Защищено:
Гапанюк Ю.Е.

"__"_____2025 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2025 г.

Рубежный контроль 2 по курсу
Парадигмы и конструкции языков программирования
ГУИМЦ

Тема работы: "Рубежный контроль представляет собой разработку тестов на языке Python."

5
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-54Б _____
(подпись)

Щетинин Д.С.

"__"_____2025 г.

Москва, МГТУ - 2025 г.

1.Тема и задание для выполнения рубежного контроля.

Условия рубежного контроля №2 по курсу ПиК ЯП Рубежный контроль представляет собой разработку тестов на языке Python. 1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования. 2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

Условия рубежного контроля №1 по курсу ПиК ЯП
Тема работы: Рубежный контроль представляет собой разработку программы на языке Python, которая выполняет следующие действия:

1) Необходимо создать два класса данных в соответствии с Вашим вариантом предметной области, которые связаны отношениями один-ко-многим и многие-комногим. Пример классов данных для предметной области Сотрудник-Отдел:

1. Класс «Сотрудник», содержащий поля:
   - ID записи о сотруднике;
   - Фамилия сотрудника;
   - Зарплата (количественный признак);
   - ID записи об отделе. (для реализации связи один-ко-многим)
2. Класс «Отдел», содержащий поля:
   - ID записи об отделе;
   - Наименование отдела.
3. (Для реализации связи многие-ко-многим) Класс «Сотрудники отдела», содержащий поля:
   - ID записи о сотруднике;
   - ID записи об отделе.

2) Необходимо создать списки объектов классов, содержащих тестовые данные (3-5 записей), таким образом, чтобы первичные и вторичные ключи соответствующих записей были связаны по идентификаторам.

3) Необходимо разработать запросы в соответствии с Вашим вариантом. Запросы сформулированы в терминах классов «Сотрудник» и «Отдел», которые используются в примере. Вам нужно перенести эти требования в Ваш вариант предметной области. При разработке запросов необходимо по возможности использовать функциональные возможности языка Python (list/dict comprehensions, функции высших порядков).

Для реализации запроса №2 введите в класс, находящийся на стороне связи «много», произвольный количественный признак, например, «зарплата сотрудника». Результатом рубежного контроля является документ в формате PDF, который содержит текст программы и результаты ее выполнения.

Вариант Д.

1. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список всех сотрудников, у которых фамилия заканчивается на «ов», и названия их отделов.
2. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список отделов со средней зарплатой сотрудников в каждом отделе, отсортированный по средней зарплате (отдельной функции вычисления среднего значения в Python нет, нужно использовать комбинацию функций вычисления суммы и количества значений).
3. «Отдел» и «Сотрудник» связаны соотношением многие-ко-многим. Выведите список всех отделов, у которых название начинается с буквы «А», и список работающих в них сотрудников.

Таблица 1. Варианты предметной области

№ варианта	Класс 1	Класс 2
34	Хранимая процедура	База данных

2.Листинг программы

Файл classes.py:

```
class StoredProcedure:
    def __init__(self, id, name, db_id):
        self.id = id
        self.name = name
        self.db_id = db_id

class Database:
    def __init__(self, id, name):
        self.id = id
        self.name = name

class StoredProcedureDatabase:
    def __init__(self, sp_id, db_id):
        self.sp_id = sp_id
        self.db_id = db_id
```

Файл queries.py:

```
def find_procedures_ending_with_ov(procedures, databases):
    return [
        (sp.name, next(db.name for db in databases if db.id == sp.db_id))
        for sp in procedures if sp.name.endswith("ов")
    ]

def average_salary_by_database(procedures, databases):
    result = []
    for db in databases:
        db_procedures = [sp for sp in procedures if sp.db_id == db.id]
        if db_procedures:
            avg_salary = sum(len(sp.name) for sp in db_procedures) / len(db_procedures)  # пример
количественного признака
            result.append((db.name, avg_salary))
    return sorted(result, key=lambda x: x[1])

def databases_starting_with_a(procedures, databases, sp_dbs):
    result = []
    for db in databases:
        if db.name.startswith("A"):
            db_procedures = [
                sp.name for sp in procedures if sp.db_id == db.id
            ]
            result.append((db.name, db_procedures))
    return result
```

Файл test_queries.py:

```
import unittest
from classes import StoredProcedure, Database, StoredProcedureDatabase
from queries import find_procedures_ending_with_ov, average_salary_by_database,
databases_starting_with_a

class TestQueries(unittest.TestCase):
```

```
    def setUp(self):
        self.databases = [
            Database(1, "Административная"),
            Database(2, "Финансовая"),
            Database(3, "Отделов")
        ]
        self.procedures = [
            StoredProcedure(1, "Процедураов", 1),  # 10 символов
            StoredProcedure(2, "Процедура", 2),  # 9 символов
            StoredProcedure(3, "Функцияов", 1),  # 11 символов
            StoredProcedure(4, "Функция", 3),  # 7 символов
        ]
        self.sp_dbs = [
            StoredProcedureDatabase(1, 1),
            StoredProcedureDatabase(2, 2),
            StoredProcedureDatabase(3, 1),
            StoredProcedureDatabase(4, 3),
        ]

    def test_find_procedures_ending_with_ov(self):
        result = find_procedures_ending_with_ov(self.procedures, self.databases)
        expected = [("Процедураов", "Административная"), ("Функцияов", "Административная")]
        self.assertEqual(result, expected)

    def test_average_salary_by_database(self):
        result = average_salary_by_database(self.procedures, self.databases)
        expected = [
            ("Отделов", 7.0),
            ("Финансовая", 9.0),
            ("Административная", 10.5)
        ]
        self.assertEqual(result, expected)

    def test_databases_starting_with_a(self):
        result = databases_starting_with_a(self.procedures, self.databases, self.sp_dbs)
        expected = [("Административная", ["Процедураов", "Функцияов"])]
        self.assertEqual(result, expected)

if __name__ == '__main__':
    unittest.main()
```

3.Результаты работы программы

```
"C:\Program Files\Python313\python.exe" "C:/Program Files/JetBrains/PyCharm 2025.2.1.1/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py" --path "C:\Users\Денис\Desktop\Отчёт РК2\test_queries.py"
Testing started at 8:47 ...
Launching unittests with arguments python -m unittest C:\Users\Денис\Desktop\Отчёт РК2\test_queries.py in C:\Users\Денис\Desktop\Отчёт РК2



[('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.5)] != [('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.0)]

Expected :[('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.0)]
Actual   :[('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.5)]
<Click to see difference>

Traceback (most recent call last):
  File "C:\Users\Денис\Desktop\Отчёт РК2\test_queries.py", line 37, in test_average_salary_by_database
    self.assertEqual(result, expected)
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^
AssertionError: Lists differ: [('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.0)] != [('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.5)]

First differing element 2:
('Административная', 10.0)
('Административная', 10.5)

- [('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.0)]
?                                                          ^

+ [('Отделов', 7.0), ('Финансовая', 9.0), ('Административная', 10.5)]
?                                                          ^



Ran 3 tests in 0.030s

FAILED (failures=1)

Process finished with exit code 1
```