Using a Database with FastAPI

Reindert-Jan Ekker

@rjekker www.codesensei.nl

Overview

SQLModel

- Built on SQLAlchemy + Pydantic

Create data model classes

Create a DB connection

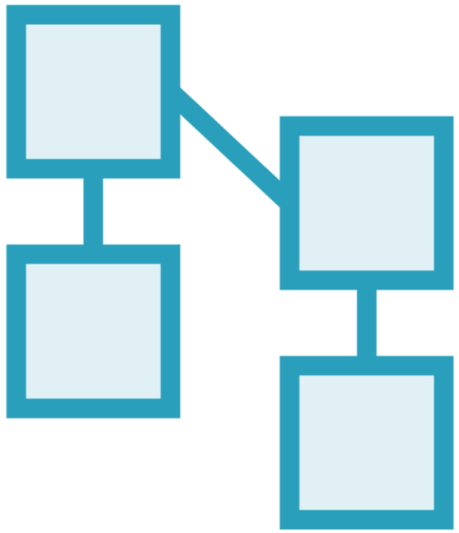
CRUD operations

Relations

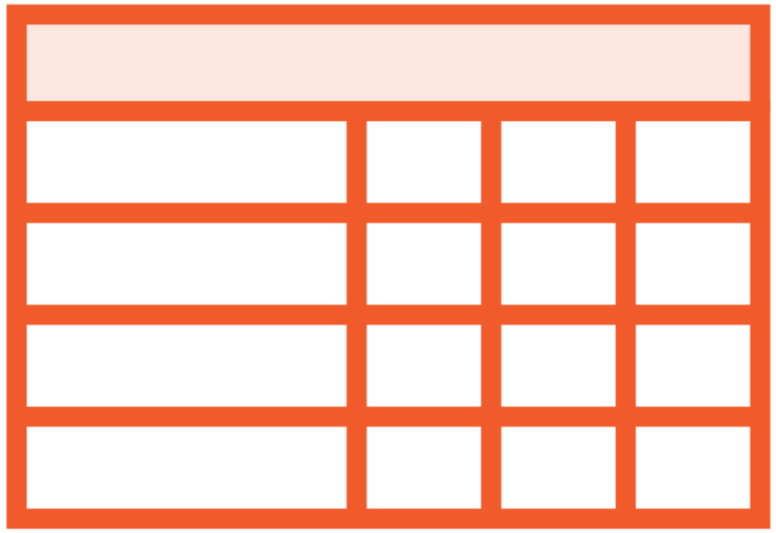
Transactions

- Session

Object-Relational Mapping



Python
Classes
Objects
Attributes



Relational DB (SQL)
Tables
Rows
Columns

Based on SQLAlchemy

- Popular ORM library
- Mature, robust
- Supports many databases

Also based on Pydantic

- Model classes are Pydantic Models
- Allows easy integration with FastAPI
- Same creator (Sebastián Ramírez)

SQLModel

- New, still being developed
- Gives access to the power of SQLAlchemy
- https://sqlmodel.tiangolo.com/

SQLModel and Async

We will not use async with SQLModel

Regular functions

FastAPI still runs those concurrently

SQLAlchemy async support still beta

Data Model Classes

```python
from sqlmodel import SQLModel, Field

# SQLModel inherits from pydantic BaseModel
# Pass table=True when creating the class to map this to a DB table


class Car(SQLModel, table=True):
    id: int | None = Field(primary_key=True, default=None)
    start: int
    end: int
    description: str
```

Database Setup

```
from sqlmodel import SQLModel, create_engine

engine = create_engine(
    "sqlite:///carsharing.db",
    connect_args={"check_same_thread": False},  # Needed for SQLite
    echo=True # Log generated SQL (don’t use in production)
)


# Create the database on startup
@app.on_event("startup")
def on_startup():
    SQLModel.metadata.create_all(engine)
```

Session

```python
from sqlmodel import Session, Depends

def get_session():
    with Session(engine) as session:
        yield session


# FastAPI will call get_session and store result in session parameter
@app.get(...)
def car_by_id(id: int, session: Session = Depends(get_session)):
    car = session.get(Car, id)
    if car:
        return car
    # else return 404
```

Adding a New Car

```
@app.post(...)
def add_car(car_input: CarInput,
            session: Session = Depends(get_session)) -> Car:

    new_car = Car.from_orm(car_input)
    session.add(new_car)
    session.commit()
    session.refresh(new_car)
    return new_car
```

Querying Cars

```
from sqlmodel import select


@app.get("/api/cars")
def get_cars(size: str | None = None, doors: int | None = None,
             session: Session = Depends(get_session)) -> list[Car]:

    query = select(Car)
    if size:
        query = query.where(Car.size == size)
    if doors:
        query = query.where(Car.doors >= doors)
    return session.exec(query).all()
```

Removing a Car

```python
@app.delete("/api/cars/{id}", status_code=204)
def remove_car(id: int, session: Session = Depends(get_session)) -> None:
    car = session.get(Car, id)
    if car:
        session.delete(car)
        session.commit()
    else:
        raise HTTPException(status_code=404)
```

Relations

```python
class Trip(TripInput, table=True):
    id: int | None = Field(default=None, primary_key=True)
    car_id: int = Field(foreign_key="car.id")
    car: "Car" = Relationship(back_populates="trips")


class Car(CarInput, table=True):
    id: int | None = Field(default=None, primary_key=True)
    trips: list[Trip] = Relationship(back_populates="car")
```

https://bit.ly/sqlalchemy_async

https://bit.ly/fastapi_sqlalchemy

https://bit.ly/fastapi_sqlmodel

https://bit.ly/fastapi_db_async

Summary

SQLModel

- Built on SQLAlchemy + Pydantic

Create data model classes

Create a DB connection

CRUD operations

Relations

Transactions

- Session