SQLAlchemy

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#python-db

SQLAlchemy Overview

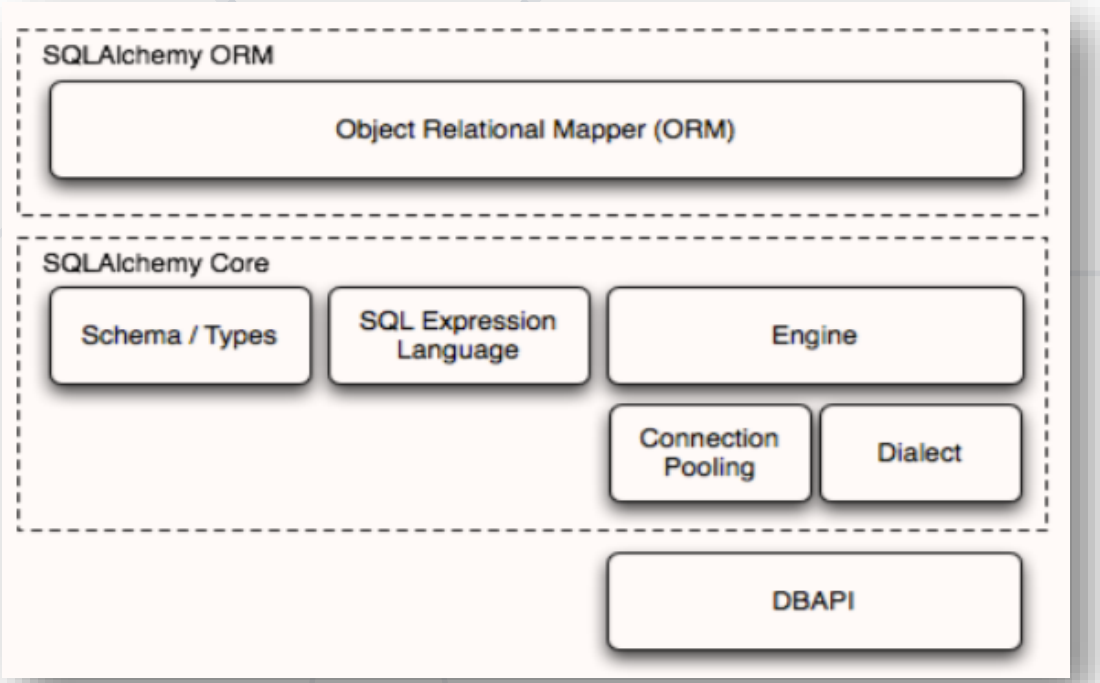
Key Concepts

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 databases using **Python** code
    - making **database operations** more **intuitive** and **less error-prone**
  - designed to work with a variety of **database systems** like
    - PostgreSQL, MySQL, SQLite, and more

SQLAlchemy Overview (2)

- **SQLAlchemy** has **two** most significant front-facing **components**
  - the **Object Relational Mapper** **(ORM)**
  - the **Core**



SQLAlchemy Key Concepts

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

SQLAlchemy Key Concepts (2)

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

SQLAlchemy Key Concepts (3)

- **Session**
  - Provides a **high-level** interface for managing **interactions** with the **database**
  - Acts as a **unit of work**, allowing you to
    - **create**, **update**, and **delete** records
    - use **Python objects**
    - **commit** changes to the database

More at: https://docs.sqlalchemy.org/en/20/tutorial/index.html

Installation & Configuration

Connecting to Database

Installation & Configuration

- Install **SQLAlchemy**

```
pip install sqlalchemy
```

- Install a **PostgreSQL** Driver

```
pip install psycopg2
```

- Import Required **Modules**

```
# main.py
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base
```

Connecting to Database

- Create a **Database Connection**
  - Use the `create_engine` function to **establish** a **connection** to your PostgreSQL database
  - Replace `your_username`, `your_password`, `your_host`, and `your_database` with your **PostgreSQL credentials** and database information

```
# main.py
DATABASE_URL =
'postgresql+psycopg2://your_username:your_password@your_host
/your_database'

engine = create_engine(DATABASE_URL)
```

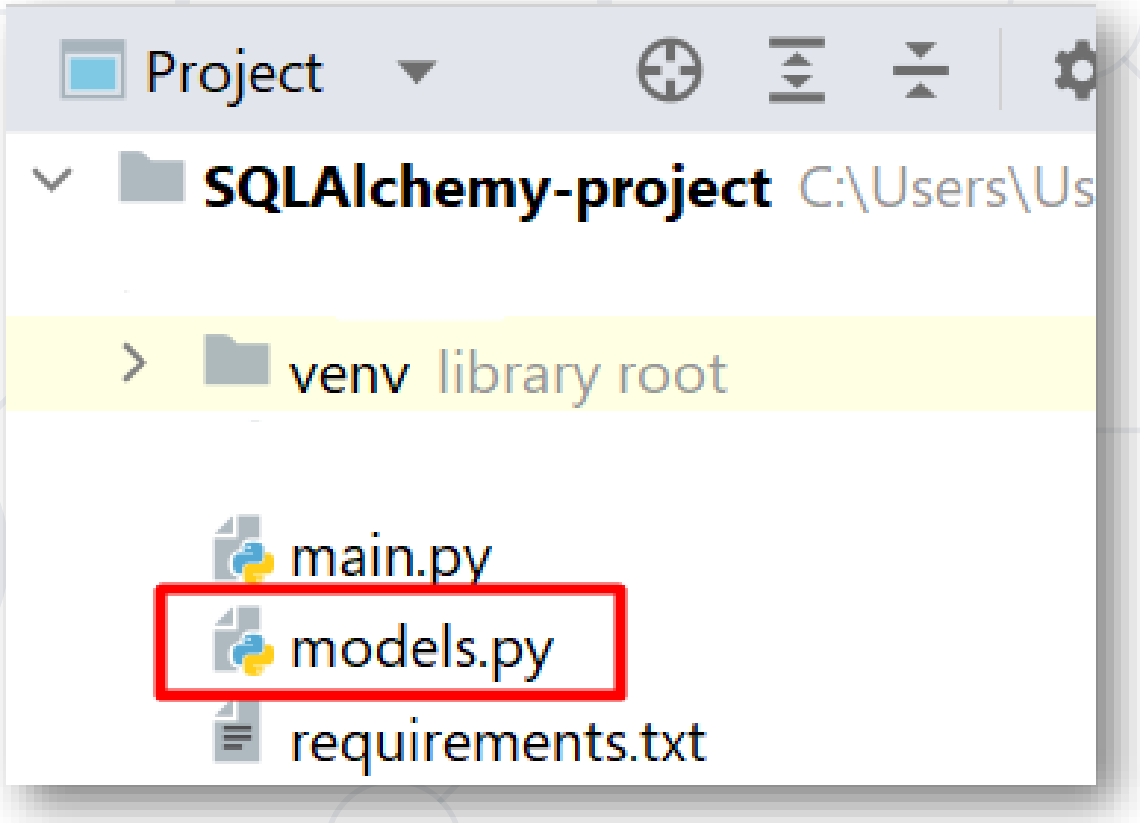
Defining Models

Defining a Model

```
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 database (no migrations management)
Base.metadata.create_all(engine)
```

Defining a Model (2)

- Create a `models.py` file in your **project** directory
- Move your **User model** there



Migrations

Alembic

Migrations

- **Migrations** are a way to **manage changes** to a database **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**
  - **Alembic** is one of these tools

Alembic

- **Alembic** is a popular **migration** **tool** for **SQLAlchemy**
- It provides a way to
  - **manage** and **apply changes**
  - to your database **schema**
  - using **Python** scripts
- **Alembic** also **supports** managing **migrations** for **multiple environments**
  - e.g., development, testing, production

Install and Configure Alembic

- **Install Alembic**

```
pip install alembic
```

- **Initialize Alembic**

```
alembic init alembic
```

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

Change the default sqlalchemy.url value

Replace with your PostgreSQL credentials here

Install and Configure Alembic (2)

- Indicate what Alembic should **compare against** when **generating migration scripts**

models.py | alembic.ini | env.py

```python
# This line sets up loggers basically.
if config.config_file_name is not None:
    fileConfig(config.config_file_name)

# add your model's MetaData object here
# for 'autogenerate' support
from models import Base
target_metadata = Base.metadata
# target_metadata = None

# other values from the config, defined by the needs
# can be acquired:
```

Using Alembic (1)

- Create a Migration

```
alembic revision --autogenerate -m "Add User Table"
```

- Apply Migrations

```
alembic upgrade head
```

- Downgrade (Rollback) Migrations

```
alembic downgrade -1
```

Using Alembic (2)

Terminal: Local × Command Prompt ×

```
(venv) C:\Users\User\                    \PythonProjects\SQLAlchemy-project>alembic revision --autogenerate -m "Add User Table"
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'users'
Generating C:\Users\User\                    \PythonProjects\SQLAlchemy-project\alembic\versions\b76c55ffc32e_add_user_table.py ...  done
```

SQLAlchemy-project 〉 alembic 〉 versions 〉 b76c55ffc32e_add_user_table.py

Project

- SQLAlchemy-project C:\Users\User\Dov
  - alembic
    - versions
      - b76c55ffc32e_add_user_table.py
    - env.py
    - README
    - script.py.mako
  - venv library root
  - alembic.ini
  - main.py

main.py × alembic.ini × env.py × b76c55ffc32e_add_user_table.py ×

```
1   """Add User Table
2
3   Revision ID: b76c55ffc32e
4   Revises:
5   Create Date: 2023-08-30 10:40:33.239515
6
7   """
8   import ...
12
```

Using Alembic (3)

Software University

.py alembic.ini env.py b76c55ffc32e_add_user_table.py

```python
# revision identifiers, used by Alembic.
revision: str = 'b76c55ffc32e'
down_revision: Union[str, None] = None
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('users',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('username', sa.String(), nullable=True),
    sa.Column('email', sa.String(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('users')
    # ### end Alembic commands ###
```

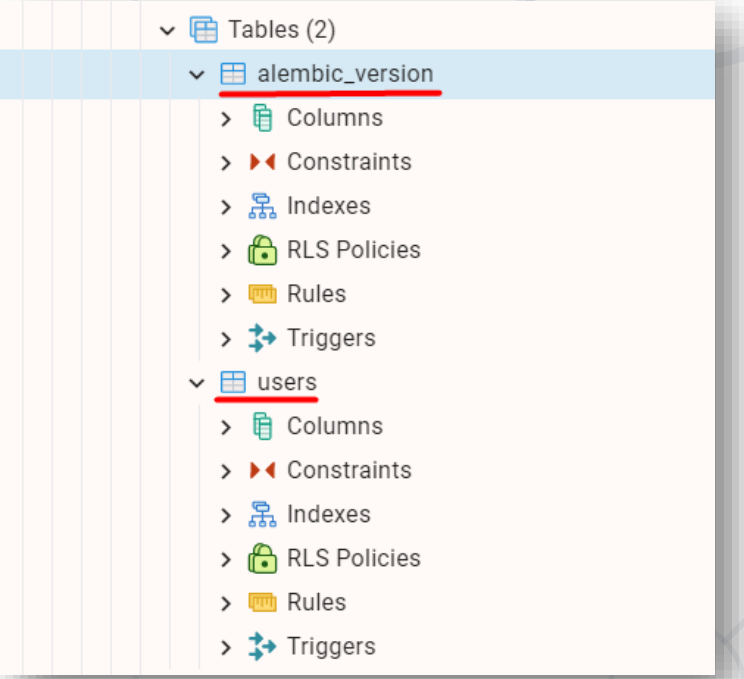
First migration

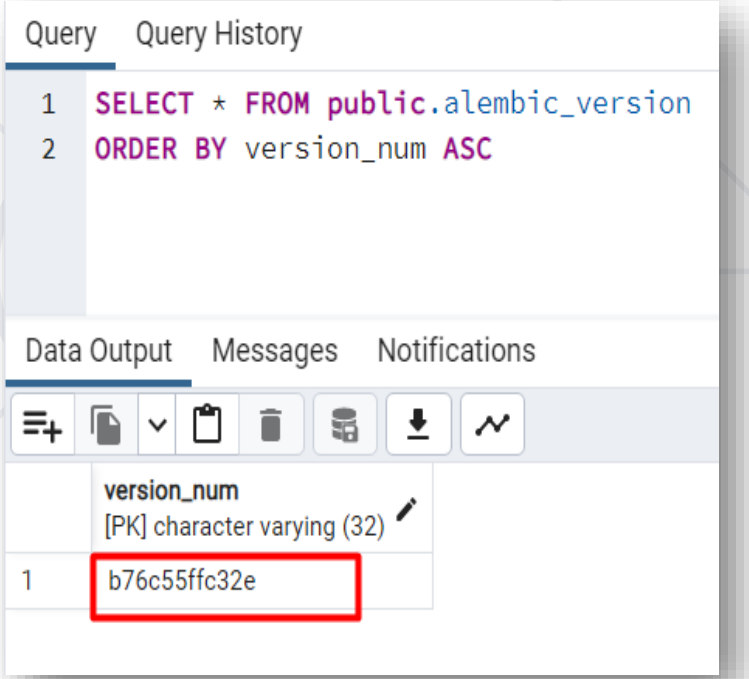
Defines upgrade

Defines downgrade

Using Alembic - Upgrade

```
(venv) C:\Users\User\                          \PythonProjects\SQLAlchemy-project>alembic upgrade head
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> b76c55ffc32e, Add User Table
```

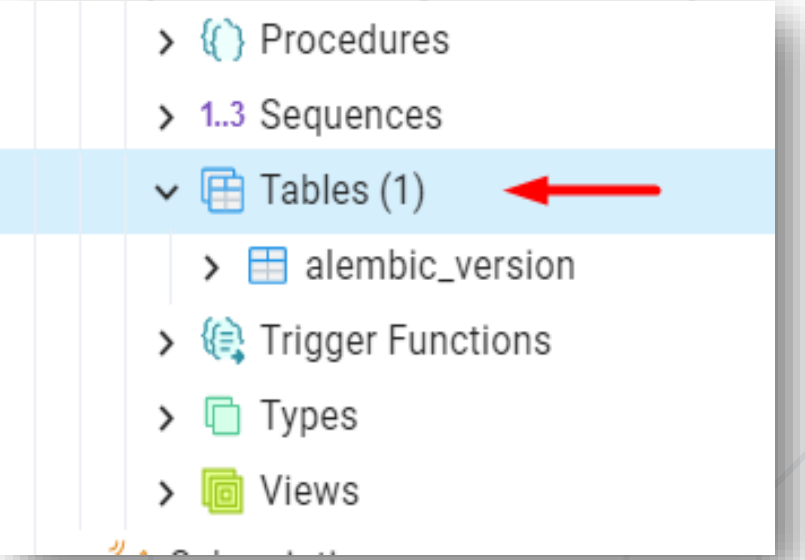


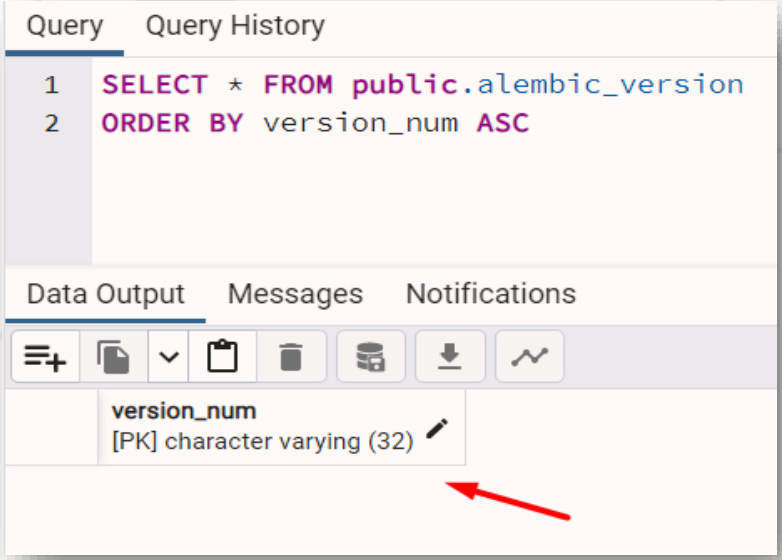


Using Alembic - Downgrade

Terminal: Local × Command Prompt × + ⌄

```
(venv) C:\Users\User\                          \PythonProjects\SQLAlchemy-project>alembic downgrade -1
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running downgrade b76c55ffc32e -> , Add User Table
```





Queries and CRUD Operations

Queries and CRUD

- Create a **Session**
  - To **interact** with the **database**, you'll need to **create** a **session**
    - using the **sessionmaker** function
    - This **session** will act as a **unit of work** for your **database operations**

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)

session = Session() # be careful when using this

with Session() as session: # a good practice
    ...
```

Queries and CRUD (2)

- Perform **Database Operations**
  - With the **session created**, you can now perform various **database operations** using **SQLAlchemy's ORM**
  - For example, to **add** a **new user** to the **database**

```
with Session() as session:
    new_user = User(username='john_doe', email='john@example.com')
    session.add(new_user)
    session.commit()
```

Queries and CRUD (3)

- **Querying Data**
  - You can also use **SQLAlchemy** to **query data** from the database
  - For example, to **retrieve all users**

```
with Session() as session:
    users = session.query(User).all()
    for user in users:
        print(user.username, user.email)
```

Queries and CRUD (4)

- **Updating** a user

```
# Query the user you want to update
user_to_update =
session.query(User).filter_by(username='john_doe').first()

# Update the user's information
if user_to_update:
    user_to_update.email = 'new_email@example.com'
    session.commit()
    print("User updated successfully")
else:
    print("User not found")
```

Queries and CRUD (5)

- **Deleting** a user

```
# Query the user you want to delete
user_to_delete =
session.query(User).filter_by(username='john_doe').first()

# Delete the user
if user_to_delete:
    session.delete(user_to_delete)
    session.commit()
    print("User deleted successfully")
else:
    print("User not found")
```

Transactions

Transactions

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

Transactions - Example

```python
from main import Session
from models import User

# Start a session
session = Session()

try:
    # Begin a transaction
    session.begin()

    # Perform database operations within the transaction
    ...

    # Commit the transaction
    session.commit()

except Exception as e:
    # Rollback the transaction if an exception occurs
    session.rollback()
    print("An error occurred:", str(e))

finally:
    # Close the session
    session.close()
```

Import the created Session

Open a session for the whole unit of work

Close the session no matter if the transaction failed or succeeded

Transactions – Example (2)

Software University

Project

- SQLAlchemy-project C:\Users\User\Dow
  - alembic
    - versions
    - env.py
    - README
    - script.py.mako
  - venv library root
  - alembic.ini
  - main.py
  - models.py
  - transactions.py
- External Libraries
- Scratches and Consoles

models.py | transactions.py | main.py

```python
from main import Session
from models import User

# Start a session
session = Session()

try:
    # Begin a transaction
    session.begin()

    # Perform database operations within the transaction
    session.query(User).delete()

    # Commit the transaction
    session.commit()
    print("All users deleted successfully")

except Exception as e:
    # Rollback the transaction if an exception occurs
    session.rollback()
    print("An error occurred:", str(e))

finally:
    # Close the session
    session.close()
```

Transactions – Result

Run: transactions

```
"C:\Users\User\
All users deleted successfully

Process finished with exit code 0
```

public.users/sqlAlchemy_db/postgres@PostgreSQL 15

No limit

Query | Query History

```
1 SELECT * FROM public.users
2 ORDER BY id ASC
```

Data Output | Messages | Notifications

id [PK] integer	username character varying	email character varying	created_on timestamp without time zone	rating double precision

Simple Relations

Defining a Relation

```python
# models.py
from sqlalchemy import Column, Integer, String, Float, DateTime, Boolean, ForeignKey
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class User(Base):
    ...
# Many-to-one relationship
class Order(Base):
    __tablename__ = 'orders'

    id = Column(Integer, primary_key=True)
    is_completed = Column(Boolean, default=False)
    user_id = Column(Integer, ForeignKey('users.id'))
    user = relationship('User')
```

Populate Order Table

```
# main.py
# Populate Order table

def populate_order_table():
    with Session() as session:

        session.add_all((Order(user_id=1), Order(user_id=2)))

        session.commit()
```

Populate with existing user id

Queries for Relationships

```
# Relationships queries

def relationship_query():
    with Session() as session:
        orders =
session.query(Order).order_by(Order.user_id.desc()).all()
        if not orders:
            print("No orders yet.")
            return
        for order in orders:
            user = order.user
            print(f'Order number {order.id}, Is completed:
{order.is_completed}, Username: {user.username}')
```

Descending order

Referring to FK

Referring to object

Database Pooling

Database Pooling

- **Database connection pooling** is a technique
  - used to **efficiently manage** and **reuse** database **connections**
- Instead of **opening** and **closing** a **new** database **connection** for **every request** or **operation**
  - a **connection pool maintains** a set of **pre-established** database **connections** that can be **reused**

DB Connection Pooling - Example

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

# Create a connection pool using SQLAlchemy
DATABASE_URL = 'postgresql://username:password@localhost/database'
engine = create_engine(DATABASE_URL, pool_size=10, max_overflow=20)

# Create a session factory
Session = sessionmaker(bind=engine)

# Use sessions as needed
session = Session()

# Perform database operations using the session
...

# Close the session
session.close()
```

Replace with your PostgreSQL credentials

Pool size sets the initial number of connections in the pool

Max overflow specifies how many additional connections can be created when the pool is exhausted

Django ORM vs SQLAlchemy

Django ORM

- Tightly integrated with the Django web framework
- High-Level Abstraction
- Built-in Migration System
- Powerful Admin Interface
- Authentication and Authorization

SQLAlchemy

- A standalone library that can be used independently
- Lower-Level Control
- No Built-in Migration Capabilities
- No Built-in Admin Interface
- Multiple Databases

SQLAlchemy - When and Why

- **SQLAlchemy** 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 transforming **complex queries** from SQL to Python
  - You're building **advanced queries dynamically**
  - The database is **not natively supported** by Django (e.g., SQL Azure, Sybase, Firebird)

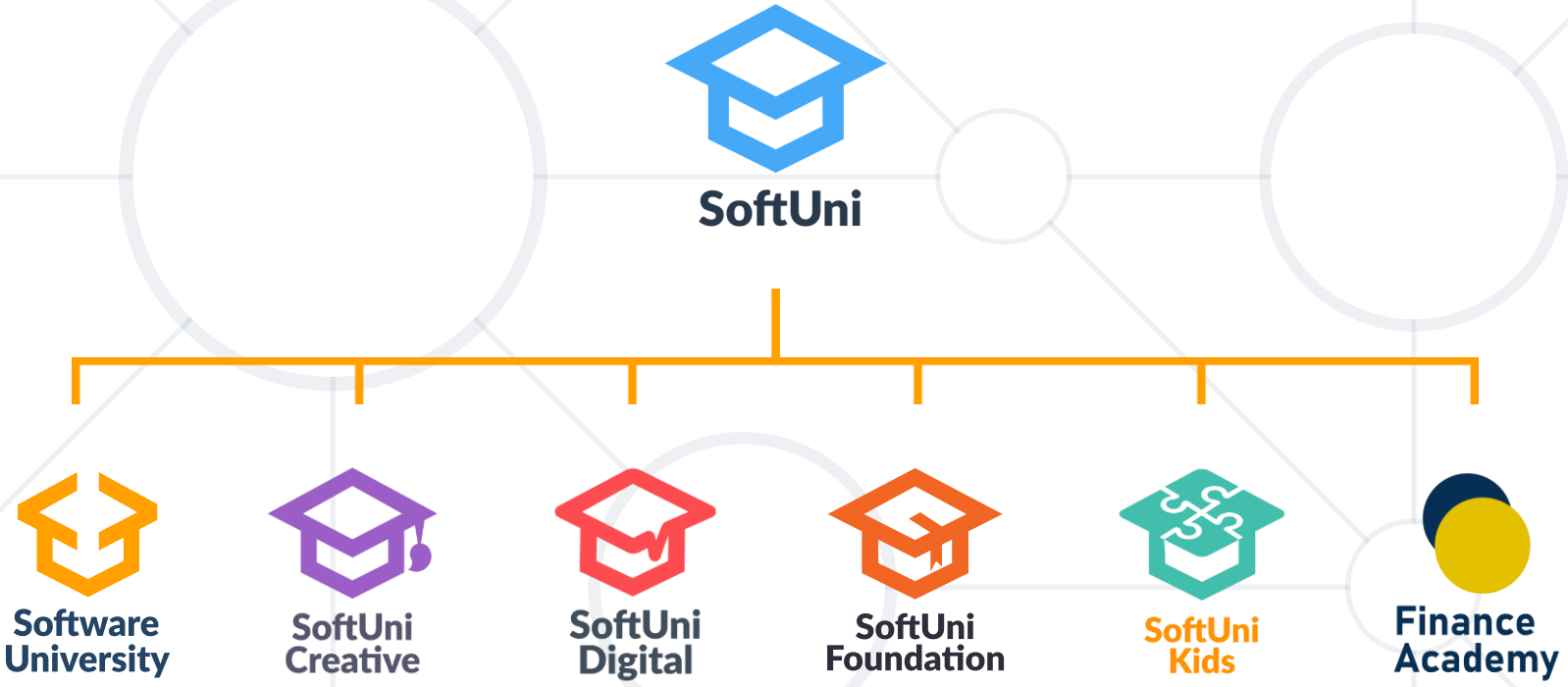
Live Demo

Live Demo in Class

Summary

- **SQLAlchemy** Overview
- **Installation** and **Configuration**
- Defining **Models**
- **Migrations**
  - Alembic
- **Queries** and **CRUD** Operations
- **Transactions**
- Simple **Relationships**
- DB **Connection Pooling**

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg