Факультет программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа №1

Преподаватель: Кореньков Юрий Дмитриевич
Выполнил: Ле Чонг Дат
Группа: P33302

2023 г.

Описание заданий

Задание 1

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида.

Порядок выполнения:

1. Спроектировать структуры данных для представления информации в оперативной памяти
    a. Для порции данных, состоящий из элементов определённого рода (см форму данных), поддержать тривиальные значения по меньшей мере следующих типов: цетырёхбайтовые целые числа и числа с плавающей точкой, текстовые строки произвольной длины, булевские значения
    b. Для информации о запросе
2. Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:
    a. Операции над схемой данных (создание и удаление элементов схемы)
    b. Базовые операции над элементами данных в соответствии с текущим состоянием схемы (над узлами или записями заданного вида)
        i. Вставка элемента данных
        ii. Перечисление элементов данных
        iii. Обновление элемента данных
        iv. Удаление элемента данных
3. Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:
    a. Добавление, удаление и получение информации о элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей
    b. Добавление нового элемента данных определённого вида
    c. Выборка набора элементов данных с учётом заданных условий и отношений со смежными элементами данных (по свойствам/полями/атрибутам и логическим связям соответственно)
    d. Обновление элементов данных, соответствующих заданным условиям
    e. Удаление элементов данных, соответствующих заданным условиям
4. Реализовать тестовую программу для демонстрации работоспособности решения
    a. Параметры для всех операций задаются посредством формирования соответствующих структур данных
    b. Показать, что при выполнении операций, результат выполнения которых не отражает отношения между элементами данных, потребление оперативной памяти стремится к O(1) независимо от общего объёма фактического затрагиваемых данных
    c. Показать, что операция вставки выполняется за O(1) независимо от размера данных, представленных в файле
    d. Показать, что операция выборки без учёта отношений (но с опциональными условиями) выполняется за O(n), где n – количество представленных элементов данных выбираемого вида
    e. Показать, что операции обновления и удаления элемента данных выполняются не более чем за $O(n*m) > t \rightarrow O(n+m)$, где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных
    f. Показать, что размер файла данных всегда пропорционален количеству фактически размещённых элементов данных
    g. Показать работоспособность решения под управлением ОС семейств Windows и *NIX
5. Результаты тестирования по п.4 представить в составе отчёта, при этом:
    a. В части 3 привести описание структур данных, разработанных в соответствии с п.1
    b. В части 4 описать решение, реализованное в соответствии с пп.2-3
    c. В часть 5 включить графики на основе тестов, демонстрирующие амортизированные показатели ресурсоёмкости по п. 4

1 Challenges and Solutions

1.1 Initial In-Memory Database Implementation

The initial task was straightforward, involving the creation of an in-memory database. Each table was akin to a 2D array (m x n):

- **Insert Query:** Adding a row at the end of the table, complexity O(1).
- **Search Query:** Iterating through each row and checking conditions, complexity O(m*n).
- **Update Query:** Similar to search, iterating and updating if conditions match, complexity O(m*n).
- **Delete Query:** Iterating through rows, marking deleted ones to be skipped in future queries.

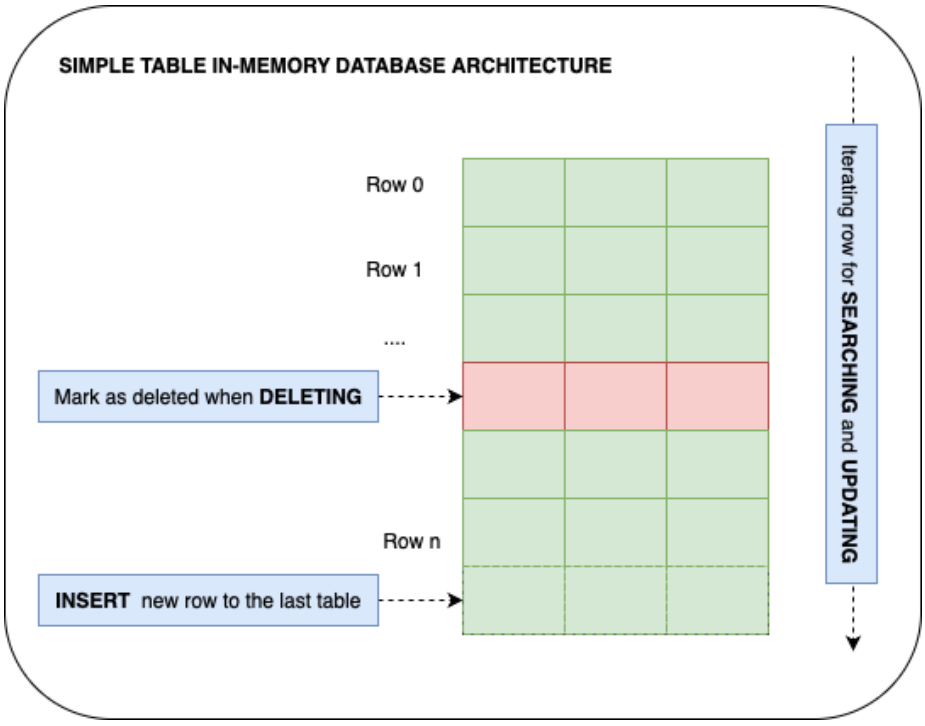

Рис. 1: SIMPLE TABLE IN-MEMORY DATABASE ARCHITECTURE

1.2 Memory Optimization Challenge

A key challenge arose in handling deleted data. Simply marking data as deleted wasn't memory-efficient, as the deleted data still occupied space. To optimize memory usage, I transitioned to using a linked list data structure for rows. In this setup, deleting a row meant removing the node and updating links in adjacent nodes. For inserts, I prepended to the linked list. The search and update operations remained similar to the 2D array approach.

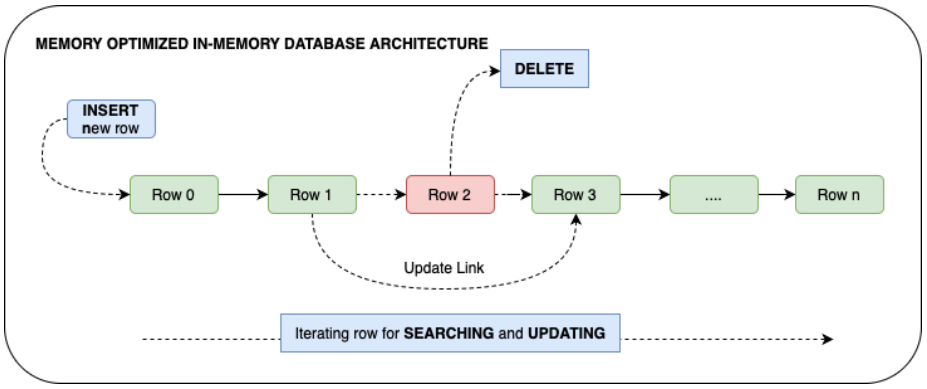

Рис. 2: MEMORY OPTIMIZED IN-MEMORY DATABASE ARCHITECTURE

1.3 Handling File-Based Storage

The complexity increased with requirements to handle file-based storage. Storing large files (up to 10GB) in memory wasn't practical. The challenge was to map the in-memory linked list to a file using offsets. While the concept was straightforward, the implementation was intricate. Let's review how queries changed with file handling:

- **Search Query:** Remained unchanged, still iterating through file rows.
- **Insert Query:** Appending to the file's end and linking the new element with the previous one.
- **Delete Query:** Direct deletion was impractical due to offset changes in subsequent elements. I chose a safer but less memory-efficient approach, skipping the row to be deleted and updating adjacent row links.
- **Update Query:** Initially, I fixed the row size, disallowing updates that increased the row size. However, this was impractical. My current approach trades off memory efficiency: creating a new row for each update, skipping the old row, and updating links to the new row. This solution is straightforward and practical but not optimal in terms of memory usage.

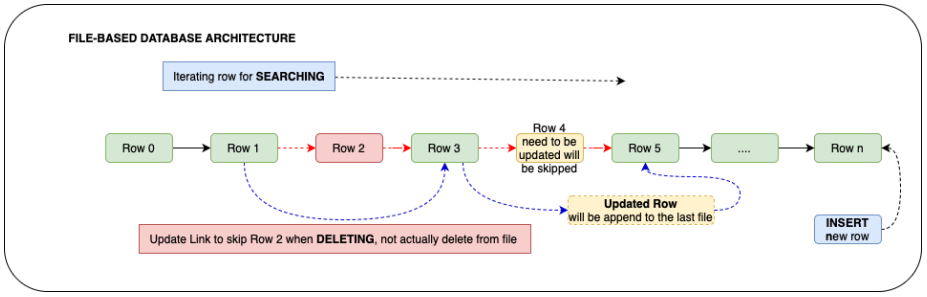

Рис. 3: FILE-BASED DATABASE ARCHITECTURE

2 Performance Testing Results

The performance of the file-based database system was rigorously evaluated through a series of tests, focusing on the efficiency of various operations such as adding, querying, updating, and deleting rows. The results of these tests provide valuable insights into the system's scalability and efficiency in handling different types of database operations.

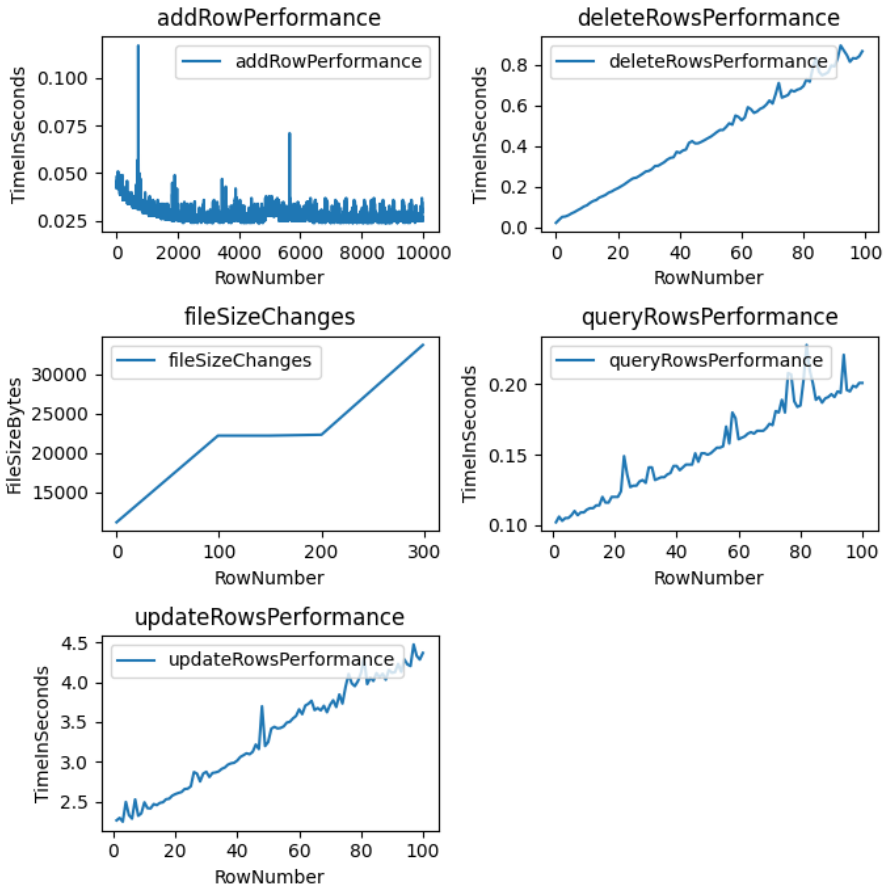

Рис. 4: FILE-BASED DATABASE ARCHITECTURE

3 Conclusion

In conclusion, this lab project on building a file-based database system has been a profound learning experience, highlighting the complexities of low-level programming and efficient data management. The challenges encountered and overcome in this project have provided valuable insights into data structures, file handling, and performance optimization. This work not only reinforces theoretical knowledge but also enhances practical skills crucial for future software development endeavors.