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

Лабораторная работа №5
По дисциплине “**Современные платформы программирования**”
Тема: “**Работа с FastAPI и SQLAlchemy**”

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

Брест 2025

Цель: приобрести практические навыки разработки API и баз данных.

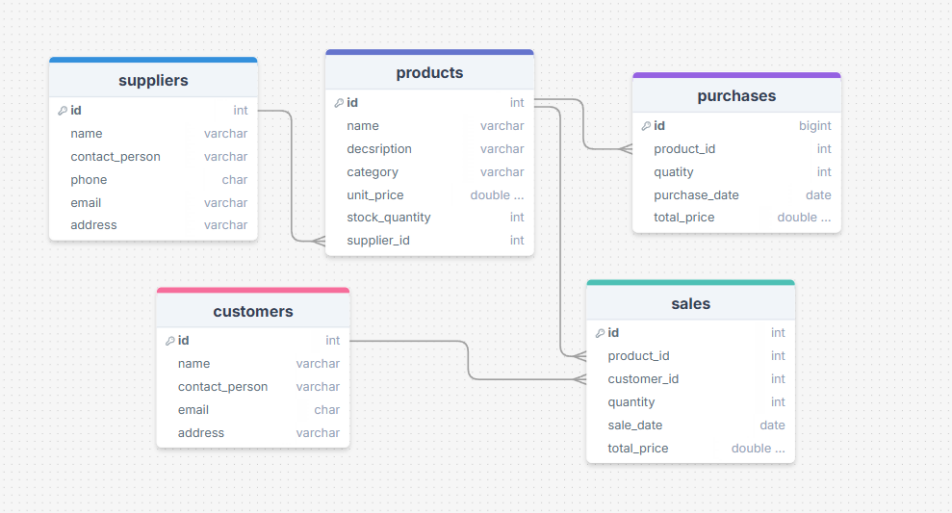
Общее задание:

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;
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;

Вариант 3

Задание. База данных: торгово-закупочная деятельность фирмы.

Ход работы



Код программы:

Database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

DATABASE_URL = "postgresql://postgres:123456789@localhost:5432/trade_db?client_encoding=utf8"
engine = create_engine(DATABASE_URL, pool_pre_ping = True)
Base = declarative_base()
SessionLocal = sessionmaker(autocommit = False, autoflush = False, bind = engine)

def create_tables():
    Base.metadata.create_all(bind = engine)
```

Models.py

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

class Supplier(Base):
    __tablename__ = "suppliers"

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

    products = relationship("Product", back_populates = "supplier", cascade = "all, delete-
orphan", passive_deletes = True)
```

```
class Product(Base):
    __tablename__ = "products"

    id = Column(Integer, primary_key = True, index = True)
    name = Column(String(100), nullable = False)
    description = Column(String(500), nullable = True)
    category = Column(String(50), nullable = True)
    unit_price = Column(Float, nullable = False)
    stock_quantity = Column(Integer, default = 0, nullable = False)
    supplier_id = Column(Integer, ForeignKey("suppliers.id", ondelete = "CASCADE"), nullable =
False)

    supplier = relationship("Supplier", back_populates = "products")
    purchases = relationship("Purchase", back_populates = "product", cascade = "all, delete-
orphan")
    sales = relationship("Sale", back_populates = "product",cascade = "all, delete-orphan")

class Purchase(Base):
    __tablename__ = "purchases"

    id = Column(Integer, primary_key = True, index = True)
    product_id = Column(Integer, ForeignKey("products.id", ondelete = "CASCADE"), nullable=False)
    quantity = Column(Integer, nullable = False)
    purchase_date = Column(Date, nullable = False)
    total_price = Column(Float, nullable = False)

    product = relationship("Product", back_populates = "purchases")

class Customer(Base):
    __tablename__ = "customers"

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

    sales = relationship("Sale", back_populates = "customer",cascade = "all, delete-orphan")

class Sale(Base):
    __tablename__ = "sales"

    id = Column(Integer, primary_key = True, index = True)
    product_id = Column(Integer, ForeignKey("products.id", ondelete = "CASCADE"), nullable =
False)
    customer_id = Column(Integer, ForeignKey("customers.id", ondelete = "CASCADE"), nullable =
False)
    quantity = Column(Integer, nullable = False)
    sale_date = Column(Date, nullable = False)
    total_price = Column(Float, nullable = False)

    product = relationship("Product", back_populates = "sales")
    customer = relationship("Customer", back_populates = "sales")
```

Main.py

```
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy.orm import Session
from pydantic import BaseModel
from typing import List, Optional
from datetime import date
from database import SessionLocal
from models import Supplier, Product, Customer, Purchase, Sale
from database import create_tables

create_tables()

app = FastAPI()
```

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

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

class SupplierResponse(SupplierCreate):
    id: int
    class Config:
        orm_mode = True

class ProductCreate(BaseModel):
    name: str
    description: Optional[str] = None
    category: Optional[str] = None
    unit_price: float
    stock_quantity: int = 0
    supplier_id: int

class ProductResponse(ProductCreate):
    id: int
    class Config:
        orm_mode = True

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

class CustomerResponse(CustomerCreate):
    id: int
    class Config:
        orm_mode = True

class PurchaseCreate(BaseModel):
    product_id: int
    quantity: int
    purchase_date: date
    total_price: float

class PurchaseResponse(PurchaseCreate):
    id: int
    class Config:
        orm_mode = True

class SaleCreate(BaseModel):
    product_id: int
    customer_id: int
    quantity: int
    sale_date: date
    total_price: float

class SaleResponse(SaleCreate):
    id: int
    class Config:
        orm_mode = True

@app.post("/suppliers/", response_model = SupplierResponse)
def create_supplier(supplier: SupplierCreate, db: Session = Depends(get_db)):
    db_supplier = Supplier(**supplier.dict())
```

```
    db.add(db_supplier)
    db.commit()
    db.refresh(db_supplier)
    return db_supplier

@app.get("/suppliers/", response_model = List[SupplierResponse])
def get_suppliers(db: Session = Depends(get_db)):
    return db.query(Supplier).all()

@app.get("/suppliers/{supplier_id}", response_model = SupplierResponse)
def get_supplier(supplier_id: int, db: Session = Depends(get_db)):
    supplier = db.query(Supplier).filter(Supplier.id == supplier_id).first()
    if not supplier:
        raise HTTPException(status_code = 404, detail = "Supplier not found")
    return supplier

@app.put("/suppliers/{supplier_id}", response_model = SupplierResponse)
def update_supplier(supplier_id: int, supplier: SupplierCreate, db: Session = Depends(get_db)):
    db_supplier = db.query(Supplier).filter(Supplier.id == supplier_id).first()
    if not db_supplier:
        raise HTTPException(status_code = 404, detail = "Supplier not found")
    for key, value in supplier.dict().items():
        setattr(db_supplier, key, value)
    db.commit()
    db.refresh(db_supplier)
    return db_supplier

@app.delete("/suppliers/{supplier_id}")
def delete_supplier(supplier_id: int, db: Session = Depends(get_db)):
    supplier = db.query(Supplier).filter(Supplier.id == supplier_id).first()
    if not supplier:
        raise HTTPException(status_code = 404, detail = "Supplier not found")
    db.delete(supplier)
    db.commit()
    return {"message": "Supplier deleted successfully"}

@app.post("/products/", response_model = ProductResponse)
def create_product(product: ProductCreate, db: Session = Depends(get_db)):
    db_product = Product(**product.dict())
    db.add(db_product)
    db.commit()
    db.refresh(db_product)
    return db_product

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

@app.get("/products/{product_id}", response_model = ProductResponse)
def get_product(product_id: int, db: Session = Depends(get_db)):
    product = db.query(Product).filter(Product.id == product_id).first()
    if not product:
        raise HTTPException(status_code = 404, detail = "Product not found")
    return product

@app.put("/products/{product_id}", response_model = ProductResponse)
def update_product(product_id: int, product: ProductCreate, db: Session = Depends(get_db)):
    db_product = db.query(Product).filter(Product.id == product_id).first()
    if not db_product:
        raise HTTPException(status_code = 404, detail = "Product not found")
    for key, value in product.dict().items():
        setattr(db_product, key, value)
    db.commit()
    db.refresh(db_product)
    return

@app.delete("/products/{product_id}")
def delete_product(product_id: int, db: Session = Depends(get_db)):
    product = db.query(Product).filter(Product.id == product_id).first()
    if not product:
```

```
        raise HTTPException(status_code = 404, detail = "Product not found")
    db.delete(product)
    db.commit()
    return {"message": "Product deleted successfully"}

@app.post("/customers/", response_model = CustomerResponse)
def create_customer(customer: CustomerCreate, db: Session = Depends(get_db)):
    db_customer = Customer(**customer.dict())
    db.add(db_customer)
    db.commit()
    db.refresh(db_customer)
    return db_customer

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

@app.get("/customers/{customer_id}", response_model = CustomerResponse)
def get_customer(customer_id: int, db: Session = Depends(get_db)):
    customer = db.query(Customer).filter(Customer.id == customer_id).first()
    if not customer:
        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: Session = Depends(get_db)):
    db_customer = db.query(Customer).filter(Customer.id == customer_id).first()
    if not db_customer:
        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)
    return db_customer

@app.delete("/customers/{customer_id}")
def delete_customer(customer_id: int, db: Session = Depends(get_db)):
    customer = db.query(Customer).filter(Customer.id == customer_id).first()
    if not customer:
        raise HTTPException(status_code = 404, detail = "Customer not found")
    db.delete(customer)
    db.commit()
    return {"message": "Customer deleted successfully"}


@app.post("/purchases/", response_model = PurchaseResponse)
def create_purchase(purchase: PurchaseCreate, db: Session = Depends(get_db)):
    product = db.query(Product).filter(Product.id == purchase.product_id).first()
    if not product:
        raise HTTPException(status_code = 404, detail = "Product not found")
    product.stock_quantity += purchase.quantity
    db_purchase = Purchase(**purchase.dict())
    db.add(db_purchase)
    db.commit()
    db.refresh(db_purchase)
    return db_purchase

@app.get("/purchases/", response_model = List[PurchaseResponse])
def get_purchases(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    return db.query(Purchase).offset(skip).limit(limit).all()

@app.get("/purchases/{purchase_id}", response_model = PurchaseResponse)
def get_purchase(purchase_id: int, db: Session = Depends(get_db)):
    purchase = db.query(Purchase).filter(Purchase.id == purchase_id).first()
    if not purchase:
        raise HTTPException(status_code = 404, detail = "Purchase not found")
    return purchase

@app.post("/sales/", response_model = SaleResponse)
def create_sale(sale: SaleCreate, db: Session = Depends(get_db)):
```

```
    product = db.query(Product).filter(Product.id == sale.product_id).first()
    if not product:
        raise HTTPException(status_code = 404, detail = "Product not found")
    if product.stock_quantity < sale.quantity:
        raise HTTPException(status_code =  400, detail = "Not enough stock")
    product.stock_quantity -= sale.quantity
    db_sale = Sale(**sale.dict())
    db.add(db_sale)
    db.commit()
    db.refresh(db_sale)
    return db_sale

@app.get("/sales/", response_model = List[SaleResponse])
def get_sales(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    return db.query(Sale).offset(skip).limit(limit).all()

@app.get("/sales/{sale_id}", response_model = SaleResponse)
def get_sale(sale_id: int, db: Session = Depends(get_db)):
    sale = db.query(Sale).filter(Sale.id == sale_id).first()
    if not sale:
        raise HTTPException(status_code = 404, detail = "Sale not found")
    return sale


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

Вывод: приобрёл практические навыки разработки API и базы данных.