**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 магазина кроссовок»**

Выполнила: ст. гр. ТКИ-541,  
Кузьмина С. П.  
Проверил: доц., к.т.н.,

Васильева М. А.

Москва 2023

Оглавление

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 6](#_Toc154399826)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 6](#_Toc154399827)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 6](#_Toc154399828)

[UML - ДИАГРАММА КЛАССОВ 7](#_Toc154399829)

[СТРУКТУРА ПРОЕКТА 8](#_Toc154399830)

[КОД ПРОЕКТА 8](#_Toc154399831)

[ORM/main.py: 8](#_Toc154399832)

[ORM/database.py: 9](#_Toc154399833)

[ORM/methods.py: 10](#_Toc154399834)

[ORM/models.py: 10](#_Toc154399835)

[ORM/tables/base.py: 13](#_Toc154399836)

[ORM/tables/customers.py: 13](#_Toc154399837)

[ORM/tables/orders.py: 13](#_Toc154399838)

[ORM/tables/reviews.py: 14](#_Toc154399839)

[ORM/tables/sneakers.py: 15](#_Toc154399840)

[ПРОХОЖДЕНИЕ ТЕСТОВ 16](#_Toc154399841)

[ORM/ test\_brands.py: 16](#_Toc154399842)

[ORM/test\_categories.py: 17](#_Toc154399843)

[ORM/test\_database.py: 19](#_Toc154399844)

[ORM/test\_models.py: 20](#_Toc154399845)

[ORM/test\_sneakers.py: 21](#_Toc154399846)

[ERD - ДИАГРАММА 24](#_Toc154399847)

[СПИСОК ЛИТЕРАТУРЫ 25](#_Toc154399848)

# ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

Разработать систему учета магазина кроссовок.

# ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ

Познакомиться с механизмом, который позволяет не допустить SQL – инъекции при работе с базами данных

# ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ

1. Язык программирования – Python
2. PostgreSQL - система управления базами данных с открытым исходным кодом.
3. Visual Studio Code - среда разработки, которая предоставляет разработчикам все необходимые инструменты для эффективной разработки, отладки и тестирования приложений.
4. DBeaver - инструмент (графическое окружение) для работы с базами данных, который поддерживает большинство популярных СУБД, включая PostgreSQL.
5. SQLAlchemy — 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

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 1 — UML — диаграмма классов

# СТРУКТУРА ПРОЕКТА

├───orm1 │ │ ├───main.py │ ├───database.py │ ├───methods.py │ ├───models.py │ ├───test\_brands.py

│ ├───test\_categories.py

│ ├───test\_database.py

│ ├───test\_models.py

│ ├───test\_sneakers.py │ ├───tables │ │ ├───base.py │ │ ├───customers.py │ │ ├───orders.py

│ │ ├───reviews.py

│ │ ├───sneakers.py

# КОД ПРОЕКТА

## ORM/main.py:

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Base, Brand, Category, Sneaker

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

engine = create\_engine(db\_url)

Base.metadata.create\_all(engine)

SessionLocal = sessionmaker(bind=engine)

db\_session = SessionLocal()

# Пример создания нового бренда

new\_brand = Brand(brand\_name='Nike')

db\_session.add(new\_brand)

db\_session.commit()

# Пример создания новой категории

new\_category = Category(category\_name='Running')

db\_session.add(new\_category)

db\_session.commit()

# Пример создания новой кроссовки

new\_sneaker = Sneaker(model='Air Max', brand\_id=new\_brand.brand\_id, category\_id=new\_category.category\_id, price='150.00')

db\_session.add(new\_sneaker)

db\_session.commit()

# Вывод информации

print("Созданный бренд:", Brand.get(db\_session, new\_brand.brand\_id))

print("Созданная категория:", Category.get(db\_session, new\_category.category\_id))

print("Созданные кроссовки:", Sneaker.get(db\_session, new\_sneaker.sneaker\_id))

db\_session.close()

## ORM/database.py:

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Base

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

engine = create\_engine(db\_url)

SessionLocal = sessionmaker(bind=engine)

session = SessionLocal()

# Создаем таблицы

Base.metadata.create\_all(engine)

## ORM/methods.py:

from sqlalchemy.orm import Session

class BaseMethod:

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

self.session = session

self.model = model

def get(self, id: int):

return self.session.query(self.model).filter(self.model.id == id).first()

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

return self.session.query(self.model).filter\_by(\*\*kwargs).all()

def get\_all(self):

return self.session.query(self.model).all()

def save(self, entity):

self.session.add(entity)

self.session.commit()

def delete(self, entity):

self.session.delete(entity)

self.session.commit()

## ORM/models.py:

from sqlalchemy import Column, Integer, String, ForeignKey, create\_engine

from sqlalchemy.orm import relationship, declarative\_base, sessionmaker

Base = declarative\_base()

class Brand(Base):

\_\_tablename\_\_ = 'brands'

brand\_id = Column(Integer, primary\_key=True, index=True)

brand\_name = Column(String, index=True)

sneakers = relationship("Sneaker", back\_populates="brand")

@classmethod

def get(cls, session, brand\_id):

return session.query(cls).filter\_by(brand\_id=brand\_id).first()

@classmethod

def find(cls, session, brand\_name):

return session.query(cls).filter\_by(brand\_name=brand\_name).first()

@classmethod

def delete(cls, session, brand\_id):

brand = cls.get(session, brand\_id)

if brand:

session.delete(brand)

session.commit()

@classmethod

def get\_all(cls, session):

return session.query(cls).all()

class Category(Base):

\_\_tablename\_\_ = 'categories'

category\_id = Column(Integer, primary\_key=True, index=True)

category\_name = Column(String, index=True)

sneakers = relationship("Sneaker", back\_populates="category")

@classmethod

def get(cls, session, category\_id):

return session.query(cls).filter\_by(category\_id=category\_id).first()

@classmethod

def find(cls, session, category\_name):

return session.query(cls).filter\_by(category\_name=category\_name).first()

@classmethod

def delete(cls, session, category\_id):

category = cls.get(session, category\_id)

if category:

session.delete(category)

session.commit()

@classmethod

def get\_all(cls, session):

return session.query(cls).all()

class Sneaker(Base):

\_\_tablename\_\_ = 'sneakers'

sneaker\_id = Column(Integer, primary\_key=True, index=True)

model = Column(String)

brand\_id = Column(Integer, ForeignKey('brands.brand\_id'))

category\_id = Column(Integer, ForeignKey('categories.category\_id'))

price = Column(String)

brand = relationship("Brand", back\_populates="sneakers")

category = relationship("Category", back\_populates="sneakers")

@classmethod

def get(cls, session, sneaker\_id):

return session.query(cls).filter\_by(sneaker\_id=sneaker\_id).first()

@classmethod

def find(cls, session, model):

return session.query(cls).filter\_by(model=model).first()

@classmethod

def delete(cls, session, sneaker\_id):

sneaker = cls.get(session, sneaker\_id)

if sneaker:

session.delete(sneaker)

session.commit()

@classmethod

def get\_all(cls, session):

return session.query(cls).all()

## ORM/tables/base.py:

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

## ORM/tables/customers.py:

from sqlalchemy.orm import relationship

from sqlalchemy import Column, Integer, String

from tables.base import Base

class Customer(Base):

\_\_tablename\_\_ = 'customers'

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, nullable=False)

email = Column(String, nullable=False)

phone\_number = Column(String, nullable=False)

address = Column(String, nullable=False)

orders = relationship("Order", back\_populates="customer")

def \_\_init\_\_(self, name: str, email: str, phone\_number: str, address: str):

self.name = name

self.email = email

self.phone\_number = phone\_number

self.address = address

## ORM/tables/orders.py:

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

from sqlalchemy.orm import relationship

from tables.base import Base

class Order(Base):

\_\_tablename\_\_ = 'orders'

id = Column(Integer, primary\_key=True, index=True)

customer\_id = Column(Integer, ForeignKey('customers.id'), nullable=False)

sneaker\_id = Column(Integer, ForeignKey('sneakers.id'), nullable=False)

order\_date = Column(Date, nullable=False)

quantity = Column(Integer, nullable=False)

total\_price = Column(Float, nullable=False)

# добавляем свойства customer и sneaker для связи с моделями Customer и Sneaker

customer = relationship("Customer", back\_populates="orders")

sneaker = relationship("Sneaker", back\_populates="orders")

def \_\_init\_\_(self, customer\_id: int, sneaker\_id: int, order\_date, quantity: int, total\_price: float):

self.customer\_id = customer\_id

self.sneaker\_id = sneaker\_id

self.order\_date = order\_date

self.quantity = quantity

self.total\_price = total\_price

## ORM/tables/reviews.py:

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

from sqlalchemy.orm import relationship

from tables.base import Base

class Review(Base):

\_\_tablename\_\_ = 'reviews'

id = Column(Integer, primary\_key=True, index=True)

sneaker\_id = Column(Integer, ForeignKey('sneakers.id'), nullable=False)

customer\_id = Column(Integer, ForeignKey('customers.id'), nullable=False)

rating = Column(Integer, nullable=False)

comment = Column(Text, nullable=False)

review\_date = Column(Date, nullable=False)

# добавляем свойства sneaker и customer для связи с моделями Sneaker и Customer

sneaker = relationship("Sneaker", back\_populates="reviews")

customer = relationship("Customer", back\_populates="reviews")

def \_\_init\_\_(self, sneaker\_id: int, customer\_id: int, rating: int, comment: str, review\_date):

self.sneaker\_id = sneaker\_id

self.customer\_id = customer\_id

self.rating = rating

self.comment = comment

self.review\_date = review\_date

## ORM/tables/sneakers.py:

from sqlalchemy.orm import relationship

from sqlalchemy import Column, Integer, String

from tables.base import Base

class Sneaker(Base):

\_\_tablename\_\_ = 'sneakers'

id = Column(Integer, primary\_key=True, index=True)

brand = Column(String, nullable=False)

model = Column(String, nullable=False)

size = Column(Integer, nullable=False)

color = Column(String, nullable=False)

price = Column(String, nullable=False)

orders = relationship("Order", back\_populates="sneaker")

def \_\_init\_\_(self, brand: str, model: str, size: int, color: str, price: str):

self.brand = brand

self.model = model

self.size = size

self.color = color

self.price = price

# ПРОХОЖДЕНИЕ ТЕСТОВ

## ORM/ test\_brands.py:

import unittest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Brand, Base

class TestBrandMethods(unittest.TestCase):

def setUp(self):

# Создаем временную базу данных в памяти для тестов

self.engine = create\_engine('sqlite:///:memory:')

Base.metadata.create\_all(bind=self.engine)

# Создаем сессию для взаимодействия с базой данных

Session = sessionmaker(bind=self.engine)

self.session = Session()

# Добавляем тестовые данные

self.nike = Brand(brand\_name='Nike')

self.adidas = Brand(brand\_name='Adidas')

self.session.add\_all([self.nike, self.adidas])

self.session.commit()

def tearDown(self):

# Закрываем сессию и удаляем временную базу данных

self.session.close()

Base.metadata.drop\_all(bind=self.engine)

def test\_get\_brand(self):

# Тест для метода get

retrieved\_nike = Brand.get(self.session, brand\_id=self.nike.brand\_id)

self.assertEqual(retrieved\_nike.brand\_name, 'Nike')

retrieved\_adidas = Brand.get(self.session, brand\_id=self.adidas.brand\_id)

self.assertEqual(retrieved\_adidas.brand\_name, 'Adidas')

not\_found\_brand = Brand.get(self.session, brand\_id=999) # Несуществующий ID

self.assertIsNone(not\_found\_brand)

def test\_find\_brand(self):

# Тест для метода find

found\_nike = Brand.find(self.session, brand\_name='Nike')

self.assertEqual(found\_nike.brand\_name, 'Nike')

found\_adidas = Brand.find(self.session, brand\_name='Adidas')

self.assertEqual(found\_adidas.brand\_name, 'Adidas')

not\_found\_brand = Brand.find(self.session, brand\_name='NonexistentBrand')

self.assertIsNone(not\_found\_brand)

def test\_get\_all\_brands(self):

# Тест для метода get\_all

all\_brands = Brand.get\_all(self.session)

self.assertEqual(len(all\_brands), 2)

brand\_names = set(brand.brand\_name for brand in all\_brands)

self.assertEqual(brand\_names, {'Nike', 'Adidas'})

def test\_delete\_brand(self):

# Тест для метода delete

Brand.delete(self.session, brand\_id=self.nike.brand\_id)

remaining\_brands = Brand.get\_all(self.session)

self.assertEqual(len(remaining\_brands), 1)

deleted\_brand = Brand.get(self.session, brand\_id=self.nike.brand\_id)

self.assertIsNone(deleted\_brand)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## ORM/test\_categories.py:

import unittest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Category, Base

class TestCategoryMethods(unittest.TestCase):

def setUp(self):

# Создаем временную базу данных в памяти для тестов

self.engine = create\_engine('sqlite:///:memory:')

Base.metadata.create\_all(bind=self.engine)

# Создаем сессию для взаимодействия с базой данных

Session = sessionmaker(bind=self.engine)

self.session = Session()

# Добавляем тестовые данные

self.running = Category(category\_name='Running')

self.basketball = Category(category\_name='Basketball')

self.session.add\_all([self.running, self.basketball])

self.session.commit()

def tearDown(self):

# Закрываем сессию и удаляем временную базу данных

self.session.close()

Base.metadata.drop\_all(bind=self.engine)

def test\_get\_category(self):

# Тест для метода get

retrieved\_running = Category.get(self.session, category\_id=self.running.category\_id)

self.assertEqual(retrieved\_running.category\_name, 'Running')

retrieved\_basketball = Category.get(self.session, category\_id=self.basketball.category\_id)

self.assertEqual(retrieved\_basketball.category\_name, 'Basketball')

not\_found\_category = Category.get(self.session, category\_id=999) # Несуществующий ID

self.assertIsNone(not\_found\_category)

def test\_find\_category(self):

# Тест для метода find

found\_running = Category.find(self.session, category\_name='Running')

self.assertEqual(found\_running.category\_name, 'Running')

found\_basketball = Category.find(self.session, category\_name='Basketball')

self.assertEqual(found\_basketball.category\_name, 'Basketball')

not\_found\_category = Category.find(self.session, category\_name='NonexistentCategory')

self.assertIsNone(not\_found\_category)

def test\_get\_all\_categories(self):

# Тест для метода get\_all

all\_categories = Category.get\_all(self.session)

self.assertEqual(len(all\_categories), 2)

category\_names = set(category.category\_name for category in all\_categories)

self.assertEqual(category\_names, {'Running', 'Basketball'})

def test\_delete\_category(self):

# Тест для метода delete

Category.delete(self.session, category\_id=self.running.category\_id)

remaining\_categories = Category.get\_all(self.session)

self.assertEqual(len(remaining\_categories), 1)

deleted\_category = Category.get(self.session, category\_id=self.running.category\_id)

self.assertIsNone(deleted\_category)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## ORM/test\_database.py:

import unittest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Base

class TestDatabase(unittest.TestCase):

def test\_database\_connection(self):

# Задаем URL вашей тестовой базы данных

db\_url = 'postgresql://postgres:123@localhost:5432/testbd1' # Обратите внимание, что добавлен суффикс \_test

# Создаем движок для подключения к базе данных

engine = create\_engine(db\_url)

# Проверяем успешность подключения к базе данных

connection = engine.connect()

self.assertIsNotNone(connection)

connection.close()

def test\_create\_session(self):

# Задаем URL вашей тестовой базы данных

db\_url = 'postgresql://postgres:123@localhost:5432/testbd1\_test' # Обратите внимание, что добавлен суффикс \_test

# Создаем движок для подключения к базе данных

engine = create\_engine(db\_url)

# Создаем сессию для взаимодействия с базой данных

SessionLocal = sessionmaker(bind=engine)

session = SessionLocal()

# Проверяем успешность создания сессии

self.assertIsNotNone(session)

session.close()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## ORM/test\_models.py:

import unittest

from models import Sneaker, Brand, Category

class TestSneaker(unittest.TestCase):

def test\_create\_sneaker(self):

sneaker = Sneaker(model='Test Model', brand\_id=1, category\_id=1, price='100.00')

self.assertEqual(sneaker.model, 'Test Model')

self.assertEqual(sneaker.brand\_id, 1)

self.assertEqual(sneaker.category\_id, 1)

self.assertEqual(sneaker.price, '100.00')

class TestBrand(unittest.TestCase):

def test\_create\_brand(self):

brand = Brand(brand\_name='Test Brand')

self.assertEqual(brand.brand\_name, 'Test Brand')

class TestCategory(unittest.TestCase):

def test\_create\_category(self):

category = Category(category\_name='Test Category')

self.assertEqual(category.category\_name, 'Test Category')

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## ORM/test\_sneakers.py:

import unittest

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Sneaker, Category, Brand, Base # Замените "your\_module" на фактическое имя вашего модуля, содержащего описания таблиц и классов

class TestSneakerMethods(unittest.TestCase):

def setUp(self):

# Создаем временную базу данных в памяти для тестов

self.engine = create\_engine('sqlite:///:memory:')

Base.metadata.create\_all(bind=self.engine)

# Создаем сессию для взаимодействия с базой данных

Session = sessionmaker(bind=self.engine)

self.session = Session()

# Добавляем тестовые данные

self.running\_category = Category(category\_name='Running')

self.basketball\_category = Category(category\_name='Basketball')

self.nike\_brand = Brand(brand\_name='Nike')

self.adidas\_brand = Brand(brand\_name='Adidas')

self.session.add\_all([self.running\_category, self.basketball\_category, self.nike\_brand, self.adidas\_brand])

self.session.commit()

self.running\_sneaker = Sneaker(model='Air Max', brand\_id=self.nike\_brand.brand\_id, category\_id=self.running\_category.category\_id, price='100.00')

self.basketball\_sneaker = Sneaker(model='Superstar', brand\_id=self.adidas\_brand.brand\_id, category\_id=self.basketball\_category.category\_id, price='80.00')

self.session.add\_all([self.running\_sneaker, self.basketball\_sneaker])

self.session.commit()

def tearDown(self):

# Закрываем сессию и удаляем временную базу данных

self.session.close()

Base.metadata.drop\_all(bind=self.engine)

def test\_get\_sneaker(self):

# Тест для метода get

retrieved\_running\_sneaker = Sneaker.get(self.session, sneaker\_id=self.running\_sneaker.sneaker\_id)

self.assertEqual(retrieved\_running\_sneaker.model, 'Air Max')

retrieved\_basketball\_sneaker = Sneaker.get(self.session, sneaker\_id=self.basketball\_sneaker.sneaker\_id)

self.assertEqual(retrieved\_basketball\_sneaker.model, 'Superstar')

not\_found\_sneaker = Sneaker.get(self.session, sneaker\_id=999) # Несуществующий ID

self.assertIsNone(not\_found\_sneaker)

def test\_find\_sneaker(self):

# Тест для метода find

found\_running\_sneaker = Sneaker.find(self.session, model='Air Max')

self.assertEqual(found\_running\_sneaker.model, 'Air Max')

found\_basketball\_sneaker = Sneaker.find(self.session, model='Superstar')

self.assertEqual(found\_basketball\_sneaker.model, 'Superstar')

not\_found\_sneaker = Sneaker.find(self.session, model='NonexistentModel')

self.assertIsNone(not\_found\_sneaker)

def test\_get\_all\_sneakers(self):

# Тест для метода get\_all

all\_sneakers = Sneaker.get\_all(self.session)

self.assertEqual(len(all\_sneakers), 2)

sneaker\_models = set(sneaker.model for sneaker in all\_sneakers)

self.assertEqual(sneaker\_models, {'Air Max', 'Superstar'})

def test\_delete\_sneaker(self):

# Тест для метода delete

Sneaker.delete(self.session, sneaker\_id=self.running\_sneaker.sneaker\_id)

remaining\_sneakers = Sneaker.get\_all(self.session)

self.assertEqual(len(remaining\_sneakers), 1)

deleted\_sneaker = Sneaker.get(self.session, sneaker\_id=self.running\_sneaker.sneaker\_id)

self.assertIsNone(deleted\_sneaker)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 2 — Результаты прохождения 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.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 3 — ERD диаграмма

# Список литературы

x

|  |  |
| --- | --- |
| 1. | Васильева М.А. Хобта Д.О. Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий. Издание второе, исправленное и дополненное ed. Москва: РУТ(МИИТ), 2023. 105 pp. |
| 2. | Васильева М.А. Ракиньцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. РУТ(МИИТ) ed. Москва. 2023 г. 63–с. pp. |
| 3. | SQLAlchemy 2023 г. URL: https:/​/​www.sqlalchemy.org/​ |

x