**Министерство транспорта Российской Федерации**

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«РУТ (МИИТ)»**

**Институт транспортной техники и систем управления**

**Кафедра «Управление и защита информации»**

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему  
«Проектирование базы данных для интернет-магазина»**

Выполнил: ст. гр. ТКИ-541,  
Гаврилин-Кирьянов И. А.  
Проверил: доц., к.т.н.,

Васильева М. А.

Москва 2023

Оглавление

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 5](#_Toc153234263)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 5](#_Toc153234264)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 5](#_Toc153234265)

[UML - ДИАГРАММА КЛАССОВ 6](#_Toc153234266)

[СТРУКТУРА ПРОЕКТА 7](#_Toc153234267)

[КОД ПРОЕКТА 8](#_Toc153234268)

[ORM/confest.py: 8](#_Toc153234269)

[ORM/database.py: 8](#_Toc153234270)

[ORM/main.py: 8](#_Toc153234271)

[ORM/methods.py: 8](#_Toc153234272)

[ORM/models.py: 8](#_Toc153234273)

[ПРОХОЖДЕНИЕ ТЕСТОВ 8](#_Toc153234274)

[ORM/test\_category.py: 8](#_Toc153234275)

[ORM/test\_database.py: 10](#_Toc153234276)

[ORM/test\_manufacturer.py: 11](#_Toc153234277)

[ORM/test\_models.py: 12](#_Toc153234278)

[ORM/test\_order.py: 14](#_Toc153234279)

[ORM/test\_product.py: 16](#_Toc153234280)

[ORM/test\_warehouse.py: 18](#_Toc153234281)

[ERD - ДИАГРАММА 20](#_Toc153234282)

[Вывод 21](#_Toc153234283)

**ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

Разработать систему учета интернет-магазина. Система должна позволять добавлять новые продукты, производителей, категории, заказы и склады; выводить информацию.

**ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ**

Познакомиться с созданием ORM на языке программирования высокого уровня. Изучить основы построения защищенных баз данных.

**ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ**

1. Язык программирования – Python 3.12.1
2. PostgreSQL 16.1 – система управления базами данных с открытым исходным кодом
3. Visual Studio Code 1.85.0 – среда разработки, которая предоставляет разработчикам все необходимые инструменты для эффективной разработки, отладки и тестирования приложений
4. SQLAlchemy 2.0.23 – это [программная библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) на языке [Python](https://ru.wikipedia.org/wiki/Python) для работы с [реляционными СУБД](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) с применением технологии [ORM](https://ru.wikipedia.org/wiki/ORM)

**UML - ДИАГРАММА КЛАССОВ**

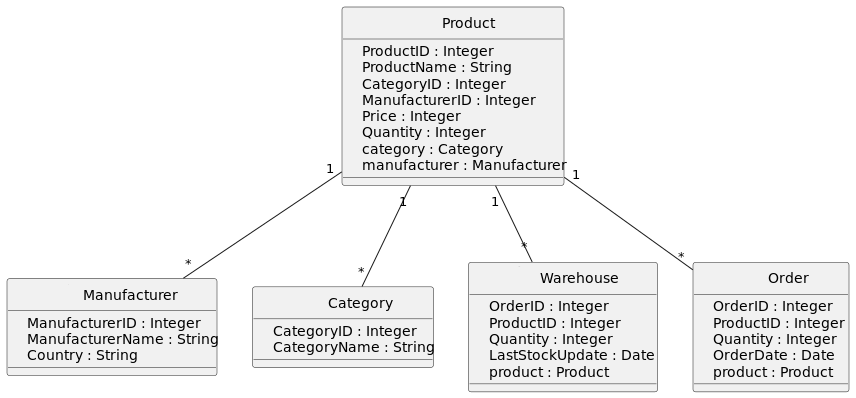


Рисунок 1 – UML-диаграмма

**СТРУКТУРА ПРОЕКТА**

├── ORM

│ ├── conftest.py

│ ├── database.py

│ ├── main.py

│ ├── methods.py

│ ├── models.py

│ ├── main.py

│ ├── test\_category.py

│ ├── test\_database.py

│ ├── test\_manufacturer.py

│ ├── test\_models.py

│ ├── test\_order.py

│ ├── test\_product.py

│ ├── test\_warehouse.py

**КОД ПРОЕКТА**

**ORM/confest.py:**

import pytest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Base, Category, Manufacturer, Order, Product, Warehouse

TEST\_DATABASE\_URL = 'postgresql://postgres:1@localhost:5432/postgres'

@pytest.fixture(scope='session')

def engine():

return create\_engine(TEST\_DATABASE\_URL)

@pytest.fixture(scope='session')

def tables(engine):

Base.metadata.create\_all(engine)

yield

Base.metadata.drop\_all(engine)

@pytest.fixture(scope='function')

def session(engine, tables):

connection = engine.connect()

transaction = connection.begin()

session = sessionmaker(bind=connection)()

yield session

session.close()

transaction.rollback()

connection.close()

@pytest.fixture(scope='function')

def test\_data(session):

new\_manufacturer = Manufacturer(ManufacturerName="Sony", Country="Japan")

new\_category = Category(CategoryName="Electronics")

session.add(new\_manufacturer)

session.add(new\_category)

session.commit()

yield

session.delete(new\_manufacturer)

session.delete(new\_category)

session.commit()

@pytest.fixture(scope='function')

def add\_test\_data(session):

new\_manufacturer = Manufacturer(ManufacturerName="Sony", Country="Japan")

new\_category = Category(CategoryName="Electronics")

new\_product = Product(ProductName="TestProduct", CategoryID=new\_category.CategoryID, ManufacturerID=new\_manufacturer.ManufacturerID, Price=100, Quantity=10)

session.add(new\_manufacturer)

session.add(new\_category)

session.flush()

session.add(new\_product)

session.commit()

yield new\_manufacturer, new\_category, new\_product

session.query(Order).filter(Order.ProductID == new\_product.ProductID).delete(synchronize\_session=False)

session.query(Warehouse).filter(Warehouse.ProductID == new\_product.ProductID).delete(synchronize\_session=False)

session.flush()

session.query(Product).filter(Product.CategoryID == new\_category.CategoryID).delete(synchronize\_session=False)

session.flush()

session.delete(new\_product)

session.flush()

session.delete(new\_category)

session.delete(new\_manufacturer)

session.commit()

**ORM/database.py:**

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Base

db\_url = 'postgresql://postgres:1@localhost:5432/postgres'

engine = create\_engine(db\_url)

SessionLocal = sessionmaker(bind=engine)

session = SessionLocal()

Base.metadata.create\_all(engine)

**ORM/main.py:**

from database import session as db\_session

from methods import BaseRepository

from models import Manufacturer, Category, Product, Warehouse, Order

manufacturer\_repository = BaseRepository(db\_session, Manufacturer)

category\_repository = BaseRepository(db\_session, Category)

product\_repository = BaseRepository(db\_session, Product)

warehouse\_repository = BaseRepository(db\_session, Warehouse)

order\_repository = BaseRepository(db\_session, Order)

**ORM/methods.py:**

from sqlalchemy.orm import Session

from models import Manufacturer, Category, Product, Warehouse, Order

class BaseRepository:

def \_\_init\_\_(self, session, model):

self.session = session

self.model = model

def create(self, \*\*kwargs):

entity = self.model(\*\*kwargs)

self.session.add(entity)

self.session.commit()

return entity

def get(self, record\_id: int):

primary\_key = self.model.\_\_mapper\_\_.primary\_key[0]

return self.session.query(self.model).filter(primary\_key == record\_id).first()

def update(self, record\_id: int, \*\*kwargs):

record = self.get(record\_id)

for key, value in kwargs.items():

setattr(record, key, value)

self.session.commit()

def delete(self, record\_id: int):

record = self.get(record\_id)

self.session.delete(record)

self.session.commit()

**ORM/models.py:**

import sqlalchemy.orm

from sqlalchemy import Column, Integer, String, Date, ForeignKey

from sqlalchemy.orm import relationship

from sqlalchemy.ext.declarative import declarative\_base

Base = sqlalchemy.orm.declarative\_base()

class Manufacturer(Base):

\_\_tablename\_\_ = 'Manufacturers'

ManufacturerID = Column(Integer, primary\_key=True)

ManufacturerName = Column(String(50), nullable=False)

Country = Column(String(50), nullable=False)

class Category(Base):

\_\_tablename\_\_ = 'Categories'

CategoryID = Column(Integer, primary\_key=True)

CategoryName = Column(String(50), nullable=False)

class Product(Base):

\_\_tablename\_\_ = 'Products'

\_\_mapper\_args\_\_ = {

'confirm\_deleted\_rows': False

}

ProductID = Column(Integer, primary\_key=True)

ProductName = Column(String(100), nullable=False)

CategoryID = Column(Integer, ForeignKey('Categories.CategoryID'))

ManufacturerID = Column(Integer, ForeignKey('Manufacturers.ManufacturerID'))

Price = Column(Integer, nullable=False)

Quantity = Column(Integer, nullable=False)

category = relationship("Category")

manufacturer = relationship("Manufacturer")

class Warehouse(Base):

\_\_tablename\_\_ = 'Warehouse'

\_\_mapper\_args\_\_ = {

'confirm\_deleted\_rows': False

}

OrderID = Column(Integer, primary\_key=True)

ProductID = Column(Integer, ForeignKey('Products.ProductID'))

Quantity = Column(Integer, nullable=False)

LastStockUpdate = Column(Date, nullable=False)

product = relationship("Product")

class Order(Base):

\_\_tablename\_\_ = 'Orders'

\_\_mapper\_args\_\_ = {

'confirm\_deleted\_rows': False

}

OrderID = Column(Integer, primary\_key=True)

ProductID = Column(Integer, ForeignKey('Products.ProductID'))

Quantity = Column(Integer, nullable=False)

OrderDate = Column(Date, nullable=False)

product = relationship("Product")

**ПРОХОЖДЕНИЕ ТЕСТОВ**

**ORM/test\_category.py:**

import pytest

from models import Category

from database import session

@pytest.fixture(scope="module")

def new\_category():

category = Category(CategoryName="Electronics")

return category

def test\_new\_category(new\_category):

assert new\_category.CategoryName == "Electronics", "Incorrect category name"

def test\_add\_category(session, new\_category):

session.add(new\_category)

session.commit()

assert new\_category in session, "Category not added to session"

def test\_get\_category(session, add\_test\_data):

new\_manufacturer, new\_category, \_ = add\_test\_data

category = session.query(Category).filter\_by(CategoryName="Electronics").first()

assert category == new\_category, "Category retrieval failed"

A black screen with white text

Description automatically generated

Рисунок 2 – Прохождение тестов

**ORM/test\_database.py:**

import pytest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from database import SessionLocal, engine

from models import Base

@pytest.fixture(scope="module")

def database\_engine():

return create\_engine('postgresql://postgres:1@localhost:5432/postgres')

@pytest.fixture(scope="module")

def session(database\_engine):

Base.metadata.create\_all(database\_engine)

SessionLocal = sessionmaker(bind=database\_engine)

return SessionLocal()

def test\_database\_connection(database\_engine):

assert database\_engine.connect() != None, "Database connection failed"

def test\_session\_creation(session):

assert session != None, "Session creation failed"

def test\_create\_tables(session):

assert len(Base.metadata.tables) > 0, "No tables created"

A black screen with white text

Description automatically generated

Рисунок 3 – Прохождение тестов

**ORM/test\_manufacturer.py:**

import pytest

from models import Manufacturer

from database import session

@pytest.fixture(scope="module")

def new\_manufacturer():

manufacturer = Manufacturer(ManufacturerName="Sony", Country="Japan")

return manufacturer

def test\_new\_manufacturer(new\_manufacturer):

assert new\_manufacturer.ManufacturerName == "Sony", "Incorrect manufacturer name"

assert new\_manufacturer.Country == "Japan", "Incorrect country"

def test\_add\_manufacturer(session, new\_manufacturer):

session.add(new\_manufacturer)

session.commit()

assert new\_manufacturer in session, "Manufacturer not added to session"

def test\_get\_manufacturer(session, add\_test\_data):

new\_manufacturer, new\_category, \_ = add\_test\_data

manufacturer = session.query(Manufacturer).filter\_by(ManufacturerName="Sony").first()

assert manufacturer == new\_manufacturer, "Manufacturer retrieval failed"

A black screen with white text

Description automatically generated

Рисунок 4 – Прохождение тестов

**ORM/test\_models.py:**

import pytest

from models import Manufacturer, Category, Product, Warehouse, Order

def test\_create\_manufacturer(session):

manufacturer = Manufacturer(ManufacturerName="TestManufacturer", Country="TestCountry")

session.add(manufacturer)

session.commit()

assert manufacturer in session

def test\_create\_category(session):

category = Category(CategoryName="TestCategory")

session.add(category)

session.commit()

assert category in session

def test\_create\_product(session, add\_test\_data):

new\_manufacturer, new\_category, \_ = add\_test\_data

product = Product(ProductName="TestProduct", CategoryID=new\_category.CategoryID, ManufacturerID=new\_manufacturer.ManufacturerID, Price=100, Quantity=10)

session.add(product)

session.commit()

assert product in session and product.category == new\_category and product.manufacturer == new\_manufacturer

def test\_create\_warehouse(session, add\_test\_data):

new\_manufacturer, new\_category, new\_product = add\_test\_data

warehouse = Warehouse(ProductID=new\_product.ProductID, Quantity=20, LastStockUpdate="2023-01-01")

session.add(warehouse)

session.commit()

assert warehouse in session

def test\_create\_order(session, add\_test\_data):

new\_manufacturer, new\_category, new\_product = add\_test\_data

order = Order(ProductID=new\_product.ProductID, Quantity=5, OrderDate="2023-01-01")

session.add(order)

session.commit()

assert order in session

A black screen with white text

Description automatically generated

Рисунок 5 – Прохождение тестов

**ORM/test\_order.py:**

import pytest

from models import Order, Product

from datetime import datetime

@pytest.fixture(scope='function')

def add\_order\_data(session, add\_test\_data):

\_, \_, new\_product = add\_test\_data

new\_order = Order(ProductID=new\_product.ProductID, Quantity=5, OrderDate=datetime.now())

session.add(new\_order)

session.commit()

yield new\_order

session.delete(new\_order)

session.commit()

def test\_create\_order(session, add\_order\_data):

new\_order = add\_order\_data

assert new\_order.OrderID is not None

def test\_read\_order(session, add\_order\_data):

new\_order = add\_order\_data

found\_order = session.query(Order).filter\_by(OrderID=new\_order.OrderID).first()

assert found\_order is not None

assert found\_order.ProductID == new\_order.ProductID

def test\_update\_order(session, add\_order\_data):

new\_order = add\_order\_data

new\_order.Quantity = 10

session.commit()

updated\_order = session.query(Order).filter\_by(OrderID=new\_order.OrderID).first()

assert updated\_order.Quantity == 10

def test\_delete\_order(session, add\_order\_data):

new\_order = add\_order\_data

session.delete(new\_order)

session.commit()

deleted\_order = session.query(Order).filter\_by(OrderID=new\_order.OrderID).first()

assert deleted\_order is None

A black screen with white text

Description automatically generated

Рисунок 6 – Прохождение тестов

**ORM/test\_product.py:**

import pytest

from models import Product

def test\_create\_product(session, add\_test\_data):

new\_manufacturer, new\_category, \_ = add\_test\_data

new\_product = Product(ProductName="NewProduct", CategoryID=new\_category.CategoryID, ManufacturerID=new\_manufacturer.ManufacturerID, Price=200, Quantity=15)

session.add(new\_product)

session.commit()

assert new\_product.ProductID is not None

def test\_read\_product(session, add\_test\_data):

\_, \_, new\_product = add\_test\_data

product = session.query(Product).filter\_by(ProductID=new\_product.ProductID).first()

assert product is not None

assert product.ProductName == new\_product.ProductName

def test\_update\_product(session, add\_test\_data):

\_, \_, new\_product = add\_test\_data

product = session.query(Product).filter\_by(ProductID=new\_product.ProductID).first()

product.Quantity = 20

session.commit()

updated\_product = session.query(Product).filter\_by(ProductID=new\_product.ProductID).first()

assert updated\_product.Quantity == 20

def test\_delete\_product(session, add\_test\_data):

\_, \_, new\_product = add\_test\_data

session.delete(new\_product)

session.commit()

deleted\_product = session.query(Product).filter\_by(ProductID=new\_product.ProductID).first()

assert deleted\_product is None

A black screen with white text

Description automatically generated

Рисунок 7 – Прохождение тестов

**ORM/test\_warehouse.py:**

import pytest

from sqlalchemy.orm import scoped\_session, sessionmaker

from models import Manufacturer, Category, Product

from methods import BaseRepository

from database import engine, Base

@pytest.fixture(scope='module')

def test\_session():

connection = engine.connect()

transaction = connection.begin()

Session = scoped\_session(sessionmaker(bind=connection))

session = Session()

Base.metadata.create\_all(bind=connection)

yield session

session.close()

Base.metadata.drop\_all(bind=connection)

transaction.rollback()

connection.close()

@pytest.fixture(scope='module')

def manufacturer\_repository(test\_session):

return BaseRepository(test\_session, Manufacturer)

@pytest.fixture(scope='module')

def category\_repository(test\_session):

return BaseRepository(test\_session, Category)

@pytest.fixture(scope='module')

def product\_repository(test\_session):

return BaseRepository(test\_session, Product)

def test\_create\_manufacturer(manufacturer\_repository):

manufacturer = manufacturer\_repository.create(ManufacturerName='Test Manufacturer', Country='Test Country')

assert manufacturer.ManufacturerName == 'Test Manufacturer'

def test\_read\_manufacturer(manufacturer\_repository):

manufacturer = manufacturer\_repository.create(ManufacturerName='Read Manufacturer', Country='Read Country')

found = manufacturer\_repository.get(manufacturer.ManufacturerID)

assert found == manufacturer

def test\_update\_manufacturer(manufacturer\_repository):

manufacturer = manufacturer\_repository.create(ManufacturerName='Update Manufacturer', Country='Update Country')

manufacturer.ManufacturerName = 'Updated Manufacturer'

manufacturer\_repository.session.commit()

updated = manufacturer\_repository.get(manufacturer.ManufacturerID)

assert updated.ManufacturerName == 'Updated Manufacturer'

def test\_delete\_manufacturer(manufacturer\_repository):

manufacturer = manufacturer\_repository.create(ManufacturerName='Delete Manufacturer', Country='Delete Country')

manufacturer\_id = manufacturer.ManufacturerID

manufacturer\_repository.session.delete(manufacturer)

manufacturer\_repository.session.commit()

assert manufacturer\_repository.get(manufacturer\_id) is None

A black screen with white text

Description automatically generated

Рисунок 8 – Прохождение тестов

**ERD - ДИАГРАММА**

Диаграмма ERD (Entity-Relationship Diagram) представляет собой графическое изображение структуры данных, используемое для высокоуровневого проектирования базы данных. Она иллюстрирует отношения между различными сущностями или объектами в базе данных. В нашем случае, для создания диаграммы ERD, мы начали с анализа основных классов и сущностей в проекте, таких как «category», «manufacturer», «order», «product» и «warehouse». Основываясь на атрибутах и свойствах каждой сущности, а также на их взаимодействии, мы определили отношения между ними. ERD-диаграмма представлена на рисунке 9.

A screenshot of a computer screen

Description automatically generated

Рисунок 9 – ERD-диаграмма

**Вывод**

В ходе данной работы я освоил создание ORM для интернет-магазина, используя Python и PostgreSQL. Результатом стал эффективно работающий прототип интернет-магазина, способный обрабатывать данные пользователей и товаров с высокой степенью безопасности и производительности.

**СПИСОК ЛИТЕРАТУРЫ**

* + - 1. Васильева М.А., Хобта Д.О., Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий: Учебно-методическое пособие. Издание второе, исправленное и дополненное–М.:РУТ(МИИТ). 2023.–105с.
      2. Васильева М.А., Меркулов Д.А. Группировка и обобщение данных. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 46–с.
      3. Васильева М.А., Ракинцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 63–с.
      4. Балакина Е.П., Васильева М.А., Филипченко К.М. Информационное обеспечение систем управления. Методические указания к курсовому проектированию. Учебно-методическое пособие. Издание второе, исправленное и дополненное, 2023.102–с.
      5. SQLAlchemy. — Текст : электронный // SQLAlchemy : [сайт]. — URL: https://www.sqlalchemy.org/ (дата обращения: 24.10.2023).
      6. PostgreSQL. — Текст : электронный // PostgreSQL : [сайт]. — URL: https://www.postgresql.org/ (дата обращения: 24.10.2023).