Московский государственный технический университет им. Н.Э. Баумана

Отчет
Рубежный контроль 1
Вариант-15

ИСПОЛНИТЕЛЬ:

Группа ИБМ3-34Б
Михайлова В.И.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

2025 г.

1. Создание классов данных

Были созданы три класса на языке Python:

- **File**: Класс, представляющий отдельный файл с его характеристиками: id (идентификатор), name (имя), size (размер в байтах - количественный признак) и catalog_id (идентификатор каталога, для связи один-ко-многим).
- **Catalog**: Класс, представляющий каталог файлов с его идентификатором (id) и наименованием (name).
- **FileCatalog**: Класс для реализации связи многие-ко-многим, содержащий пары идентификаторов catalog_id и file_id.

2. Инициализация тестовых данных

Созданы списки объектов для каждого из классов (files, catalogs, files_catalogs), обеспечивающие логическую связь между записями через идентификаторы.

3. Реализация запросов (Вариант А)

Для реализации запросов я использовала функциональные возможности Python, такие как: генераторы списков (list comprehension), анонимные функции (lambda), функции высшего порядка (filter, sum) и модуль operator (itemgetter) для сортировки.

- **А1 (Связь 1:М, Сортировка):** выполнено прямое соединение списков файлов и каталогов (по принципу один-ко-многим) с последующей сортировкой по имени каталога.
- **А2 (Связь 1:М, Агрегация):** для каждого каталога были отфильтрованы соответствующие файлы. Вычислен суммарный размер файлов (агрегация количественного признака) для каждого каталога. Результат отсортирован по убыванию суммарного размера.
- **А3 (Связь М:М, Фильтрация по имени):** сначала выполнено соединение «многие-ко-многим» для получения полной картины. Затем осуществлена фильтрация каталогов по наличию слова «каталог» в названии. Для каждого отобранного каталога сформирован список имен связанных с ним файлов.

Код:

```
# Импортируем itemgetter для удобной сортировки

from operator import itemgetter

class File:

    """Файл (аналог Сотрудника)"""

    def __init__(self, id, name, size, catalog_id):

        self.id = id          # ID файла
```

```
        self.name = name       # Имя файла
        self.size = size       # Размер в байтах - это количественный признак
        self.catalog_id = catalog_id # ID каталога (для связи 1:M)
# Класс 2: Каталог (сторона "один")
class Catalog:
    """Каталог файлов (аналог Отдела)"""
    def __init__(self, id, name):
        self.id = id           # ID каталога
        self.name = name       # Наименование каталога
# Класс для связи многие-ко-многим
class FileCatalog:
    """Связь файлов и каталогов (M:M)"""
    def __init__(self, catalog_id, file_id):
        self.catalog_id = catalog_id
        self.file_id = file_id
# Тестовые данные
catalogs = [
    Catalog(1, 'Системный каталог'),
    Catalog(2, 'Рабочий каталог проекта'),
    Catalog(3, 'Архив документов'),
    Catalog(11, 'Каталог (старый)'),
    Catalog(22, 'Рабочий каталог (тестовый)'),
    Catalog(33, 'Архив (временный)'),
]
files = [
    File(1, 'boot.ini', 500, 1),
```

```
    File(2, 'main.py', 12000, 2),
    File(3, 'report_Q1.docx', 80000, 3),
    File(4, 'config.sys', 1500, 3),
    File(5, 'temp.log', 200, 3),
]
files_catalogs = [
    FileCatalog(1, 1),
    FileCatalog(2, 2),
    FileCatalog(3, 3),
    FileCatalog(3, 4),
    FileCatalog(3, 5),
    FileCatalog(11, 1), # Связь M:M
    FileCatalog(22, 2), # Связь M:M
    FileCatalog(33, 3), # Связь M:M
    FileCatalog(33, 4), # Связь M:M
    FileCatalog(33, 5), # Связь M:M
]
def main():
    """Основная функция, реализующая запросы"""
    # 1. Соединение данных один-ко-многим: (Имя файла, Размер, Имя каталога)
    one_to_many = [(f.name, f.size, c.name)
                   for c in catalogs
                   for f in files
                   if f.catalog_id == c.id]
    # 2. Соединение данных многие-ко-многим: (Имя файла, Размер, Имя каталога)
    # Шаг 1: (Имя каталога, ID файла)
```

```
    many_to_many_temp = [(c.name, fcd.file_id)
                         for c in catalogs
                         for fcd in files_catalogs
                         if c.id == fcd.catalog_id]
    # Шаг 2: (Имя файла, Размер, Имя каталога)
    many_to_many = [(f.name, f.size, cat_name)
                    for cat_name, file_id in many_to_many_temp
                    for f in files if f.id == file_id]
    # Запрос A1
    # Список всех связанных файлов и каталогов, отсортированный по каталогам (1:M)
    res_a1 = sorted(one_to_many, key=itemgetter(2))
    # Запрос A2
    # Список каталогов с суммарным размером файлов, отсортированный по сумме
(1:M)
    res_a2_unsorted = []
    for c in catalogs:
        # Фильтруем файлы для текущего каталога
        c_files = list(filter(lambda i: i[2] == c.name, one_to_many))
        if len(c_files) > 0:
            # Вычисляем суммарный размер файлов (индекс 1 в кортеже)
            c_sizes_sum = sum([size for _, size, _ in c_files])
            res_a2_unsorted.append((c.name, c_sizes_sum))

    # Сортировка по суммарному размеру (по убыванию)
    res_a2 = sorted(res_a2_unsorted, key=itemgetter(1), reverse=True)
    # --- Запрос A3 ---
```

```
    # Список каталогов, содержащих слово "каталог", и список их файлов (M:M)
    res_a3 = {}
    for c in catalogs:
        # Фильтрация по наличию слова "каталог" в названии
        if 'каталог' in c.name.lower():
            # Фильтруем связанные записи по имени каталога
            c_files = list(filter(lambda i: i[2] == c.name, many_to_many))
            # Составляем список имен файлов (индекс 0)
            c_file_names = [name for name, _, _ in c_files]
            res_a3[c.name] = c_file_names
    print('*** РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЗАПРОСОВ (ВАРИАНТ А) ***')
    print('\n[ЗАПРОС А1] Список файлов, отсортированный по каталогам (связь 1:M):')
    print(res_a1)
    print('\n[ЗАПРОС А2] Каталоги и суммарный размер их файлов (связь 1:M,
сортировка по размеру):')
    print(res_a2)
    print('\n[ЗАПРОС А3] Каталоги, содержащие слово "каталог", и список их файлов
(связь M:M):')
    print(res_a3)
if __name__ == '__main__':
    main()
```

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
*** РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЗАПРОСОВ (ВАРИАНТ А) ***

[ЗАПРОС А1] Список файлов, отсортированный по каталогам (связь 1:М):
[('report_Q1.docx', 80000, 'Архив документов'), ('config.sys', 1500, 'Архив документов'), ('temp.log', 200,

[ЗАПРОС А2] Каталоги и суммарный размер их файлов (связь 1:М, сортировка по размеру):
[('Архив документов', 81700), ('Рабочий каталог проекта', 12000), ('Системный каталог', 500)]

[ЗАПРОС А3] Каталоги, содержащие слово "каталог", и список их файлов (связь М:М):
{'Системный каталог': ['boot.ini'], 'Рабочий каталог проекта': ['main.py'], 'Каталог (старый)': ['boot.ini'

Process finished with exit code 0
```

Продолжение вывода кода справа (что не влезло)

```
тов'), ('temp.log', 200, 'Архив документов'), ('main.py', 12000, 'Рабочий каталог проекта'), ('boot.ini', 500, 'Системный каталог')]

):
ог', 500)]


г (старый)': ['boot.ini'], 'Рабочий каталог (тестовый)': ['main.py']}
```

Вывод:

1. Эффективность связей и сортировки (запрос А1)

Запрос А1, использующий связь **один-ко-многим**, подтвердил, что сортировка по классу «Каталог» является эффективным способом логической группировки данных. Файлы (boot.ini, main.py, report_Q1.docx, config.sys, temp.log) были сгруппированы под своими основными каталогами (Системный каталог, Рабочий каталог проекта, Архив документов), что обеспечило быстрый обзор содержимого каждого каталога.

2. Доминирование «Архива документов» по объему (запрос А2)

Запрос А2, использующий **агрегацию** (суммарный размер файлов), выявил, что:

- «Архив документов» (и его копия «Архив (временный)») является наиболее объемным каталогом, содержащим 81700 байт данных. Это указывает на то, что самые крупные или многочисленные файлы в системе хранятся именно в архивных папках.
- «Рабочий каталог проекта» занимает второе место (12000 байт), а «Системный каталог» и «Каталог (старый)» — наименьший объем (500 байт), что логично для системных и конфигурационных файлов, которые обычно небольшие.

Этот результат позволяет сделать вывод о **распределении нагрузки по хранилищу** и потенциально выявить области, требующие оптимизации или более частого резервного копирования (в данном случае, архивные каталоги).

3. Дублирование и системные файлы в каталогах (запрос А3)

Запрос А3, использующий связь **многие ко многим** и фильтрацию по имени, показал:

- Системный файл boot.ini содержится как в «Системный каталог», так и в «Каталог (старый)». Это является примером дублирования (то есть использования файла в разных контекстах), это стало возможным благодаря связи многие ко многим.
- Рабочий файл main.py также дублируется в «Рабочий каталог проекта» и «Рабочий каталог (тестовый)».

Результаты этого запроса показывают, что структура М:М успешно моделирует ситуации, когда один и тот же файл может быть частью нескольких различных каталогов. Это является важным аспектом управления файловой системой.

Общий вывод: разработанная модель данных эффективно обрабатывает основные типы связей (один-ко-многим, многие-ко-многим) и реализует аналитические операции, такие как сортировка, агрегация (то есть суммирование размера) и фильтрация, что позволяет нам делать выводы о структуре и распределении файлов.