МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №5
По дисциплине “**Современные платформы программирования**”

Выполнил:
студент группы ПО-11
Сымоник И.А.
Проверил:
Козик И. Д.

Брест 2025

Цель: приобрести практические навыки разработки API и баз данных.

Вариант 18

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

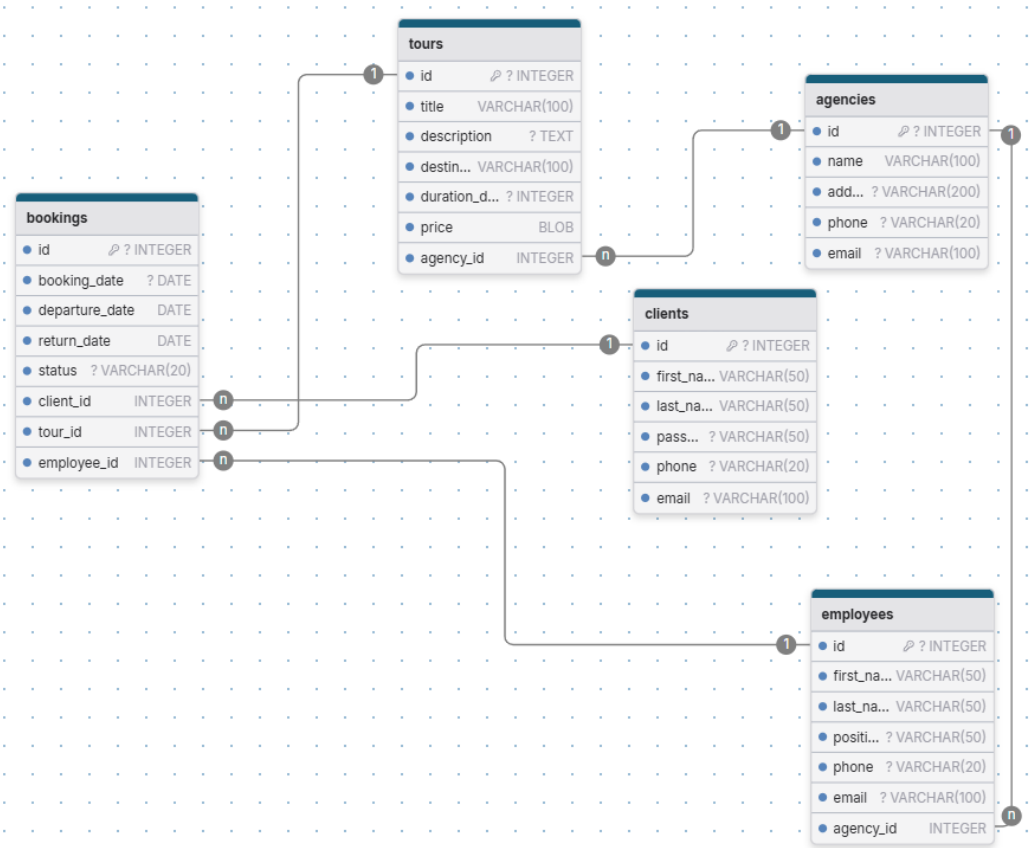
3. На языке Python с использованием SQLAlchemy реализовать подключение к БД;

4. Реализовать основные операции с данными (выборку, добавление, удаление, модификацию);

5. Для каждой реализованной операции с использованием FastAPI реализовать отдельный эндпойнт;

База данных Туристические агентства

Схема БД:



Код программы:

Api/__init__.py

```
from fastapi import APIRouter
from .agency import router as agency_router
from .tour import router as tour_router
from .client import router as client_router
from .employee import router as employee_router
from .booking import router as booking_router

router = APIRouter()
router.include_router(agency_router)
router.include_router(tour_router)
router.include_router(client_router)
```

```
router.include_router(employee_router)
router.include_router(booking_router)

__all__ = ["router"]
```

Api/agency.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from database import get_db
from schemas.agency import Agency, AgencyCreate
from crud.agency import (
    get_agency, get_agencies, create_agency,
    update_agency, delete_agency
)

router = APIRouter(prefix="/agencies", tags=["agencies"])

@router.post("/", response_model=Agency, status_code=201)
def create_agency_endpoint(agency: AgencyCreate, db: Session = Depends(get_db)):
    return create_agency(db=db, agency=agency)

@router.get("/", response_model=List[Agency])
def read_agencies_endpoint(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    return get_agencies(db, skip=skip, limit=limit)

@router.get("/{agency_id}", response_model=Agency)
def read_agency_endpoint(agency_id: int, db: Session = Depends(get_db)):
    db_agency = get_agency(db, agency_id=agency_id)
    if db_agency is None:
        raise HTTPException(status_code=404, detail="Agency not found")
    return db_agency

@router.put("/{agency_id}", response_model=Agency)
def update_agency_endpoint(
    agency_id: int,
    agency: AgencyCreate,
    db: Session = Depends(get_db)
):
    db_agency = update_agency(db, agency_id=agency_id, agency_data=agency)
    if db_agency is None:
        raise HTTPException(status_code=404, detail="Agency not found")
    return db_agency

@router.delete("/{agency_id}")
def delete_agency_endpoint(agency_id: int, db: Session = Depends(get_db)):
    success = delete_agency(db, agency_id=agency_id)
    if not success:
        raise HTTPException(status_code=404, detail="Agency not found")
    return {"message": "Agency deleted successfully"}
```

api/booking.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from datetime import date
from database import get_db
from schemas.booking import Booking, BookingCreate
from crud.booking import (
    get_booking, get_bookings, get_bookings_by_client,
    get_bookings_by_employee, get_upcoming_bookings,
    create_booking, update_booking, delete_booking
)
```

```
router = APIRouter(prefix="/bookings", tags=["bookings"])

@router.post("/", response_model=Booking, status_code=201)
def create_booking_endpoint(
    booking: BookingCreate,
    db: Session = Depends(get_db)
):
    return create_booking(db=db, booking=booking)

@router.get("/", response_model=List[Booking])
def read_bookings_endpoint(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    return get_bookings(db, skip=skip, limit=limit)

@router.get("/upcoming", response_model=List[Booking])
def read_upcoming_bookings_endpoint(db: Session = Depends(get_db)):
    bookings = get_upcoming_bookings(db)
    if not bookings:
        raise HTTPException(
            status_code=404,
            detail="No upcoming bookings found"
        )
    return bookings

@router.get("/client/{client_id}", response_model=List[Booking])
def read_bookings_by_client_endpoint(
    client_id: int,
    db: Session = Depends(get_db)
):
    bookings = get_bookings_by_client(db, client_id=client_id)
    if not bookings:
        raise HTTPException(
            status_code=404,
            detail="No bookings found for this client"
        )
    return bookings

@router.get("/employee/{employee_id}", response_model=List[Booking])
def read_bookings_by_employee_endpoint(
    employee_id: int,
    db: Session = Depends(get_db)
):
    bookings = get_bookings_by_employee(db, employee_id=employee_id)
    if not bookings:
        raise HTTPException(
            status_code=404,
            detail="No bookings found for this employee"
        )
    return bookings

@router.get("/{booking_id}", response_model=Booking)
def read_booking_endpoint(
    booking_id: int,
    db: Session = Depends(get_db)
):
    db_booking = get_booking(db, booking_id=booking_id)
    if db_booking is None:
        raise HTTPException(status_code=404, detail="Booking not found")
    return db_booking

@router.put("/{booking_id}", response_model=Booking)
def update_booking_endpoint(
    booking_id: int,
    booking: BookingCreate,
    db: Session = Depends(get_db)
):
    db_booking = update_booking(db, booking_id=booking_id, booking_data=booking)
```

```
    if db_booking is None:
        raise HTTPException(status_code=404, detail="Booking not found")
    return db_booking

@router.delete("/{booking_id}")
def delete_booking_endpoint(
    booking_id: int,
    db: Session = Depends(get_db)
):
    success = delete_booking(db, booking_id=booking_id)
    if not success:
        raise HTTPException(status_code=404, detail="Booking not found")
    return {"message": "Booking deleted successfully"}
```

api/client.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from database import get_db
from schemas.client import Client, ClientCreate
from crud.client import (
    get_client, get_client_by_passport, get_clients,
    create_client, update_client, delete_client
)

router = APIRouter(prefix="/clients", tags=["clients"])

@router.post("/", response_model=Client, status_code=201)
def create_client_endpoint(client: ClientCreate, db: Session = Depends(get_db)):
    db_client = get_client_by_passport(db, passport_number=client.passport_number)
    if db_client:
        raise HTTPException(
            status_code=400,
            detail="Client with this passport already exists"
        )
    return create_client(db=db, client=client)

@router.get("/", response_model=List[Client])
def read_clients_endpoint(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    return get_clients(db, skip=skip, limit=limit)

@router.get("/{client_id}", response_model=Client)
def read_client_endpoint(client_id: int, db: Session = Depends(get_db)):
    db_client = get_client(db, client_id=client_id)
    if db_client is None:
        raise HTTPException(status_code=404, detail="Client not found")
    return db_client

@router.get("/by-passport/{passport_number}", response_model=Client)
def read_client_by_passport_endpoint(
    passport_number: str,
    db: Session = Depends(get_db)
):
    db_client = get_client_by_passport(db, passport_number=passport_number)
    if db_client is None:
        raise HTTPException(status_code=404, detail="Client not found")
    return db_client

@router.put("/{client_id}", response_model=Client)
def update_client_endpoint(
    client_id: int,
    client: ClientCreate,
    db: Session = Depends(get_db)
):
    db_client = update_client(db, client_id=client_id, client_data=client)
```

```
    if db_client is None:
        raise HTTPException(status_code=404, detail="Client not found")
    return db_client

@router.delete("/{client_id}")
def delete_client_endpoint(client_id: int, db: Session = Depends(get_db)):
    success = delete_client(db, client_id=client_id)
    if not success:
        raise HTTPException(status_code=404, detail="Client not found")
    return {"message": "Client deleted successfully"}
```

api/employee.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from database import get_db
from schemas.employee import Employee, EmployeeCreate
from crud.employee import (
    get_employee, get_employees, get_employees_by_agency,
    create_employee, update_employee, delete_employee
)

router = APIRouter(prefix="/employees", tags=["employees"])

@router.post("/", response_model=Employee, status_code=201)
def create_employee_endpoint(
    employee: EmployeeCreate,
    db: Session = Depends(get_db)
):
    return create_employee(db=db, employee=employee)

@router.get("/", response_model=List[Employee])
def read_employees_endpoint(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    return get_employees(db, skip=skip, limit=limit)

@router.get("/agency/{agency_id}", response_model=List[Employee])
def read_employees_by_agency_endpoint(
    agency_id: int,
    db: Session = Depends(get_db)
):
    employees = get_employees_by_agency(db, agency_id=agency_id)
    if not employees:
        raise HTTPException(
            status_code=404,
            detail="No employees found for this agency"
        )
    return employees

@router.get("/{employee_id}", response_model=Employee)
def read_employee_endpoint(
    employee_id: int,
    db: Session = Depends(get_db)
):
    db_employee = get_employee(db, employee_id=employee_id)
    if db_employee is None:
        raise HTTPException(status_code=404, detail="Employee not found")
    return db_employee

@router.put("/{employee_id}", response_model=Employee)
def update_employee_endpoint(
    employee_id: int,
    employee: EmployeeCreate,
    db: Session = Depends(get_db)
):
    db_employee = update_employee(db, employee_id=employee_id, employee_data=employee)
```

```
    if db_employee is None:
        raise HTTPException(status_code=404, detail="Employee not found")
    return db_employee

@router.delete("/{employee_id}")
def delete_employee_endpoint(
    employee_id: int,
    db: Session = Depends(get_db)
):
    success = delete_employee(db, employee_id=employee_id)
    if not success:
        raise HTTPException(status_code=404, detail="Employee not found")
    return {"message": "Employee deleted successfully"}
```

api/tour.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from database import get_db
from schemas.tour import Tour, TourCreate
from crud.tour import (
    get_tour, get_tours, get_tours_by_agency,
    create_tour, update_tour, delete_tour
)

router = APIRouter(prefix="/tours", tags=["tours"])

@router.post("/", response_model=Tour, status_code=201)
def create_tour_endpoint(tour: TourCreate, db: Session = Depends(get_db)):
    return create_tour(db=db, tour=tour)

@router.get("/", response_model=List[Tour])
def read_tours_endpoint(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    return get_tours(db, skip=skip, limit=limit)

@router.get("/agency/{agency_id}", response_model=List[Tour])
def read_tours_by_agency_endpoint(
    agency_id: int,
    db: Session = Depends(get_db)
):
    tours = get_tours_by_agency(db, agency_id=agency_id)
    if not tours:
        raise HTTPException(
            status_code=404,
            detail="No tours found for this agency"
        )
    return tours

@router.get("/{tour_id}", response_model=Tour)
def read_tour_endpoint(tour_id: int, db: Session = Depends(get_db)):
    db_tour = get_tour(db, tour_id=tour_id)
    if db_tour is None:
        raise HTTPException(status_code=404, detail="Tour not found")
    return db_tour

@router.put("/{tour_id}", response_model=Tour)
def update_tour_endpoint(
    tour_id: int,
    tour: TourCreate,
    db: Session = Depends(get_db)
):
    db_tour = update_tour(db, tour_id=tour_id, tour_data=tour)
    if db_tour is None:
        raise HTTPException(status_code=404, detail="Tour not found")
    return db_tour
```

```
@router.delete("/{tour_id}")
def delete_tour_endpoint(tour_id: int, db: Session = Depends(get_db)):
    success = delete_tour(db, tour_id=tour_id)
    if not success:
        raise HTTPException(status_code=404, detail="Tour not found")
    return {"message": "Tour deleted successfully"}
```

crud/__init__.py

```
from .agency import (
    get_agency, get_agencies, create_agency,
    update_agency, delete_agency
)
from .tour import (
    get_tour, get_tours, get_tours_by_agency,
    create_tour, update_tour, delete_tour
)
from .client import (
    get_client, get_client_by_passport, get_clients,
    create_client, update_client, delete_client
)
from .employee import (
    get_employee, get_employees, get_employees_by_agency,
    create_employee, update_employee, delete_employee
)
from .booking import (
    get_booking, get_bookings, get_bookings_by_client,
    get_bookings_by_employee, get_upcoming_bookings,
    create_booking, update_booking, delete_booking
)

__all__ = [
    # Agency
    "get_agency", "get_agencies", "create_agency",
    "update_agency", "delete_agency",

    # Tour
    "get_tour", "get_tours", "get_tours_by_agency",
    "create_tour", "update_tour", "delete_tour",

    # Client
    "get_client", "get_client_by_passport", "get_clients",
    "create_client", "update_client", "delete_client",

    # Employee
    "get_employee", "get_employees", "get_employees_by_agency",
    "create_employee", "update_employee", "delete_employee",

    # Booking
    "get_booking", "get_bookings", "get_bookings_by_client",
    "get_bookings_by_employee", "get_upcoming_bookings",
    "create_booking", "update_booking", "delete_booking"
]
```

Crud/agency.py

```
from sqlalchemy.orm import Session
from models.agency import Agency
from schemas.agency import AgencyCreate

def get_agency(db: Session, agency_id: int):
    return db.query(Agency).filter(Agency.id == agency_id).first()

def get_agencies(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Agency).offset(skip).limit(limit).all()

def create_agency(db: Session, agency: AgencyCreate):
    db_agency = Agency(**agency.dict())
    db.add(db_agency)
```

```
    db.commit()
    db.refresh(db_agency)
    return db_agency

def update_agency(db: Session, agency_id: int, agency_data: AgencyCreate):
    db_agency = db.query(Agency).filter(Agency.id == agency_id).first()
    if db_agency:
        for key, value in agency_data.dict().items():
            setattr(db_agency, key, value)
        db.commit()
        db.refresh(db_agency)
    return db_agency

def delete_agency(db: Session, agency_id: int):
    db_agency = db.query(Agency).filter(Agency.id == agency_id).first()
    if db_agency:
        db.delete(db_agency)
        db.commit()
    return db_agency
```

crud/booking.py

```
from sqlalchemy.orm import Session
from models.booking import Booking
from schemas.booking import BookingCreate
from datetime import date

def get_booking(db: Session, booking_id: int):
    return db.query(Booking).filter(Booking.id == booking_id).first()

def get_bookings(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Booking).offset(skip).limit(limit).all()

def get_bookings_by_client(db: Session, client_id: int):
    return db.query(Booking).filter(Booking.client_id == client_id).all()

def get_bookings_by_employee(db: Session, employee_id: int):
    return db.query(Booking).filter(Booking.employee_id == employee_id).all()

def get_upcoming_bookings(db: Session):
    return db.query(Booking).filter(Booking.departure_date >= date.today()).all()

def create_booking(db: Session, booking: BookingCreate):
    db_booking = Booking(**booking.dict())
    db.add(db_booking)
    db.commit()
    db.refresh(db_booking)
    return db_booking

def update_booking(db: Session, booking_id: int, booking_data: BookingCreate):
    db_booking = db.query(Booking).filter(Booking.id == booking_id).first()
    if db_booking:
        for key, value in booking_data.dict().items():
            setattr(db_booking, key, value)
        db.commit()
        db.refresh(db_booking)
    return db_booking

def delete_booking(db: Session, booking_id: int):
    db_booking = db.query(Booking).filter(Booking.id == booking_id).first()
    if db_booking:
        db.delete(db_booking)
        db.commit()
    return db_booking
```

crud/client.py

```
from sqlalchemy.orm import Session
from models.client import Client
from schemas.client import ClientCreate
```

```
def get_client(db: Session, client_id: int):
    return db.query(Client).filter(Client.id == client_id).first()

def get_client_by_passport(db: Session, passport_number: str):
    return db.query(Client).filter(Client.passport_number == passport_number).first()

def get_clients(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Client).offset(skip).limit(limit).all()

def create_client(db: Session, client: ClientCreate):
    db_client = Client(**client.dict())
    db.add(db_client)
    db.commit()
    db.refresh(db_client)
    return db_client

def update_client(db: Session, client_id: int, client_data: ClientCreate):
    db_client = db.query(Client).filter(Client.id == client_id).first()
    if db_client:
        for key, value in client_data.dict().items():
            setattr(db_client, key, value)
        db.commit()
        db.refresh(db_client)
    return db_client

def delete_client(db: Session, client_id: int):
    db_client = db.query(Client).filter(Client.id == client_id).first()
    if db_client:
        db.delete(db_client)
        db.commit()
    return db_client
```

crud/employee.py

```
from sqlalchemy.orm import Session
from models.employee import Employee
from schemas.employee import EmployeeCreate

def get_employee(db: Session, employee_id: int):
    return db.query(Employee).filter(Employee.id == employee_id).first()

def get_employees(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Employee).offset(skip).limit(limit).all()

def get_employees_by_agency(db: Session, agency_id: int):
    return db.query(Employee).filter(Employee.agency_id == agency_id).all()

def create_employee(db: Session, employee: EmployeeCreate):
    db_employee = Employee(**employee.dict())
    db.add(db_employee)
    db.commit()
    db.refresh(db_employee)
    return db_employee

def update_employee(db: Session, employee_id: int, employee_data: EmployeeCreate):
    db_employee = db.query(Employee).filter(Employee.id == employee_id).first()
    if db_employee:
        for key, value in employee_data.dict().items():
            setattr(db_employee, key, value)
        db.commit()
        db.refresh(db_employee)
    return db_employee

def delete_employee(db: Session, employee_id: int):
    db_employee = db.query(Employee).filter(Employee.id == employee_id).first()
    if db_employee:
        db.delete(db_employee)
        db.commit()
    return db_employee
```

curd/tour.py

```
from sqlalchemy.orm import Session
from models.tour import Tour
from schemas.tour import TourCreate

def get_tour(db: Session, tour_id: int):
    return db.query(Tour).filter(Tour.id == tour_id).first()

def get_tours(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Tour).offset(skip).limit(limit).all()

def get_tours_by_agency(db: Session, agency_id: int):
    return db.query(Tour).filter(Tour.agency_id == agency_id).all()

def create_tour(db: Session, tour: TourCreate):
    db_tour = Tour(**tour.dict())
    db.add(db_tour)
    db.commit()
    db.refresh(db_tour)
    return db_tour

def update_tour(db: Session, tour_id: int, tour_data: TourCreate):
    db_tour = db.query(Tour).filter(Tour.id == tour_id).first()
    if db_tour:
        for key, value in tour_data.dict().items():
            setattr(db_tour, key, value)
        db.commit()
        db.refresh(db_tour)
    return db_tour

def delete_tour(db: Session, tour_id: int):
    db_tour = db.query(Tour).filter(Tour.id == tour_id).first()
    if db_tour:
        db.delete(db_tour)
        db.commit()
    return db_tour
```

models/agency.py

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import relationship
from database import Base

class Agency(Base):
    __tablename__ = 'agencies'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    address = Column(String(200))
    phone = Column(String(20))
    email = Column(String(100))

    employees = relationship("Employee", back_populates="agency")
    tours = relationship("Tour", back_populates="agency")

    def __repr__(self):
        return f"<Agency(id={self.id}, name='{self.name}')>"

from pydantic import BaseModel
from typing import Optional

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

class AgencyCreate(AgencyBase):
```

```
    pass

class Agency(AgencyBase):
    id: int

    class Config:
        orm_mode = True
```

models/booking.py

```
from sqlalchemy import Column, Integer, Date, String, ForeignKey
from sqlalchemy.orm import relationship
from database import Base
from datetime import date

class Booking(Base):
    __tablename__ = 'bookings'

    id = Column(Integer, primary_key=True, index=True)
    booking_date = Column(Date, default=date.today())
    departure_date = Column(Date)
    return_date = Column(Date)
    status = Column(String(20), default='confirmed')

    client_id = Column(Integer, ForeignKey('clients.id'))
    tour_id = Column(Integer, ForeignKey('tours.id'))
    employee_id = Column(Integer, ForeignKey('employees.id'))

    client = relationship("Client", back_populates="bookings")
    tour = relationship("Tour", back_populates="bookings")
    employee = relationship("Employee", back_populates="bookings")

    def __repr__(self):
        return f"<Booking(id={self.id}, tour_id={self.tour_id}, status='{self.status}')>"
from pydantic import BaseModel
from datetime import date
from typing import Optional

class BookingBase(BaseModel):
    departure_date: date
    return_date: date
    client_id: int
    tour_id: int
    employee_id: int
    status: Optional[str] = 'confirmed'

class BookingCreate(BookingBase):
    pass

class Booking(BookingBase):
    id: int
    booking_date: date

    class Config:
        orm_mode = True
```

models/client.py

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import relationship
from database import Base

class Client(Base):
    __tablename__ = 'clients'

    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    passport_number = Column(String(50), unique=True)
    phone = Column(String(20))
```

```
    email = Column(String(100))

    bookings = relationship("Booking", back_populates="client")

    def __repr__(self):
        return f"<Client(id={self.id}, name='{self.first_name} {self.last_name}')>"
from pydantic import BaseModel
from typing import Optional

class ClientBase(BaseModel):
    first_name: str
    last_name: str
    passport_number: Optional[str] = None
    phone: Optional[str] = None
    email: Optional[str] = None

class ClientCreate(ClientBase):
    pass

class Client(ClientBase):
    id: int

    class Config:
        orm_mode = True
```

models/employee.py

```
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from database import Base

class Employee(Base):
    __tablename__ = 'employees'

    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    position = Column(String(50))
    phone = Column(String(20))
    email = Column(String(100))
    agency_id = Column(Integer, ForeignKey('agencies.id'))

    agency = relationship("Agency", back_populates="employees")
    bookings = relationship("Booking", back_populates="employee")

    def __repr__(self):
        return f"<Employee(id={self.id}, name='{self.first_name} {self.last_name}')>"
from pydantic import BaseModel
from typing import Optional

class EmployeeBase(BaseModel):
    first_name: str
    last_name: str
    position: Optional[str] = None
    phone: Optional[str] = None
    email: Optional[str] = None
    agency_id: int

class EmployeeCreate(EmployeeBase):
    pass

class Employee(EmployeeBase):
    id: int

    class Config:
        orm_mode = True
```

models/tour.py

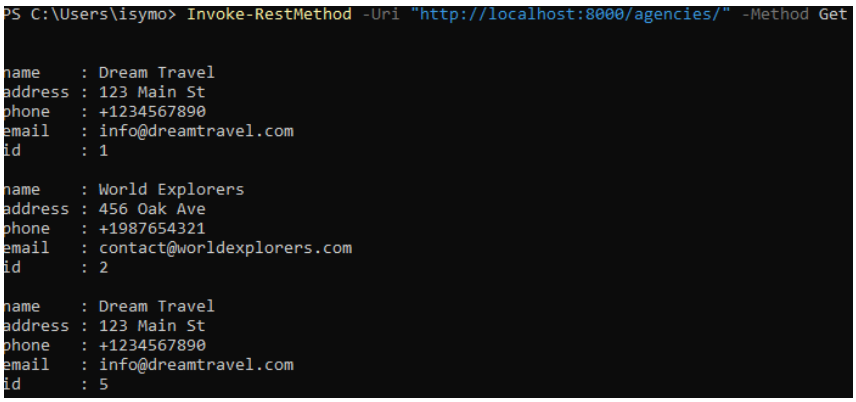
```
from sqlalchemy import Column, Integer, String, Float, ForeignKey
```

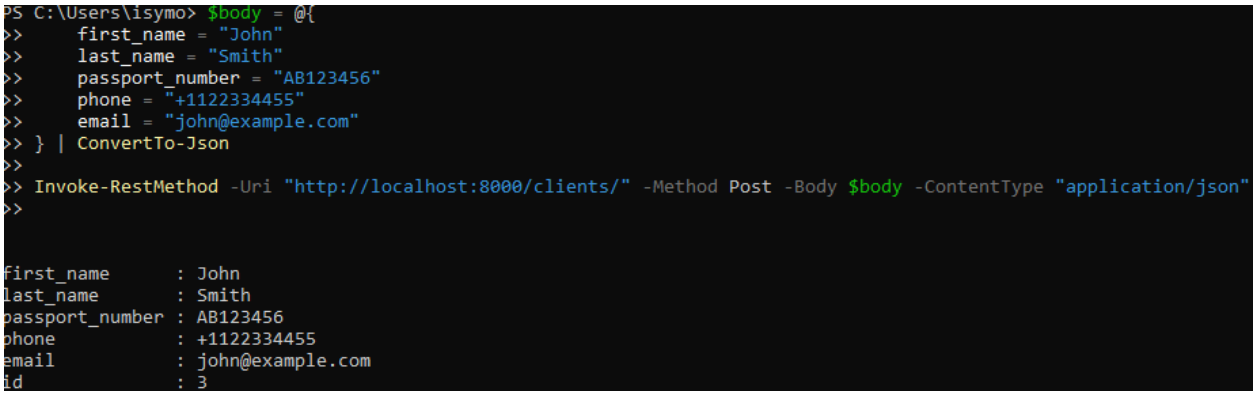
```
from sqlalchemy.orm import relationship
from database import Base

class Tour(Base):
    __tablename__ = 'tours'

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String(100), nullable=False)
    description = Column(String(500))
    destination = Column(String(100), nullable=False)
    duration_days = Column(Integer)
    price = Column(Float, nullable=False)
    agency_id = Column(Integer, ForeignKey('agencies.id'))

    agency = relationship("Agency", back_populates="tours")
    bookings = relationship("Booking", back_populates="tour")

    def __repr__(self):
        return f"<Tour(id={self.id}, title='{self.title}', price={self.price}>"
from pydantic import BaseModel
from typing import Optional

class TourBase(BaseModel):
    title: str
    description: Optional[str] = None
    destination: str
    duration_days: Optional[int] = None
    price: float
    agency_id: int

class TourCreate(TourBase):
    pass

class Tour(TourBase):
    id: int

    class Config:
        orm_mode = True


from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from config import settings

# Используем settings.DATABASE_URL (в верхнем регистре)
engine = create_engine(settings.DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

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

main.py

```
from fastapi import FastAPI
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

from models.agency import Agency
from models.tour import Tour
from models.client import Client
```

```
from models.employee import Employee
from models.booking import Booking

load_dotenv()

app = FastAPI(
    title="Travel Agency API",
    description="API для управления туристическим агентством",
    version="1.0.0",
    docs_url="/docs",
    redoc_url="/redoc"
)

DATABASE_URL = os.getenv("DATABASE_URL", "sqlite:///./travel_agency.db")
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

from api.agency import router as agency_router
from api.tour import router as tour_router
from api.client import router as client_router
from api.employee import router as employee_router
from api.booking import router as booking_router
from services.initialization import router as init_router

Base.metadata.create_all(bind=engine)

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

app.include_router(agency_router)
app.include_router(tour_router)
app.include_router(client_router)
app.include_router(employee_router)
app.include_router(booking_router)
app.include_router(init_router)

@app.get("/", tags=["Root"])
def read_root():
    return {
        "message": "Добро пожаловать в API туристического агентства",
        "endpoints": {
            "agencies": "/agencies/",
            "tours": "/tours/",
            "clients": "/clients/",
            "employees": "/employees/",
            "bookings": "/bookings/",
            "init_db": "/init-db/"
        },
        "documentation": {
            "swagger": "/docs",
            "redoc": "/redoc"
        }
    }

@app.get("/health", tags=["Utility"])
def health_check():
    return {"status": "OK", "message": "Сервис работает нормально"}

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





Вывод: приобрели практические навыки разработки API и баз данных.