS language) |
| Ready to Install | Review & click **Next** |
| Installation Progress | Wait 2–5 minutes |
| Stack Builder | You can uncheck this. Not required right now |
| Finish | Click **Finish** |

**Optional: Verify PostgreSQL Installation**

Open **pgAdmin** (installed along with PostgreSQL)  
OR  
Use CMD:

psql -U postgres -h localhost -p 5432

It should ask for your password and connect to the shell.

**📦 Part 2: Installing psycopg2 Python Library**

Open PowerShell or CMD and type:

pip install psycopg2-binary

✅ This installs the PostgreSQL driver needed for Python.

**🧠 Part 3: Understanding the Architecture**

+-------------+ +---------------------+ +--------------------+

| Python App | <-----> | psycopg2 (DB Driver) | <-----> | PostgreSQL Server |

+-------------+ +---------------------+ +--------------------+

* **Python** sends SQL commands through **psycopg2**
* **psycopg2** connects to **PostgreSQL** server running locally or remotely
* **SQL commands** are interpreted, executed, and results are returned

**📌 Part 4: Connecting to PostgreSQL in Python (Ultra Deep Breakdown)**

**🧑‍💻 Code:**

import psycopg2

# Step 1: Create connection

conn = psycopg2.connect(

database="your\_database\_name",

user="your\_username",

password="your\_password",

host="your\_host",

port="your\_port"

)

# Step 2: Enable autocommit

conn.autocommit = True

# Step 3: Create cursor

cur = conn.cursor()

# Step 4: Execute SQL here

# Step 5: Close everything

cur.close()

conn.close()

**🔍 Line-by-Line Explanation:**

**🔸 import psycopg2**

* Loads the library so we can connect to PostgreSQL and send SQL commands

**🔸 psycopg2.connect(...)**

* Establishes the connection to your PostgreSQL database server
* Required parameters:
  + database: The DB name you're connecting to
  + user: Username (e.g., postgres)
  + password: Password for the user
  + host: localhost for local DB
  + port: Default is 5432

**🔸 conn.autocommit = True**

* Makes sure that every SQL statement is automatically committed
* Without this, you need to explicitly call conn.commit() after each statement

**🔸 cur = conn.cursor()**

* Creates a **cursor** object that lets us run SQL queries

**🔸 cur.close() and conn.close()**

* Always close both to free up server connections and memory

**🎯 Part 5: Create a Database from Python**

cur.execute('CREATE DATABASE new\_database;')

* This SQL command tells PostgreSQL to create a new database
* You must run this **while connected to a different database** (like postgres), because you can't create a DB you're currently inside of

**Part 6: View Existing Databases**

cur.execute('SELECT datname FROM pg\_database;')

databases = cur.fetchall()

print(databases)

* Queries the pg\_database system table to fetch all existing database names
* fetchall() returns a list of all result rows

**🧱 Part 7: Create a Table**

cur.execute('''

CREATE TABLE my\_table (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

age INT

);

''')

**🔍 Deep Dive:**

* id SERIAL PRIMARY KEY: Auto-incremented ID (like AUTO\_INCREMENT)
* name VARCHAR(100): Text column max 100 characters
* age INT: Integer column
* PRIMARY KEY: Makes sure ID is unique for each row

**➕ Part 8: Insert Data**

cur.execute("INSERT INTO my\_table (name, age) VALUES (%s, %s);", ('John Doe', 30))

**🔍 Explanation:**

* %s placeholders protect against SQL injection
* Values are passed as a tuple: ('John Doe', 30)

**📂 Part 9: View All Tables in Current DB**

cur.execute("SELECT table\_name FROM information\_schema.tables WHERE table\_schema='public';")

tables = cur.fetchall()

print(tables)

* This queries the information\_schema.tables system view
* Filters on public schema which is the default in PostgreSQL

**👁 Part 10: View Data in the Table**

cur.execute("SELECT \* FROM my\_table;")

data = cur.fetchall()

for row in data:

print(row)

* SELECT \* FROM my\_table fetches all rows
* fetchall() retrieves all result rows
* Looping over results lets you process or display each row

**🔐 Part 11: Exception Handling (Best Practice)**

try:

conn = psycopg2.connect(...)

cur = conn.cursor()

cur.execute("...")

except Exception as e:

print("Something went wrong:", e)

finally:

if cur: cur.close()

if conn: conn.close()

✅ Prevents your script from crashing due to:

* Wrong credentials
* Invalid SQL
* Connection refused

**Complete Practical Script (Put it All Together)**

import psycopg2

try:

conn = psycopg2.connect(

database="postgres",

user="postgres",

password="admin123",

host="localhost",

port="5432"

)

conn.autocommit = True

cur = conn.cursor()

cur.execute("CREATE DATABASE demo\_db;")

print("Database 'demo\_db' created!")

cur.close()

conn.close()

conn = psycopg2.connect(

database="demo\_db",

user="postgres",

password="admin123",

host="localhost",

port="5432"

)

conn.autocommit = True

cur = conn.cursor()

cur.execute('''

CREATE TABLE my\_table (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

age INT

);

''')

cur.execute("INSERT INTO my\_table (name, age) VALUES (%s, %s);", ('John Doe', 30))

cur.execute("INSERT INTO my\_table (name, age) VALUES (%s, %s);", ('Alice', 25))

cur.execute("SELECT \* FROM my\_table;")

rows = cur.fetchall()

for row in rows:

print(row)

except Exception as e:

print("Error:", e)

finally:

if cur:

cur.close()

if conn:

conn.close()

**⚠️ Common Errors**

| **Error** | **Cause** | **Fix** |
| --- | --- | --- |
| FATAL: password authentication failed | Wrong password | Use correct password |
| connection refused | PostgreSQL not running | Start PostgreSQL service |
| duplicate database | Already exists | Drop or use a different name |
| relation already exists | Table already exists | Use DROP TABLE IF EXISTS |

**📌 Best Practices Checklist**

✅ Use psycopg2-binary  
✅ Always close cursor and connection  
✅ Use try-except-finally blocks  
✅ Use %s placeholders to avoid SQL injection  
✅ Keep sensitive data like passwords in .env files (next lesson)  
✅ Autocommit only when necessary (for critical writes, disable it)

**✅ Summary**

* Installed PostgreSQL + pgAdmin
* Installed psycopg2
* Connected from Python
* Created DB, Table, Inserted Data, and Read from Table
* Learned clean coding with error handling

**PART-2: Deep PostgreSQL Interactions via Python**

**📘 Goals:**

* Connect to an existing PostgreSQL database (demo\_db)
* Create a table (employees)
* Insert multiple rows of data
* Read and display data
* Handle exceptions properly
* Understand the **why** behind every step

**🧱 Step 1: Create & Connect to Your Working Database (demo\_db)**

Make sure this database is already created (from Part-1). If not, refer back or run:

CREATE DATABASE demo\_db;

**✅ Step 2: Full Python Code to Create Table + Insert + Read**

We’ll start with the entire working Python script, then **explain it piece by piece**.

**🧑‍💻 Full Script:**

import psycopg2

try:

# Connect to existing database

conn = psycopg2.connect(

database="demo\_db",

user="postgres",

password="admin123",

host="localhost",

port="5432"

)

conn.autocommit = True

cur = conn.cursor()

# Step 1: Create a table

cur.execute("""

CREATE TABLE IF NOT EXISTS employees (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

role VARCHAR(100),

age INT CHECK (age > 0)

);

""")

print("Table 'employees' created.")

# Step 2: Insert some employees

employees = [

('Alice', 'DevOps Engineer', 28),

('Bob', 'Python Developer', 32),

('Charlie', 'SysAdmin', 29)

]

for emp in employees:

cur.execute("INSERT INTO employees (name, role, age) VALUES (%s, %s, %s);", emp)

print("Inserted", len(employees), "records.")

# Step 3: Fetch all records

cur.execute("SELECT \* FROM employees;")

rows = cur.fetchall()

print("Employee Records:")

for row in rows:

print(row)

except Exception as e:

print("Error:", e)

finally:

if cur:

cur.close()

if conn:

conn.close()

**🔍 Line-by-Line Ultra Deep Explanation**

**🔸 import psycopg2**

* This loads the PostgreSQL driver so Python can communicate with PostgreSQL.
* Required for all DB work.

**🔸 conn = psycopg2.connect(...)**

conn = psycopg2.connect(

database="demo\_db",

user="postgres",

password="admin123",

host="localhost",

port="5432"

)

* This opens a **connection session** with the demo\_db database.
* Replace admin123 if your password is different.
* If connection fails, the except block will catch the error.

**🔸 conn.autocommit = True**

* PostgreSQL requires an explicit commit for DML operations (INSERT, UPDATE, DELETE).
* Setting this to True means: **auto-save after every query**.
* For production, you may want to use transactions instead (safer control).

**🔸 cur = conn.cursor()**

* This creates a **cursor object**: a "control panel" for sending SQL commands.

**🔸 Table Creation**

cur.execute("""

CREATE TABLE IF NOT EXISTS employees (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

role VARCHAR(100),

age INT CHECK (age > 0)

);

""")

| **Part** | **Meaning** |
| --- | --- |
| IF NOT EXISTS | Prevents error if the table already exists |
| id SERIAL | Auto-incremented primary key |
| name VARCHAR(100) NOT NULL | Text field, must be filled |
| role VARCHAR(100) | Optional string field |
| age INT CHECK (age > 0) | Integer with constraint: age must be positive |

**🔸 Insert Multiple Records**

employees = [

('Alice', 'DevOps Engineer', 28),

('Bob', 'Python Developer', 32),

('Charlie', 'SysAdmin', 29)

]

for emp in employees:

cur.execute("INSERT INTO employees (name, role, age) VALUES (%s, %s, %s);", emp)

* A list of tuples, where each tuple is a row of data.
* Using a loop to insert each row securely.
* %s are **safe parameter placeholders** (prevents SQL injection).
* The loop feeds in values dynamically from the employees list.

**🔸 Fetch and Display All Records**

cur.execute("SELECT \* FROM employees;")

rows = cur.fetchall()

* SQL: SELECT \* retrieves all rows and columns from employees.
* fetchall() returns a list of tuples (each tuple = one row).

for row in rows:

print(row)

* Iterates over each row and prints it.
* Sample output:

(1, 'Alice', 'DevOps Engineer', 28)

(2, 'Bob', 'Python Developer', 32)

**🔸 Exception Handling**

except Exception as e:

print("Error:", e)

* Any DB error (e.g., bad SQL, wrong credentials) will show here.

**🔸 Clean Up**

finally:

if cur:

cur.close()

if conn:

conn.close()

* Always close your cursor and connection.
* Prevents memory leaks and connection exhaustion.

**🧪 Run It and Verify**

After running the script:

* You should see the employees table in your demo\_db
* You’ll see the printed rows in console output
* You can also connect via **pgAdmin** and browse the data

**🧯 Bonus: How to Update & Delete Data**

**🔁 Update:**

cur.execute("UPDATE employees SET age = %s WHERE name = %s;", (30, 'Alice'))

**❌ Delete:**

cur.execute("DELETE FROM employees WHERE name = %s;", ('Charlie',))