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
Е. А. Германович
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.

Задание:

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 (пример, схема на рис. 1);
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;

Базу данные можно реализовать в любой СУБД (MySQL, PostgreSQL, SQLite и др.)

4) База данных Городской транспорт

Код программы:

```
import os
from datetime import date, time
from typing import List

from fastapi import FastAPI, HTTPException, Depends
from sqlalchemy import (
    Boolean,
    Column,
    Date,
    Float,
    ForeignKey,
    Integer,
    String,
    Time,
    create_engine,
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
from pydantic import BaseModel
from typing import Optional

# Создание базы данных SQLite
DATABASE_URL = "sqlite:///./transport.db"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()


# Модели SQLAlchemy
class Vehicle(Base):
    __tablename__ = "vehicles"

    id = Column(Integer, primary_key=True, index=True)
    vehicle_number = Column(String(20), nullable=False)
    model = Column(String(100), nullable=False)
    capacity = Column(Integer, nullable=False)
    year_of_manufacture = Column(Integer, nullable=False)
    last_maintenance_date = Column(Date, nullable=True)
```

```
    schedules = relationship("Schedule", back_populates="vehicle")


class Route(Base):
    __tablename__ = "routes"

    id = Column(Integer, primary_key=True, index=True)
    route_number = Column(String(10), nullable=False)
    start_point = Column(String(100), nullable=False)
    end_point = Column(String(100), nullable=False)
    distance_km = Column(Float, nullable=False)
    estimated_time_minutes = Column(Integer, nullable=False)

    schedules = relationship("Schedule", back_populates="route")
    stops = relationship("Stop", secondary="route_stops", back_populates="routes")


class Driver(Base):
    __tablename__ = "drivers"

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

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


class Schedule(Base):
    __tablename__ = "schedules"

    id = Column(Integer, primary_key=True, index=True)
    route_id = Column(Integer, ForeignKey("routes.id"))
    vehicle_id = Column(Integer, ForeignKey("vehicles.id"))
    driver_id = Column(Integer, ForeignKey("drivers.id"))
    is_weekend = Column(Boolean, default=False)

    route = relationship("Route", back_populates="schedules")
    vehicle = relationship("Vehicle", back_populates="schedules")
    driver = relationship("Driver", back_populates="schedules")


class Stop(Base):
    __tablename__ = "stops"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    address = Column(String(200), nullable=False)
    latitude = Column(Float, nullable=False)
    longitude = Column(Float, nullable=False)

    routes = relationship("Route", secondary="route_stops", back_populates="stops")


class RouteStop(Base):
    __tablename__ = "route_stops"
```

```
    route_id = Column(Integer, ForeignKey("routes.id"), primary_key=True)
    stop_id = Column(Integer, ForeignKey("stops.id"), primary_key=True)
    stop_order = Column(Integer, nullable=False)


# Создание таблиц
Base.metadata.create_all(bind=engine)

app = FastAPI()


# Модели Pydantic для валидации данных
class VehicleBase(BaseModel):
    vehicle_number: str
    model: str
    capacity: int
    year_of_manufacture: int
    last_maintenance_date: Optional[date] = None


class VehicleCreate(VehicleBase):
    pass


class VehicleResponse(VehicleBase):
    id: int

    class Config:
        from_attributes = True


class DriverBase(BaseModel):
    first_name: str
    last_name: str
    license_number: str
    experience_years: int
    phone_number: str


class DriverCreate(DriverBase):
    pass


class DriverResponse(DriverBase):
    id: int

    class Config:
        from_attributes = True


class RouteBase(BaseModel):
    route_number: str
    start_point: str
    end_point: str
    distance_km: float
    estimated_time_minutes: int
```

```
class RouteCreate(RouteBase):
    pass


class RouteResponse(RouteBase):
    id: int

    class Config:
        from_attributes = True


class StopBase(BaseModel):
    name: str
    address: str
    latitude: float
    longitude: float


class StopCreate(StopBase):
    pass


class StopResponse(StopBase):
    id: int

    class Config:
        from_attributes = True


class ScheduleBase(BaseModel):
    route_id: int
    vehicle_id: int
    driver_id: int
    is_weekend: bool = False


class ScheduleCreate(ScheduleBase):
    pass


class ScheduleResponse(ScheduleBase):
    id: int

    class Config:
        from_attributes = True


# Вспомогательная функция для получения сессии базы данных
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()


# Эндпоинты для Vehicle
@app.post("/vehicles/", response_model=VehicleResponse)
def create_vehicle(vehicle: VehicleCreate, db: SessionLocal = Depends(get_db)):
```

```
    db_vehicle = Vehicle(
        vehicle_number=vehicle.vehicle_number,
        model=vehicle.model,
        capacity=vehicle.capacity,
        year_of_manufacture=vehicle.year_of_manufacture,
        last_maintenance_date=vehicle.last_maintenance_date
    )
    db.add(db_vehicle)
    db.commit()
    db.refresh(db_vehicle)
    return db_vehicle


@app.get("/vehicles/", response_model=List[VehicleResponse])
def get_vehicles(db: SessionLocal = Depends(get_db)):
    return db.query(Vehicle).all()


@app.get("/vehicles/{vehicle_id}", response_model=VehicleResponse)
def get_vehicle(vehicle_id: int, db: SessionLocal = Depends(get_db)):
    vehicle = db.query(Vehicle).filter(Vehicle.id == vehicle_id).first()
    if vehicle is None:
        raise HTTPException(status_code=404, detail="Vehicle not found")
    return vehicle


@app.put("/vehicles/{vehicle_id}", response_model=VehicleResponse)
def update_vehicle(vehicle_id: int, vehicle: VehicleCreate, db: SessionLocal = Depends(get_db)):
    db_vehicle = db.query(Vehicle).filter(Vehicle.id == vehicle_id).first()
    if db_vehicle is None:
        raise HTTPException(status_code=404, detail="Vehicle not found")

    db_vehicle.vehicle_number = vehicle.vehicle_number
    db_vehicle.model = vehicle.model
    db_vehicle.capacity = vehicle.capacity
    db_vehicle.year_of_manufacture = vehicle.year_of_manufacture
    db_vehicle.last_maintenance_date = vehicle.last_maintenance_date

    db.commit()
    return db_vehicle


@app.delete("/vehicles/{vehicle_id}")
def delete_vehicle(vehicle_id: int, db: SessionLocal = Depends(get_db)):
    vehicle = db.query(Vehicle).filter(Vehicle.id == vehicle_id).first()
    if vehicle is None:
        raise HTTPException(status_code=404, detail="Vehicle not found")
    db.delete(vehicle)
    db.commit()
    return {"message": "Vehicle deleted"}


# Эндпоинты для Route
@app.post("/routes/")
def create_route(route: dict):
    db = SessionLocal()
    db_route = Route(**route)
    db.add(db_route)
    db.commit()
```

```
    db.refresh(db_route)
    return db_route


@app.get("/routes/")
def get_routes():
    db = SessionLocal()
    return db.query(Route).all()


@app.get("/routes/{route_id}")
def get_route(route_id: int):
    db = SessionLocal()
    route = db.query(Route).filter(Route.id == route_id).first()
    if route is None:
        raise HTTPException(status_code=404, detail="Route not found")
    return route


@app.put("/routes/{route_id}")
def update_route(route_id: int, route: dict):
    db = SessionLocal()
    db_route = db.query(Route).filter(Route.id == route_id).first()
    if db_route is None:
        raise HTTPException(status_code=404, detail="Route not found")
    for key, value in route.items():
        setattr(db_route, key, value)
    db.commit()
    return db_route


@app.delete("/routes/{route_id}")
def delete_route(route_id: int):
    db = SessionLocal()
    route = db.query(Route).filter(Route.id == route_id).first()
    if route is None:
        raise HTTPException(status_code=404, detail="Route not found")
    db.delete(route)
    db.commit()
    return {"message": "Route deleted"}


# Эндпоинты для Driver
@app.post("/drivers/")
def create_driver(driver: dict):
    db = SessionLocal()
    db_driver = Driver(**driver)
    db.add(db_driver)
    db.commit()
    db.refresh(db_driver)
    return db_driver


@app.get("/drivers/")
def get_drivers():
    db = SessionLocal()
    return db.query(Driver).all()
```

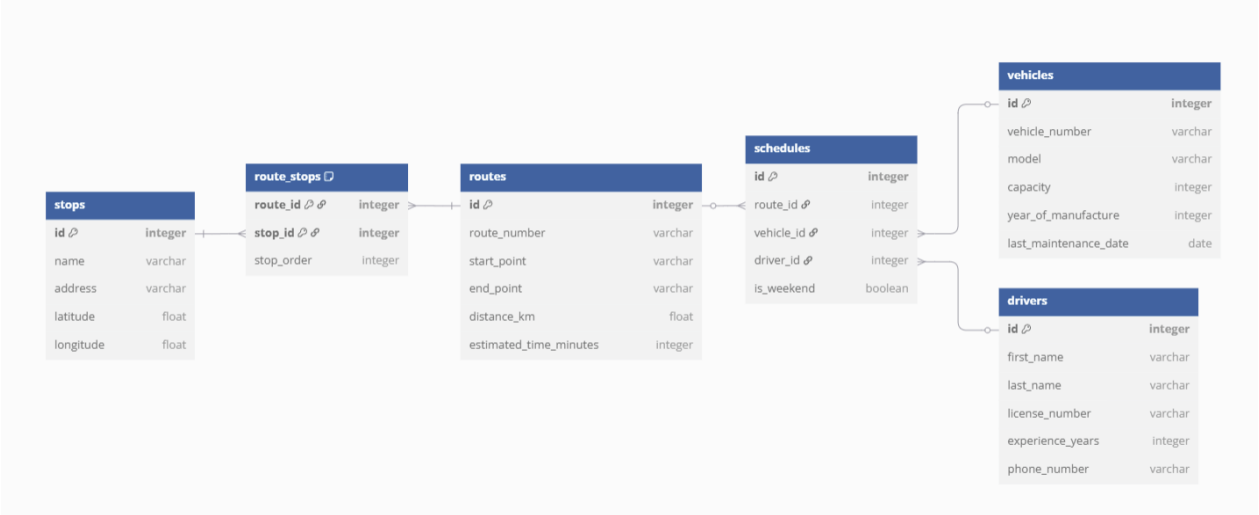
```
@app.get("/drivers/{driver_id}")
def get_driver(driver_id: int):
    db = SessionLocal()
    driver = db.query(Driver).filter(Driver.id == driver_id).first()
    if driver is None:
        raise HTTPException(status_code=404, detail="Driver not found")
    return driver


@app.put("/drivers/{driver_id}")
def update_driver(driver_id: int, driver: dict):
    db = SessionLocal()
    db_driver = db.query(Driver).filter(Driver.id == driver_id).first()
    if db_driver is None:
        raise HTTPException(status_code=404, detail="Driver not found")
    for key, value in driver.items():
        setattr(db_driver, key, value)
    db.commit()
    return db_driver


@app.delete("/drivers/{driver_id}")
def delete_driver(driver_id: int):
    db = SessionLocal()
    driver = db.query(Driver).filter(Driver.id == driver_id).first()
    if driver is None:
        raise HTTPException(status_code=404, detail="Driver not found")
    db.delete(driver)
    db.commit()
    return {"message": "Driver deleted"}


# Эндпоинты для Schedule
@app.post("/schedules/", response_model=ScheduleResponse)
def create_schedule(schedule: ScheduleCreate, db: SessionLocal = Depends(get_db)):
    db_schedule = Schedule(
        route_id=schedule.route_id,
        vehicle_id=schedule.vehicle_id,
        driver_id=schedule.driver_id,
        is_weekend=schedule.is_weekend
    )
    db.add(db_schedule)
    db.commit()
    db.refresh(db_schedule)
    return db_schedule


@app.get("/schedules/", response_model=List[ScheduleResponse])
def get_schedules(db: SessionLocal = Depends(get_db)):
    return db.query(Schedule).all()


@app.get("/schedules/{schedule_id}", response_model=ScheduleResponse)
def get_schedule(schedule_id: int, db: SessionLocal = Depends(get_db)):
    schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")
    return schedule
```

```
@app.put("/schedules/{schedule_id}", response_model=ScheduleResponse)
def update_schedule(schedule_id: int, schedule: ScheduleCreate, db: SessionLocal = Depends(get_db)):
    db_schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if db_schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")

    db_schedule.route_id = schedule.route_id
    db_schedule.vehicle_id = schedule.vehicle_id
    db_schedule.driver_id = schedule.driver_id
    db_schedule.is_weekend = schedule.is_weekend

    db.commit()
    return db_schedule


@app.delete("/schedules/{schedule_id}")
def delete_schedule(schedule_id: int, db: SessionLocal = Depends(get_db)):
    schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")
    db.delete(schedule)
    db.commit()
    return {"message": "Schedule deleted"}


# Эндпоинты для Stop
@app.post("/stops/")
def create_stop(stop: dict):
    db = SessionLocal()
    db_stop = Stop(**stop)
    db.add(db_stop)
    db.commit()
    db.refresh(db_stop)
    return db_stop


@app.get("/stops/")
def get_stops():
    db = SessionLocal()
    return db.query(Stop).all()


@app.get("/stops/{stop_id}")
def get_stop(stop_id: int):
    db = SessionLocal()
    stop = db.query(Stop).filter(Stop.id == stop_id).first()
    if stop is None:
        raise HTTPException(status_code=404, detail="Stop not found")
    return stop


@app.put("/stops/{stop_id}")
def update_stop(stop_id: int, stop: dict):
    db = SessionLocal()
    db_stop = db.query(Stop).filter(Stop.id == stop_id).first()
    if db_stop is None:
        raise HTTPException(status_code=404, detail="Stop not found")
    for key, value in stop.items():
```

```
        setattr(db_stop, key, value)
    db.commit()
    return db_stop


@app.delete("/stops/{stop_id}")
def delete_stop(stop_id: int):
    db = SessionLocal()
    stop = db.query(Stop).filter(Stop.id == stop_id).first()
    if stop is None:
        raise HTTPException(status_code=404, detail="Stop not found")
    db.delete(stop)
    db.commit()
    return {"message": "Stop deleted"}


if __name__ == "__main__":
    import uvicorn

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

Рисунки с результатами работы программы:

```
PS C:\OSPanel\domains\spp_po11\reports\Germanovich Egor\Lab5\src> python Lab5.py
C:\OSPanel\domains\spp_po11\reports\Germanovich Egor\Lab5\src\Lab5.py:26: MovedIn20Warning: The ``declarative_base()`` fu
ction is now available as sqlalchemy.orm.declarative_base(). (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: ht
ps://sqlalche.me/e/b8d9)
  Base = declarative_base()
INFO:     Started server process [3076]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```



Вывод: приобрел практические навыки разработки API и баз данных.