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
Лесько М.И.
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
02.05.2025 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.)

12) База данных Европейские футбольные чемпионаты

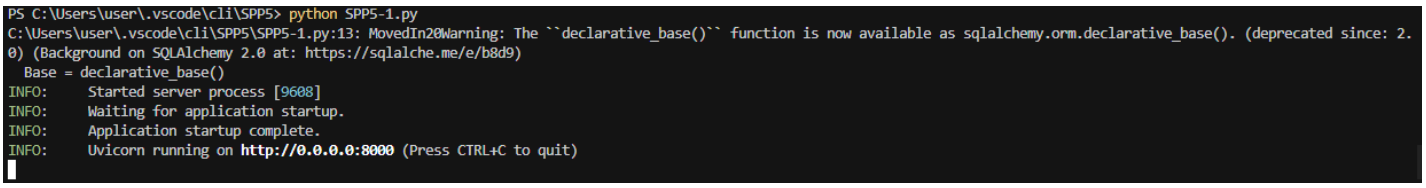
Код программы:

```
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, Date, Float, Boolean
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Optional
from datetime import date

# Database setup
SQLALCHEMY_DATABASE_URL = "sqlite:///./football_championships.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# Database Models
class Championship(Base):
    __tablename__ = "championships"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True)
    start_date = Column(Date)
    end_date = Column(Date)
    host_country = Column(String)

    matches = relationship("Match", back_populates="championship")

class Team(Base):
    __tablename__ = "teams"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True)
    country = Column(String)
    coach = Column(String)

    players = relationship("Player", back_populates="team")
    home_matches = relationship("Match", foreign_keys="Match.home_team_id", back_populates="home_team")
    away_matches = relationship("Match", foreign_keys="Match.away_team_id", back_populates="away_team")

class Player(Base):
    __tablename__ = "players"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    position = Column(String)
    age = Column(Integer)
    team_id = Column(Integer, ForeignKey("teams.id"))

    team = relationship("Team", back_populates="players")

class Stadium(Base):
```

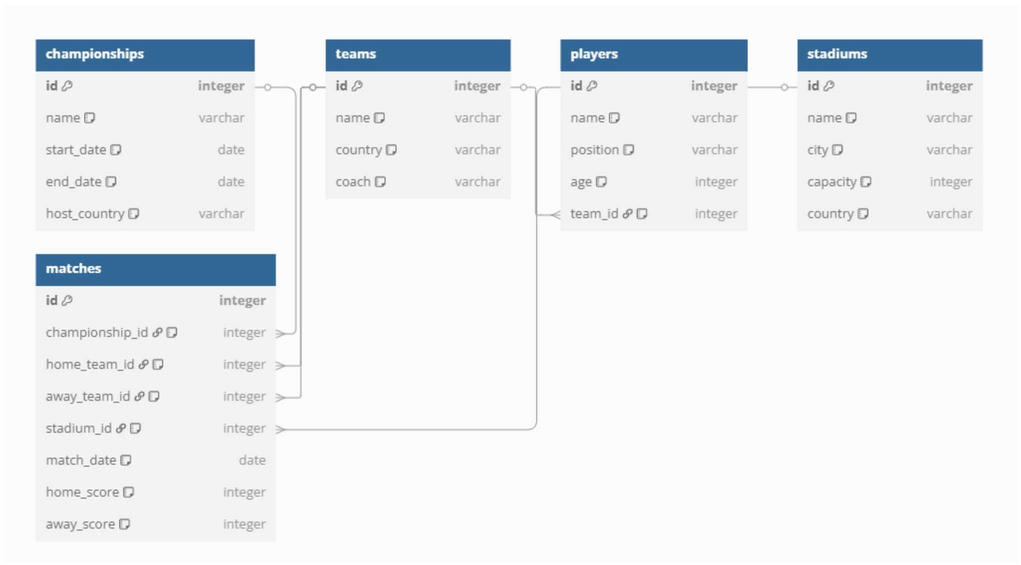
```python
    __tablename__ = "stadiums"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    city = Column(String)
    capacity = Column(Integer)
    country = Column(String)

    matches = relationship("Match", back_populates="stadium")

class Match(Base):
    __tablename__ = "matches"

    id = Column(Integer, primary_key=True, index=True)
    championship_id = Column(Integer, ForeignKey("championships.id"))
    home_team_id = Column(Integer, ForeignKey("teams.id"))
    away_team_id = Column(Integer, ForeignKey("teams.id"))
    stadium_id = Column(Integer, ForeignKey("stadiums.id"))
    match_date = Column(Date)
    home_score = Column(Integer)
    away_score = Column(Integer)

    championship = relationship("Championship", back_populates="matches")
    home_team = relationship("Team", foreign_keys=[home_team_id], back_populates="home_matches")
    away_team = relationship("Team", foreign_keys=[away_team_id], back_populates="away_matches")
    stadium = relationship("Stadium", back_populates="matches")

# Create database tables
Base.metadata.create_all(bind=engine)

# Pydantic models for request/response
class ChampionshipCreate(BaseModel):
    name: str
    start_date: date
    end_date: date
    host_country: str

class TeamCreate(BaseModel):
    name: str
    country: str
    coach: str

class PlayerCreate(BaseModel):
    name: str
    position: str
    age: int
    team_id: int

class StadiumCreate(BaseModel):
    name: str
    city: str
    capacity: int
    country: str

class MatchCreate(BaseModel):
    championship_id: int
    home_team_id: int
    away_team_id: int
    stadium_id: int
    match_date: date
    home_score: int
    away_score: int

# FastAPI app
app = FastAPI(title="European Football Championships API")

# Root endpoint
@app.get("/")
def read_root():
    return {
        "message": "Welcome to European Football Championships API",
        "available_endpoints": {
            "championships": "/championships/",
            "teams": "/teams/",
            "players": "/players/",
            "stadiums": "/stadiums/",
            "matches": "/matches/"
        },
        "documentation": "/docs",
        "openapi": "/openapi.json"
    }

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

# Championship endpoints
@app.post("/championships/", response_model=ChampionshipCreate)
def create_championship(championship: ChampionshipCreate, db: SessionLocal = Depends(get_db)):
    db_championship = Championship(**championship.dict())
    db.add(db_championship)
    db.commit()
    db.refresh(db_championship)
    return db_championship

@app.get("/championships/", response_model=List[ChampionshipCreate])
def read_championships(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    championships = db.query(Championship).offset(skip).limit(limit).all()
    return championships

@app.delete("/championships/{championship_id}")
def delete_championship(championship_id: int, db: SessionLocal = Depends(get_db)):
    db_championship = db.query(Championship).filter(Championship.id == championship_id).first()
    if db_championship is None:
        raise HTTPException(status_code=404, detail="Championship not found")
    db.delete(db_championship)
    db.commit()
    return {"message": "Championship deleted successfully"}

@app.put("/championships/{championship_id}", response_model=ChampionshipCreate)
def update_championship(championship_id: int, championship: ChampionshipCreate, db: SessionLocal = Depends(get_db)):
    db_championship = db.query(Championship).filter(Championship.id == championship_id).first()
    if db_championship is None:
        raise HTTPException(status_code=404, detail="Championship not found")

    for key, value in championship.dict().items():
        setattr(db_championship, key, value)

    db.commit()
    db.refresh(db_championship)
    return db_championship

# Team endpoints
@app.post("/teams/", response_model=TeamCreate)
def create_team(team: TeamCreate, db: SessionLocal = Depends(get_db)):
    db_team = Team(**team.dict())
    db.add(db_team)
    db.commit()
    db.refresh(db_team)
    return db_team

@app.get("/teams/", response_model=List[TeamCreate])
def read_teams(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    teams = db.query(Team).offset(skip).limit(limit).all()
    return teams

@app.delete("/teams/{team_id}")
def delete_team(team_id: int, db: SessionLocal = Depends(get_db)):
    db_team = db.query(Team).filter(Team.id == team_id).first()
    if db_team is None:
        raise HTTPException(status_code=404, detail="Team not found")
    db.delete(db_team)
    db.commit()
    return {"message": "Team deleted successfully"}

@app.put("/teams/{team_id}", response_model=TeamCreate)
def update_team(team_id: int, team: TeamCreate, db: SessionLocal = Depends(get_db)):
    db_team = db.query(Team).filter(Team.id == team_id).first()
    if db_team is None:
        raise HTTPException(status_code=404, detail="Team not found")
    for key, value in team.dict().items():
        setattr(db_team, key, value)

    db.commit()
    db.refresh(db_team)
    return db_team

# Player endpoints
@app.post("/players/", response_model=PlayerCreate)
def create_player(player: PlayerCreate, db: SessionLocal = Depends(get_db)):
    db_player = Player(**player.dict())
    db.add(db_player)
    db.commit()
    db.refresh(db_player)
    return db_player

@app.get("/players/", response_model=List[PlayerCreate])
def read_players(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    players = db.query(Player).offset(skip).limit(limit).all()
    return players

@app.delete("/players/{player_id}")
def delete_player(player_id: int, db: SessionLocal = Depends(get_db)):
    db_player = db.query(Player).filter(Player.id == player_id).first()
    if db_player is None:
        raise HTTPException(status_code=404, detail="Player not found")
    db.delete(db_player)
    db.commit()
    return {"message": "Player deleted successfully"}

@app.put("/players/{player_id}", response_model=PlayerCreate)
def update_player(player_id: int, player: PlayerCreate, db: SessionLocal = Depends(get_db)):
    db_player = db.query(Player).filter(Player.id == player_id).first()
    if db_player is None:
        raise HTTPException(status_code=404, detail="Player not found")

    for key, value in player.dict().items():
        setattr(db_player, key, value)

    db.commit()
    db.refresh(db_player)
    return db_player

# Stadium endpoints
@app.post("/stadiums/", response_model=StadiumCreate)
def create_stadium(stadium: StadiumCreate, db: SessionLocal = Depends(get_db)):
    db_stadium = Stadium(**stadium.dict())
    db.add(db_stadium)
    db.commit()
    db.refresh(db_stadium)
    return db_stadium

@app.get("/stadiums/", response_model=List[StadiumCreate])
def read_stadiums(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    stadiums = db.query(Stadium).offset(skip).limit(limit).all()
    return stadiums

@app.delete("/stadiums/{stadium_id}")
def delete_stadium(stadium_id: int, db: SessionLocal = Depends(get_db)):
    db_stadium = db.query(Stadium).filter(Stadium.id == stadium_id).first()
    if db_stadium is None:
        raise HTTPException(status_code=404, detail="Stadium not found")
    db.delete(db_stadium)
    db.commit()
    return {"message": "Stadium deleted successfully"}

@app.put("/stadiums/{stadium_id}", response_model=StadiumCreate)
def update_stadium(stadium_id: int, stadium: StadiumCreate, db: SessionLocal = Depends(get_db)):
    db_stadium = db.query(Stadium).filter(Stadium.id == stadium_id).first()
    if db_stadium is None:
        raise HTTPException(status_code=404, detail="Stadium not found")

    for key, value in stadium.dict().items():
        setattr(db_stadium, key, value)

    db.commit()
```

```python
    db.refresh(db_stadium)
    return db_stadium

# Match endpoints
@app.post("/matches/", response_model=MatchCreate)
def create_match(match: MatchCreate, db: SessionLocal = Depends(get_db)):
    db_match = Match(**match.dict())
    db.add(db_match)
    db.commit()
    db.refresh(db_match)
    return db_match

@app.get("/matches/", response_model=List[MatchCreate])
def read_matches(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    matches = db.query(Match).offset(skip).limit(limit).all()
    return matches

@app.delete("/matches/{match_id}")
def delete_match(match_id: int, db: SessionLocal = Depends(get_db)):
    db_match = db.query(Match).filter(Match.id == match_id).first()
    if db_match is None:
        raise HTTPException(status_code=404, detail="Match not found")
    db.delete(db_match)
    db.commit()
    return {"message": "Match deleted successfully"}

@app.put("/matches/{match_id}", response_model=MatchCreate)
def update_match(match_id: int, match: MatchCreate, db: SessionLocal = Depends(get_db)):
    db_match = db.query(Match).filter(Match.id == match_id).first()
    if db_match is None:
        raise HTTPException(status_code=404, detail="Match not found")

    for key, value in match.dict().items():
        setattr(db_match, key, value)

    db.commit()
    db.refresh(db_match)
    return db_match

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Рисунки с результатами работы программы:



```
PS C:\Users\user\.vscode\cli\SPP5> python SPP5-1.py
C:\Users\user\.vscode\cli\SPP5\SPP5-1.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_base(). (deprecated since: 2.
0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
INFO:     Started server process [9608]
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.