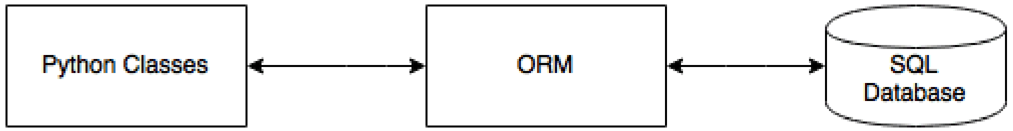
Object Relational Mapping (ORM)
in python

Introduction:

- **O**bject **R**elational **M**apping is a system of mapping objects to a database. That means it automates the transfer of data stored in relational databases tables into objects that are commonly used in application code.

- An object relational mapper maps a relational database system to objects. The ORM is independent of which relational database system is used. From within Python, you can talk to objects and the ORM will map it to the database.



- **ORM**s provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to interact (create, read, update and delete data and schemas) with database.

The mapping like this…

- Python Class == SQL Table
- Instance of the Class == Row in the Table

- Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

- There are many ORM implementations written in Python, including
  - **SQLAlchemy**
  - Peewee
  - The Django ORM
  - PonyORM
  - SQLObject
  - Tortoise ORM

- We are going to discuss about **SQLAlchemy**,it pronounced as **SQL**-**All**-**Chemy**.

SQLAlchemy:

- **SQLAlchemy** is a library used to interact with a wide variety of databases. It enables you to create data models and queries in a manner that feels like normal Python classes and statements.

- It can be used to connect to most common databases such as Postgres, MySQL, SQLite, Oracle, and many others.

- **SQLAlchemy** is a popular SQL toolkit and Object Relational Mapper. It is written in Python and gives full power and flexibility of SQL to an application developer.

- It is necessary to install SQLAlchemy. To install we have to use following code at Terminal or CMD.

  pip install sqlalchemy

- To check if SQLAlchemy is properly installed or not, enter the following command in the Python prompt

  >>>import sqlalchemy

- If the above statement was executed with no errors, “sqlalchemy " is installed and ready to be used.

Connecting to Database:

- To connect with database using SQLAlchemy, we have to create engine for this purpose SQLAlchemy supports one function is **create_engine().**
- The **create_engine()** function is used to create engine; it takes overall responsibilities of database connectivity.

Syntax:

```
Database-server[+driver]://user:password@host/dbname
```

Example:

```
mysql+mysqldb://root:root@localhost/collegedb
```

- The main objective of the ORM-API of SQLAlchemy is to facilitate associating user-defined Python classes with database tables, and objects of those classes with rows in their corresponding tables.

Declare Mapping:

- First of all, create_engine() function is called to set up an engine object which is subsequently used to perform SQL operations.

To create engine in case of MySQL:

Example:

```
from sqlalchemy import create_engine
engine = create_engine('mysql+mysqldb://root:@localhost/Collegedb')
```

- When using ORM, we first configure database tables that we will be using. Then we define classes that will be mapped to them. Modern SQLAlchemy uses *Declarative* system to do these tasks.
- A *declarative base class* is created, which maintains a catalog of classes and tables. A declarative base class is created with the declarative_base() function.
- *The declarative_base() function is used to create base class. This function is defined in* ***sqlalchemy.ext.declarative*** *module.*

To create declarative base class:

Example:

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

<u>Example:</u> **tabledef.py**

```
from sqlalchemy import Column, Integer, String
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
# create a engine
engine =create_engine('mysql+mysqldb://root:@localhost/Sampledb‘)
# create a declarative base class
Base = declarative_base()

class Students(Base):
   __tablename__ = 'students'
   id = Column(Integer, primary_key=True)
   name = Column(String(10))
   address = Column(String(10))
   email = Column(String(10))

Base.metadata.create_all(engine)
```