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: А
Вариант предметной области: 33

Выполнил:
студент группы ИУ5-32Б
Грачев К. А.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Москва, 2025 г.

Вариант запросов А. Предметная область 33.

1. «Таблица данных» и «Строка данных» связаны соотношением один-ко-многим. Выведите список всех связанных строк и таблиц, отсортированный по именам таблиц, сортировка по сотрудникам произвольная.
2. «Таблица данных» и «Строка данных» связаны соотношением один-ко-многим. Выведите список таблиц с суммой числового поля в каждой таблице, отсортированный по данной сумме.
3. «Таблица данных» и «Строка данных» связаны соотношением многие-ко-многим. Выведите список всех таблиц, у которых в названии присутствует символ «С», и содержимое текстовых полей в их строках.

Листинг программы.
Файл main.py

```
from operator import itemgetter


class Row:
    """Строка данных"""
    def __init__(self, id, digit_field, string_field, table_id):
        self.id = id
        self.digit_field = digit_field
        self.string_field = string_field
        self.table_id = table_id


class Table:
    """Таблица данных"""
    def __init__(self, id, name):
        self.id = id
        self.name = name


class TableRow:
    """
    "Строки таблицы" для реализации
    связи многие-ко-многим
    """
    def __init__(self, row_id, table_id):
        self.row_id = row_id
        self.table_id = table_id


tables = [
    Table(1, 'Сотрудники'),
    Table(2, 'Отделы'),
```

```
    Table(3, 'Студенты'),
    Table(4, 'Empty table')
]

rows = [
    Row(1, 100000, 'Иванов', 1),
    Row(2, 120000, 'Петров', 1),
    Row(3, 123, 'Отдел кадров', 2),
    Row(4, 1234, 'Бухгалтерия', 2),
    Row(5, 4000, 'Баранкин', 3),
    Row(6, 7000, 'Бубликов', 3)
]

table_rows = [
    TableRow(1, 1),
    TableRow(2, 1),
    TableRow(3, 2),
    TableRow(4, 2),
    TableRow(5, 3),
    TableRow(6, 3),
    TableRow(1, 3),
    TableRow(2, 3)
]


def task_1(one_to_many):
    "Задание А1"
    return sorted(one_to_many, key=itemgetter(0))


def task_2(one_to_many):
    "Задание А2"
    result = {}
    for table, salary, _ in one_to_many: # Перебираем всех сотрудников
        if table in result.keys(): # Проверяем, есть ли запись о соответсвующем сотруднику отделе
            result[table] += salary # Увеличиваем сумму зарплат
        else:
            result[table] = salary # Добавляем запись об отделе
    return sorted(result.items(), key=lambda item: item[1], reverse=True) # Возвращаем,
отсортированный по сумме зарплат, результат


def task_3(many_to_many):
    "Задание А3"
    result = {}
    for table in tables:
        if 'С' in table.name:
            # Добавляем в словарь список имен сотрудников отдела
            # ключ - отдел, значение - список фамилий
            result[table.name] = [item[2] for item in filter(lambda x: x[0] == table.name, many_to_many)]
    return result
```

```
def create_one_many_connection(tables, rows):
    result = [(table.name, row.digit_field, row.string_field)
              for table in tables
              for row in rows
              if table.id == row.table_id
              ]
    return result


def create_many_many_connection(table_rows, tables, rows):
    result = [(table.name, row.digit_field, row.string_field)
              for table_row in table_rows
              for table in tables if table.id == table_row.table_id
              for row in rows if row.id == table_row.row_id
              ]
    return result


def main():
    """Основная функция"""

    # Соединение данных один-ко-многим
    one_to_many = create_one_many_connection(tables, rows)

    # Соединение данных многие-ко-многим
    many_to_many = create_many_many_connection(table_rows, tables, rows)

    print('Задание А1')
    print(task_1(one_to_many))

    print('\nЗадание А2')
    print(task_2(one_to_many))

    print('\nЗадание А3')
    print(task_3(many_to_many))


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

Файл test.py

```
import unittest

from main import Row, Table, TableRow, task_1, task_2, task_3, create_many_many_connection,
create_one_many_connection


class Test(unittest.TestCase):
    def setUp(self):
        self.tables = [
            Table(1, 'Сотрудники'),
            Table(2, 'Отделы'),
```

```
            Table(3, 'Студенты'),
            Table(4, 'Empty table')
        ]

        self.rows = [
            Row(1, 100000, 'Иванов', 1),
            Row(2, 120000, 'Петров', 1),
            Row(3, 123, 'Отдел кадров', 2),
            Row(4, 1234, 'Бухгалтерия', 2),
            Row(5, 4000, 'Баранкин', 3),
            Row(6, 7000, 'Бубликов', 3)
        ]

        self.table_rows = [
            TableRow(1, 1),
            TableRow(2, 1),
            TableRow(3, 2),
            TableRow(4, 2),
            TableRow(5, 3),
            TableRow(6, 3),
            TableRow(1, 3),
            TableRow(2, 3)
        ]

        self.one_to_many = create_one_many_connection(self.tables, self.rows)
        self.many_to_many = create_many_many_connection(self.table_rows, self.tables, self.rows)

        return super().setUp()

    def test_task_1(self):
        """"""
        result = task_1(self.one_to_many)
        correct = ['Отделы', 'Отделы', 'Сотрудники', 'Сотрудники', 'Студенты', 'Студенты']

        for i in range(len(correct)):
            self.assertEqual(result[i][0], correct[i]) # Проверяем, что результат отсортирован по названию

    def test_task_2(self):
        """"""
        result = task_2(self.one_to_many)
        for i in range(len(result) - 1):
            self.assertFalse(result[i][1] < result[i + 1][1]) # Проверяем сортировку по убыванию

        correct = [220000, 11000, 1357]
        for i in range(len(result)):
            self.assertEqual(result[i][1], correct[i]) # Проверяем сортировку по убыванию

    def test_task_3(self):
        """"""
        result = task_3(self.many_to_many)

        # Проверяем фильтр
```

```
        correct = ['Сотрудники', 'Студенты']
        wrong = ['Отделы', 'Empty table']
        for i in range(len(correct)):
            self.assertIn(correct[i], result.keys())
        for i in range(len(wrong)):
            self.assertNotIn(wrong[i], result.keys())

        # Проверяем содержимое
        correct = {'Сотрудники': ['Иванов', 'Петров'], 'Студенты': ['Баранкин', 'Бубликов', 'Иванов',
'Петров']}
        for key, value in correct.items():
            self.assertEqual(len(value), len(result[key]))
            for item in value:
                self.assertIn(item, result[key])


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

Результат выполнения.