Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5
По дисциплине: “Современные платформы программирования”
Тема: “Разработка API и базы данных”

Выполнил:
Студент 3 курса
Группы ПО-11
Янущик Д.Д.
Проверил:
Козик И.Д.

Брест 2025

Цель: Приобрести практические навыки разработки API и баз данных.

Вариант 24

База данных: Прокат DVD-дисков.

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;

Код программы:

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

app = FastAPI()

SQLALCHEMY_DATABASE_URL = "sqlite:///./dvd_rental.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

# Модели базы данных

class DVDStatus(enum.Enum):
    available = "available"
    rented = "rented"
    damaged = "damaged"
    lost = "lost"


class Genre(Base):
    __tablename__ = "genres"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, nullable=False)

    movies = relationship("Movie", back_populates="genre")


class Movie(Base):
    __tablename__ = "movies"

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

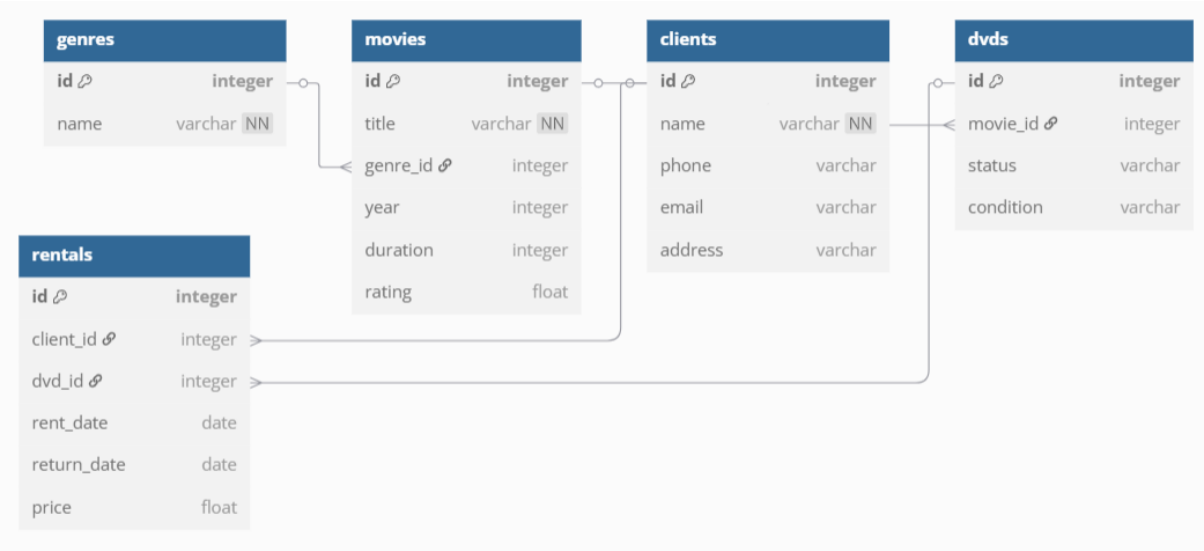
```
    genre_id = Column(Integer, ForeignKey("genres.id"))
    year = Column(Integer)
    duration = Column(Integer)  # в минутах
    rating = Column(Float)

    genre = relationship("Genre", back_populates="movies")
    dvds = relationship("DVD", back_populates="movie")


class Client(Base):
    __tablename__ = "clients"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    phone = Column(String)
    email = Column(String)
    address = Column(String)

    rentals = relationship("Rental", back_populates="client")


class DVD(Base):
    __tablename__ = "dvds"

    id = Column(Integer, primary_key=True, index=True)
    movie_id = Column(Integer, ForeignKey("movies.id"))
    status = Column(Enum(DVDStatus), default=DVDStatus.available)
    condition = Column(String)  # например, "new", "good", "worn"

    movie = relationship("Movie", back_populates="dvds")
    rentals = relationship("Rental", back_populates="dvd")


class Rental(Base):
    __tablename__ = "rentals"

    id = Column(Integer, primary_key=True, index=True)
    client_id = Column(Integer, ForeignKey("clients.id"))
    dvd_id = Column(Integer, ForeignKey("dvds.id"))
    rent_date = Column(Date, default=date.today())
    return_date = Column(Date, nullable=True)
    price = Column(Float)

    client = relationship("Client", back_populates="rentals")
    dvd = relationship("DVD", back_populates="rentals")

Base.metadata.create_all(bind=engine)

# Pydantic модели для запросов и ответов

class GenreCreate(BaseModel):
    name: str


class GenreResponse(BaseModel):
    id: int
    name: str

    class Config:
        from_attributes = True

class MovieCreate(BaseModel):
    title: str
    genre_id: int
```

```
    year: int
    duration: int
    rating: float


class MovieResponse(BaseModel):
    id: int
    title: str
    genre_id: int
    year: int
    duration: int
    rating: float

    class Config:
        from_attributes = True

// Остальные Pydantic модели(Client, DVD, Rental)
```

Эндпоинты

```
@app.post("/genres/", response_model=GenreResponse)
def create_genre(genre: GenreCreate):
  db = SessionLocal()
  db_genre = Genre(**genre.dict())
  db.add(db_genre)
  db.commit()
  db.refresh(db_genre)
  db.close()
  return db_genre


@app.get("/genres/", response_model=List[GenreResponse])
def read_genres():
  db = SessionLocal()
  genres = db.query(Genre).all()
  db.close()
  return genres


@app.get("/genres/{genre_id}", response_model=GenreResponse)
def read_genre(genre_id: int):
  db = SessionLocal()
  genre = db.query(Genre).filter(Genre.id == genre_id).first()
  db.close()
  if genre is None:
    raise HTTPException(status_code=404, detail="Genre not found")
  return genre


@app.put("/genres/{genre_id}", response_model=GenreResponse)
def update_genre(genre_id: int, genre: GenreCreate):
  db = SessionLocal()
  db_genre = db.query(Genre).filter(Genre.id == genre_id).first()
  if db_genre is None:
    db.close()
    raise HTTPException(status_code=404, detail="Genre not found")
  for key, value in genre.dict().items():
    setattr(db_genre, key, value)
  db.commit()
  db.refresh(db_genre)
  db.close()
  return db_genre
```

```
@app.delete("/genres/{genre_id}")
def delete_genre(genre_id: int):
    db = SessionLocal()
    genre = db.query(Genre).filter(Genre.id == genre_id).first()
    if genre is None:
        db.close()
        raise HTTPException(status_code=404, detail="Genre not found")
    db.delete(genre)
    db.commit()
    db.close()
    return {"message": "Genre deleted"}

// Эндпоинты для (Movie, Client, DVD, Rental)
```



Вывод: Приобрел практические навыки разработки API и баз данных.