Санкт-Петербургский национальный исследовательский университет ИТМО

Низкоуровневое программирование

Лабораторная работа №1

Выполнил:

Ненов Владислав Александрович

Преподаватель:

Кореньков Юрий Дмитриевич

Вариант 5

Группа №P33082

Санкт-Петербург
2023

Задание

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 выбранного вида
- e. Показать, что операции обновления и удаления элемента данных выполняются не более чем за O(n*m) > t O(n+m), где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных
- g. Показать работоспособность решения под управлением ОС семейств Windows и *NIX

Выполнение

Структуры данных

Отображение памяти

`struct Chunk` – структура, хранящая указатель на отображенную память, размер и отступ от начала файла в страницах, а также количество ссылок на данный фрагмент памяти. Хранятся как связный список в `MemoryManager.`

`struct UserChunk` – структура, хранящая ссылку на `Chunk` и реальный размер, а также отступ от начала чанка. Был введен на этапе адаптации кода под платформу Windows, так как на ней отступы при отображении файла должны кратными 64 кб. Это вынуждаем использовать большие чанки и `UserChunk` как прослойку, отдаваемую пользователю.

`struct MemoryManager` – структура, хранящая все данные, необходимые для отображения фрагментов файла-хранилища в память. Принимается в качестве параметра почти всеми методами данного логического слоя.

Работа со страницами

`struct PageMeta` – метаданные страницы, включая отступ от начала файла.

`struct PageRecord` – структура без выравнивания, используемая для записи и чтения заголовков страниц. По содержимому является клоном структуры `PageMeta.`

`struct PageRow` – структура, используемая для записи и чтения объектов, хранящихся в страницах.

`enum RowReadStatus, enum RowWriteStatus, enum RowRemoveStatus` – перечисления, возвращающие статус операций чтения, записи и удаления объектов соответственно.

Таблицы

`enum TableDatatype` – перечисление всех типов, доступных для хранения в базе данных.

`struct TableScheme` – структура, используемая для создания схемы таблицы в процессе создания таблицы пользователем.

`struct SchemeItem` – структура, используемая для хранения данных (тип, название и тд) поля таблицы.

`struct Table` – структура, используемая при создании новой таблицы в базе данных пользователем.
`struct TableRecord` – структура без выравнивания, используемая для записи и чтения метаданных о таблице. Отображается в память и хранится по ссылке в объекте `OpenedTable.`

Хранилище

`struct FileHeader` – структура, хранящая общую мета-информацию о файле базы данных. Хранит текущее кол-во страниц и таблиц в файле.
`struct Storage` – структура, хранящая ссылки на отображения всех важных фрагментов файла. Таких как заголовок файла, таблица-таблиц, таблица схем и таблица пустых страниц. А также структуры, необходимые для взаимодействия с другими модулями программы. Принимается в качестве параметра почти всеми методами данного логического слоя.
`struct OpenedTable` – структура-представление таблицы, необходимая для любого взаимодействия с ней пользователем. Хранит схему и ссылку на отображенные в память метаданные.

Запросы

`struct RowsIterator` – структура для итерации по строкам таблицы пользователем. Поддерживает неограниченное количество передаваемых условий. Операции получения, удаления и обновления данных происходят при взаимодействии с данной структурой.
`enum RowsIteratorResult` – перечисления для результатов работы итератора.
`struct RequestFilter` – структура, используемая для добавления опциональных условий `RowsIterator.`
`struct Join` – структура, используемая для осуществления операции join.

Описание решения

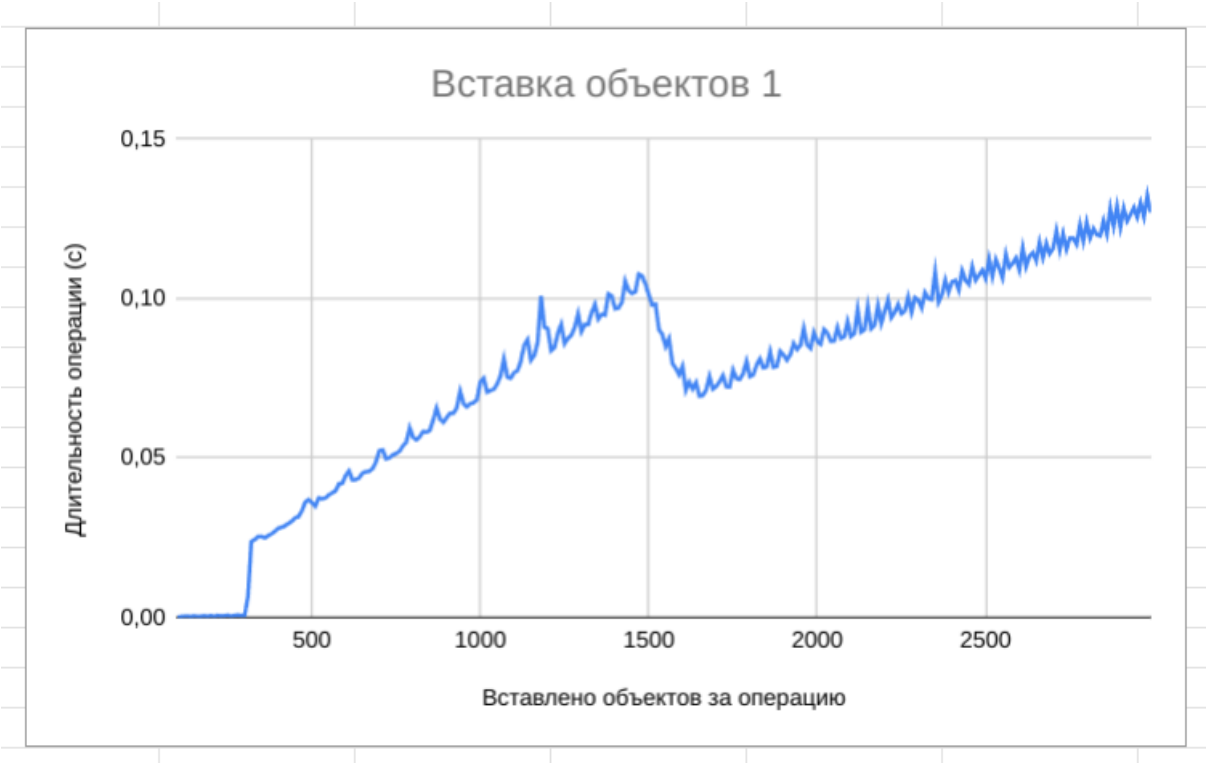
Файл базы данных поделен на страницы, каждая из которых состоит из заголовка с метаданными, битовой маски занятости строк и некоторого количества строк, помещающихся на страницу. Страницы могут быть любого размера, кратного размеру системной страницы. Они закрепляются за определенными таблицами и выступают в роли двусвязного списка, так как хранят отступ предыдущей страницы в таблице и следующей. При это страницы делятся на 2 вида: полностью занятые, и не полностью. Страницы каждого из двух видов хранят адреса только страниц своего вида, образуя два связных списка. Отступ первой страницы каждого из них хранится в метаданных таблицы и обновляется по мере необходимости. Это обеспечивает быстрое получение незаполненной страницы при вставке, а также легкость перемещения страниц между этими списками. Когда страница полностью освобождается в процессе удаления содержимого таблицы, она удаляется из своего списка и попадает в таблицу для свободных (то есть не привязанных ни к одной таблице) страниц. Далее при вставке проверяется - есть ли подходящая свободная страница. И если да - то мы используем ее, вместо создания новой.

Строки не фиксированной длины хранятся в отдельных таблицах. Такие таблицы создаются для строк размером в диапазоне +-50 символов. А в таблице, куда первоначально происходит вставка хранится только отступ целевой страницы и строки (где по-настоящему хранится строка).
Все операции над содержимым таблиц производятся пользователем с использованием API RowsIterator'a. Он предоставляет возможность добавлять условия выборки строк и итерировать по ним, получая их содержимое, изменяя или удаляя его.

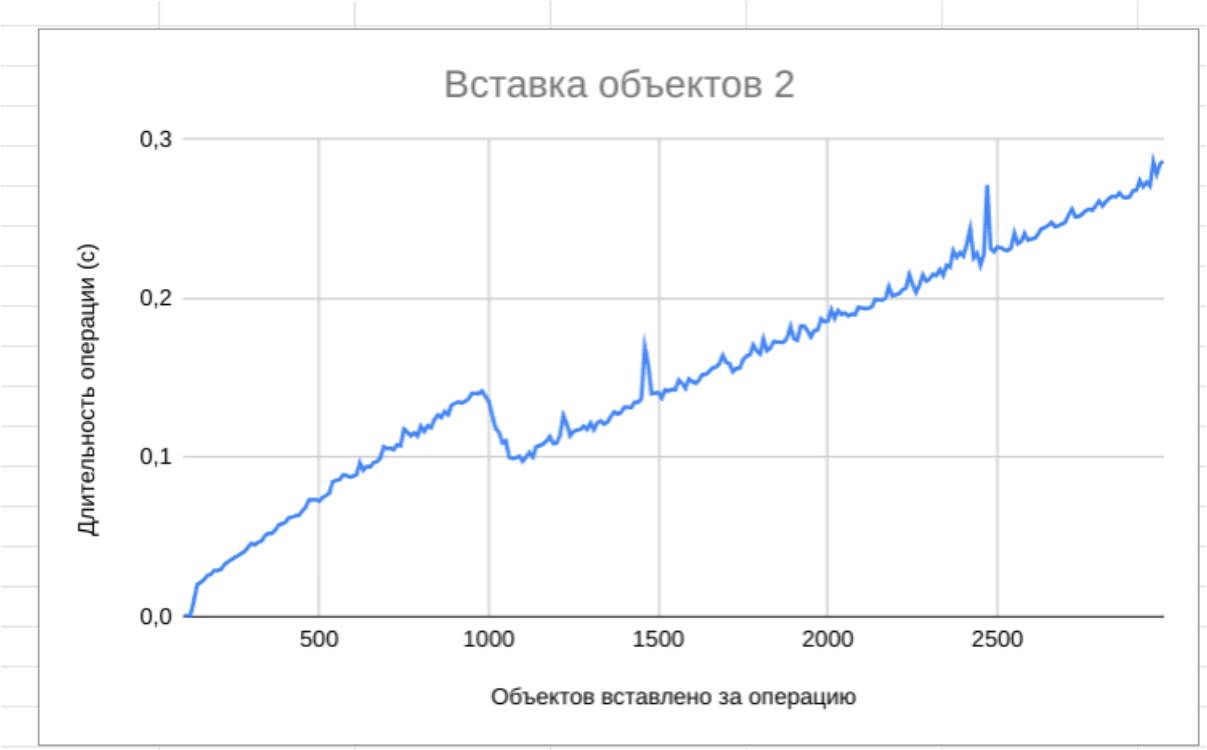
Графики и тестирование

Вставка

В базу данных вставляются пачки по n строк, n линейно растет. Таким образом время вставки также должно расти линейно.



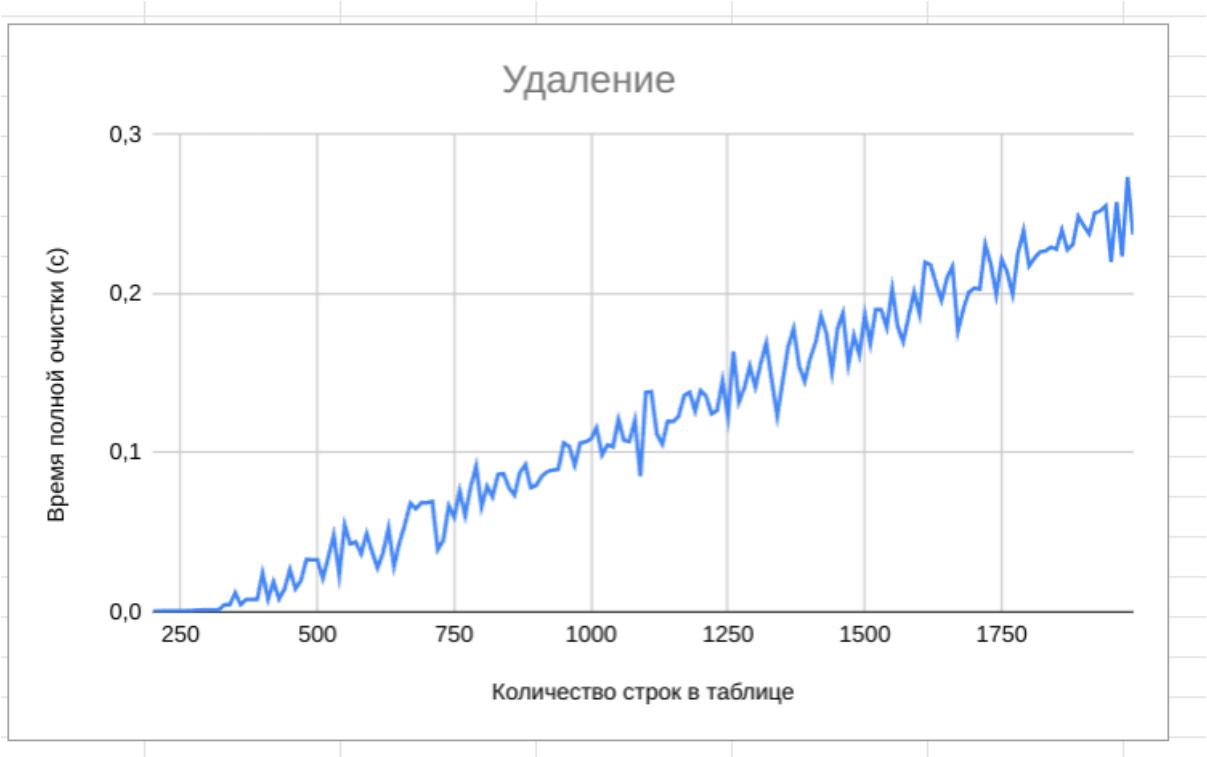
Вставка объектов без строк произвольной длины



Вставка объектов со строками произвольной длины

Удаление

В базу данных загружается n строк, далее они все удаляются. Операции повторяются с линейным увеличением переменной n.



Вставка-удаление

В базу данных вставляется 500 строк, далее случайным образом удаляются 400. Замеры времени происходят каждые 100 операций. Всего в процессе теста выполнено 327700 операций вставок.