**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 Поликлиника»**

Выполнил: ст. гр. ТКИ-541,  
Куминов В.П.  
  
Проверил: доц., к.т.н.,

Васильева М. А.

Москва 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 5](#_Toc150526274)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 5](#_Toc150526275)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 5](#_Toc150526276)

[UML - ДИАГРАММА КЛАССОВ 6](#_Toc150526277)

[СТРУКТУРА ПРОЕКТА 7](#_Toc150526278)

[КОД ПРОЕКТА 7](#_Toc150526279)

[ПРОХОЖДЕНИЕ ТЕСТОВ 13](#_Toc150526280)

[ERD - ДИАГРАММА 18](#_Toc150526281)

[ВЫВОД 19](#_Toc150526282)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc150526283)

# ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

Разработать БД «Поликлиника». Система должна позволять добавлять новых пациентов, назначать им врача, дату приёма и кабинет.

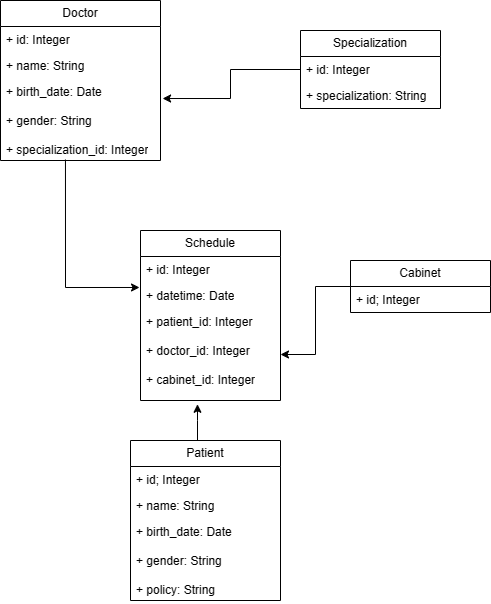
# ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ

Познакомиться с механизмом, который не позволяет 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



**СТРУКТУРА ПРОЕКТА**

├───Polyclinic

│ ├───base.py │ ├───cabinet.py │ ├───doctor.py │ ├───patient.py │ ├───schedule.py

│ ├───specialization.py

│ ├───methods.py │ ├───test │ │ ├───database.py │ │ ├───test\_cabinet.py │ │ ├───test\_doctor.py

│ │ ├───test\_patient.py │ │ ├───test\_schedule.py │ │ ├───test\_specialization.py

# КОД ПРОЕКТА

**Polyclinic/base.py**

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

**Polyclinic/cabinet.py**

from sqlalchemy import Column, Integer

from sqlalchemy.orm import relationship

from base import Base

class Cabinet(Base):

\_\_tablename\_\_ = 'cabinets'

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

schedules = relationship('Schedule', back\_populates='cabinet')

def \_\_init\_\_(self, id):

self.id = id

**Polyclinic/doctor.py**

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

from sqlalchemy.orm import relationship

from base import Base

class Doctor(Base):

\_\_tablename\_\_ = 'doctors'

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

name = Column(String)

birth\_date = Column(Date)

gender = Column(String)

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

schedules = relationship('Schedule', back\_populates='doctor')

specialization = relationship('Specialization', back\_populates='doctors')

def \_\_init\_\_(self, name: str, birth\_date: Date, gender: str, specialization\_id: int):

self.name = name

self.birth\_date = birth\_date

self.gender = gender

self.specialization\_id = specialization\_id

**Polyclinic/patient.py**

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

from sqlalchemy.orm import relationship

from base import Base

from schedule import Schedule

class Patient(Base):

\_\_tablename\_\_ = 'patients'

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

name = Column(String)

birth\_date = Column(Date)

gender = Column(String)

policy = Column(String)

schedules = relationship('Schedule', back\_populates='patient')

def \_\_init\_\_(self, name: str, birth\_date: Date, gender: str, policy: str):

self.name = name

self.birth\_date = birth\_date

self.gender = gender

self.policy = policy

**Polyclinic/schedule.py**

from sqlalchemy import Column, Integer, DateTime, ForeignKey

from sqlalchemy.orm import relationship

from base import Base

class Schedule(Base):

\_\_tablename\_\_ = 'schedules'

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

datetime = Column(DateTime)

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

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

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

patient = relationship('Patient', back\_populates='schedules')

doctor = relationship('Doctor', back\_populates='schedules')

cabinet = relationship('Cabinet', back\_populates='schedules')

def \_\_init\_\_(self, datetime: DateTime, patient\_id: int, doctor\_id: int, cabinet\_id: int):

self.datetime = datetime

self.patient\_id = patient\_id

self.doctor\_id = doctor\_id

self.cabinet\_id = cabinet\_id

**Polyclinic/specialization.py**

from sqlalchemy import Column, Integer, String

from sqlalchemy.orm import relationship

from base import Base

class Specialization(Base):

\_\_tablename\_\_ = 'specializations'

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

specialization = Column(String)

doctors = relationship('Doctor', back\_populates='specialization')

def \_\_init\_\_(self, specialization: str):

self.specialization = specialization

**Polyclinic/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()

# ПРОХОЖДЕНИЕ ТЕСТОВ

**Polyclinic/test/test\_cabinet.py**

import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

import unittest

from datetime import date

from patient import Patient

from schedule import Schedule

from doctor import Doctor

from cabinet import Cabinet

from specialization import Specialization

class TestCabinet(unittest.TestCase):

def setUp(self):

self.cabinet = Cabinet(101)

def test\_init(self):

self.assertEqual(self.cabinet.id, 101)

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

unittest.main()

**Polyclinic/test/test\_doctor.py**

import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

import unittest

from datetime import date

from patient import Patient

from schedule import Schedule

from doctor import Doctor

from cabinet import Cabinet

from specialization import Specialization

class TestDoctor(unittest.TestCase):

def setUp(self):

self.doctor = Doctor('Доктор Хаус', date(1980, 1, 1), 'М', 1)

def test\_init(self):

self.assertEqual(self.doctor.name, 'Доктор Хаус')

self.assertEqual(self.doctor.birth\_date, date(1980, 1, 1))

self.assertEqual(self.doctor.gender, 'М')

self.assertEqual(self.doctor.specialization\_id, 1)

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

unittest.main()

**Polyclinic/test/test\_patient.py**

import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

import unittest

from datetime import date

from patient import Patient

from schedule import Schedule

from doctor import Doctor

from cabinet import Cabinet

from specialization import Specialization

class TestPatient(unittest.TestCase):

def setUp(self):

self.patient = Patient('Иван Иванов', date(1980, 1, 1), 'М', '1234567890')

def test\_init(self):

self.assertEqual(self.patient.name, 'Иван Иванов')

self.assertEqual(self.patient.birth\_date, date(1980, 1, 1))

self.assertEqual(self.patient.gender, 'М')

self.assertEqual(self.patient.policy, '1234567890')

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

unittest.main()

**Polyclinic/test/test\_schedule.py**

import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

import unittest

from datetime import datetime

from patient import Patient

from schedule import Schedule

from doctor import Doctor

from cabinet import Cabinet

from specialization import Specialization

class TestSchedule(unittest.TestCase):

def setUp(self):

self.schedule = Schedule(datetime.now(), 1, 1, 1)

def test\_init(self):

self.assertEqual(self.schedule.patient\_id, 1)

self.assertEqual(self.schedule.doctor\_id, 1)

self.assertEqual(self.schedule.cabinet\_id, 1)

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

unittest.main()

**Polyclinic/test/test\_specialization.py**

import sys

import unittest

from datetime import datetime

from patient import Patient

from schedule import Schedule

from doctor import Doctor

from cabinet import Cabinet

from specialization import Specialization

class TestSpecialization(unittest.TestCase):

def setUp(self):

self.specialization = Specialization('Терапевт')

def test\_init(self):

self.assertEqual(self.specialization.specialization, 'Терапевт')

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

unittest.main()

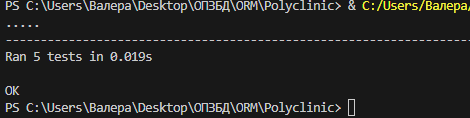


Рисунок 2 - Результаты прохождения тестов

# 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. В нашем случае, для создания диаграммы ERD, мы начали с анализа основных классов и сущностей в нашем проекте, основываясь на атрибутах и свойствах каждой сущности, а также на их взаимодействии, мы определили отношения между ними.

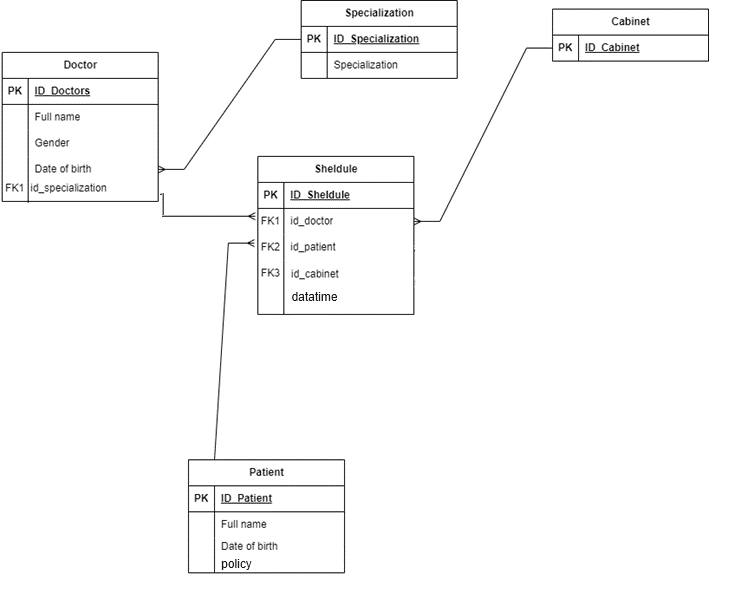


Рисунок 3 - ERD-диаграмма

# ВЫВОД

В ходе выполнения работы была создана модели для базы данных «Поликлиника», используя SQLAlchemy ORM. Модели включают Cabinet, Doctor, Patient, Schedule и Specialization. Каждая из этих моделей представляет собой таблицу в базе данных и содержит различные поля и связи, соответствующие структуре данных поликлиники.

Также создан класс Methods, который предоставляет базовые методы для работы с этими моделями, включая получение, поиск, сохранение и удаление записей. Все классы были успешно протестированы с использованием модуля unittest в Python.

В процессе работы пришлось столкнуться с некоторыми проблемами, включая ошибки импорта и необходимость определения дополнительных функций, таких как init\_db. Однако удалось успешно решить эти проблемы и выполнить работу в полном объёме.

# 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., 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.

3. SQLAlchemy. — Текст : электронный // SQLAlchemy : [сайт]. — URL: https://www.sqlalchemy.org/ (дата обращения: 10.11.2023).

4. PostgreSQL. — Текст : электронный // PostgreSQL : [сайт]. — URL: https://www.postgresql.org/ (дата обращения: 10.11.2023).

5. Python Documentation. — Текст : электронный // Python: [сайт]. — URL: https://www.python.org/doc/ (дата обращения: 10.11.2023).