МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №5

Специальность ПО11

Выполнил
И. А. Гурин
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 г.

Брест 2025

Цель работы: приобрести практические навыки разработки API и баз данных

Общее задание:

1. Реализовать базу данных из не менее 5 таблиц на заданную тематику. При реализации продумать типизацию полей и внешние ключи в таблицах;

2. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними;

3. На языке Python с использованием SQLAlchemy реализовать подключение к БД;

4. Реализовать основные операции с данными (выборку, добавление, удаление, модификацию);

5. Для каждой реализованной операции с использованием FastAPI реализовать отдельный эндпойнт;

Код программы:

main.py:

```python
from datetime import time
from fastapi import FastAPI, HTTPException
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

from const import Const
from database import SessionLocal
from models import Course, Teacher, Classroom, StudentGroup, Schedule

DATABASE_URL = Const.DATABASE_URL
app = FastAPI()

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

@app.post("/courses/")
def create_course(course: dict):
    db = SessionLocal()
    db_course = Course(**course)
    db.add(db_course)
    db.commit()
    db.refresh(db_course)
    return db_course

@app.get("/courses/")
def get_courses():
    db = SessionLocal()
    return db.query(Course).all()

@app.get("/courses/{course_id}")
def get_course(course_id: int):
    db = SessionLocal()
    course = db.query(Course).filter(Course.id == course_id).first()
    if not course:
        raise HTTPException(status_code=404, detail="Course not found")
    return course

@app.put("/courses/{course_id}")
def update_course(course_id: int, course: dict):
    db = SessionLocal()
    db_course = db.query(Course).filter(Course.id == course_id).first()
    if not db_course:
        raise HTTPException(status_code=404, detail="Course not found")
    for key, value in course.items():
        setattr(db_course, key, value)
    db.commit()
    return db_course

@app.delete("/courses/{course_id}")
def delete_course(course_id: int):
```

```
    db = SessionLocal()
    course = db.query(Course).filter(Course.id == course_id).first()
    if not course:
        raise HTTPException(status_code=404, detail="Course not found")
    db.delete(course)
    db.commit()
    return {"message": "Course deleted"}

@app.post("/teachers/")
def create_teacher(teacher: dict):
    db = SessionLocal()
    db_teacher = Teacher(**teacher)
    db.add(db_teacher)
    db.commit()
    db.refresh(db_teacher)
    return db_teacher

@app.get("/teachers/")
def get_teachers():
    db = SessionLocal()
    return db.query(Teacher).all()

@app.get("/teachers/{teacher_id}")
def get_teacher(teacher_id: int):
    db = SessionLocal()
    teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not teacher:
        raise HTTPException(status_code=404, detail="Teacher not found")
    return teacher

@app.put("/teachers/{teacher_id}")
def update_teacher(teacher_id: int, teacher: dict):
    db = SessionLocal()
    db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not db_teacher:
        raise HTTPException(status_code=404, detail="Teacher not found")
    for key, value in teacher.items():
        setattr(db_teacher, key, value)
    db.commit()
    return db_teacher

@app.delete("/teachers/{teacher_id}")
def delete_teacher(teacher_id: int):
    db = SessionLocal()
    teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not teacher:
        raise HTTPException(status_code=404, detail="Teacher not found")
    db.delete(teacher)
    db.commit()
    return {"message": "Teacher deleted"}

@app.post("/classrooms/")
def create_classroom(classroom: dict):
    db = SessionLocal()
    db_classroom = Classroom(**classroom)
    db.add(db_classroom)
    db.commit()
    db.refresh(db_classroom)
    return db_classroom

@app.get("/classrooms/")
def get_classrooms():
    db = SessionLocal()
    return db.query(Classroom).all()

@app.get("/classrooms/{classroom_id}")
def get_classroom(classroom_id: int):
    db = SessionLocal()
    classroom = db.query(Classroom).filter(Classroom.id == classroom_id).first()
    if not classroom:
        raise HTTPException(status_code=404, detail="Classroom not found")
```

```
    return classroom

@app.put("/classrooms/{classroom_id}")
def update_classroom(classroom_id: int, classroom: dict):
    db = SessionLocal()
    db_classroom = db.query(Classroom).filter(Classroom.id == classroom_id).first()
    if not db_classroom:
        raise HTTPException(status_code=404, detail="Classroom not found")
    for key, value in classroom.items():
        setattr(db_classroom, key, value)
    db.commit()
    return db_classroom

@app.delete("/classrooms/{classroom_id}")
def delete_classroom(classroom_id: int):
    db = SessionLocal()
    classroom = db.query(Classroom).filter(Classroom.id == classroom_id).first()
    if not classroom:
        raise HTTPException(status_code=404, detail="Classroom not found")
    db.delete(classroom)
    db.commit()
    return {"message": "Classroom deleted"}

@app.post("/student_groups/")
def create_student_group(student_group: dict):
    db = SessionLocal()
    db_group = StudentGroup(**student_group)
    db.add(db_group)
    db.commit()
    db.refresh(db_group)
    return db_group

@app.get("/student_groups/")
def get_student_groups():
    db = SessionLocal()
    return db.query(StudentGroup).all()

@app.get("/student_groups/{group_id}")
def get_student_group(group_id: int):
    db = SessionLocal()
    group = db.query(StudentGroup).filter(StudentGroup.id == group_id).first()
    if not group:
        raise HTTPException(status_code=404, detail="Student group not found")
    return group

@app.put("/student_groups/{group_id}")
def update_student_group(group_id: int, student_group: dict):
    db = SessionLocal()
    db_group = db.query(StudentGroup).filter(StudentGroup.id == group_id).first()
    if not db_group:
        raise HTTPException(status_code=404, detail="Student group not found")
    for key, value in student_group.items():
        setattr(db_group, key, value)
    db.commit()
    return db_group

@app.delete("/student_groups/{group_id}")
def delete_student_group(group_id: int):
    db = SessionLocal()
    group = db.query(StudentGroup).filter(StudentGroup.id == group_id).first()
    if not group:
        raise HTTPException(status_code=404, detail="Student group not found")
    db.delete(group)
    db.commit()
    return {"message": "Student group deleted"}

@app.post("/schedules/")
def create_schedule(schedule: dict):
    db = SessionLocal()
    db_schedule = Schedule(**schedule)
    db.add(db_schedule)
```

```
    db.commit()
    db.refresh(db_schedule)
    return db_schedule

@app.get("/schedules/")
def get_schedules():
    db = SessionLocal()
    return db.query(Schedule).all()

@app.get("/schedules/{schedule_id}")
def get_schedule(schedule_id: int):
    db = SessionLocal()
    schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if not schedule:
        raise HTTPException(status_code=404, detail="Schedule not found")
    return schedule

@app.put("/schedules/{schedule_id}")
def update_schedule(schedule_id: int, schedule: dict):
    db = SessionLocal()
    db_schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if not db_schedule:
        raise HTTPException(status_code=404, detail="Schedule not found")
    for key, value in schedule.items():
        setattr(db_schedule, key, value)
    db.commit()
    return db_schedule

@app.delete("/schedules/{schedule_id}")
def delete_schedule(schedule_id: int):
    db = SessionLocal()
    schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if not schedule:
        raise HTTPException(status_code=404, detail="Schedule not found")
    db.delete(schedule)
    db.commit()
    return {"message": "Schedule deleted"}

@app.post("/populate_test_data/")
def populate_test_data():
    db = SessionLocal()

    course1 = Course(title="СПП", description="СПП description")
    course2 = Course(title="БД", description="БД description")

    teacher1 = Teacher(first_name="Петров", last_name="Пётр", email="petr@example.com",
phone_number="1234567890")
    teacher2 = Teacher(first_name="Иванов", last_name="Иван", email="ivan@example.com",
phone_number="0987654321")

    classroom1 = Classroom(room_number="101", building="1", capacity=50)
    classroom2 = Classroom(room_number="202", building="2", capacity=40)

    group1 = StudentGroup(group_name="ПО-1")
    group2 = StudentGroup(group_name="ПО-2")

    db.add_all([course1, course2, teacher1, teacher2, classroom1, classroom2, group1, group2])
    db.commit()

    schedule1 = Schedule(
        course_id=course1.id,
        teacher_id=teacher1.id,
        classroom_id=classroom1.id,
        group_id=group1.id,
        day_of_week="Monday",
        start_time=time(9, 0),
        end_time=time(10, 30)
    )

    schedule2 = Schedule(
        course_id=course2.id,
```

```
        teacher_id=teacher2.id,
        classroom_id=classroom2.id,
        group_id=group2.id,
        day_of_week="Tuesday",
        start_time=time(11, 0),
        end_time=time(12, 30)
    )

    db.add_all([schedule1, schedule2])
    db.commit()

    return {"message": "Test data populated successfully"}

if __name__ == "__main__":
    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=8000)
```

const.py:

```
from dataclasses import dataclass

@dataclass
class Const:
    DATABASE_URL: str = "sqlite:///./university_schedule.db"
```

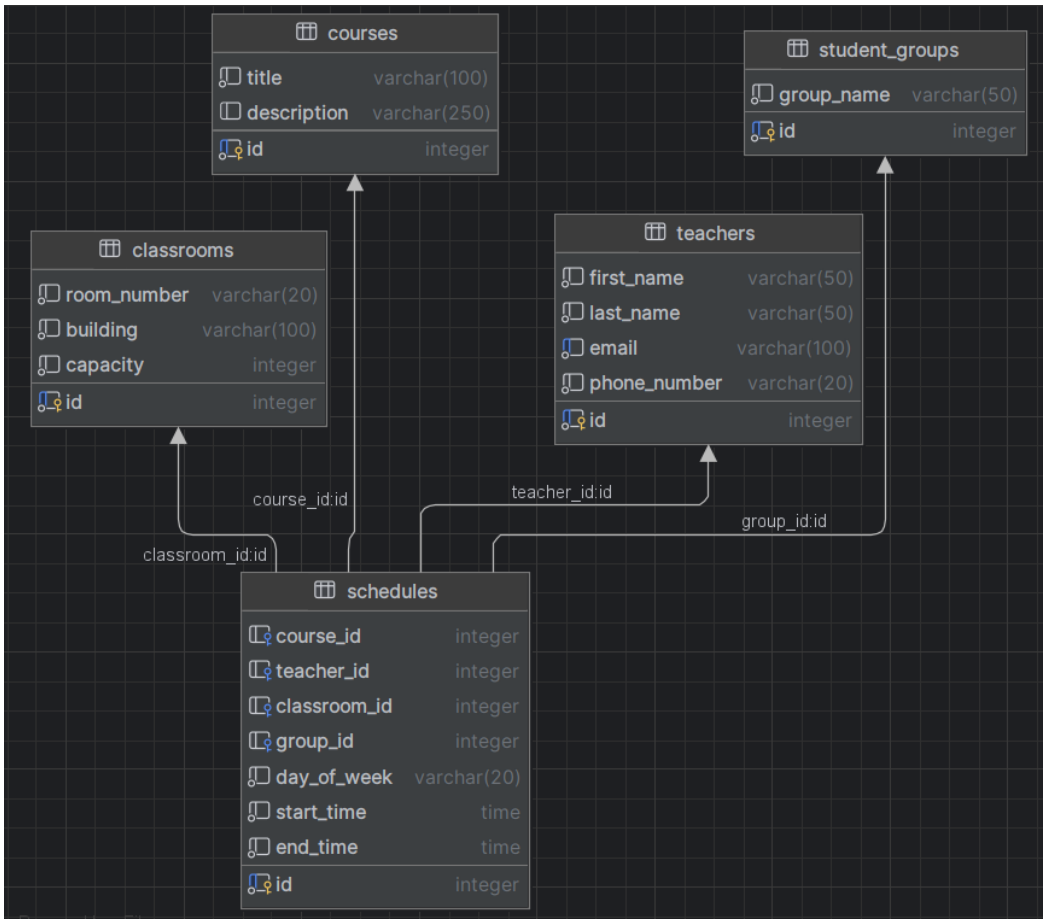
database.py:

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

from const import Const

engine = create_engine(Const.DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

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

models.py:

```
from sqlalchemy import Column, Integer, String, ForeignKey, Time
from sqlalchemy.orm import relationship

from database import Base

class Course(Base):
    __tablename__ = "courses"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String(100), nullable=False)
    description = Column(String(250))

    schedules = relationship("Schedule", back_populates="course")

class Teacher(Base):
    __tablename__ = "teachers"

    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    email = Column(String(100), nullable=False, unique=True)
    phone_number = Column(String(20), nullable=False)

    schedules = relationship("Schedule", back_populates="teacher")

class Classroom(Base):
    __tablename__ = "classrooms"

    id = Column(Integer, primary_key=True, index=True)
    room_number = Column(String(20), nullable=False)
```

```
    building = Column(String(100), nullable=False)
    capacity = Column(Integer, nullable=False)

    schedules = relationship("Schedule", back_populates="classroom")

class StudentGroup(Base):
    __tablename__ = "student_groups"

    id = Column(Integer, primary_key=True, index=True)
    group_name = Column(String(50), nullable=False)

    schedules = relationship("Schedule", back_populates="student_group")

class Schedule(Base):
    __tablename__ = "schedules"

    id = Column(Integer, primary_key=True, index=True)
    course_id = Column(Integer, ForeignKey("courses.id"))
    teacher_id = Column(Integer, ForeignKey("teachers.id"))
    classroom_id = Column(Integer, ForeignKey("classrooms.id"))
    group_id = Column(Integer, ForeignKey("student_groups.id"))
    day_of_week = Column(String(20), nullable=False)
    start_time = Column(Time, nullable=False)
    end_time = Column(Time, nullable=False)

    course = relationship("Course", back_populates="schedules")
    teacher = relationship("Teacher", back_populates="schedules")
    classroom = relationship("Classroom", back_populates="schedules")
    student_group = relationship("StudentGroup", back_populates="schedules")
```



Вывод: приобрёл практические навыки разработки API и базы данных