

# SQLAlchemy



**SQLAlchemy**

**SoftUni Team**  
**Technical Trainers**



**SoftUni**



**Software University**

<https://softuni.bg>

sli.do

**#python-db**

# Table of Contents

1. SQLAlchemy Overview
  - Key Concepts
2. Installation and Configuration
  - Connecting to Database
3. Defining Models
4. Migrations
5. Queries and CRUD Operations
6. Transactions
7. Simple Relations
8. Database Pooling





# SQLAlchemy Overview

Key Concepts

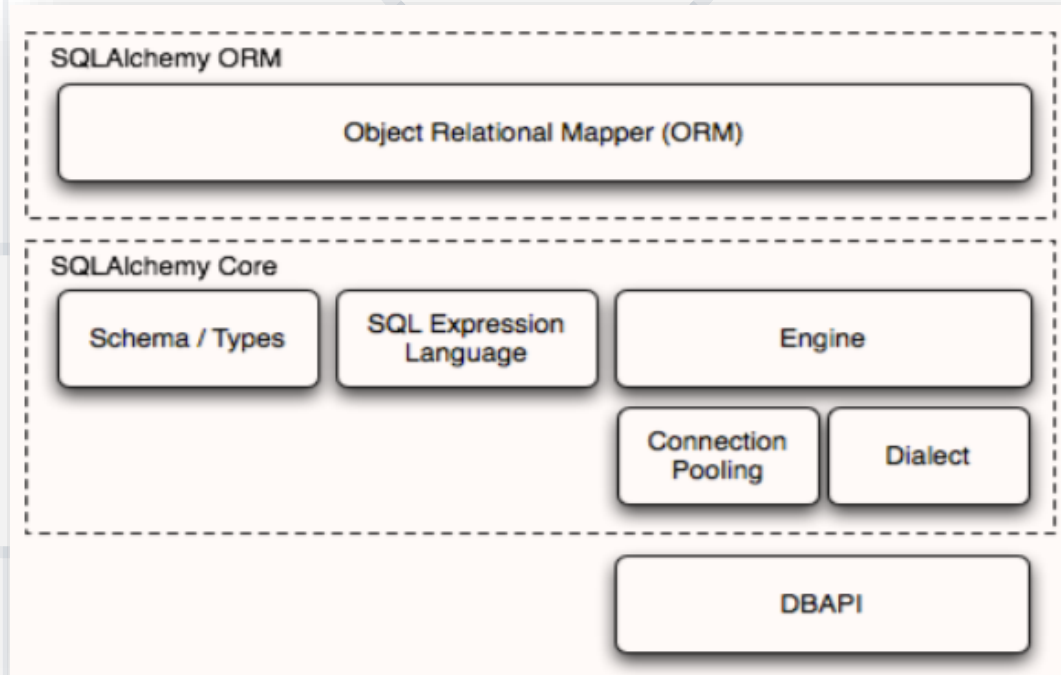
# SQLAlchemy Overview

- **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

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

- **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

- **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>

The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light beige oval, which is itself inside a dark blue circle. The background of the slide is white with a faint, light gray geometric pattern of lines and circles.

SQLA

# Installation & Configuration

Connecting to Database

- 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
```

- 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)
```

The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light pink oval, which is itself inside a dark blue circle. The background of the slide has a light gray geometric pattern of lines and circles.

SQLA

**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)
```

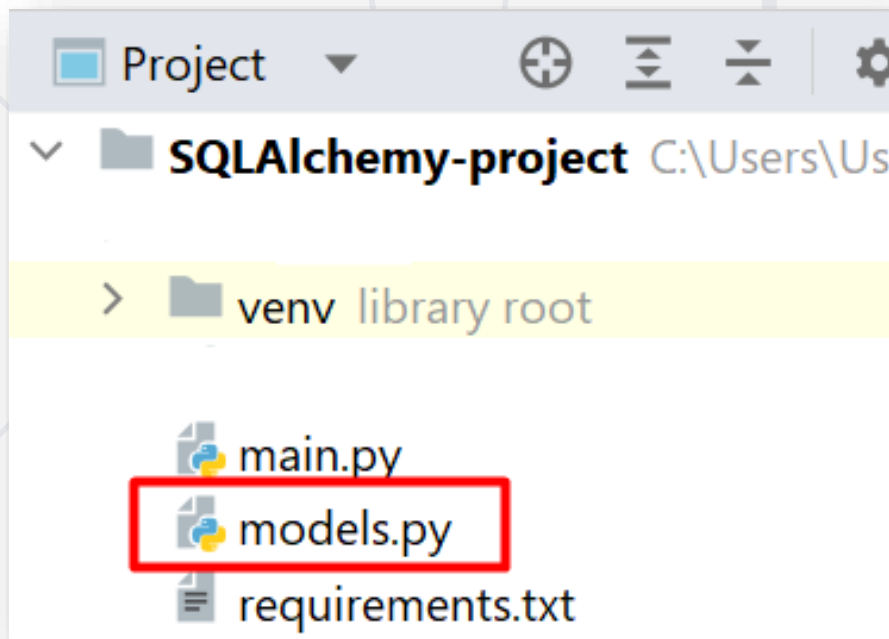
```
# Create tables in the database (if there is no migrations management)
```

```
Base.metadata.create_all(engine)
```

A manual approach for creating tables (not convenient when changes in models occur)

# Defining a Model

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



The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light pink oval, which is itself inside a dark blue circle. The background of the slide is white with a faint, light gray geometric pattern of lines and circles.

SQLA

**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
```

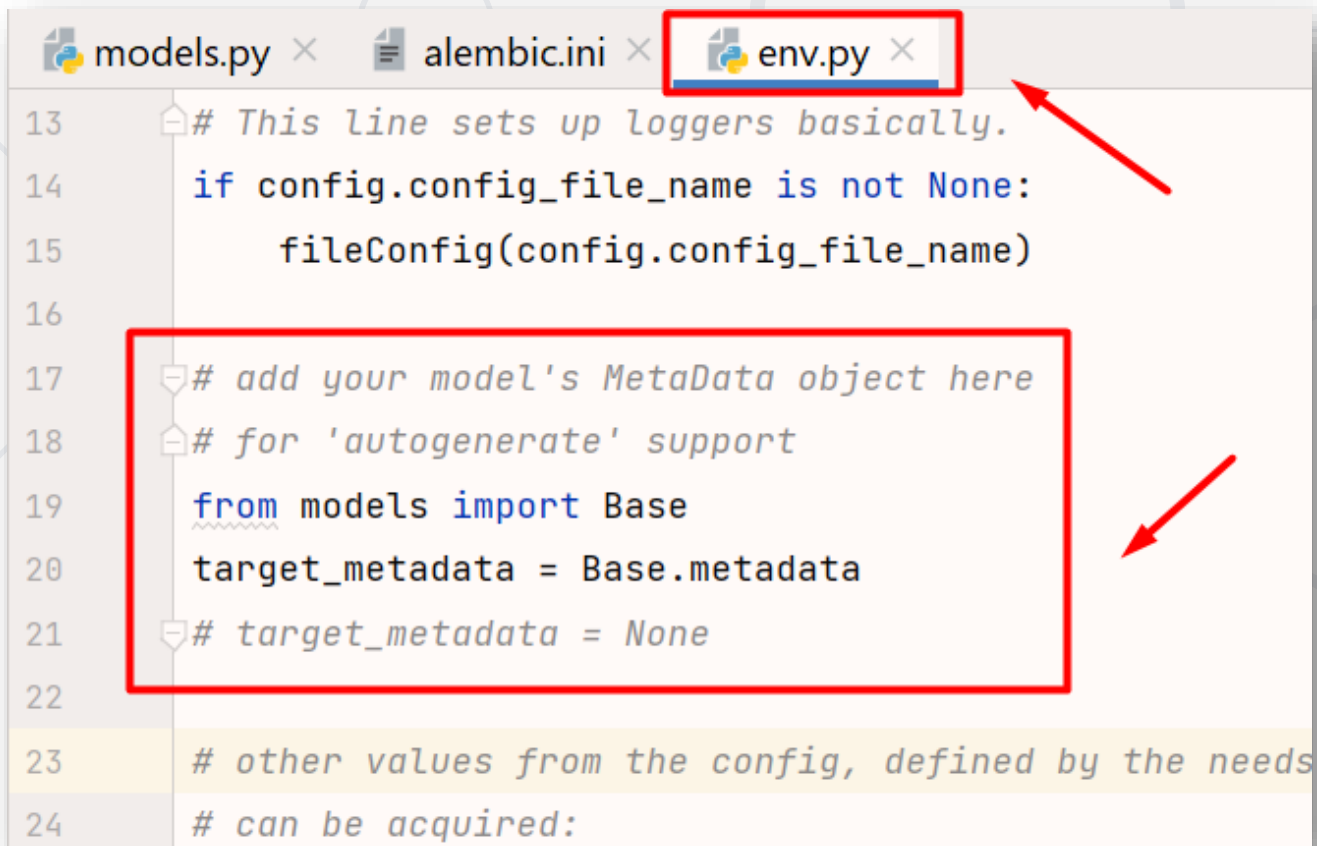
```
# alembic.ini  
sqlalchemy.url =  
postgresql+psycopg2://username:password@localhost/db_name
```

Change the default  
sqlalchemy.url value

Replace with your  
PostgreSQL credentials here

# Install and Configure Alembic

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



```
13 # This line sets up loggers basically.
14 if config.config_file_name is not None:
15     fileConfig(config.config_file_name)
16
17 # add your model's MetaData object here
18 # for 'autogenerate' support
19 from models import Base
20 target_metadata = Base.metadata
21 # target_metadata = None
22
23 # other values from the config, defined by the needs
24 # can be acquired:
```

- Create a **Migration**

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

- Apply **Migrations**

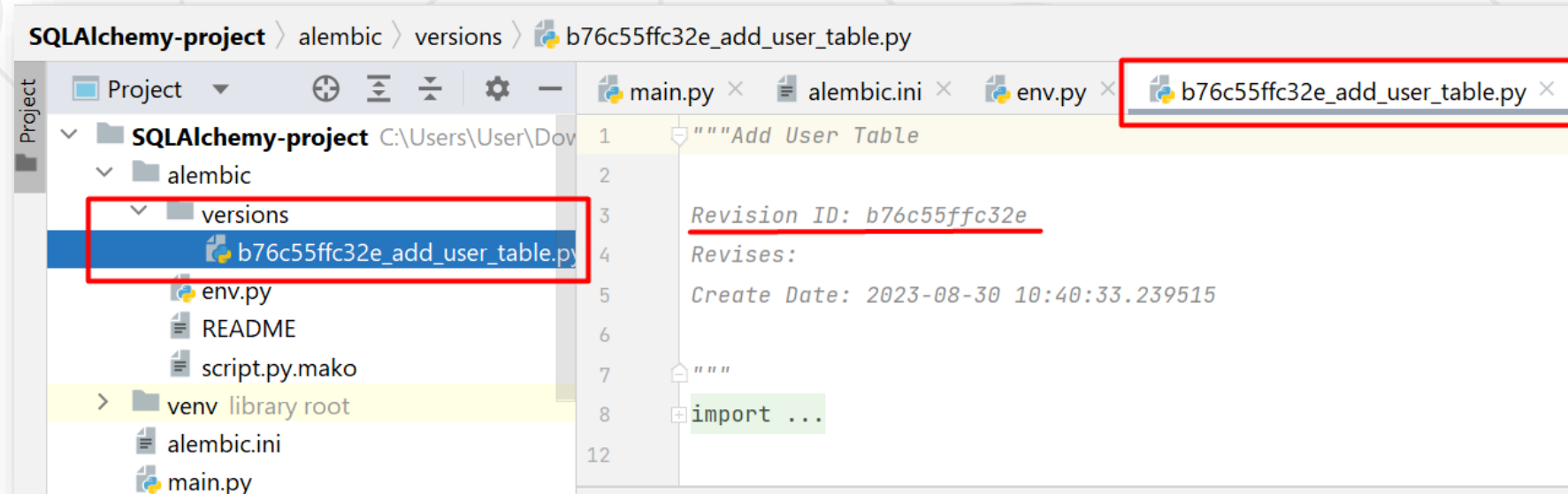
```
alembic upgrade head
```

- Downgrade (Rollback) **Migrations**

```
alembic downgrade -1
```

# Using Alembic

```
Terminal: Local x Command Prompt x + v
(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
```



```
n.py x alembic.ini x env.py x b76c55ffc32e_add_user_table.py x
# 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 ###
```

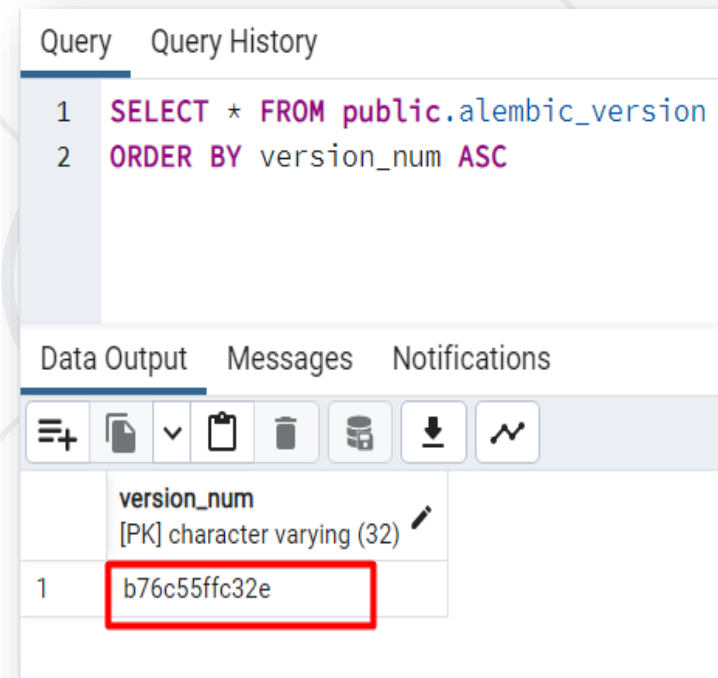
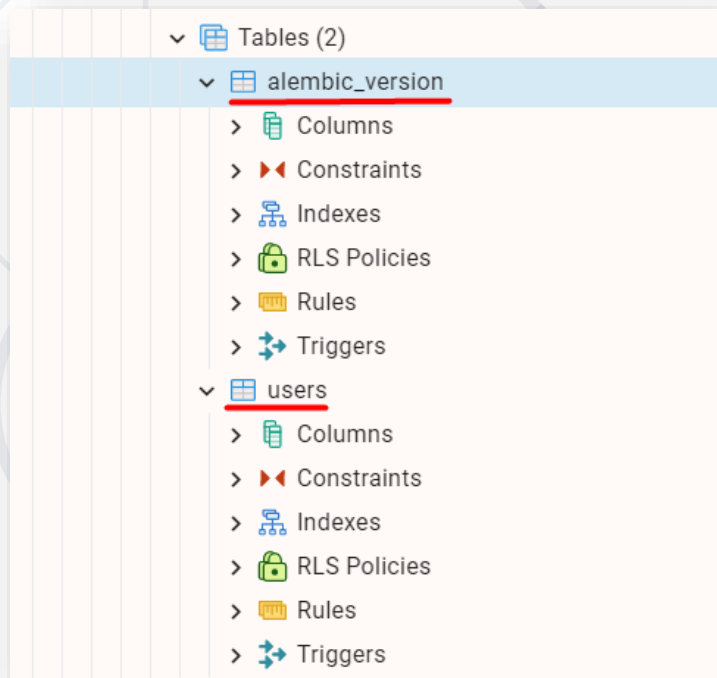
First migration

Defines upgrade

Defines  
downgrade

# Using Alembic - Upgrade

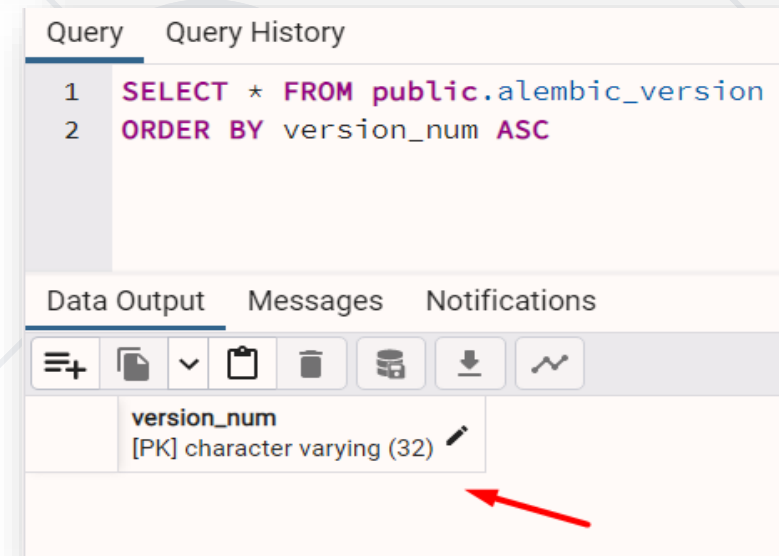
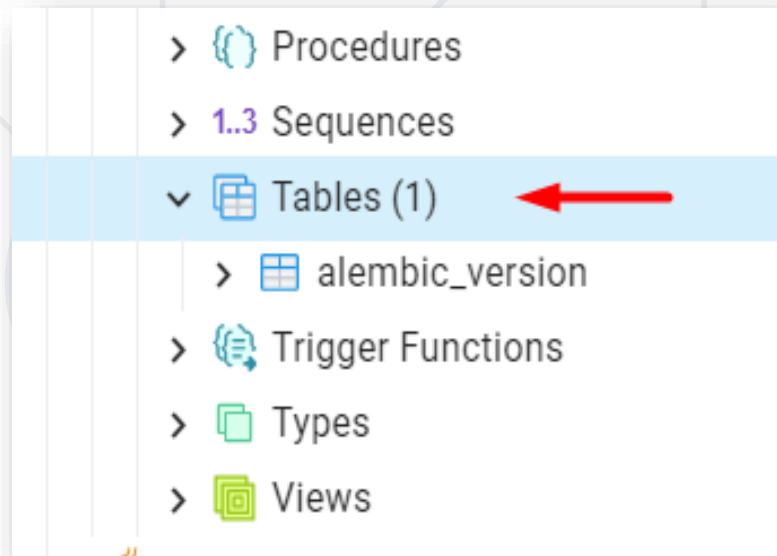
```
(venv) C:\Users\User\... \PythonProjects\SQLAlchemy-project>alembic upgrade head  
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.  
INFO [alembic.runtime.migration] Will assume transactional DDL.  
INFO [alembic.runtime.migration] Running upgrade -> b76c55ffc32e, Add User Table
```





# Using Alembic - Downgrade

```
Terminal: Local x Command Prompt x + v
(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 b76c55fffc32e -> , Add User Table
```



The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light beige oval, which is itself surrounded by a thick dark blue ring. The background of the slide is white with a faint, light gray geometric pattern of lines and circles.

SQLA

# Queries and CRUD Operations

- 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  
    ...
```

- 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()
```

- **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)
```

- **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")
```

- **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")
```

The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light pink oval, which is itself inside a dark blue circle. The background of the entire image is a light gray network of lines and circles.

SQLA

**Transactions**



# Transactions

- A **transaction** is a **sequence** of one or more database **operations**
  - 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

```
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:
```

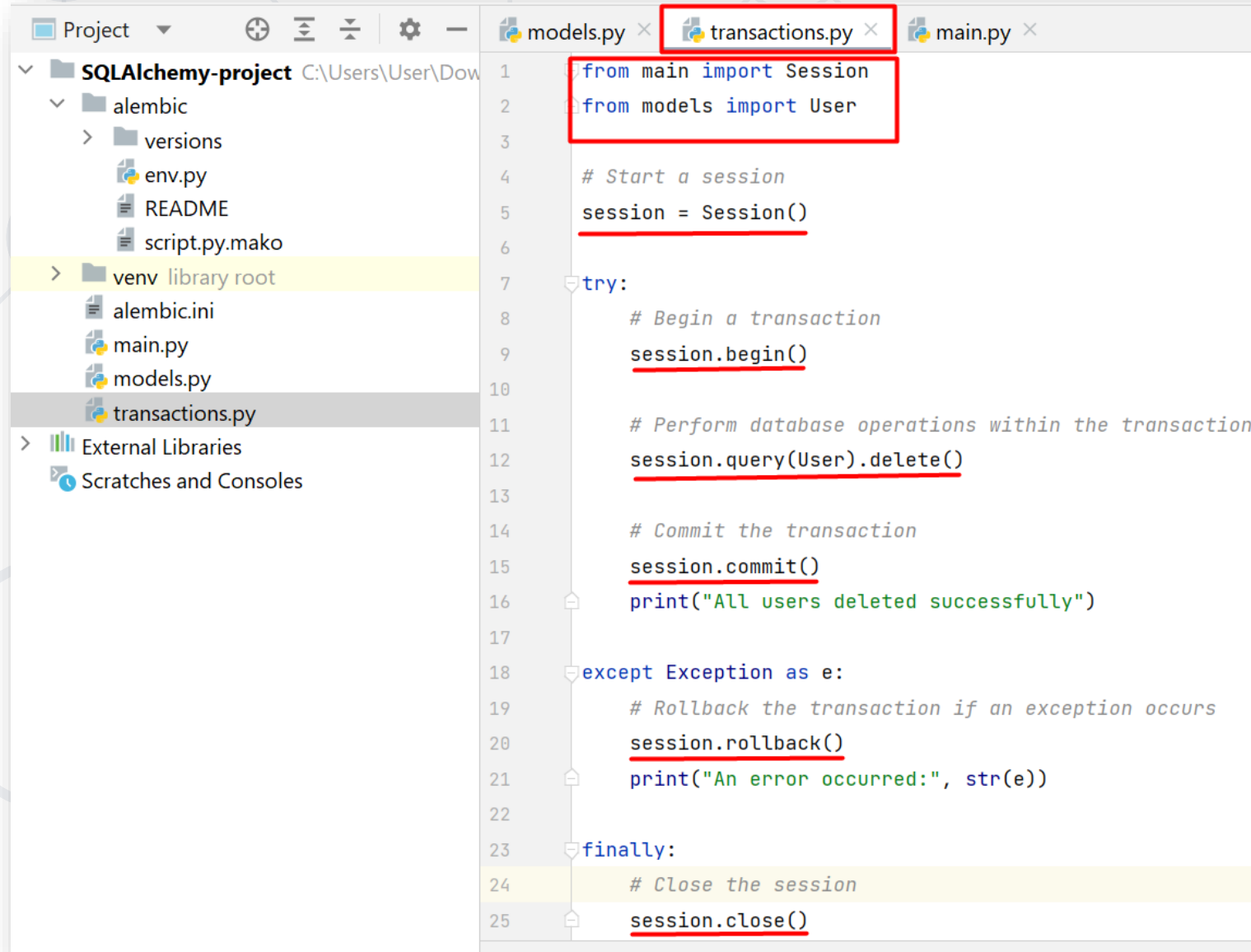
```
# Close the session
session.close()
```

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



The screenshot shows an IDE window with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'SQLAlchemy-project' with a subdirectory 'alembic' containing 'versions', 'env.py', 'README', and 'script.py.mako'. A 'venv library root' is also shown with 'alembic.ini', 'main.py', 'models.py', and 'transactions.py'. The code editor shows the 'transactions.py' file with the following code:

```
1 from main import Session
2 from models import User
3
4 # Start a session
5 session = Session()
6
7 try:
8     # Begin a transaction
9     session.begin()
10
11     # Perform database operations within the transaction
12     session.query(User).delete()
13
14     # Commit the transaction
15     session.commit()
16     print("All users deleted successfully")
17
18 except Exception as e:
19     # Rollback the transaction if an exception occurs
20     session.rollback()
21     print("An error occurred:", str(e))
22
23 finally:
24     # Close the session
25     session.close()
```

# Transactions - Result

The screenshot shows a Python IDE window titled "Run: transactions". The output console displays the following text:

```
"C:\Users\User\...  
All users deleted successfully  
Process finished with exit code 0
```

Below the output console, a database client window is open, showing a connection to "public.users/sqlAlchemy\_db/postgres@PostgreSQL 15". The query editor contains the following SQL query:

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

The "Data Output" tab is active, showing a table with the following columns and data types:

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

A red arrow points to the "created\_on" column header.

The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light beige oval, which is itself inside a dark blue circle. The background of the slide is white with a faint, light gray geometric pattern of lines and circles.

SQLA

**Simple Relations**

# Defining a Relation

```
# 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

## # 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



The logo features the text "SQLA" in a serif font, with "SQL" in black and "A" in red. It is centered within a light pink oval, which is itself inside a dark blue circle. The background of the slide is white with a light gray geometric pattern of lines and circles.

SQLA

# 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

```
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

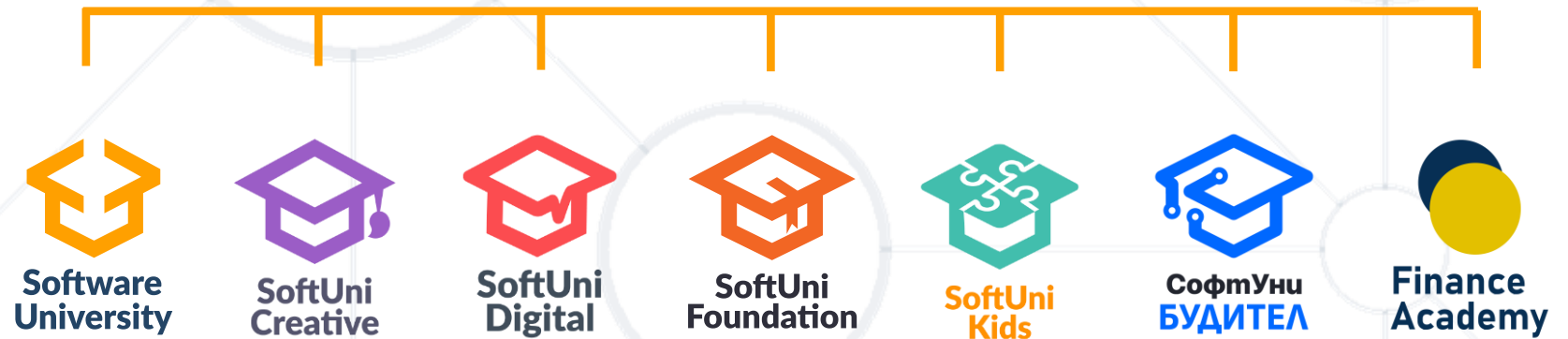
- **SQLAlchemy** can be more convenient when:
  - Your app **mostly works** with **aggregations**
  - You have **large sets** of data
  - You need **precise** and **performant** queries
  - You're transforming **complex** queries from SQL to Python
  - You're building **advanced queries dynamically**
  - The database is **not** **natively supported** by Django (e.g., SQL Azure, Sybase, Firebird)



- **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 – High-Quality Education, Profession and Job for Software Developers
  - [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)
- Software University Foundation
  - [softuni.foundation](http://softuni.foundation)
- Software University @ Facebook
  - [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)



- 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>

