

**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 рубежному контролю №1

Вариант 23А: «Язык программирования» и «Синтаксическая конструкция»

Выполнил:

ФИО: Цыпышев Т. А.

Группа: ИУ5-31Б

Дата: 29.10.23

Подпись:

Принял:

ФИО: Гапанюк Ю. Е.

Должность: Преподаватель

Дата:

Подпись:

Постановка задачи

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-ко-многим.

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.

2. (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 А.

1. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список всех связанных сотрудников и отделов, отсортированный по отделам, сортировка по сотрудникам произвольная.
2. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список отделов с суммарной зарплатой сотрудников в каждом отделе, отсортированный по суммарной зарплате.

3. «Отдел» и «Сотрудник» связаны соотношением многие-ко-многим. Выведите список всех отделов, у которых в названии присутствует слово «отдел», и список работающих в них сотрудников.

Текст программы

main.cpp

```
from collections import defaultdict

# Класс «Язык программирования»
class ProgrammingLanguage:
    def __init__(self, language_id, name):
        self.language_id = language_id
        self.name = name

# Класс «Синтаксическая конструкция»
class SyntaxConstruction:
    def __init__(self, construction_id, construction_type, complexity, language_id):
        self.construction_id = construction_id
        self.construction_type = construction_type
        self.complexity = complexity # Сложность синтаксической конструкции
        self.language_id = language_id

# Класс реализация связи многие ео многим
class LanguageSyntaxMap:
    def __init__(self, language_id, construction_id):
        self.language_id = language_id
        self.construction_id = construction_id

# Генерация тестовых данных
def get_test_data():
    python = ProgrammingLanguage(1, "Python")
    c_sharp = ProgrammingLanguage(2, "C#")
    cpp = ProgrammingLanguage(3, "C++")
    go = ProgrammingLanguage(4, "GoLang")

    constructions = [
        SyntaxConstruction(1, "Loop", 2, 1),
        SyntaxConstruction(2, "Condition", 3, 1),
        SyntaxConstruction(3, "Function", 4, 2),
        SyntaxConstruction(4, "Loop", 2, 2),
        SyntaxConstruction(5, "Condition", 3, 3),
        SyntaxConstruction(6, "Function", 4, 3),
        SyntaxConstruction(7, "Loop", 2, 4),
        SyntaxConstruction(8, "Condition", 3, 4),
        SyntaxConstruction(9, "Function", 4, 4),
    ]

    language_syntax_map = [
        LanguageSyntaxMap(1, 1),
        LanguageSyntaxMap(1, 2),
        LanguageSyntaxMap(2, 3),
        LanguageSyntaxMap(2, 4),
        LanguageSyntaxMap(3, 5),
```

```

        LanguageSyntaxMap(3, 6),
        LanguageSyntaxMap(4, 7),
        LanguageSyntaxMap(4, 8),
        LanguageSyntaxMap(4, 9),
    ]

    return [python, c_sharp, cpp, go], constructions, language_syntax_map

# Запрос 1: Список связанных синтаксических конструкций и языков программирования,
отсортированный по языкам
def get_languages_with_constructions(languages, constructions):
    language_constructions = defaultdict(list)
    for construction in constructions:

language_constructions[construction.language_id].append(construction.construction_type)

    result = [(language.name, language_constructions[language.language_id]) for language
in languages if
        language.language_id in language_constructions]
    result.sort(key=lambda x: x[0]) # Сортировка по языкам
    return result

# Запрос 2: Список языков программирования с ключевым словом и связанных с ними
синтаксических конструкций
def get_avg_complexity_by_language(languages, constructions):
    language_complexity = defaultdict(int)
    language_count = defaultdict(int)

    for construction in constructions:
        language_id = construction.language_id
        language_complexity[language_id] += construction.complexity
        language_count[language_id] += 1

    result = [(language.name, language_complexity[language.language_id] /
language_count[language.language_id]) for
        language in languages if language.language_id in language_complexity]
    result.sort(key=lambda x: x[1]) # Сортировка по средней сложности
    return result

# Запрос 3: Вывести список всех языков программирования, у которых в названии
присутствует ключевое слово,
# и список связанных с ними синтаксических конструкций.
def get_languages_with_related_constructions(languages, constructions, keyword):
    result = [(language.name, [construction.construction_type for construction in
constructions if
        construction.language_id == language.language_id]) for
language in languages if
        keyword.lower() in language.name.lower()]
    return result

def main():
    # Инициализируем данные
    languages, constructions, language_syntax_map = get_test_data()

    # Выполнение запросов
    print("Запрос 1: Список связанных синтаксических конструкций и языков
программирования, отсортированный по языкам.")
    result1 = get_languages_with_constructions(languages, constructions)
    for language, constructions_list in result1:
        print(f"\t{language}: {' '.join(constructions_list)}")

```

```

print("\nЗапрос 2: Список языков программирования с средней сложностью синтаксических
конструкций.")
result2 = get_avg_complexity_by_language(languages, constructions)
for language, avg_complexity in result2:
    print(f"\t{language}: {avg_complexity:.2f}")

print(
    "\nЗапрос 3: Список языков программирования, содержащих в своём названии ключевое
слово, "
    "и связанных с ними синтаксических конструкций.")
keyword = "Lang"
result3 = get_languages_with_related_constructions(languages, constructions, keyword)
for language, constructions_list in result3:
    print(f"\t{language}: {' '.join(constructions_list)}")

if __name__ == '__main__':
    main()

```

Результат выполнения

Запрос 1: Список связанных синтаксических конструкций и языков программирования, отсортированный по языкам.

```

C#: Function, Loop
C++: Condition, Function
GoLang: Loop, Condition, Function
Python: Loop, Condition

```

Запрос 2: Список языков программирования с средней сложностью синтаксических конструкций.

```

Python: 2.50
C#: 3.00
GoLang: 3.00
C++: 3.50

```

Запрос 3: Список языков программирования, содержащих в своём названии ключевое слово, и связанных с ними синтаксических конструкций.

```

GoLang: Loop, Condition, Function

```

Process finished with exit code 0