Database Access in Python
Relational and Non-Relational Databases

Advanced Python for Data Science

Relational Databases (SQL)

Relational databases store data in structured tables consisting of rows and columns. They use **Structured Query Language (SQL)** for data manipulation.

- Data is organized into tables
- Relationships exist between tables
- Ensures data integrity using constraints

Common Relational Databases

- **SQLite** – Lightweight, file-based database
- **MySQL / MariaDB** – Popular open-source databases
- **PostgreSQL** – Advanced open-source database
- **Oracle** – Enterprise-level database
- **SQL Server** – Microsoft relational database

Python Libraries for SQL Databases

Library	Purpose
sqlite3	Built-in support for SQLite
psycopg2	PostgreSQL database access
mysql-connector-python	MySQL database access
pyodbc	ODBC-based database connectivity
SQLAlchemy	ORM and SQL toolkit
pandas	Read/write SQL tables

Create a Database in SQLite3

Connect to or create a new SQLite database

```
import sqlite3

conn = sqlite3.connect('school.db')  # Database file is create
cursor = conn.cursor()

print("Database created successfully")
conn.close()
```

Notes:

- sqlite3.connect() creates a new database file if it does not exist.
- cursor is used to execute SQL commands.
- Always close the connection using conn.close().

Create a Table in SQLite3

Define and create a table in the SQLite database

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY,
    name TEXT,
    marks INTEGER
)
''')
print("Table created successfully")
conn.commit()
conn.close()
```

Insert One Record in SQLite3

Add a single record to the students table

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute("INSERT INTO students (id, name, marks)
            VALUES (?, ?, ?)", (1, 'Sita', 85))

print("Record inserted successfully")
conn.commit()
conn.close()
```

Insert Multiple Records in SQLite3

Insert multiple records into the students table

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
students = [
    (2, 'Ram', 78),
    (3, 'Hari', 92),
    (4, 'Suman', 88)
]
cursor.executemany("INSERT INTO students (id, name, marks)
    VALUES (?, ?, ?)", students)

print("Multiple records inserted successfully")
conn.commit()
conn.close()
```

Print All Records in SQLite3

Retrieve and display all records from the students table

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()

for row in rows:
    print(f"ID: {row[0]}, Name: {row[1]}, Marks: {row[2]}")

conn.close()
```

Select a Particular Record in SQLite3

Retrieve and display a specific record from the students table

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute("SELECT * FROM students WHERE name = ?",
    ("Sita",))
row = cursor.fetchone()

if row:
    print(f"ID: {row[0]}, Name: {row[1]}, Marks: {row[2]}")
else:
    print("Record not found")

conn.close()
```

Delete a Particular Record in SQLite3

Remove a specific record from the students table

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute("DELETE FROM students
          WHERE name = ?", ("Ram",))

conn.commit()
print("Record deleted successfully")
conn.close()
```

Delete (Drop) a Table in SQLite3

Remove the entire students table and all its records

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute("DROP TABLE IF EXISTS students")

print("Table deleted successfully")
conn.commit()
conn.close()
```

What is SQLAlchemy?

- SQLAlchemy is a Python **SQL toolkit** and **Object Relational Mapper (ORM)**.
- Allows Python programs to interact with relational databases using **Python classes and objects**.
- Supports multiple relational databases: SQLite, MySQL, PostgreSQL, Oracle, SQL Server, etc.

Insert Multiple Records in SQLite3

Add multiple records to the students table

```python
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

students = [
    (2, 'Ram', 78),
    (3, 'Hari', 92),
    (4, 'Suman', 88)
]

cursor.executemany("INSERT INTO students (id, name, marks) VAI

print("Multiple records inserted successfully")
conn.commit()
conn.close()
```

Key Features of SQLAlchemy

- **ORM (Object Relational Mapper)**
  - Maps Python classes to database tables.
  - Maps Python objects to table rows.
  - Example: a `Student` class represents the `students` table.
- **SQL Expression Language**: write SQL queries in Python syntax.
- **Database Agnostic**: code works with multiple relational databases.
- **Transaction Management**: uses Session objects for safe inserts, updates, deletes, commits.
- **Integration with Pandas**: works with `pd.read_sql()` and `DataFrame.to_sql()`.

Example: Simple ORM Class

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///school.db')
Base = declarative_base()

class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    marks = Column(Integer)
```

Notes:

- Student $\rightarrow$ Python class representing the students table.
- id, name, marks $\rightarrow$ Columns in the table.
- SQLAlchemy handles table creation and mapping objects to rows.

Define a Table (Part 1)

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    marks = Column(Integer)
```

```
engine = create_engine('sqlite:///school.db', echo=True)
Base.metadata.create_all(engine)

print("Table created successfully")
```

Print All Records from Table

Query and display all records using SQLAlchemy ORM

```
# Create session
Session = sessionmaker(bind=engine)
session = Session()

# Query all records
students = session.query(Student).all()

# Print all records
for student in students:
    print(f"ID: {student.id}, Name: {student.name},
      Marks: {student.marks}")
```

Select a Particular Record in SQLAlchemy

Query and display a specific record from the table

```
student = session.query(Student).filter
        (Student.name == "Sita").first()

if student:
    print(f"ID: {student.id}, Name: {student.name},
    Marks: {student.marks}")
else:
    print("Record not found")
```

Delete a specific record from the table using ORM

```
student_to_delete = session.query(Student).
     filter(Student.name == "Ram").first()

if student_to_delete:
    session.delete(student_to_delete)
    session.commit()
    print("Record deleted successfully")
else:
    print("Record not found")
```

```
Student.__table__.drop(engine)
print("Table deleted successfully")
```

SQL Key Takeaways

- SQL databases are ideal for structured data
- Python provides multiple libraries for SQL access
- SQLite is suitable for learning and small projects
- SQLAlchemy is preferred for scalable applications

Non-Relational Databases (NoSQL)

NoSQL databases store data in flexible and schema-less formats.

- Schema-less or dynamic schema
- Supports JSON, key-value, graph, and column formats
- Highly scalable and distributed

Types of NoSQL Databases

Type	Database	Use Case
Document	MongoDB	Semi-structured data
Key-Value	Redis	Caching
Column	Cassandra	Big data
Graph	Neo4j	Relationships

MongoDB for Data Science

- Popular document-based NoSQL database
- Stores data in JSON-like format (BSON)
- Used for logs, text, and API data
- Easy integration with Python

Python Library for MongoDB

PyMongo is the official Python driver for MongoDB.

Installation

```
pip install pymongo
```

MongoDB Access in Python

Connecting to MongoDB and inserting data

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["school"]
collection = db["students"]

collection.insert_one({
    "id": 1,
    "name": "Sita",
    "marks": 85
})
```

Find students with marks greater than 80

```
result = collection.find({"marks": {"$gt": 80}})

for doc in result:
    print(doc)
```

NoSQL Key Takeaways

- NoSQL handles unstructured data efficiently
- MongoDB is common in data science workflows
- PyMongo enables easy Python integration
- SQL and NoSQL are often used together

What is a Collection in MongoDB?

In **MongoDB**, a **collection** is a group of documents stored inside a database.

SQL vs MongoDB Comparison

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

PyMongo: Creating a Database

MongoDB creates a database automatically when data is inserted.

```python
from pymongo import MongoClient

# Connect to MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# Create (or access) a database
db = client["school"]

# Create a collection and insert a document
collection = db["students"]
collection.insert_one({
    "id": 1,
    "name": "Sita",
    "marks": 85
})
```

PyMongo: Insert Multiple Documents

```
collection.insert_many([
    {"id": 2, "name": "Ram", "marks": 78},
    {"id": 3, "name": "Hari", "marks": 92}
])

print("Two documents inserted successfully")
```

Print All Documents in a Collection

Retrieve and display all documents in a MongoDB collection

```
result = collection.find()

for doc in result:
    print(doc)
```

Find a Particular Person in MongoDB

Retrieve documents where the name is "Sita"

```
result = collection.find({"name": "Sita"})

# Display results
for doc in result:
    print(doc)
```

Delete One Document in MongoDB

Delete a document based on specified criteria

```python
# Delete one document where name is "Ram"
collection.delete_one({"name": "Ram"})
```

Delete an Entire Collection in MongoDB

Remove all documents and the collection itself

```
collection.drop()

print("Collection deleted successfully")
```