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

Лабораторная работа №5
По дисциплине «Современные платформы программирования»

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

Брест 2025

Цель работы: приобрести практические навыки разработки API и баз данных.

Вариант 14

Общее задание

1. Реализовать базу данных из не менее 5 таблиц на заданную тематику.
1. 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:

```
# main.py
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
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship

# Инициализация FastAPI
app = FastAPI()

# Подключение к SQLite
SQLALCHEMY_DATABASE_URL = "sqlite:///./computer_store.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

# Модели базы данных
class Customer(Base):
    __tablename__ = "customers"

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

    orders = relationship("Order", back_populates="customer")

class Manufacturer(Base):
    __tablename__ = "manufacturers"

    manufacturer_id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    country = Column(String)
    contact_person = Column(String)
    website = Column(String)

    products = relationship("Product", back_populates="manufacturer")

class Product(Base):
    __tablename__ = "products"

    product_id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    category = Column(String)
    price = Column(Float)
    quantity_in_stock = Column(Integer)
    warranty_period = Column(Integer)  # в месяцах
    manufacturer_id = Column(Integer, ForeignKey("manufacturers.manufacturer_id"))

    manufacturer = relationship("Manufacturer", back_populates="products")
    order_items = relationship("OrderItem", back_populates="product")
```

```
class Order(Base):
    __tablename__ = "orders"

    order_id = Column(Integer, primary_key=True, index=True)
    customer_id = Column(Integer, ForeignKey("customers.customer_id"))
    order_date = Column(Date)
    total_amount = Column(Float)
    status = Column(String)  # "pending", "completed", "cancelled"

    customer = relationship("Customer", back_populates="orders")
    items = relationship("OrderItem", back_populates="order")

class OrderItem(Base):
    __tablename__ = "order_items"

    order_item_id = Column(Integer, primary_key=True, index=True)
    order_id = Column(Integer, ForeignKey("orders.order_id"))
    product_id = Column(Integer, ForeignKey("products.product_id"))
    quantity = Column(Integer)

    order = relationship("Order", back_populates="items")
    product = relationship("Product", back_populates="order_items")

# Создание таблиц
Base.metadata.create_all(bind=engine)

# Pydantic модели для запросов и ответов
class CustomerCreate(BaseModel):
    name: str
    address: Optional[str] = None
    phone: Optional[str] = None
    email: Optional[str] = None
    registration_date: Optional[date] = None

class CustomerResponse(CustomerCreate):
    customer_id: int

    class Config:
        orm_mode = True

class ManufacturerCreate(BaseModel):
    name: str
    country: Optional[str] = None
    contact_person: Optional[str] = None
    website: Optional[str] = None

class ManufacturerResponse(ManufacturerCreate):
    manufacturer_id: int

    class Config:
        orm_mode = True

class ProductCreate(BaseModel):
    name: str
    category: Optional[str] = None
    price: float
    quantity_in_stock: int
    warranty_period: Optional[int] = None
    manufacturer_id: int

class ProductResponse(ProductCreate):
    product_id: int

    class Config:
        orm_mode = True

class OrderCreate(BaseModel):
    customer_id: int
    order_date: date
    total_amount: float
    status: str

class OrderResponse(OrderCreate):
    order_id: int

    class Config:
        orm_mode = True

class OrderItemCreate(BaseModel):
    order_id: int
    product_id: int
```

```
    quantity: int

class OrderItemResponse(OrderItemCreate):
    order_item_id: int

    class Config:
        orm_mode = True

# Эндпоинты для клиентов
@app.post("/customers/", response_model=CustomerResponse)
def create_customer(customer: CustomerCreate):
    db = SessionLocal()
    db_customer = Customer(**customer.dict())
    db.add(db_customer)
    db.commit()
    db.refresh(db_customer)
    db.close()
    return db_customer

@app.get("/customers/", response_model=List[CustomerResponse])
def read_customers(skip: int = 0, limit: int = 100):
    db = SessionLocal()
    customers = db.query(Customer).offset(skip).limit(limit).all()
    db.close()
    return customers

@app.get("/customers/{customer_id}", response_model=CustomerResponse)
def read_customer(customer_id: int):
    db = SessionLocal()
    customer = db.query(Customer).filter(Customer.customer_id == customer_id).first()
    db.close()
    if customer is None:
        raise HTTPException(status_code=404, detail="Customer not found")
    return customer

@app.put("/customers/{customer_id}", response_model=CustomerResponse)
def update_customer(customer_id: int, customer: CustomerCreate):
    db = SessionLocal()
    db_customer = db.query(Customer).filter(Customer.customer_id == customer_id).first()
    if db_customer is None:
        raise HTTPException(status_code=404, detail="Customer not found")
    for key, value in customer.dict().items():
        setattr(db_customer, key, value)
    db.commit()
    db.refresh(db_customer)
    db.close()
    return db_customer

@app.delete("/customers/{customer_id}")
def delete_customer(customer_id: int):
    db = SessionLocal()
    db_customer = db.query(Customer).filter(Customer.customer_id == customer_id).first()
    if db_customer is None:
        raise HTTPException(status_code=404, detail="Customer not found")
    db.delete(db_customer)
    db.commit()
    db.close()
    return {"message": "Customer deleted successfully"}

# Эндпоинты для производителей
@app.post("/manufacturers/", response_model=ManufacturerResponse)
def create_manufacturer(manufacturer: ManufacturerCreate):
    db = SessionLocal()
    db_manufacturer = Manufacturer(**manufacturer.dict())
    db.add(db_manufacturer)
    db.commit()
    db.refresh(db_manufacturer)
    db.close()
    return db_manufacturer

@app.get("/manufacturers/", response_model=List[ManufacturerResponse])
def read_manufacturers(skip: int = 0, limit: int = 100):
    db = SessionLocal()
    manufacturers = db.query(Manufacturer).offset(skip).limit(limit).all()
    db.close()
    return manufacturers

@app.get("/manufacturers/{manufacturer_id}", response_model=ManufacturerResponse)
def read_manufacturer(manufacturer_id: int):
    db = SessionLocal()
    manufacturer = db.query(Manufacturer).filter(Manufacturer.manufacturer_id == manufacturer_id).first()
    db.close()
```

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

# Эндпоинты для товаров
@app.post("/products/", response_model=ProductResponse)
def create_product(product: ProductCreate):
    db = SessionLocal()
    db_product = Product(**product.dict())
    db.add(db_product)
    db.commit()
    db.refresh(db_product)
    db.close()
    return db_product

@app.get("/products/", response_model=List[ProductResponse])
def read_products(skip: int = 0, limit: int = 100):
    db = SessionLocal()
    products = db.query(Product).offset(skip).limit(limit).all()
    db.close()
    return products

@app.get("/products/{product_id}", response_model=ProductResponse)
def read_product(product_id: int):
    db = SessionLocal()
    product = db.query(Product).filter(Product.product_id == product_id).first()
    db.close()
    if product is None:
        raise HTTPException(status_code=404, detail="Product not found")
    return product

# Эндпоинты для заказов
@app.post("/orders/", response_model=OrderResponse)
def create_order(order: OrderCreate):
    db = SessionLocal()
    db_order = Order(**order.dict())
    db.add(db_order)
    db.commit()
    db.refresh(db_order)
    db.close()
    return db_order

@app.get("/orders/", response_model=List[OrderResponse])
def read_orders(skip: int = 0, limit: int = 100):
    db = SessionLocal()
    orders = db.query(Order).offset(skip).limit(limit).all()
    db.close()
    return orders

# Эндпоинты для составов заказов
@app.post("/order_items/", response_model=OrderItemResponse)
def create_order_item(order_item: OrderItemCreate):
    db = SessionLocal()
    db_order_item = OrderItem(**order_item.dict())
    db.add(db_order_item)
    db.commit()
    db.refresh(db_order_item)
    db.close()
    return db_order_item

@app.get("/order_items/", response_model=List[OrderItemResponse])
def read_order_items(skip: int = 0, limit: int = 100):
    db = SessionLocal()
    order_items = db.query(OrderItem).offset(skip).limit(limit).all()
    db.close()
    return order_items
```

Результат:

Code	Details
200	Response body


```
{
  "name": "Иванов Иван Иванович",
  "address": "г. Москва, ул. Ленина, д. 10, кв. 25",
  "phone": "+7 (495) 123-45-67",
  "email": "ivanov@example.com",
  "registration_date": "2020-01-15",
  "customer_id": 1
}
```


Download

Response headers

```
content-length: 223
content-type: application/json
date: Fri,18 Apr 2025 06:54:03 GMT
server: uvicorn
```

Code	Details
200	Response body


```
[
  {
    "name": "Иванов Иван Иванович",
    "address": "г. Москва, ул. Ленина, д. 10, кв. 25",
    "phone": "+7 (495) 123-45-67",
    "email": "ivanov@example.com",
    "registration_date": "2020-01-15",
    "customer_id": 1
  }
]
```


Download

Response headers

```
content-length: 225
content-type: application/json
date: Fri,18 Apr 2025 06:54:44 GMT
server: uvicorn
```

Таблицы (5)	
customers	CREATE TABLE customers (customer_id INTEGER NOT NULL, name VARC
manufacturers	CREATE TABLE manufacturers (manufacturer_id INTEGER NOT NULL, nam
order_items	CREATE TABLE order_items (order_item_id INTEGER NOT NULL, order_id
orders	CREATE TABLE orders (order_id INTEGER NOT NULL, customer_id INTEGE
products	CREATE TABLE products (product_id INTEGER NOT NULL, name VARCHAI
Индексы (5)	
ix_customers_customer_id	CREATE INDEX ix_customers_customer_id ON customers (customer_id)
ix_manufacturers_manufact...	CREATE INDEX ix_manufacturers_manufacturer_id ON manufacturers (mar
ix_order_items_order_item_id	CREATE INDEX ix_order_items_order_item_id ON order_items (order_item
ix_orders_order_id	CREATE INDEX ix_orders_order_id ON orders (order_id)
ix_products_product_id	CREATE INDEX ix_products_product_id ON products (product_id)

Вывод: приобрел практические навыки разработки API и баз данных.