## ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

## направление подготовки

**09.03.02 «Информационные системы и технологии»**

Лабораторная №9

**Дисциплина:** Backend-разработка

# **Тема:** SQLAlchemy

## Выполнил(а): студент(ка) группы 221-3711

Морозов К.А.

(Фамилия И.О.)

## Дата, подпись

(Дата) (Подпись)

## Проверил:

(Фамилия И.О., степень, звание) **(Оценка)**

## Дата, подпись

(Дата) (Подпись)

**Замечания:**

**Москва 2024**

# Лабораторная №9

«SQLAlchemy»

Ссылка на git:

Цель работы:

Освоить основные принципы работы с базой данных через SQLAlchemy: подключение к базе данных, создание таблиц, выполнение запросов и интеграция с веб-приложением.

Задание:

Часть 1: Подключение к базе данных и создание таблиц

Выбор базы данных:

Выберите одну из баз данных: MSSQL, SQLite, PostgreSQL, MySQL.

Установите необходимые библиотеки для работы с выбранной базой данных и SQLAlchemy.

Создание модели данных:

Опишите модель данных, состоящую из двух таблиц: Users и Posts.

Таблица Users должна содержать следующие поля:

id (целое число, первичный ключ, автоинкремент)

username (строка, уникальное значение)

email (строка, уникальное значение)

password (строка)

Таблица Posts должна содержать следующие поля:

id (целое число, первичный ключ, автоинкремент)

title (строка)

content (текст)

user\_id (целое число, внешний ключ, ссылающийся на поле id таблицы Users)

Создание таблиц:

Напишите программу на Python, которая подключается к выбранной базе данных и создает таблицы Users и Posts на основе описанной модели данных.

Часть 2: Взаимодействие с базой данных

Добавление данных:

Напишите программу, которая добавляет в таблицу Users несколько записей с разными значениями полей username, email и password.

Напишите программу, которая добавляет в таблицу Posts несколько записей, связанных с пользователями из таблицы Users.

Извлечение данных:

Напишите программу, которая извлекает все записи из таблицы Users.

Напишите программу, которая извлекает все записи из таблицы Posts, включая информацию о пользователях, которые их создали.

Напишите программу, которая извлекает записи из таблицы Posts, созданные конкретным пользователем.

Обновление данных:

Напишите программу, которая обновляет поле email у одного из пользователей.

Напишите программу, которая обновляет поле content у одного из постов.

Удаление данных:

Напишите программу, которая удаляет один из постов.

Напишите программу, которая удаляет пользователя и все его посты.

Часть 3: Базовые операции с базой данных в веб-приложении

Создание веб-приложения:

Создайте простое веб-приложение на FastAPI.

Интегрируйте SQLAlchemy в ваше веб-приложение.

Реализация CRUD-операций:

Реализуйте веб-страницы для выполнения CRUD-операций (создание, чтение, обновление, удаление) с записями в таблицах Users и Posts.

Страницы должны включать:

Форму для создания нового пользователя/поста.

Список всех пользователей/постов с возможностью редактирования и удаления.

Страницу для редактирования информации о пользователе/посте.

|  |
| --- |
| from sqlalchemy import create\_engine, Column, Integer, String, Text, ForeignKey  from sqlalchemy.ext.declarative import declarative\_base  from sqlalchemy.orm import sessionmaker  from sqlalchemy.orm import Session, relationship  from pydantic import BaseModel  from fastapi import FastAPI, Depends, HTTPException  from typing import List  from typing import Optional  app = FastAPI()  # Database SQLite  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()  #Модель пользователя  class User(Base):      \_\_tablename\_\_ = "users"      id = Column(Integer, primary\_key=True, index=True)      name = Column(String, index=True)      email = Column(String, unique=True, index=True)      password = Column(String, nullable=False)      posts = relationship("Post", back\_populates="user", cascade="all, delete")  #Модель поста  class Post(Base):      \_\_tablename\_\_ = "posts"      id = Column(Integer, primary\_key=True, index=True)      title = Column(String, nullable=False)      content = Column(Text, nullable=False)      user\_id = Column(Integer, ForeignKey("users.id"), nullable=False)      user = relationship("User", back\_populates="posts")  #Создание таблицы в базе данных  Base.metadata.create\_all(bind=engine)  #Получение сессии базы данных  def get\_db():      db = SessionLocal()      try:          yield db      finally:          db.close()  class UserCreate(BaseModel):      name: str      email: str      password: str  class UserResponse(BaseModel):      id: int      name: str      email: str      password: str  class PostCreate(BaseModel):      title: str      content: str      user\_id: int  class PostResponse(BaseModel):      id: int      title: str      content: str      user\_id: int      user: UserResponse      class Config:          orm\_mode = True  @app.post("/users/", response\_model=UserResponse)  def create\_user(user: UserCreate, db: Session = Depends(get\_db)):      db\_user = User(name=user.name, email=user.email, password=user.password)      db.add(db\_user)      db.commit()      db.refresh(db\_user)      return db\_user  @app.get("/users/", response\_model=List[UserResponse])  def read\_users(skip: int = 0, Limit: int = 10, db: Session = Depends(get\_db)):      users = db.query(User).offset(skip).limit(Limit).all()      return users  @app.post("/posts/", response\_model=PostResponse)  def create\_post(post: PostCreate, db: Session = Depends(get\_db)):      db\_user = db.query(User).filter(User.id == post.user\_id).first()      if not db\_user:          raise HTTPException(status\_code=404, detail="User not found")      db\_post = Post(title=post.title, content=post.content, user\_id=post.user\_id)      db.add(db\_post)      db.commit()      db.refresh(db\_post)      return db\_post  @app.get("/posts/", response\_model=List[PostResponse])  def read\_posts(skip: int = 0, limit: int = 10, db: Session = Depends(get\_db)):      posts = db.query(Post).offset(skip).limit(limit).all()      return posts  @app.get("/posts/user/{user\_id}", response\_model=List[PostResponse])  def read\_posts\_by\_user(user\_id: int, db: Session = Depends(get\_db)):      posts = db.query(Post).filter(Post.user\_id == user\_id).all()      return posts  @app.put("/posts/{post\_id}", response\_model=PostResponse)  def update\_post(post\_id: int, post: PostCreate, db: Session = Depends(get\_db)):      db\_post = db.query(Post).filter(Post.id == post\_id).first()      if not db\_post:          raise HTTPException(status\_code=404, detail="Post not found")      db\_post.title = post.title      db\_post.content = post.content      db.commit()      db.refresh(db\_post)      return db\_post  @app.delete("/posts/{post\_id}", response\_model=PostResponse)  def delete\_post(post\_id: int, db: Session = Depends(get\_db)):      db\_post = db.query(Post).filter(Post.id == post\_id).first()      if not db\_post:          raise HTTPException(status\_code=404, detail="Post not found")      db.delete(db\_post)      db.commit()      return db\_post  @app.get("/users/{user\_id}", response\_model=UserResponse)  def read\_user(user\_id: int, db: Session = Depends(get\_db)):      user = db.query(User).filter(User.id == user\_id).first()      if user is None:          raise HTTPException(status\_code=404, detail="User not found")      return user  class UserUpdate(BaseModel):      name: Optional[str] = None      email: Optional[str] = None  @app.put("/users/{user\_id}", response\_model=UserResponse)  def update\_user(user\_id: int, user: UserUpdate, db: Session = Depends(get\_db)):      db\_user = db.query(User).filter(User.id == user\_id).first()      if db\_user is None:          raise HTTPException(status\_code=404, detail= "User not found")        db\_user.name = user.name if user.name is not None else db\_user.name      db\_user.email = user.email if user.email is not None else db\_user.email      db.commit()      db.refresh(db\_user)      return db\_user  @app.delete("/users/{user\_id}", response\_model=UserResponse)  def delete\_user(user\_id: int, db: Session = Depends(get\_db)):      db\_user = db.query(User).filter(User.id == user\_id).first()      if db\_user is None:          raise HTTPException(status\_code=404, detail= "User not found")        db.delete(db\_user)      db.commit()      return db\_user |























