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
Сильчук Д.А.
Проверил:
Козик И. Д.

Брест 2025

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

**Вариант 16**

**Общее задание**

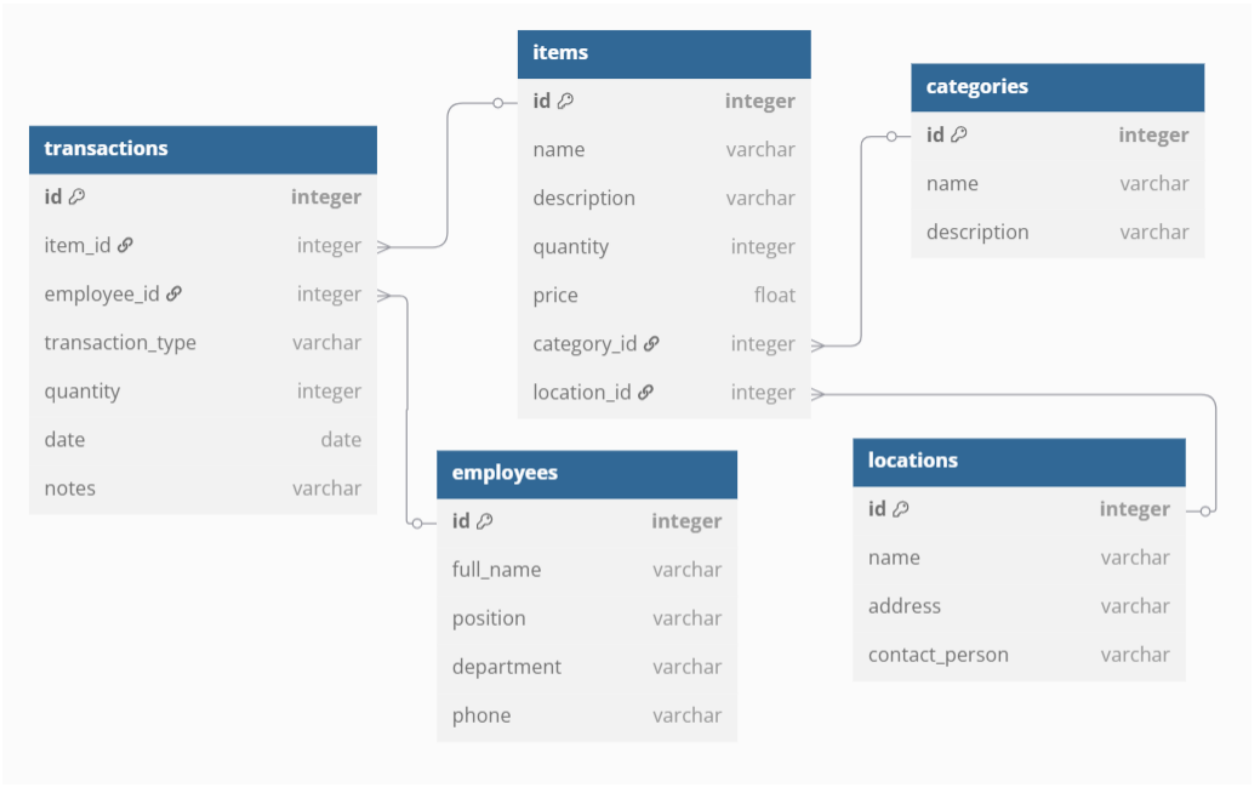
1. Реализовать базу данных из не менее 5 таблиц на заданную тематику. При реализации продумать типизацию полей и внешние ключи в таблицах;
2. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними (пример, схема на рис. 1);
3. На языке Python с использованием 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;

**Задание 1.** 16) База данных Учет материальных ценностей

**Код программы:**

```
# -*- coding: cp1251 -*-
# uvicorn spp5:app --reload
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Optional
from sqlalchemy import create_engine, Column, Integer, String, Float, ForeignKey, Date
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship
import datetime

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Material Assets Management API is running!"}

SQLALCHEMY_DATABASE_URL = "sqlite:///./inventory.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Category(Base):
    __tablename__ = "categories"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True)
    description = Column(String)

    items = relationship("Item", back_populates="category")

class Location(Base):
    __tablename__ = "locations"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    address = Column(String)
    contact_person = Column(String)

    items = relationship("Item", back_populates="location")

class Employee(Base):
    __tablename__ = "employees"
    id = Column(Integer, primary_key=True, index=True)
    full_name = Column(String)
    position = Column(String)
    department = Column(String)
    phone = Column(String)

    transactions = relationship("Transaction", back_populates="employee")

class Item(Base):
    __tablename__ = "items"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    description = Column(String)
    quantity = Column(Integer)
    price = Column(Float)
    category_id = Column(Integer, ForeignKey("categories.id"))
    location_id = Column(Integer, ForeignKey("locations.id"))

    category = relationship("Category", back_populates="items")
    location = relationship("Location", back_populates="items")
    transactions = relationship("Transaction", back_populates="item")

class Transaction(Base):
    __tablename__ = "transactions"
    id = Column(Integer, primary_key=True, index=True)
    item_id = Column(Integer, ForeignKey("items.id"))
    employee_id = Column(Integer, ForeignKey("employees.id"))
    transaction_type = Column(String)  # "in" / "out"
    quantity = Column(Integer)
    date = Column(Date, default=datetime.date.today())
    notes = Column(String)

    item = relationship("Item", back_populates="transactions")
    employee = relationship("Employee", back_populates="transactions")

Base.metadata.create_all(bind=engine)

class CategoryCreate(BaseModel):
    name: str
    description: Optional[str] = None

class CategoryResponse(CategoryCreate):
    id: int

    class Config:
        orm_mode = True

class LocationCreate(BaseModel):
    name: str
    address: str
    contact_person: str

class LocationResponse(LocationCreate):
    id: int

    class Config:
        orm_mode = True

class EmployeeCreate(BaseModel):
    full_name: str
    position: str
    department: str
    phone: str

class EmployeeResponse(EmployeeCreate):
    id: int

    class Config:
        orm_mode = True

class ItemCreate(BaseModel):
    name: str
    description: Optional[str] = None
    quantity: int
    price: float
    category_id: int
    location_id: int

class ItemResponse(ItemCreate):
    id: int

    class Config:
        from_attributes = True

class TransactionCreate(BaseModel):
    item_id: int
    employee_id: int
    transaction_type: str
    quantity: int
    notes: Optional[str] = None

class TransactionResponse(TransactionCreate):
    id: int
    date: datetime.date

    class Config:
        orm_mode = True

@app.post("/categories/", response_model=CategoryResponse)
def create_category(category: CategoryCreate):
    db = SessionLocal()
    db_category = Category(**category.dict())
    db.add(db_category)
    db.commit()
    db.refresh(db_category)
    db.close()
    return db_category

@app.get("/categories/", response_model=List[CategoryResponse])
def read_categories():
    db = SessionLocal()
    categories = db.query(Category).all()
    db.close()
    return categories

@app.get("/categories/{category_id}", response_model=CategoryResponse)
def read_category(category_id: int):
    db = SessionLocal()
    category = db.query(Category).filter(Category.id == category_id).first()
    db.close()
    if category is None:
        raise HTTPException(status_code=404, detail="Category not found")
    return category

@app.put("/categories/{category_id}", response_model=CategoryResponse)
def update_category(category_id: int, category: CategoryCreate):
    db = SessionLocal()
    db_category = db.query(Category).filter(Category.id == category_id).first()
    if db_category is None:
        db.close()
        raise HTTPException(status_code=404, detail="Category not found")
    for key, value in category.dict().items():
        setattr(db_category, key, value)
    db.commit()
    db.refresh(db_category)
    db.close()
    return db_category

@app.delete("/categories/{category_id}")
def delete_category(category_id: int):
    db = SessionLocal()
    db_category = db.query(Category).filter(Category.id == category_id).first()
    if db_category is None:
        db.close()
        raise HTTPException(status_code=404, detail="Category not found")
    db.delete(db_category)
    db.commit()
    db.close()
    return {"message": "Category deleted"}

@app.post("/locations/", response_model=LocationResponse)
def create_location(location: LocationCreate):
    db = SessionLocal()
    db_location = Location(**location.dict())
    db.add(db_location)
    db.commit()
    db.refresh(db_location)
    db.close()
    return db_location

@app.get("/locations/", response_model=List[LocationResponse])
def read_locations():
    db = SessionLocal()
    locations = db.query(Location).all()
    db.close()
    return locations

@app.get("/locations/{location_id}", response_model=LocationResponse)
def read_location(location_id: int):
    db = SessionLocal()
    location = db.query(Location).filter(Location.id == location_id).first()
    db.close()
    if location is None:
        raise HTTPException(status_code=404, detail="Location not found")
    return location

@app.put("/locations/{location_id}", response_model=LocationResponse)
def update_location(location_id: int, location: LocationCreate):
    db = SessionLocal()
    db_location = db.query(Location).filter(Location.id == location_id).first()
    if db_location is None:
        db.close()
        raise HTTPException(status_code=404, detail="Location not found")
    for key, value in location.dict().items():
        setattr(db_location, key, value)
    db.commit()
    db.refresh(db_location)
    db.close()
    return db_location

@app.delete("/locations/{location_id}")
def delete_location(location_id: int):
    db = SessionLocal()
    db_location = db.query(Location).filter(Location.id == location_id).first()
    if db_location is None:
        db.close()
        raise HTTPException(status_code=404, detail="Location not found")
    db.delete(db_location)
    db.commit()
    db.close()
    return {"message": "Location deleted"}

@app.post("/employees/", response_model=EmployeeResponse)
def create_employee(employee: EmployeeCreate):
    db = SessionLocal()
    db_employee = Employee(**employee.dict())
    db.add(db_employee)
    db.commit()
    db.refresh(db_employee)
    db.close()
    return db_employee

@app.get("/employees/", response_model=List[EmployeeResponse])
def read_employees():
    db = SessionLocal()
    employees = db.query(Employee).all()
    db.close()
    return employees

@app.get("/employees/{employee_id}", response_model=EmployeeResponse)
def read_employee(employee_id: int):
    db = SessionLocal()
    employee = db.query(Employee).filter(Employee.id == employee_id).first()
    db.close()
    if employee is None:
        raise HTTPException(status_code=404, detail="Employee not found")
    return employee

@app.put("/employees/{employee_id}", response_model=EmployeeResponse)
def update_employee(employee_id: int, employee: EmployeeCreate):
    db = SessionLocal()
    db_employee = db.query(Employee).filter(Employee.id == employee_id).first()
    if db_employee is None:
        db.close()
        raise HTTPException(status_code=404, detail="Employee not found")
    for key, value in employee.dict().items():
        setattr(db_employee, key, value)
    db.commit()
    db.refresh(db_employee)
    db.close()
    return db_employee

@app.delete("/employees/{employee_id}")
def delete_employee(employee_id: int):
    db = SessionLocal()
    db_employee = db.query(Employee).filter(Employee.id == employee_id).first()
    if db_employee is None:
        db.close()
        raise HTTPException(status_code=404, detail="Employee not found")
    db.delete(db_employee)
    db.commit()
    db.close()
    return {"message": "Employee deleted"}

@app.post("/items/", response_model=ItemResponse)
def create_item(item: ItemCreate):
    db = SessionLocal()
    db_item = Item(**item.dict())
    db.add(db_item)
    db.commit()
    db.refresh(db_item)
    db.close()
    return db_item

@app.get("/items/", response_model=List[ItemResponse])
def read_items():
    db = SessionLocal()
    items = db.query(Item).all()
    db.close()
    return items

@app.get("/items/{item_id}", response_model=ItemResponse)
def read_item(item_id: int):
    db = SessionLocal()
    item = db.query(Item).filter(Item.id == item_id).first()
    db.close()
    if item is None:
        raise HTTPException(status_code=404, detail="Item not found")
    return item

@app.put("/items/{item_id}", response_model=ItemResponse)
def update_item(item_id: int, item: ItemCreate):
    db = SessionLocal()
    db_item = db.query(Item).filter(Item.id == item_id).first()
    if db_item is None:
        db.close()
        raise HTTPException(status_code=404, detail="Item not found")
    for key, value in item.dict().items():
        setattr(db_item, key, value)
    db.commit()
    db.refresh(db_item)
    db.close()
    return db_item

@app.delete("/items/{item_id}")
def delete_item(item_id: int):
    db = SessionLocal()
    db_item = db.query(Item).filter(Item.id == item_id).first()
    if db_item is None:
        db.close()
        raise HTTPException(status_code=404, detail="Item not found")
    db.delete(db_item)
    db.commit()
    db.close()
    return {"message": "Item deleted"}

@app.post("/transactions/", response_model=TransactionResponse)
def create_transaction(transaction: TransactionCreate):
    db = SessionLocal()

    item = db.query(Item).filter(Item.id == transaction.item_id).first()
    if item is None:
        db.close()
        raise HTTPException(status_code=404, detail="Item not found")

    employee = db.query(Employee).filter(Employee.id == transaction.employee_id).first()
    if employee is None:
        db.close()
        raise HTTPException(status_code=404, detail="Employee not found")

    if transaction.transaction_type == "in":
        item.quantity += transaction.quantity
    elif transaction.transaction_type == "out":
        if item.quantity < transaction.quantity:
            db.close()
            raise HTTPException(status_code=400, detail="Not enough items in stock")
        item.quantity -= transaction.quantity
    else:
        db.close()
        raise HTTPException(status_code=400, detail="Invalid transaction type. Use 'in' or 'out'")

    db_transaction = Transaction(**transaction.dict())
    db.add(db_transaction)
    db.commit()
    db.refresh(db_transaction)
    db.close()
    return db_transaction

@app.get("/transactions/", response_model=List[TransactionResponse])
def read_transactions():
    db = SessionLocal()
    transactions = db.query(Transaction).all()
    db.close()
    return transactions
```

```
@app.get("/transactions/{transaction_id}", response_model=TransactionResponse)
def read_transaction(transaction_id: int):
    db = SessionLocal()
    transaction = db.query(Transaction).filter(Transaction.id == transaction_id).first()
    db.close()
    if transaction is None:
        raise HTTPException(status_code=404, detail="Transaction not found")
    return transaction

@app.delete("/transactions/{transaction_id}")
def delete_transaction(transaction_id: int):
    db = SessionLocal()
    transaction = db.query(Transaction).filter(Transaction.id == transaction_id).first()
    if transaction is None:
        db.close()
        raise HTTPException(status_code=404, detail="Transaction not found")

    item = db.query(Item).filter(Item.id == transaction.item_id).first()
    if transaction.transaction_type == "in":
        item.quantity -= transaction.quantity
    else:
        item.quantity += transaction.quantity

    db.delete(transaction)
    db.commit()
    db.close()
    return {"message": "Transaction deleted and stock updated"}
```
Схема:



items
id integer
name varchar
description varchar
quantity integer
price float
category_id integer
location_id integer
categories
id integer
name varchar
description varchar
transactions
id integer
item_id integer
employee_id integer
transaction_type varchar
quantity integer
date date
notes varchar
employees
id integer
full_name varchar
position varchar
department varchar
phone varchar
locations
id integer
name varchar
address varchar
contact_person varchar