**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»**

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

Васильева М. А.

Москва 2023

Оглавление

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 6](#_Toc149071925)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 6](#_Toc149071926)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 6](#_Toc149071927)

[UML - ДИАГРАММА КЛАССОВ 7](#_Toc149071928)

[СТРУКТУРА ПРОЕКТА 8](#_Toc149071929)

[КОД ПРОЕКТА 8](#_Toc149071930)

[models/base.py 8](#_Toc149071931)

[models/ employee\_equipment.py 8](#_Toc149071932)

[models/employee.py 9](#_Toc149071933)

[models/equipment.py 11](#_Toc149071934)

[models/position.py 12](#_Toc149071935)

[models/time\_track.py 13](#_Toc149071936)

[ПРОХОЖДЕНИЕ ТЕСТОВ 14](#_Toc149071937)

[tests/ test\_employee\_equipment\_model.py 14](#_Toc149071938)

[tests/ test\_employee\_model.py 17](#_Toc149071939)

[tests/ test\_equipment\_model.py 20](#_Toc149071940)

[tests/ test\_mapping.py 23](#_Toc149071941)

[tests/ test\_position\_model.py 25](#_Toc149071942)

[tests/ test\_time\_track\_model.py 27](#_Toc149071943)

[ERD - ДИАГРАММА 30](#_Toc149071944)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc149071945)

# ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

Разработать систему учета сотрудников и выданного им оборудования в компании. Система должна позволять добавлять новых сотрудников, регистрировать выдачу им оборудования и привязывать их к определенным должностям. Каждый сотрудник имеет уникальные данные, такие как имя, фамилия, отчество, контактная информация и связанные с ним отметки времени. Оборудование также имеет свои характеристики, включая название и список сотрудников, которым оно было выдано.

# ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ

Научиться эффективно работать с базами данных, используя подход ORM (Object-Relational Mapping), для создания, модификации, запроса и управления данными в структурированной и объектно-ориентированной манере.

# ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ

1. Python: основной язык программирования, на котором написан предоставленный код.

2. SQLAlchemy: ORM-библиотека для Python, которая позволяет взаимодействовать с базами данных на высоком уровне, используя объектно-ориентированный подход. С ее помощью можно создавать модели, связи между ними и формировать запросы к базе данных.

3. SQL: язык структурированных запросов, используемый для определения и управления данными в реляционных базах данных. В коде он не присутствует напрямую, но SQLAlchemy генерирует SQL-запросы на основе объектных моделей и методов.

4. DBeaver: универсальный инструмент для работы с базами данных, который поддерживает множество различных СУБД. DBeaver предоставляет графический интерфейс для управления базами данных, выполнения SQL-запросов, просмотра структуры базы данных, экспорта и импорта данных и многих других функций.

Этот стек технологий позволяет эффективно работать с базами данных, избегая необходимости написания "сырых" SQL-запросов и предоставляя высокоуровневые инструменты для работы с данными.

# UML - ДИАГРАММА КЛАССОВ

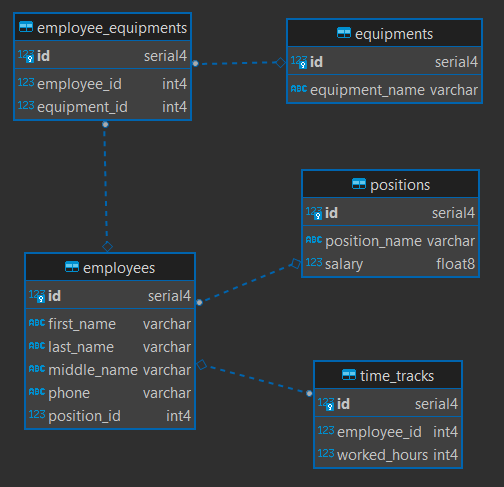
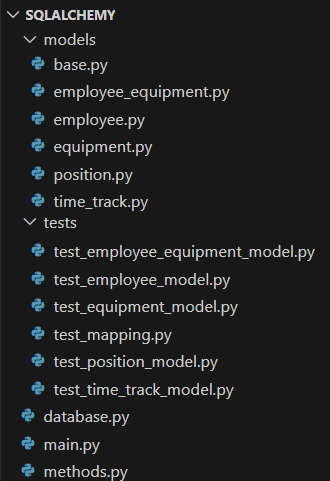


Рисунок 1 - UML-диаграмма классов

# СТРУКТУРА ПРОЕКТА



# КОД ПРОЕКТА

## models/base.py

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

## models/ employee\_equipment.py

from sqlalchemy import Column, Integer, ForeignKey

from sqlalchemy.orm import relationship

from .base import Base

class EmployeeEquipment(Base):

"""

Модель для связи между сотрудниками и оборудованием.

Отражает, какое оборудование выдано каждому сотруднику.

Атрибуты:

- id: Уникальный идентификатор записи.

- employee\_id: Идентификатор сотрудника.

- equipment\_id: Идентификатор оборудования.

- employee: Связь с моделью сотрудника.

- equipment: Связь с моделью оборудования.

"""

\_\_tablename\_\_ = 'employee\_equipments'

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

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

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

employee = relationship("Employee", back\_populates="equipments")

equipment = relationship("Equipment", back\_populates="employees")

## models/employee.py

from sqlalchemy import Column, String, Integer, ForeignKey

from sqlalchemy.orm import relationship

from .base import Base

class Employee(Base):

"""

Модель сотрудника.

Описывает основную информацию о сотруднике, его должность и связь с другими сущностями.

Атрибуты:

- id: Уникальный идентификатор сотрудника.

- first\_name: Имя сотрудника.

- last\_name: Фамилия сотрудника.

- middle\_name: Отчество сотрудника.

- phone: Номер телефона сотрудника.

- position\_id: Идентификатор должности сотрудника.

- position: Связь с моделью должности.

- time\_tracks: Список отметок времени сотрудника.

- equipments: Список оборудования, выданного сотруднику.

"""

\_\_tablename\_\_ = 'employees'

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

first\_name = Column(String)

last\_name = Column(String)

middle\_name = Column(String)

phone = Column(String)

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

position = relationship("Position", back\_populates="employees")

time\_tracks = relationship("TimeTrack", back\_populates="employee")

equipments = relationship("EmployeeEquipment", back\_populates="employee")

## models/equipment.py

from sqlalchemy import Column, String, Integer

from sqlalchemy.orm import relationship

from .base import Base

class Equipment(Base):

"""

Модель оборудования.

Описывает основную информацию о единице оборудования и связь с сотрудниками, которым это оборудование выдано.

Атрибуты:

- id: Уникальный идентификатор оборудования.

- equipment\_name: Название оборудования.

- employees: Список сотрудников, которым выдано данное оборудование.

"""

\_\_tablename\_\_ = 'equipments'

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

equipment\_name = Column(String)

employees = relationship("EmployeeEquipment", back\_populates="equipment")

## models/position.py

from sqlalchemy import Column, String, Integer, Float

from sqlalchemy.orm import relationship

from .base import Base

class Position(Base):

"""

Модель должности сотрудника.

Описывает основную информацию о должности, включая зарплату и связанных с этой должностью сотрудников.

Атрибуты:

- id: Уникальный идентификатор должности.

- position\_name: Название должности (уникальное).

- salary: Зарплата, связанная с этой должностью.

- employees: Список сотрудников, занимающих эту должность.

"""

\_\_tablename\_\_ = 'positions'

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

position\_name = Column(String, unique=True)

salary = Column(Float)

employees = relationship("Employee", back\_populates="position")

## models/time\_track.py

from sqlalchemy import Column, Integer, ForeignKey

from sqlalchemy.orm import relationship

from .base import Base

class TimeTrack(Base):

"""

Модель учета рабочего времени сотрудника.

Описывает количество отработанных часов конкретным сотрудником.

Атрибуты:

- id: Уникальный идентификатор записи.

- employee\_id: Идентификатор сотрудника.

- worked\_hours: Количество отработанных часов.

- employee: Связь с моделью сотрудника.

"""

\_\_tablename\_\_ = 'time\_tracks'

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

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

worked\_hours = Column(Integer)

employee = relationship("Employee", back\_populates="time\_tracks")

# ПРОХОЖДЕНИЕ ТЕСТОВ

## tests/ test\_employee\_equipment\_model.py

import unittest

from models.employee\_equipment import EmployeeEquipment

from models.employee import Employee

from models.equipment import Equipment

from methods import BaseMethod

from database import SessionLocal, init\_db

class TestEmployeeEquipmentMethods(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

self.emp\_equip\_method = BaseMethod(self.session, EmployeeEquipment)

self.employee\_method = BaseMethod(self.session, Employee)

self.equipment\_method = BaseMethod(self.session, Equipment)

def test\_get(self):

employee = Employee(first\_name="John Doe")

equipment = Equipment(equipment\_name="Компьютер")

self.session.add(employee)

self.session.add(equipment)

self.session.commit()

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

retrieved = self.emp\_equip\_method.get(emp\_equip.id)

self.assertEqual(emp\_equip, retrieved)

def test\_find(self):

employee = Employee(first\_name="John Doe")

equipment = Equipment(equipment\_name="Мышь")

self.session.add(employee)

self.session.add(equipment)

self.session.commit()

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

results = self.emp\_equip\_method.find(employee\_id=employee.id)

self.assertIn(emp\_equip, results)

def test\_get\_all(self):

employee = Employee(first\_name="Jane Doe")

equipment = Equipment(equipment\_name="Монитор")

self.session.add(employee)

self.session.add(equipment)

self.session.commit()

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

all\_records = self.emp\_equip\_method.get\_all()

self.assertIn(emp\_equip, all\_records)

def test\_save(self):

employee = Employee(first\_name="Jane Smith")

equipment = Equipment(equipment\_name="Клавиатура")

self.session.add(employee)

self.session.add(equipment)

self.session.commit()

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

saved\_record = self.emp\_equip\_method.get(emp\_equip.id)

self.assertEqual(saved\_record, emp\_equip)

def test\_delete(self):

employee = Employee(first\_name="Mike Johnson")

equipment = Equipment(equipment\_name="Наушники")

self.session.add(employee)

self.session.add(equipment)

self.session.commit()

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

self.emp\_equip\_method.delete(emp\_equip)

self.assertIsNone(self.emp\_equip\_method.get(emp\_equip.id))

def tearDown(self):

self.session.query(EmployeeEquipment).delete()

self.session.query(Employee).delete()

self.session.query(Equipment).delete()

self.session.commit()

self.session.close()

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

unittest.main()

## tests/ test\_employee\_model.py

import sys

sys.path.append("D:/Документы/Учеба/Учеба\_5к1с/ОПЗБД/SQLAlchemy")

import unittest

from models.employee import Employee

from methods import BaseMethod

from database import SessionLocal, init\_db

class TestEmployeeModel(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

self.employee\_method = BaseMethod(self.session, Employee)

def test\_get\_existing\_employee(self):

# Тест получения существующего сотрудника по ID

employee = Employee(first\_name="Иван", last\_name="Иванов", phone="+79123456789")

self.employee\_method.save(employee)

retrieved\_employee = self.employee\_method.get(employee.id)

self.assertEqual(retrieved\_employee.first\_name, "Иван")

def test\_find\_existing\_employee(self):

# Тест поиска существующего сотрудника по имени

employee = Employee(first\_name="Анна", last\_name="Сергеева", phone="+79123456789")

self.employee\_method.save(employee)

employees = self.employee\_method.find(first\_name="Анна")

self.assertEqual(len(employees), 1)

self.assertEqual(employees[0].last\_name, "Сергеева")

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

# Тест получения всех сотрудников

employees = [

Employee(first\_name="Иван", last\_name="Иванов", phone="+79123456789"),

Employee(first\_name="Петр", last\_name="Петров", phone="+79123456789"),

Employee(first\_name="Анна", last\_name="Сергеева", phone="+79123456789"),

]

for employee in employees:

self.employee\_method.save(employee)

all\_employees = self.employee\_method.get\_all()

self.assertEqual(len(all\_employees), 3)

def test\_update\_employee(self):

# Тест обновления сотрудника

employee = Employee(first\_name="Олег", last\_name="Олегов", phone="+79123456789")

self.employee\_method.save(employee)

employee.first\_name = "Новое имя"

self.employee\_method.save(employee)

updated\_employee = self.employee\_method.get(employee.id)

self.assertEqual(updated\_employee.first\_name, "Новое имя")

def test\_delete\_existing\_employee(self):

# Тест удаления существующего сотрудника

employee = Employee(first\_name="Иван", last\_name="Иванов", phone="+79123456789")

self.employee\_method.save(employee)

self.employee\_method.delete(employee)

deleted\_employee = self.employee\_method.get(employee.id)

self.assertIsNone(deleted\_employee)

def tearDown(self):

self.session.query(Employee).delete()

self.session.commit()

self.session.close()

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

unittest.main()

## tests/ test\_equipment\_model.py

import sys

sys.path.append("D:/Документы/Учеба/Учеба\_5к1с/ОПЗБД/SQLAlchemy")

import unittest

from models.employee\_equipment import EmployeeEquipment

from models.employee import Employee

from models.equipment import Equipment

from methods import BaseMethod

from database import SessionLocal, init\_db

class TestEmployeeEquipmentModel(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

self.emp\_equip\_method = BaseMethod(self.session, EmployeeEquipment)

self.employee\_method = BaseMethod(self.session, Employee)

self.equipment\_method = BaseMethod(self.session, Equipment)

def test\_get\_existing\_emp\_equip(self):

# Тест получения существующей связи сотрудника и оборудования по ID

employee = Employee(first\_name="John Doe")

equipment = Equipment(equipment\_name="Компьютер")

self.employee\_method.save(employee)

self.equipment\_method.save(equipment)

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

retrieved = self.emp\_equip\_method.get(emp\_equip.id)

self.assertEqual(retrieved, emp\_equip)

def test\_find\_existing\_emp\_equip(self):

# Тест поиска существующей связи сотрудника и оборудования по ID сотрудника

employee = Employee(first\_name="John Doe")

equipment = Equipment(equipment\_name="Мышь")

self.employee\_method.save(employee)

self.equipment\_method.save(equipment)

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

results = self.emp\_equip\_method.find(employee\_id=employee.id)

self.assertIn(emp\_equip, results)

def test\_get\_all\_emp\_equip(self):

# Тест получения всех связей сотрудника и оборудования

employee = Employee(first\_name="Jane Doe")

equipment = Equipment(equipment\_name="Монитор")

self.employee\_method.save(employee)

self.equipment\_method.save(equipment)

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

all\_records = self.emp\_equip\_method.get\_all()

self.assertIn(emp\_equip, all\_records)

def test\_update\_emp\_equip(self):

# Тест обновления связи сотрудника и оборудования

employee1 = Employee(first\_name="Jane Smith")

employee2 = Employee(first\_name="Mike Johnson")

equipment = Equipment(equipment\_name="Клавиатура")

self.employee\_method.save(employee1)

self.employee\_method.save(employee2)

self.equipment\_method.save(equipment)

emp\_equip = EmployeeEquipment(employee\_id=employee1.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

emp\_equip.employee\_id = employee2.id

self.emp\_equip\_method.save(emp\_equip)

updated\_record = self.emp\_equip\_method.get(emp\_equip.id)

self.assertEqual(updated\_record.employee\_id, employee2.id)

def test\_delete\_existing\_emp\_equip(self):

# Тест удаления существующей связи сотрудника и оборудования

employee = Employee(first\_name="Mike Johnson")

equipment = Equipment(equipment\_name="Наушники")

self.employee\_method.save(employee)

self.equipment\_method.save(equipment)

emp\_equip = EmployeeEquipment(employee\_id=employee.id, equipment\_id=equipment.id)

self.emp\_equip\_method.save(emp\_equip)

self.emp\_equip\_method.delete(emp\_equip)

self.assertIsNone(self.emp\_equip\_method.get(emp\_equip.id))

def tearDown(self):

self.session.query(EmployeeEquipment).delete()

self.session.query(Employee).delete()

self.session.query(Equipment).delete()

self.session.commit()

self.session.close()

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

unittest.main()

## tests/ test\_mapping.py

import sys

sys.path.append("D:/Документы/Учеба/Учеба\_5к1с/ОПЗБД/SQLAlchemy")

import unittest

from sqlalchemy.inspection import inspect

from models.employee\_equipment import EmployeeEquipment

from models.employee import Employee

from models.equipment import Equipment

from models.position import Position

from models.time\_track import TimeTrack

from database import SessionLocal, init\_db

from methods import BaseMethod

class TestTableMapping(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

def test\_employee\_equipment\_table\_mapping(self):

employee\_equipment\_mapper = inspect(EmployeeEquipment)

self.assertEqual(employee\_equipment\_mapper.local\_table.name, "employee\_equipments")

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

employee\_mapper = inspect(Employee)

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

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

equipment\_mapper = inspect(Equipment)

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

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

position\_mapper = inspect(Position)

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

def test\_time\_track\_table\_mapping(self):

time\_track\_mapper = inspect(TimeTrack)

self.assertEqual(time\_track\_mapper.local\_table.name, "time\_tracks")

def tearDown(self):

self.session.close()

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

unittest.main()

## tests/ test\_position\_model.py

import sys

sys.path.append("D:/Документы/Учеба/Учеба\_5к1с/ОПЗБД/SQLAlchemy")

import unittest

from models.position import Position

from methods import BaseMethod

from database import SessionLocal, init\_db

class TestPositionModel(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

self.position\_method = BaseMethod(self.session, Position)

def test\_get\_position(self):

# Тест получения должности по ID

position = Position(position\_name="Менеджер", salary=100000)

self.position\_method.save(position)

retrieved\_position = self.position\_method.get(position.id)

self.assertEqual(retrieved\_position.position\_name, "Менеджер")

self.assertEqual(retrieved\_position.salary, 100000)

def test\_find\_position(self):

# Тест поиска должности по параметрам

position = Position(position\_name="Программист", salary=150000)

self.position\_method.save(position)

found\_positions = self.position\_method.find(position\_name="Программист")

self.assertEqual(len(found\_positions), 1)

self.assertEqual(found\_positions[0].salary, 150000)

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

# Тест получения всех должностей

position\_list = [

Position(position\_name="Аналитик", salary=120000),

Position(position\_name="Дизайнер", salary=80000),

Position(position\_name="Тестировщик", salary=90000),

]

for position in position\_list:

self.position\_method.save(position)

all\_positions = self.position\_method.get\_all()

self.assertEqual(len(all\_positions), len(position\_list))

def test\_update\_position(self):

# Тест обновления информации о должности

position = Position(position\_name="Бухгалтер", salary=110000)

self.position\_method.save(position)

position.salary = 115000

self.position\_method.save(position)

updated\_position = self.position\_method.get(position.id)

self.assertEqual(updated\_position.salary, 115000)

def test\_delete\_position(self):

# Тест удаления должности

position = Position(position\_name="Юрист", salary=130000)

self.position\_method.save(position)

self.position\_method.delete(position)

deleted\_position = self.position\_method.get(position.id)

self.assertIsNone(deleted\_position)

def tearDown(self):

self.session.query(Position).delete()

self.session.commit()

self.session.close()

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

unittest.main()

## tests/ test\_time\_track\_model.py

import sys

sys.path.append("D:/Документы/Учеба/Учеба\_5к1с/ОПЗБД/SQLAlchemy")

import unittest

from models.time\_track import TimeTrack

from models.employee import Employee

from methods import BaseMethod

from database import SessionLocal, init\_db

class TestTimeTrackModel(unittest.TestCase):

def setUp(self):

init\_db()

self.session = SessionLocal()

self.timetrack\_method = BaseMethod(self.session, TimeTrack)

def create\_employee(self):

employee = Employee(first\_name="Иван", last\_name="Иванов", phone="+79123456789")

self.session.add(employee)

self.session.commit()

return employee

def test\_get\_timetrack(self):

employee = self.create\_employee()

timetrack = TimeTrack(employee\_id=employee.id, worked\_hours=8)

self.timetrack\_method.save(timetrack)

retrieved\_timetrack = self.timetrack\_method.get(timetrack.id)

self.assertEqual(retrieved\_timetrack.worked\_hours, 8)

def test\_find\_timetrack\_by\_hours(self):

employee = self.create\_employee()

timetrack = TimeTrack(employee\_id=employee.id, worked\_hours=5)

self.timetrack\_method.save(timetrack)

found\_timetracks = self.timetrack\_method.find(worked\_hours=5)

self.assertEqual(len(found\_timetracks), 1)

self.assertEqual(found\_timetracks[0].worked\_hours, 5)

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

employee = self.create\_employee()

timetracks = [TimeTrack(employee\_id=employee.id, worked\_hours=i) for i in range(3)]

for timetrack in timetracks:

self.timetrack\_method.save(timetrack)

all\_timetracks = self.timetrack\_method.get\_all()

self.assertEqual(len(all\_timetracks), len(timetracks))

def test\_update\_timetrack(self):

employee = self.create\_employee()

timetrack = TimeTrack(employee\_id=employee.id, worked\_hours=4)

self.timetrack\_method.save(timetrack)

timetrack.worked\_hours = 7

self.timetrack\_method.save(timetrack)

updated\_timetrack = self.timetrack\_method.get(timetrack.id)

self.assertEqual(updated\_timetrack.worked\_hours, 7)

def test\_delete\_timetrack(self):

employee = self.create\_employee()

timetrack = TimeTrack(employee\_id=employee.id, worked\_hours=6)

self.timetrack\_method.save(timetrack)

self.timetrack\_method.delete(timetrack)

deleted\_timetrack = self.timetrack\_method.get(timetrack.id)

self.assertIsNone(deleted\_timetrack)

def tearDown(self):

self.session.query(TimeTrack).delete()

self.session.query(Employee).delete()

self.session.commit()

self.session.close()

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

unittest.main()

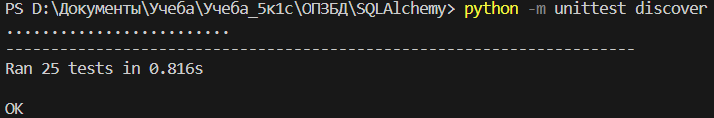


Рисунок 2 - Результаты прохождения тестов

# 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.

1. EmployeeEquipment (Оборудование сотрудника):

-Связано с Employee (Сотрудник) через поле employee\_id.

-Связано с Equipment (Оборудование) через поле equipment\_id.

2. Employee (Сотрудник):

-Связано с Position (Должность) через поле position\_id.

-Связано с TimeTrack (Учет времени) через поле employee\_id.

3. Position (Должность):

-Связано с Employee (Сотрудник) через поле position\_id.

4. Equipment (Оборудование):

-Связано с EmployeeEquipment (Оборудование сотрудника) через поле equipment\_id.

5. TimeTrack (Учет времени):

-Связано с Employee (Сотрудник) через поле employee\_id.

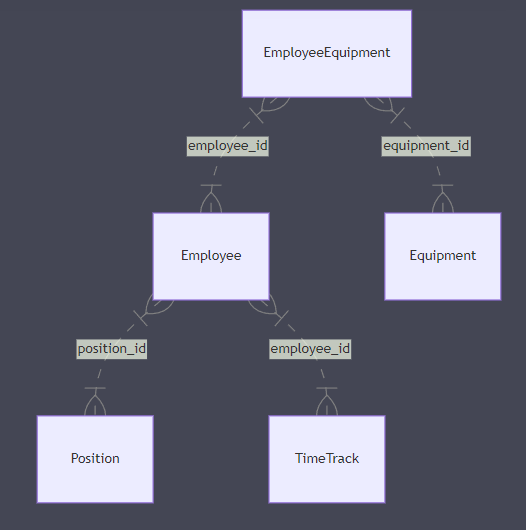


Рисунок 3 - ERD-диаграмма

# 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.