SQL Alchemy :-

4:36:00

- It is a Python ORM.
- Databases supported by Sqlalchemy :-
  - PostgreSQL
  - MySQL
  - SQLite
  - Oracle
  - Microsoft SQL Server , etc

- Installation :- [pip install sqlalchemy

- SQLAlchemy cannot directly perform operations on database. To work with, we also need (to install & use) the Database Adapter or ODBC for our particular database, such as :-
  psycopg for PostgreSQL
  mysql.connector for MySQL

4:38:20

- In our CRUD project, we will create a seperate file named database.py, and here will be the code for handling connection to database.
  It is the conventional workflow with Sqlalchemy

In database.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = 'postgresql://<username>:<password>@<ip-address/hostname>/
<database_name>'

engine = create_engine(SQLALCHEMY_DATABASE_URL)
```

both are same

⇒ Above is the starting template for SQLalchemy connection

⇒ First, we need to pass a connection string to Sqlalchemy, that we stored above in SQLALCHEMY_DATABASE_URL.
Above shown string is the syntax, where the <...> placeholders must be replaced with the data specified.

⇒ Then, we need to create an engine.
The Engine is responsible for connection of Sqlalchemy to postgresql database.

For our case, the connection string will be :-
SQL..... URL = "postgresql://postgres:Ultimate99@localhost/fastapi_orm"
In production, we will replace the credentials with Environment variables.

⇒ To actually communicate with the database, we need to create & use a Session. This step is to be done after creating the engine.

```
SessionLocal = sessionmaker(autocommit = False,
                            autoflush = False, bind = engine)
```
engine → our created engine.

⇒ Then, we need to have the following code :-

```
Base = declarative_base()
```
declarative_base() → imported at beginning
Base ↑

This Base class must be extended / inherited by all the model classes that we will define to create tables in our Postgres database.

4:42:30

- The documentation of these things can be found online at :- https://fastapi.tiangolo.com/tutorial/sql-databases

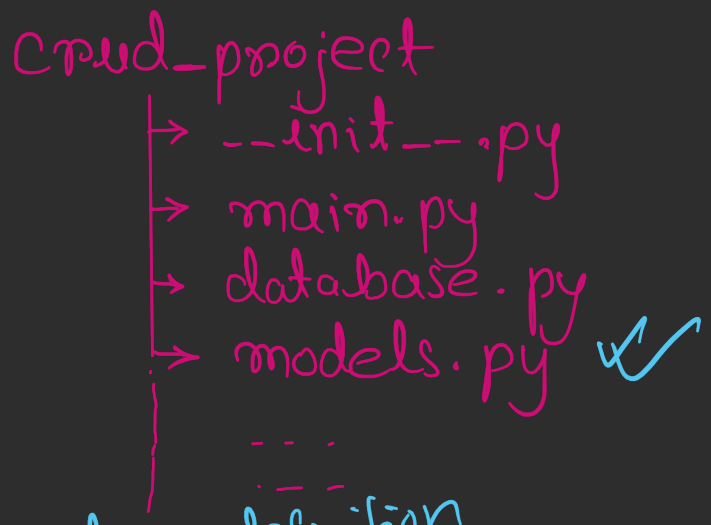
- Using sessionmaker() method, we are making the SessionLocal named class. Now, each object of this SessionLocal class will be a database session. The class itself is not a database session yet.

- The codes written till now in database.py file can be copy-pasted in each project, which uses connection with Postgresql database, using SQLAlchemy.

Making Model Classes for Tables :-

- Now, we will define model classes, that will be connected to Postgres database tables.

- Creating model classes will be done in a file named conventionally as models.py. It will be in the same directory as database.py. The folders structure will be as follows :-



model class definition looks like this

```python
from sqlalchemy import Column, ForeignKey, Integer, String, Boolean
from sqlalchemy.orm import relationship

from .database import Base

class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key = True, nullable = False)
    title = Column(String, nullable = False)
    content = Column(String, nullable = False)
    published = Column(Boolean, nullable = False, default=True)
```

not null constraint

server_default = 'TRUE'

- Each model class must inherit the Base class imported
- The name of each Base class is for use in Python code. But, the table name corresponding to it in the database can be set to something else. The table name (in database) is to be assigned to the __tablename__ variable, as a string, as a member variable of model class.

- For making columns in table, we need to make static member variables in class. Names of columns in table will be same as names of variables.

⇒ For making columns, each variable should be assigned with a Column() object.
The Column() constructor essentially takes the datatype as 1st argument. Some datatypes are :- Integer, String, Boolean.

⇒ Details such as Not null, Primary key constraints, Default value can be passed as keyword arguments to Column()

4:47:57

⇒ Now, in our main.py file, we need to write the following line, to create all our models :-

```
from .database import SessionLocal, engine
from . import models

models.Base.metadata.create_all(bind=engine)

app = FastAPI()
```

this line creates tables from orm models, when executed. (Table will be created only if it do not exists)

The create the dependency :-

```
def get_db():                    #Dependency
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

⇒ The SessionLocal object creates a connection with the database. So, the get_db() function will create a connection session to the database.

Thus, everytime the API gets a request, it will create a session to the database, execute SQL queries and operations on the database, and after that, the session / connection to the database will be closed.

By calling the get_db() function, we can perform the above said thing.

⇒ Now, we have got our Dependency function, to every Path Operation function, we need to pass another parameter that will handle connections to database, (through ORM library). The parameter will be as follow :-

```
db: Session = Depends(get_db)
```

For using the above parameter:

```
from FastApi import Depends
from sqlalchemy.orm import Session
```

Issue in Sqlalchemy :-

⇒ With all the above coding, we saw that Sqlalchemy creates the database table only if it does not exist beforehand. If the table exist, then the

library will do nothing.

So, suppose, we have the table already created. Now, we made some changes to any column(s) of the model class, in terms of Datatypes or Constraints. After doing that, if we again restart our program, the changes will not be reflected in the table.
This is because, once Sqlalchemy finds that table already exists, it does nothing further.

To track these changes automatically, such that if any changes is made to database schema (such as changing of column names, datatypes or constraints) in the model class, then the actual database tables should also get updated accordingly automatically. To do this, we need another library, called Alembic.

⇒ Model class definition, with "created_at" field:-

```python
from sqlalchemy import Column, Integer, String, Boolean
from sqlalchemy.sql.expression import text
from sqlalchemy.sql.sqltypes import TIMESTAMP

from .database import Base


class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, nullable=False)
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
    published = Column(Boolean, server_default='TRUE', nullable=False)
    created_at = Column(TIMESTAMP(timezone=True), nullable=False, server_default=text('now()'))
```

- The datatype of the created_at field is a special object called TIMESTAMP. Its constructor takes argument, using which, use of timezone is set.

- As the default value, we want to set an SQL database function. So, to set that, the function is passed as string to a function called text() and its return value will be the default value for each record.