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()

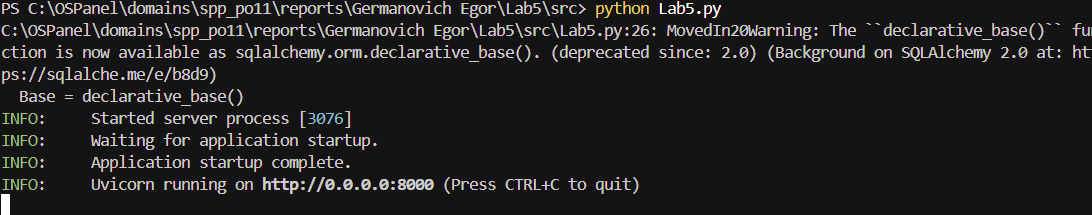
return {"message": "Stop deleted"}

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

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

**Рисунки с результатами работы программы:**

****

****

**Вывод:** приобрел практические навыки разработки API и баз данных.