The SQLAlchemy :

**SQLAlchemy** is a powerful Python SQL toolkit and Object Relational Mapper (ORM) that provides developers with the flexibility to work with SQL databases using Pythonic domain language. It allows developers to interact with databases using Python objects, eliminating the need to write separate SQL queries. This makes it easier to build high-performance SQL-based applications.

**Key Features**

1. **Object Relational Mapper (ORM)**: SQLAlchemy's ORM allows developers to map Python classes to database tables, enabling seamless interaction between Python objects and database records. This simplifies database operations and enhances code readability.
2. **Database Agnostic**: SQLAlchemy abstracts the differences between various database engines, allowing developers to write database code in a database-agnostic manner. This means you can easily switch between different database systems such as SQLite, PostgreSQL, MySQL, or Microsoft SQL Server without changing your code.
3. **SQL Expression Language**: SQLAlchemy provides a SQL Expression Language that allows developers to construct SQL queries using Python objects. This makes it easier to build complex queries programmatically.

**Installation**

To install SQLAlchemy, you can use the Python Package Manager (pip) or the Anaconda distribution

**What is Object-Relational Mapping (ORM) in DBMS?**

With Object-Relational Mapping, it becomes much easier to work with an object-oriented programming language and relational database. Fundamentally, it acts as a translator, translating data between the database and the application without any hitch. ORM enables developers to work with objects in their programming language which are mapped to corresponding database entities, such as tables, views, or stored procedures.

Key Concepts of ORM:

* **Object-Oriented Paradigm:** ORM focuses on OOP principles that data and behavior are encapsulated within objects. In ORM, database entities are mapped to objects and developers use objects to interact and manipulate data in a fairly easy way.
* **Mapping:** The main purpose of ORM is object mapping to database tables and back. The mapping is defined through the metadata which represents interconnections between objects and the corresponding database schemas. Metadata of ORM frameworks is used for the generation of SQL queries and management of data flow between the application and the database.
* **CRUD Operations:** ORM makes CRUD operations easier. Developers can perform such operations on objects in their programming language; the ORM framework takes care of the translation of the operations to their corresponding SQL statements for the underlying database.

# from sqlalchemy import create\_engine, ForeignKey, column, String, Integer, CHAR

from sqlalchemy import \*

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

#?? Base Declaration

Base = declarative\_base()

class Person(Base):

    #!! Table name will be people

    \_\_tablename\_\_ = 'people'

    #!!! Column names ( column name , datatype, optional null/pk/fk)

    ssn = Column('ssn', Integer, primary\_key=True)

    firstname = Column('firstname',String)

    lastname = Column('lastname',String)

    gender = Column('gender',CHAR)

    age = Column('age',Integer)

    def \_\_init\_\_(self,ssn, firstname , lastname , gender,age):

        self.ssn = ssn

        self.firstname = firstname

        self.lastname = lastname

        self.gender = gender

        self.age = age

    def \_\_repr\_\_(self):

        return f'{self.ssn}, { self.firstname}, {self.gender}, {self.age}'

##?? connect to sqlite data base

engine = create\_engine('sqlite:///mydb.db',echo=True)

Base.metadata.create\_all(bind=engine)

Session = sessionmaker(bind=engine)

session = Session()

person = Person(100,'Pranav','Sirsufale', 'M',21)

session.add(person)

session.commit()

p1 = Person(101,'Pooja','Sirsufale','F',20)

p2 = Person(102,'Pallavi','Tadaskar','F',25)

p3 = Person(103,'Rohan','Magar','M',23)

session.add\_all([p1,p2,p3])

session.commit()

Output :

f:\DSA\practical\python\alchemy\alchemeypro.py:7: MovedIn20Warning: The ``declarative\_base()`` function is now available as sqlalchemy.orm.declarative\_base(). (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)

  Base = declarative\_base()

2025-02-08 07:54:37,515 INFO sqlalchemy.engine.Engine BEGIN (implicit)

2025-02-08 07:54:37,516 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("people")

2025-02-08 07:54:37,517 INFO sqlalchemy.engine.Engine [raw sql] ()

2025-02-08 07:54:37,522 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("people")

2025-02-08 07:54:37,523 INFO sqlalchemy.engine.Engine [raw sql] ()

2025-02-08 07:54:37,525 INFO sqlalchemy.engine.Engine

CREATE TABLE people (

    ssn INTEGER NOT NULL,

    firstname VARCHAR,

    lastname VARCHAR,

    gender CHAR,

    age INTEGER,

    PRIMARY KEY (ssn)

)

2025-02-08 07:54:37,525 INFO sqlalchemy.engine.Engine [no key 0.00030s] ()

2025-02-08 07:54:37,675 INFO sqlalchemy.engine.Engine COMMIT

2025-02-08 07:54:37,683 INFO sqlalchemy.engine.Engine BEGIN (implicit)

2025-02-08 07:54:37,691 INFO sqlalchemy.engine.Engine INSERT INTO people (ssn, firstname, lastname, gender, age) VALUES (?, ?, ?, ?, ?)

2025-02-08 07:54:37,691 INFO sqlalchemy.engine.Engine [generated in 0.00050s] (100, 'Pranav', 'Sirsufale', 'M', 21)

2025-02-08 07:54:37,694 INFO sqlalchemy.engine.Engine COMMIT

2025-02-08 07:54:37,867 INFO sqlalchemy.engine.Engine BEGIN (implicit)

2025-02-08 07:54:37,869 INFO sqlalchemy.engine.Engine INSERT INTO people (ssn, firstname, lastname, gender, age) VALUES (?, ?, ?, ?, ?)

2025-02-08 07:54:37,869 INFO sqlalchemy.engine.Engine [generated in 0.00059s] [(101, 'Pooja', 'Sirsufale', 'F', 20), (102, 'Pallavi', 'Tadaskar', 'F', 25), (103, 'Rohan', 'Magar', 'M', 23)]

2025-02-08 07:54:37,873 INFO sqlalchemy.engine.Engine COMMIT