**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 Судоходный порт»**

Выполнил: ст. гр. ТКИ-541,  
Барсегян А. А.  
Проверил: доц., к.т.н.,

Васильева М. А.

Москва 2023

Оглавление

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 5](#_Toc160553647)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 5](#_Toc160553648)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 5](#_Toc160553649)

[UML - ДИАГРАММА КЛАССОВ 6](#_Toc160553650)

[СТРУКТУРА ПРОЕКТА 7](#_Toc160553651)

[КОД ПРОЕКТА 7](#_Toc160553652)

[ORM/main.py: 7](#_Toc160553653)

[ORM/database.py: 9](#_Toc160553654)

[ORM/methods.py: 10](#_Toc160553655)

[ORM/comp/base.py: 11](#_Toc160553656)

[ORM/comp/ship.py: 11](#_Toc160553657)

[ORM/comp/voyage.py: 11](#_Toc160553658)

[ПРОХОЖДЕНИЕ ТЕСТОВ 12](#_Toc160553659)

[ORM/ test\_ship.py: 12](#_Toc160553660)

[ORM/test\_voyage.py: 15](#_Toc160553661)

[ERD - ДИАГРАММА 18](#_Toc160553662)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc160553663)

# ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

Разработать систему учета кораблей и их маршрутов. Система должна позволять добавлять новые корабли, регистрировать их конечные точки и привязывать к уникальным номерам маршрутов.

# 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

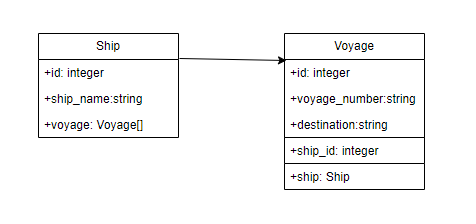


Рисунок 1 - UML-диаграмма классов

# СТРУКТУРА ПРОЕКТА

├───ORM │ │ ├───main.py │ ├───database.py │ ├───methods.py │ ├───test\_ship.py │ ├───test\_voyage.py │ ├───comp │ │ ├───voyage.py │ │ ├───ship.py │ │ ├───base.py

# КОД ПРОЕКТА

## ORM/main.py:

from database import SessionLocal, init\_db

from methods import BaseMethod

from comp.ship import Ship

from comp.voyage import Voyage

# Создание таблиц

init\_db()

# Создание сессии

session = SessionLocal()

ship\_repository = BaseMethod(session, Ship)

voyage\_repository = BaseMethod(session, Voyage)

# Добавление данных о судах

titanic = Ship(ship\_name="Titanic")

queen\_mary = Ship(ship\_name="Queen Mary")

symphony\_of\_the\_seas = Ship(ship\_name="Symphony of the Seas")

viking\_sky = Ship(ship\_name="Viking Sky")

ship\_repository.save(titanic)

ship\_repository.save(queen\_mary)

ship\_repository.save(symphony\_of\_the\_seas)

ship\_repository.save(viking\_sky)

# Добавление данных о маршрутах

voyage1 = Voyage(voyage\_number="VY101", destination="Barcelona", ship\_id=1)

voyage2 = Voyage(voyage\_number="QM256", destination="New York", ship\_id=2)

voyage3 = Voyage(voyage\_number="SO345", destination="Miami", ship\_id=3)

voyage4 = Voyage(voyage\_number="VK464", destination="Oslo", ship\_id=4)

voyage\_repository.save(voyage1)

voyage\_repository.save(voyage2)

voyage\_repository.save(voyage3)

voyage\_repository.save(voyage4)

# Вывод информации о маршрутах и судах

all\_voyages = voyage\_repository.get\_all()

for voyage in all\_voyages:

    ship = ship\_repository.get(voyage.ship\_id)

    print(f"Номер маршрута: {voyage.voyage\_number}")

    print(f"Место назначения: {voyage.destination}")

    print(f"Имя судна: {ship.ship\_name}")

    print("-" \* 50)

session.close()

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – В терминале видим что БД заполнена

## ORM/database.py:

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from comp.base import Base

DATABASE\_URL = "postgresql://postgres:123@localhost:5432/arutdb"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def init\_db():

    """

    Инициализация базы данных.

    """

    Base.metadata.create\_all(bind=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/comp/base.py:

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

## ORM/comp/ship.py:

from sqlalchemy import Column, Integer, String

from sqlalchemy.orm import relationship

from comp.base import Base

class Ship(Base):

    \_\_tablename\_\_ = 'ship'

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

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

    # Связь с моделью Voyage

    voyages = relationship("Voyage", back\_populates="ship")

    def \_\_init\_\_(self, ship\_name: str):

        self.ship\_name = ship\_name

## ORM/comp/voyage.py:

from sqlalchemy import Column, Integer, String, ForeignKey

from sqlalchemy.orm import relationship

from comp.base import Base

class Voyage(Base):

    \_\_tablename\_\_ = 'voyage'

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

    voyage\_number = Column(String, index=True)

    destination = Column(String)

    ship\_id = Column(Integer, ForeignKey('ship.id'))

    # Связь обратно к модели Ship

    ship = relationship("Ship", back\_populates="voyages")

    def \_\_init\_\_(self, voyage\_number: str, destination: str, ship\_id: int):

        self.voyage\_number = voyage\_number

        self.destination = destination

        self.ship\_id = ship\_id

# ПРОХОЖДЕНИЕ ТЕСТОВ

## ORM/ test\_ship.py:

import unittest

from sqlalchemy.inspection import inspect

from sqlalchemy.exc import IntegrityError

from comp.ship import Ship

from comp.voyage import Voyage

from database import SessionLocal, init\_db

from methods import BaseMethod

class TestShipMethods(unittest.TestCase):

    def setUp(self):

        init\_db()

        self.session = SessionLocal()

        self.ship\_method = BaseMethod(self.session, Ship)

    def test\_get(self):

        ship = Ship(ship\_name="Titanic")

        self.session.add(ship)

        self.session.commit()

        retrieved = self.ship\_method.get(ship.id)

        self.assertEqual(ship, retrieved)

    def test\_find(self):

        ship1 = Ship(ship\_name="Queen Mary")

        ship2 = Ship(ship\_name="Symphony of the Seas")

        self.session.add(ship1)

        self.session.add(ship2)

        self.session.commit()

        results = self.ship\_method.find(ship\_name="Queen Mary")

        self.assertIn(ship1, results)

        self.assertNotIn(ship2, results)

    def test\_get\_all(self):

        ship1 = Ship(ship\_name="Viking Sky")

        ship2 = Ship(ship\_name="Carnival Sunshine")

        self.session.add(ship1)

        self.session.add(ship2)

        self.session.commit()

        all\_records = self.ship\_method.get\_all()

        self.assertIn(ship1, all\_records)

        self.assertIn(ship2, all\_records)

    def test\_save(self):

        ship = Ship(ship\_name="Norwegian Bliss")

        self.ship\_method.save(ship)

        saved\_record = self.ship\_method.get(ship.id)

        self.assertEqual(saved\_record, ship)

    def test\_delete(self):

        ship = Ship(ship\_name="Harmony of the Seas")

        self.session.add(ship)

        self.session.commit()

        self.ship\_method.delete(ship)

        self.assertIsNone(self.ship\_method.get(ship.id))

    def tearDown(self):

        self.session.query(Voyage).delete()  # Удаляем все записи из таблицы voyage для соблюдения целостности

        self.session.query(Ship).delete()  # Удаляем все записи из таблицы ship

        self.session.commit()

        self.session.close()

    def test\_ship\_table\_mapping(self):

        ship\_mapper = inspect(Ship)

        self.assertEqual(ship\_mapper.local\_table.name, "ship")

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

    unittest.main()

## ORM/test\_voyage.py:

import unittest

from sqlalchemy.inspection import inspect

from comp.voyage import Voyage

from comp.ship import Ship

from database import SessionLocal, init\_db

from methods import BaseMethod

class TestVoyageMethods(unittest.TestCase):

    def setUp(self):

        init\_db()

        self.session = SessionLocal()

        self.voyage\_method = BaseMethod(self.session, Voyage)

        self.test\_ship = Ship(ship\_name="Test Ship")

        self.session.add(self.test\_ship)

        self.session.commit()

    def test\_get(self):

        voyage = Voyage(voyage\_number="VY100", destination="Hawaii", ship\_id=self.test\_ship.id)

        self.session.add(voyage)

        self.session.commit()

        retrieved = self.voyage\_method.get(voyage.id)

        self.assertEqual(voyage, retrieved)

    def test\_find(self):

        voyage1 = Voyage(voyage\_number="VM101", destination="Alaska", ship\_id=self.test\_ship.id)

        voyage2 = Voyage(voyage\_number="VS202", destination="Bahamas", ship\_id=self.test\_ship.id)

        self.session.add(voyage1)

        self.session.add(voyage2)

        self.session.commit()

        results = self.voyage\_method.find(destination="Alaska")

        self.assertIn(voyage1, results)

        self.assertNotIn(voyage2, results)

    def test\_get\_all(self):

        voyage1 = Voyage(voyage\_number="VC303", destination="Caribbean", ship\_id=self.test\_ship.id)

        voyage2 = Voyage(voyage\_number="VE404", destination="Mediterranean", ship\_id=self.test\_ship.id)

        self.session.add(voyage1)

        self.session.add(voyage2)

        self.session.commit()

        all\_records = self.voyage\_method.get\_all()

        self.assertIn(voyage1, all\_records)

        self.assertIn(voyage2, all\_records)

    def test\_save(self):

        voyage = Voyage(voyage\_number="VN505", destination="Norwegian Fjords", ship\_id=self.test\_ship.id)

        self.voyage\_method.save(voyage)

        saved\_record = self.voyage\_method.get(voyage.id)

        self.assertEqual(saved\_record, voyage)

    def test\_delete(self):

        voyage = Voyage(voyage\_number="VB606", destination="Baltic Sea", ship\_id=self.test\_ship.id)

        self.session.add(voyage)

        self.session.commit()

        self.voyage\_method.delete(voyage)

        self.assertIsNone(self.voyage\_method.get(voyage.id))

    def tearDown(self):

        self.session.query(Voyage).delete()

        self.session.query(Ship).delete()

        self.session.commit()

        self.session.close()

    def test\_voyage\_table\_mapping(self):

        voyage\_mapper = inspect(Voyage)

        self.assertEqual(voyage\_mapper.local\_table.name, "voyage")

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

    unittest.main()

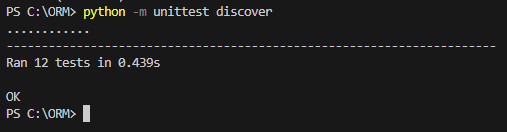


Рисунок 3 - Результаты прохождения тестов

# 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. В нашем случае, для создания диаграммы ERD, мы начали с анализа основных классов и сущностей в вашем проекте, таких как «ship» и «voyage». Основываясь на атрибутах и свойствах каждой сущности, а также на их взаимодействии, мы определили отношения между ними.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 4 - ERD-диаграмма

# 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).