Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по РК №2
Вариант запросов Б
Вариант предметной области 34

Выполнил:
студент группы ИУ5-33Б
Маргорина Ирина

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Москва, 2025 г.

Листинг кода

main.py

```
from dataclasses import dataclass
from operator import itemgetter
from typing import List, Tuple, Dict


@dataclass
class StoredProcedure:
    id: int
    name: str
    execution_time: int
    db_id: int


@dataclass
class Database:
    id: int
    name: str


@dataclass
class ProcedureDatabase:
    db_id: int
    proc_id: int


def create_sample_data() -> Tuple[List[Database], List[StoredProcedure],
List[ProcedureDatabase]]:
    databases = [
        Database(1, 'Основная база'),
        Database(2, 'Архивная база'),
        Database(3, 'База отчетов'),
        Database(4, 'Тестовая база'),
    ]

    procedures = [
        StoredProcedure(1, 'расчет_отчета', 150, 1),
        StoredProcedure(2, 'резервное_копирование', 300, 2),
        StoredProcedure(3, 'генерация_статистики', 200, 3),
        StoredProcedure(4, 'очистка_данных', 250, 3),
        StoredProcedure(5, 'проверка_ввода', 100, 3),
        StoredProcedure(6, 'формирование_отчетов', 180, 1),
    ]

    procedures_databases = [
        ProcedureDatabase(1, 1),
        ProcedureDatabase(2, 2),
        ProcedureDatabase(3, 3),
        ProcedureDatabase(3, 4),
        ProcedureDatabase(3, 5),
        ProcedureDatabase(4, 1),
        ProcedureDatabase(4, 3),
        ProcedureDatabase(1, 6),
        ProcedureDatabase(3, 6),
    ]
```

```
    return databases, procedures, procedures_databases


def get_one_to_many(databases: List[Database],
                    procedures: List[StoredProcedure]) -> List[Tuple]:
    return [(p.name, p.execution_time, d.name)
            for d in databases
            for p in procedures
            if p.db_id == d.id]


def get_many_to_many(databases: List[Database],
                     procedures: List[StoredProcedure],
                     relations: List[ProcedureDatabase]) -> List[Tuple]:
    many_to_many_temp = [(d.name, pd.db_id, pd.proc_id)
                         for d in databases
                         for pd in relations
                         if d.id == pd.db_id]

    return [(p.name, p.execution_time, db_name)
            for db_name, db_id, proc_id in many_to_many_temp
            for p in procedures if p.id == proc_id]


def task1_sorted_procedures(databases: List[Database],
                            procedures: List[StoredProcedure]) -> List[Tuple]:
    one_to_many = get_one_to_many(databases, procedures)
    return sorted(one_to_many, key=itemgetter(0))


def task2_database_procedure_counts(databases: List[Database],
                                    procedures: List[StoredProcedure]) ->
List[Tuple]:
    result = []
    for d in databases:
        # Подсчитываем процедуры для данной БД
        procedure_count = sum(1 for proc in procedures if proc.db_id == d.id)
        if procedure_count > 0:
            result.append((d.name, procedure_count))

    # Сортируем по количеству процедур (возрастание)
    return sorted(result, key=itemgetter(1))


def task3_procedures_ending_with_ov(databases: List[Database],
                                    procedures: List[StoredProcedure],
                                    relations: List[ProcedureDatabase]) ->
Dict[str, List[str]]:
    many_to_many = get_many_to_many(databases, procedures, relations)

    result = {}
    for proc_name, exec_time, db_name in many_to_many:
        if proc_name.endswith('ов'):
            if proc_name not in result:
                result[proc_name] = []
            # Добавляем БД, если её ещё нет в списке
            if db_name not in result[proc_name]:
                result[proc_name].append(db_name)
```

```
    return result


def print_results(databases: List[Database],
                  procedures: List[StoredProcedure],
                  relations: List[ProcedureDatabase]) -> None:
    print("ЗАДАНИЕ 1: Все процедуры, отсортированные по имени")
    res1 = task1_sorted_procedures(databases, procedures)
    print(f"{'Процедура':<25} {'Время (ms)':<12} {'База данных':<20}")
    for proc_name, exec_time, db_name in res1:
        print(f"{proc_name:<25} {exec_time:<12} {db_name:<20}")

    print("ЗАДАНИЕ 2: Количество процедур в каждой БД")
    res2 = task2_database_procedure_counts(databases, procedures)
    print(f"{'База данных':<25} {'Кол-во процедур':<15}")
    for db_name, count in res2:
        print(f"{db_name:<25} {count:<15}")

    print("ЗАДАНИЕ 3: Процедуры, оканчивающиеся на 'ов' и БД, где они работают")
    res3 = task3_procedures_ending_with_ov(databases, procedures, relations)
    if res3:
        for proc_name, db_list in res3.items():
            print(f"\nПроцедура: {proc_name}")
            print(f"Работает в БД: {', '.join(db_list)}")
    else:
        print("Нет процедур, оканчивающихся на 'ов'")


def main() -> None:
    # Получаем тестовые данные
    databases, procedures, relations = create_sample_data()

    # Выводим результаты
    print_results(databases, procedures, relations)


if __name__ == "__main__":
    main()
```

test_main.py

```
import unittest
from main import (
    Database,
    StoredProcedure,
    ProcedureDatabase,
    create_sample_data,
    task1_sorted_procedures,
    task2_database_procedure_counts,
    task3_procedures_ending_with_ov
)


class TestTask1(unittest.TestCase):
    """Тесты для Задания 1: Отсортированные процедуры"""

    def test_sorted_procedures_with_sample_data(self):
        """Тест 1: Проверка сортировки процедур на тестовых данных"""
        # Получаем тестовые данные
```

```
        databases, procedures, _ = create_sample_data()

        # Выполняем задание 1
        result = task1_sorted_procedures(databases, procedures)

        # Проверяем, что результат не пустой
        self.assertGreater(len(result), 0, "Результат не должен быть пустым")

        # Проверяем количество элементов
        self.assertEqual(len(result), 6, "Должно быть 6 процедур")

        # Проверяем сортировку по алфавиту (по имени процедуры)
        procedure_names = [item[0] for item in result]
        sorted_names = sorted(procedure_names)
        self.assertEqual(procedure_names, sorted_names,
                         "Процедуры должны быть отсортированы по алфавиту")

        # Проверяем структуру данных
        for proc_name, exec_time, db_name in result:
            self.assertIsInstance(proc_name, str, "Имя процедуры должно быть
строкой")
            self.assertIsInstance(exec_time, int, "Время выполнения должно быть
числом")
            self.assertIsInstance(db_name, str, "Имя БД должно быть строкой")
            self.assertGreater(exec_time, 0, "Время выполнения должно быть
положительным")

    def test_sorted_procedures_empty_data(self):
        """Тест: Проверка с пустыми данными"""
        databases = []
        procedures = []

        result = task1_sorted_procedures(databases, procedures)

        self.assertEqual(result, [], "При пустых данных должен быть пустой
список")


class TestTask2(unittest.TestCase):
    """Тесты для Задания 2: Количество процедур в БД"""

    def test_database_procedure_counts_with_sample_data(self):
        """Тест 2: Проверка подсчета процедур в БД на тестовых данных"""
        databases, procedures, _ = create_sample_data()

        result = task2_database_procedure_counts(databases, procedures)

        # Проверяем количество БД с процедурами
        self.assertEqual(len(result), 3, "Должно быть 3 БД с процедурами")

        # Проверяем конкретные значения (сортируем для сравнения)
        expected_results = [
            ('Архивная база', 1),
            ('Основная база', 2),
            ('База отчетов', 3)
        ]

        # Сортируем expected_results по количеству процедур
        expected_sorted = sorted(expected_results, key=lambda x: x[1])
```

```
        # Проверяем каждую пару
        for (expected_db, expected_count), (actual_db, actual_count) in
zip(expected_sorted, result):
            self.assertEqual(expected_db, actual_db,
                             f"Ожидалась БД '{expected_db}', получена
'{actual_db}'")
            self.assertEqual(expected_count, actual_count,
                             f"Для БД '{actual_db}' ожидалось {expected_count}
процедур, получено {actual_count}")

        # Проверяем сортировку по количеству процедур (возрастание)
        counts = [item[1] for item in result]
        self.assertEqual(counts, sorted(counts),
                         "Результат должен быть отсортирован по количеству
процедур")

    def test_single_database_with_multiple_procedures(self):
        """Тест: Одна БД с несколькими процедурами"""
        databases = [Database(1, 'Тестовая БД')]
        procedures = [
            StoredProcedure(1, 'процедура1', 100, 1),
            StoredProcedure(2, 'процедура2', 200, 1),
            StoredProcedure(3, 'процедура3', 300, 1),
        ]

        result = task2_database_procedure_counts(databases, procedures)

        self.assertEqual(len(result), 1, "Должна быть одна БД")
        self.assertEqual(result[0][0], 'Тестовая БД', "Название БД должно
совпадать")
        self.assertEqual(result[0][1], 3, "Должно быть 3 процедуры")


class TestTask3(unittest.TestCase):
    """Тесты для Задания 3: Процедуры, оканчивающиеся на 'ов'"""

    def test_procedures_ending_with_ov_with_sample_data(self):
        """Тест 3: Проверка поиска процедур, оканчивающихся на 'ов'"""
        databases, procedures, relations = create_sample_data()

        result = task3_procedures_ending_with_ov(databases, procedures,
relations)

        # Проверяем, что найдена хотя бы одна процедура
        self.assertGreater(len(result), 0, "Должна быть найдена хотя бы одна
процедура")

        # Проверяем конкретную процедуру
        self.assertIn('формирование_отчетов', result,
                      "Должна быть найдена процедура 'формирование_отчетов'")

        # Проверяем, что процедуры действительно оканчиваются на 'ов'
        for proc_name in result.keys():
            self.assertTrue(proc_name.endswith('ов'),
                            f"Процедура '{proc_name}' должна оканчиваться на 'ов'")

        # Проверяем БД для процедуры 'формирование_отчетов'
        expected_dbs_for_reporting = ['Основная база', 'База отчетов']
```

```
        actual_dbs = result['формирование_отчетов']

        # Проверяем количество БД
        self.assertEqual(len(actual_dbs), 2,
                         f"Процедура 'формирование_отчетов' должна работать в 2
БД")

        # Проверяем конкретные БД (порядок не важен)
        for db_name in expected_dbs_for_reporting:
            self.assertIn(db_name, actual_dbs,
                          f"БД '{db_name}' должна быть в списке для процедуры
'формирование_отчетов'")

    def test_no_procedures_ending_with_ov(self):
        """Тест: Нет процедур, оканчивающихся на 'ов'"""
        databases = [Database(1, 'БД1'), Database(2, 'БД2')]
        procedures = [
            StoredProcedure(1, 'процедура_тест', 100, 1),
            StoredProcedure(2, 'тестовая', 200, 2),
        ]
        relations = [
            ProcedureDatabase(1, 1),
            ProcedureDatabase(2, 2),
        ]

        result = task3_procedures_ending_with_ov(databases, procedures,
relations)

        self.assertEqual(result, {}, "При отсутствии процедур на 'ов' должен быть
пустой словарь")


class TestIntegration(unittest.TestCase):
    """Интеграционные тесты"""

    def test_all_tasks_with_consistent_data(self):
        """Тест: Проверка согласованности данных между заданиями"""
        databases, procedures, relations = create_sample_data()

        # Выполняем все задания
        result1 = task1_sorted_procedures(databases, procedures)
        result2 = task2_database_procedure_counts(databases, procedures)
        result3 = task3_procedures_ending_with_ov(databases, procedures,
relations)

        # Проверяем, что все процедуры из task3 есть в task1
        procedures_from_task1 = {item[0] for item in result1}
        for proc_name in result3.keys():
            self.assertIn(proc_name, procedures_from_task1,
                          f"Процедура '{proc_name}' из task3 должна быть в task1")

        # Проверяем, что БД из task2 существуют
        db_names_from_task2 = {item[0] for item in result2}
        all_db_names = {db.name for db in databases}
        for db_name in db_names_from_task2:
            self.assertIn(db_name, all_db_names,
                          f"БД '{db_name}' из task2 должна существовать")
```

```
def run_tests():
    """Запуск всех тестов"""
    # Создаем test suite
    loader = unittest.TestLoader()

    # Добавляем все тестовые классы
    suite = unittest.TestSuite()
    suite.addTest(loader.loadTestsFromTestCase(TestTask1))
    suite.addTest(loader.loadTestsFromTestCase(TestTask2))
    suite.addTest(loader.loadTestsFromTestCase(TestTask3))
    suite.addTest(loader.loadTestsFromTestCase(TestIntegration))

    # Запускаем тесты
    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)

    # Выводим статистику
    print("РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ")
    print(f"Всего тестов: {result.testsRun}")
    print(f"Успешно: {result.testsRun - len(result.failures) -
len(result.errors)}")
    print(f"Провалено: {len(result.failures)}")
    print(f"Ошибок: {len(result.errors)}")

    return result


if __name__ == "__main__":
    # Запускаем все тесты
    test_result = run_tests()

    # Возвращаем код выхода: 0 если все тесты прошли, 1 если есть ошибки
    exit_code = 0 if test_result.wasSuccessful() else 1
    exit(exit_code)
```

Скриншот работы приложения: