1.SQLAlchemy Overview:
- SQLAlchemy is an open-source SQL toolkit and Object-Relational Mapper that:
    - gives developers the full power and flexibility of SQL
    - provides a set of high-level abstractions that allow you to interact with DB using Python code,
        make DB operations more intuitive and less error-prone

1.1. ORM
- The ORM component is optional and can be used independently
- The ORM allows you to define Python classes(models) that correspond to DB tables:
    - encapsulating the schema
    - providing an object-oriented way to interact with the DB
- The ORM also handles the translation between Python objects and DB records

1.2. Engine
- Engine is the core of SQLAlchemy
- Provides a source of connectivity to a DB
- It manages the connection pool
- Handles the low-level details of DB communication

1.3. SQL Expression Language
- Allows you to build and manipulate SQL queries using Pythonic syntax
- Makes it easier to construct complex queries without writing raw SQL strings

1.4.Session
- Provides a high-level interface for managing interactions with the DB
- Acts as an unit of work, allowing you to:
    - CUD records
    - Use Python objects
    - commit changes to the DB

2.Installation and Configuration

```
pip install sqlalchemy
pip install psycopg2
```

- In main.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base

DATABASE_URL = 'postgresql+psycopg2://your_username:your_password@your_host/your_database'
engine = create_engine(DATABASE_URL)
```

3.Defining Models

```
from sqlalchemy.orm import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    username = Column(String)
    email = Column(string)

#Create tables in the DB (no migrations management)
Base.metadata.create_all(engine)
```

4.Migrations:
- Migrations are a way to manage changes to a DB schema over time
- In SQLAlchemy, migrations are not a built-in feature(like they are in Django)
- There are tools and libraries that work alongside SQLAlchemy to handle migrations(like Alembic)

4.1.Alembic: a popular migration tool for SQLAlchemy. Provides way to:
- manage and apply changes to your DB schema using Python scripts
- also supports managing migrations for multiple environments(e.g.: development, testing, production)

```
pip install alembic
alembic init alembic

sqlalchemy.url = postgresql+psycopg2://username:password@localhost/db_name
```

5.Queries and CRUD Operations:
5.1.Create a Session: to interact with the DB, you'll need to create a session
- using the sessionmaker function
- this session will act as a unit of work for your DB operations

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaer(bind=engine)
session = Session()
with Session() as session:
    ...
```

6.Transactions:
- A transaction is a sequence of one or more DB operations that are executed as a single unit of work
- Transactions are used to ensure data integrity and consistency in the DB
- In SQLAlchemy you can use transactions:
  - to group a series of DB operations together
  - ensure that they are either all executed successfully or none of them are

7.Simple Relations

8.DB Pooling:
- DB connection pooling is a technique used to efficiently manage and reuse DB connections
- Instead of opening and closing a new DB connection for every request or operation, a connection pool maintains a set of pre-established DB connections that can be reused

9.Django ORM vs SQLAlchemy:
9.1. Django ORM:
- tightly integrated with the Django web framework
- high-level abstraction
- built-in migration system
- powerful admin interface
- authentication and authorization

9.2. SQLAlchemy:
- a standalone library that can be used independently
- lower-level control
- no built-in migration capabilities
- no built-in admin interface

- multiple DB

Can be more convenient when:
- your app mostly works with aggregations
- you have a lot of data
- you need precise and performant queries
- you're transformnig complex queries from SQL to Python
- your DB is not natively supported by Django(SQL Azure, Sybase, Firebird)