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

class ClientCreate(BaseModel):
  name: str
  phone: Optional[str] = None
  email: Optional[str] = None
  address: Optional[str] = None


class ClientResponse(BaseModel):
  id: int
  name: str
  phone: Optional[str] = None
  email: Optional[str] = None
  address: Optional[str] = None

  class Config:
    from_attributes = True


class DVDCreate(BaseModel):
  movie_id: int
  status: DVDStatus = DVDStatus.available
  condition: str


class DVDResponse(BaseModel):
  id: int
  movie_id: int
  status: DVDStatus
  condition: str

  class Config:
    from_attributes = True


class RentalCreate(BaseModel):
  client_id: int
  dvd_id: int
  rent_date: date = date.today()
  return_date: Optional[date] = None
  price: float


class RentalResponse(BaseModel):
  id: int
  client_id: int
  dvd_id: int
```

```
    rent_date: date
    return_date: Optional[date] = None
    price: float

    class Config:
        from_attributes = True

# 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"}

@app.post("/movies/", response_model=MovieResponse)
def create_movie(movie: MovieCreate):
    db = SessionLocal()
    db_movie = Movie(**movie.dict())
    db.add(db_movie)
    db.commit()
    db.refresh(db_movie)
    db.close()
    return db_movie


@app.get("/movies/", response_model=List[MovieResponse])
def read_movies():
    db = SessionLocal()
    movies = db.query(Movie).all()
    db.close()
    return movies


@app.get("/movies/{movie_id}", response_model=MovieResponse)
def read_movie(movie_id: int):
    db = SessionLocal()
    movie = db.query(Movie).filter(Movie.id == movie_id).first()
    db.close()
    if movie is None:
        raise HTTPException(status_code=404, detail="Movie not found")
    return movie


@app.put("/movies/{movie_id}", response_model=MovieResponse)
def update_movie(movie_id: int, movie: MovieCreate):
    db = SessionLocal()
    db_movie = db.query(Movie).filter(Movie.id == movie_id).first()
    if db_movie is None:
        db.close()
        raise HTTPException(status_code=404, detail="Movie not found")
    for key, value in movie.dict().items():
        setattr(db_movie, key, value)
    db.commit()
    db.refresh(db_movie)
    db.close()
    return db_movie


@app.delete("/movies/{movie_id}")
def delete_movie(movie_id: int):
    db = SessionLocal()
    movie = db.query(Movie).filter(Movie.id == movie_id).first()
    if movie is None:
        db.close()
        raise HTTPException(status_code=404, detail="Movie not found")
    db.delete(movie)
    db.commit()
    db.close()
    return {"message": "Movie deleted"}


@app.post("/clients/", response_model=ClientResponse)
def create_client(client: ClientCreate):
    db = SessionLocal()
```

```
    db_client = Client(**client.dict())
    db.add(db_client)
    db.commit()
    db.refresh(db_client)
    db.close()
    return db_client


@app.get("/clients/", response_model=List[ClientResponse])
def read_clients():
    db = SessionLocal()
    clients = db.query(Client).all()
    db.close()
    return clients


@app.get("/clients/{client_id}", response_model=ClientResponse)
def read_client(client_id: int):
    db = SessionLocal()
    client = db.query(Client).filter(Client.id == client_id).first()
    db.close()
    if client is None:
        raise HTTPException(status_code=404, detail="Client not found")
    return client


@app.put("/clients/{client_id}", response_model=ClientResponse)
def update_client(client_id: int, client: ClientCreate):
    db = SessionLocal()
    db_client = db.query(Client).filter(Client.id == client_id).first()
    if db_client is None:
        db.close()
        raise HTTPException(status_code=404, detail="Client not found")
    for key, value in client.dict().items():
        setattr(db_client, key, value)
    db.commit()
    db.refresh(db_client)
    db.close()
    return db_client


@app.delete("/clients/{client_id}")
def delete_client(client_id: int):
    db = SessionLocal()
    client = db.query(Client).filter(Client.id == client_id).first()
    if client is None:
        db.close()
        raise HTTPException(status_code=404, detail="Client not found")
    db.delete(client)
    db.commit()
    db.close()
    return {"message": "Client deleted"}


@app.post("/dvds/", response_model=DVDResponse)
def create_dvd(dvd: DVDCreate):
    db = SessionLocal()
    db_dvd = DVD(**dvd.dict())
    db.add(db_dvd)
    db.commit()
    db.refresh(db_dvd)
    db.close()
    return db_dvd
```

```python
@app.get("/dvds/", response_model=List[DVDResponse])
def read_dvds():
    db = SessionLocal()
    dvds = db.query(DVD).all()
    db.close()
    return dvds


@app.get("/dvds/{dvd_id}", response_model=DVDResponse)
def read_dvd(dvd_id: int):
    db = SessionLocal()
    dvd = db.query(DVD).filter(DVD.id == dvd_id).first()
    db.close()
    if dvd is None:
        raise HTTPException(status_code=404, detail="DVD not found")
    return dvd


@app.put("/dvds/{dvd_id}", response_model=DVDResponse)
def update_dvd(dvd_id: int, dvd: DVDCreate):
    db = SessionLocal()
    db_dvd = db.query(DVD).filter(DVD.id == dvd_id).first()
    if db_dvd is None:
        db.close()
        raise HTTPException(status_code=404, detail="DVD not found")
    for key, value in dvd.dict().items():
        setattr(db_dvd, key, value)
    db.commit()
    db.refresh(db_dvd)
    db.close()
    return db_dvd


@app.delete("/dvds/{dvd_id}")
def delete_dvd(dvd_id: int):
    db = SessionLocal()
    dvd = db.query(DVD).filter(DVD.id == dvd_id).first()
    if dvd is None:
        db.close()
        raise HTTPException(status_code=404, detail="DVD not found")
    db.delete(dvd)
    db.commit()
    db.close()
    return {"message": "DVD deleted"}


@app.post("/rentals/", response_model=RentalResponse)
def create_rental(rental: RentalCreate):
    db = SessionLocal()

    client = db.query(Client).filter(Client.id == rental.client_id).first()
    if client is None:
        db.close()
        raise HTTPException(status_code=404, detail="Client not found")

    dvd = db.query(DVD).filter(DVD.id == rental.dvd_id).first()
    if dvd is None:
        db.close()
        raise HTTPException(status_code=404, detail="DVD not found")

    if dvd.status != DVDStatus.available:
        db.close()
        raise HTTPException(status_code=400, detail="DVD is not available for rent")
```

```
    db_rental = Rental(**rental.dict())
    db.add(db_rental)

    dvd.status = DVDStatus.rented

    db.commit()
    db.refresh(db_rental)
    db.close()
    return db_rental


@app.get("/rentals/", response_model=List[RentalResponse])
def read_rentals():
    db = SessionLocal()
    rentals = db.query(Rental).all()
    db.close()
    return rentals


@app.get("/rentals/{rental_id}", response_model=RentalResponse)
def read_rental(rental_id: int):
    db = SessionLocal()
    rental = db.query(Rental).filter(Rental.id == rental_id).first()
    db.close()
    if rental is None:
        raise HTTPException(status_code=404, detail="Rental not found")
    return rental


@app.put("/rentals/{rental_id}", response_model=RentalResponse)
def update_rental(rental_id: int, rental: RentalCreate):
    db = SessionLocal()
    db_rental = db.query(Rental).filter(Rental.id == rental_id).first()
    if db_rental is None:
        db.close()
        raise HTTPException(status_code=404, detail="Rental not found")

    for key, value in rental.dict().items():
        setattr(db_rental, key, value)

    db.commit()
    db.refresh(db_rental)
    db.close()
    return db_rental


@app.delete("/rentals/{rental_id}")
def delete_rental(rental_id: int):
    db = SessionLocal()
    rental = db.query(Rental).filter(Rental.id == rental_id).first()
    if rental is None:
        db.close()
        raise HTTPException(status_code=404, detail="Rental not found")

    db.delete(rental)
    db.commit()
    db.close()
    return {"message": "Rental deleted"}


@app.post("/rentals/{rental_id}/return")
def return_rental(rental_id: int, return_date: date = date.today()):
    db = SessionLocal()
```

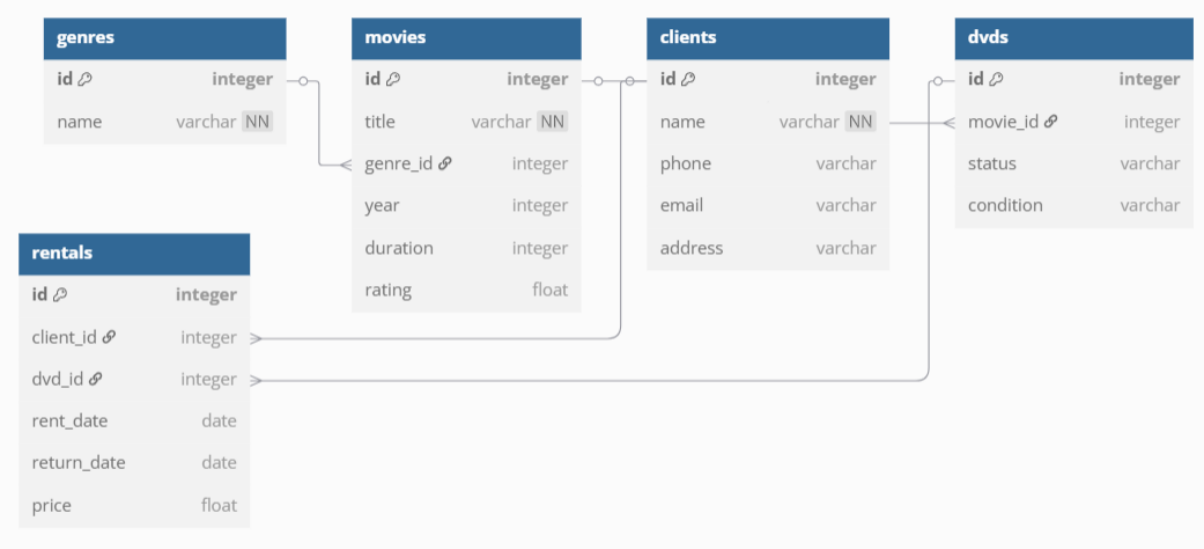
```
rental = db.query(Rental).filter(Rental.id == rental_id).first()
if rental is None:
    db.close()
    raise HTTPException(status_code=404, detail="Rental not found")

if rental.return_date is not None:
    db.close()
    raise HTTPException(status_code=400, detail="DVD already returned")

rental.return_date = return_date

dvd = db.query(DVD).filter(DVD.id == rental.dvd_id).first()
if dvd:
    dvd.status = DVDStatus.available

db.commit()
db.close()
return {"message": "DVD returned successfully"}
```



Вывод: Приобрел практические навыки разработки API и баз данных.