📘 SQLAlchemy Notes + FastAPI Blog CRUD

PART 1: SQLAlchemy Core Concepts

✂️ What is SQLAlchemy?

SQLAlchemy is a Python ORM (Object Relational Mapper) that allows us to interact with databases using Python classes instead of raw SQL.

It supports: - ORM (high level) - Core (SQL expression language)

In FastAPI, we usually use ORM.

✒️ Basic Setup

```
pip install sqlalchemy fastapi uvicorn
```

🛏️ Database Configuration

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

DATABASE_URL = "sqlite:///./test.db"

engine = create_engine(
    DATABASE_URL, connect_args={"check_same_thread": False}
)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

🌂 Creating a Model (Table)

```
from sqlalchemy import Column, Integer, String, Text

class Blog(Base):
    __tablename__ = "blogs"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    content = Column(Text)
```

Create tables:

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

🌡️ Database Dependency (FastAPI)

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


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

PART 2: CRUD OPERATIONS IN SQLAlchemy

🩺 CREATE (Insert)

```
new_blog = Blog(title="My Blog", content="Content here")
db.add(new_blog)
db.commit()
db.refresh(new_blog)
```

Explanation: - `add()` → stage object - `commit()` → save to DB - `refresh()` → get updated values (like id)

🕖 READ (Select)

Read all

```
blogs = db.query(Blog).all()
```

Read one by ID

```
blog = db.query(Blog).filter(Blog.id == blog_id).first()
```

Read with multiple conditions

```
blog = db.query(Blog).filter(
    Blog.title == "Test",
    Blog.id > 5
).all()
```

🧾 UPDATE

```
blog = db.query(Blog).filter(Blog.id == blog_id).first()

if blog:
    blog.title = "Updated Title"
    blog.content = "Updated content"
    db.commit()
    db.refresh(blog)
```

🕘 DELETE

```
blog = db.query(Blog).filter(Blog.id == blog_id).first()

if blog:
    db.delete(blog)
    db.commit()
```

PART 3: FASTAPI BLOG CRUD APPLICATION

📁 Project Structure

```
app.py
```

Complete Working Example

```python
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy import create_engine, Column, Integer, String, Text
from sqlalchemy.orm import sessionmaker, declarative_base, Session

DATABASE_URL = "sqlite:///./blog.db"

engine = create_engine(
    DATABASE_URL, connect_args={"check_same_thread": False}
)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

app = FastAPI()

# -----------------
# Model
# -----------------
class Blog(Base):
    __tablename__ = "blogs"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    content = Column(Text)

Base.metadata.create_all(bind=engine)

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

```python
# -----------------
# CREATE
# -----------------
@app.post("/blogs")
def create_blog(title: str, content: str, db: Session = Depends(get_db)):
    blog = Blog(title=title, content=content)
    db.add(blog)
    db.commit()
    db.refresh(blog)
    return blog

# -----------------
# READ ALL
# -----------------
@app.get("/blogs")
def get_blogs(db: Session = Depends(get_db)):
    return db.query(Blog).all()

# -----------------
# READ ONE
# -----------------
@app.get("/blogs/{blog_id}")
def get_blog(blog_id: int, db: Session = Depends(get_db)):
    blog = db.query(Blog).filter(Blog.id == blog_id).first()
    if not blog:
        raise HTTPException(status_code=404, detail="Blog not found")
    return blog

# -----------------
# UPDATE
# -----------------
@app.put("/blogs/{blog_id}")
def update_blog(blog_id: int, title: str, content: str, db: Session =
Depends(get_db)):
    blog = db.query(Blog).filter(Blog.id == blog_id).first()

    if not blog:
        raise HTTPException(status_code=404, detail="Blog not found")

    blog.title = title
    blog.content = content
    db.commit()
    db.refresh(blog)

    return blog

# -----------------
# DELETE
# -----------------
@app.delete("/blogs/{blog_id}")
```

```python
def delete_blog(blog_id: int, db: Session = Depends(get_db)):
    blog = db.query(Blog).filter(Blog.id == blog_id).first()

    if not blog:
        raise HTTPException(status_code=404, detail="Blog not found")

    db.delete(blog)
    db.commit()

    return {"message": "Blog deleted successfully"}
```

🗜️ Run the App

```
uvicorn app:app --reload
```

Open:

```
http://127.0.0.1:8000/docs
```

📀 Summary

Operation	SQLAlchemy Method
Create	add() + commit()
Read	query().filter().first()/all()
Update	modify fields + commit()
Delete	delete() + commit()

If you want next: - Add Pydantic schemas (proper production way) - Add authentication - Use PostgreSQL instead of SQLite - Convert to async SQLAlchemy 2.0 style

Just tell me 🚀