УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерных технологий
Направление подготовки: Информатика и вычислительная техника

Низкоуровневое программирование

Лабораторная работа №1

Вариант 2

Выполнил:

Иевлев К. В.
Группа № P33302

Преподаватель:
Кореньков Ю. Д.

г. Санкт-Петербург

2023

Цель:

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

Задачи:

- Спроектировать структуры данных для представления информации в оперативной памяти
- 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)
- Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:
- Реализовать тестовую программу для демонстрации работоспособности решения

Описание:

Примеры структур и интерфейсов:

```
struct page_header {
    bool is_dirty;
    char table_name[MAX_TABLE_NAME_LENGTH];
    uint16_t remaining_space;
    uint32_t page_number;
    uint32_t write_ptr;
    uint32_t real_number;
    uint32_t next_page_number;
};

struct database_header {
    char name[MAX_DATABASE_NAME_LENGTH];
    struct database* database;
    uint32_t table_count;
    uint32_t page_count;
    uint32_t page_size;
    uint32_t last_page_number;
};

struct database {
    struct database_header* database_header;
    FILE* source_file;
};


struct table_header {
    bool is_available;

    char name[20];
```

```
    struct database* database;
    struct table* table;
    struct schema schema;

    uint32_t page_number_first;
    uint32_t page_number_last;
    uint32_t page_count;
    uint32_t real_number;
};

struct table {
    struct table_header* table_header;
    struct schema* schema;
};

struct page_header* page_create(struct database_header* database_header,
struct table_header* table_header);
struct database* db_get(const char *const file, const enum database_state
state);
struct table* table_create(struct schema* schema, const char* table_name,
struct database*  database);
void db_close(struct database* database);
void table_close(struct table* table);
struct query* query_make(enum query_types operation, struct table* table,
char* columns[], void* vals[], int32_t cnt);
void query_execute(struct query *query, bool show_output);
void query_close(struct query* query);
void attribute_add(struct row* row, char* name, enum data_type content_type,
void* value);
struct schema* schema_create();
```

Пример добавления данных:

```
struct schema* test_schema = schema_create();
test_schema = schema_add_column(test_schema,"id", INTEGER);
struct table* table = table_create(test_schema, "test", db );
struct row* row_test = row_create(table);
attribute_add(row_test, "id", INTEGER, (void *)&j);
row_insert(row_test);
struct query* query = query_make(SELECT, table, column_name, value);
query_execute(query);
```

Вывод join:

```
1 Petr F 175.500000 job2
2 Petrucha T 192.600000 job3
3 Petrosyan F 168.100000 job4
4 Petych T 180.100000 job5
5 Petya F 175.500000 job6
```

Где в одной таблице вместо job -job_id, а в другой находятся работы.

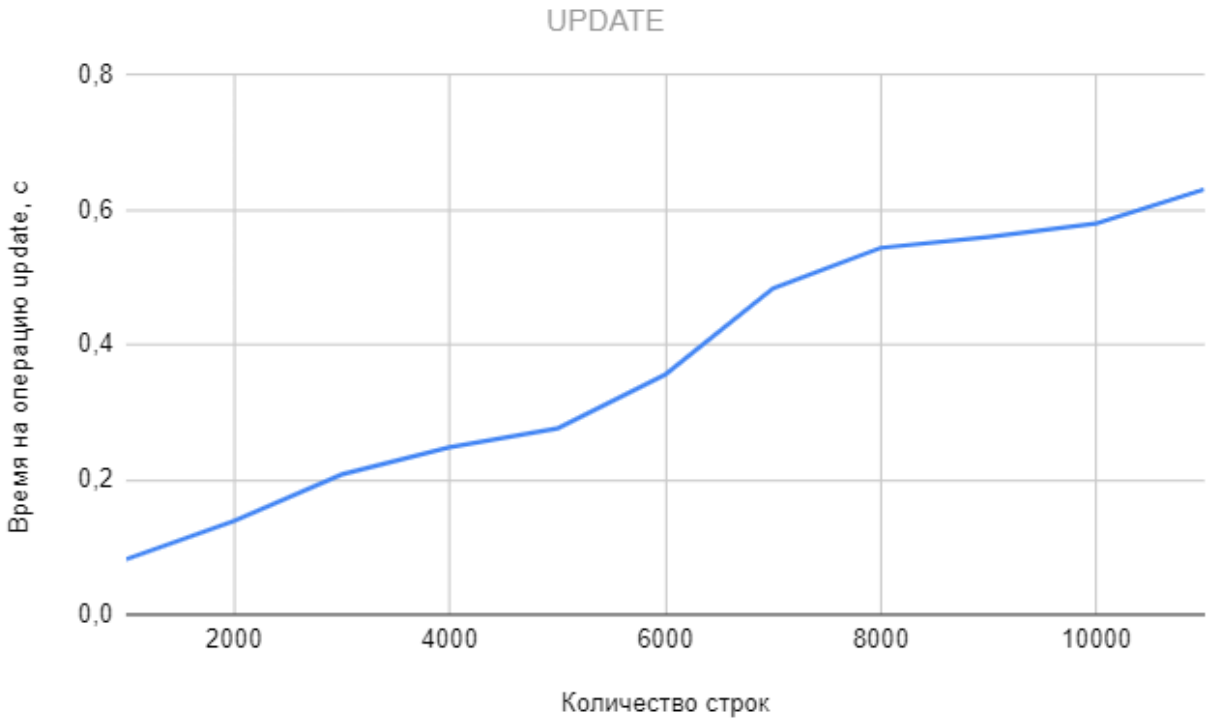
Аспекты реализации:

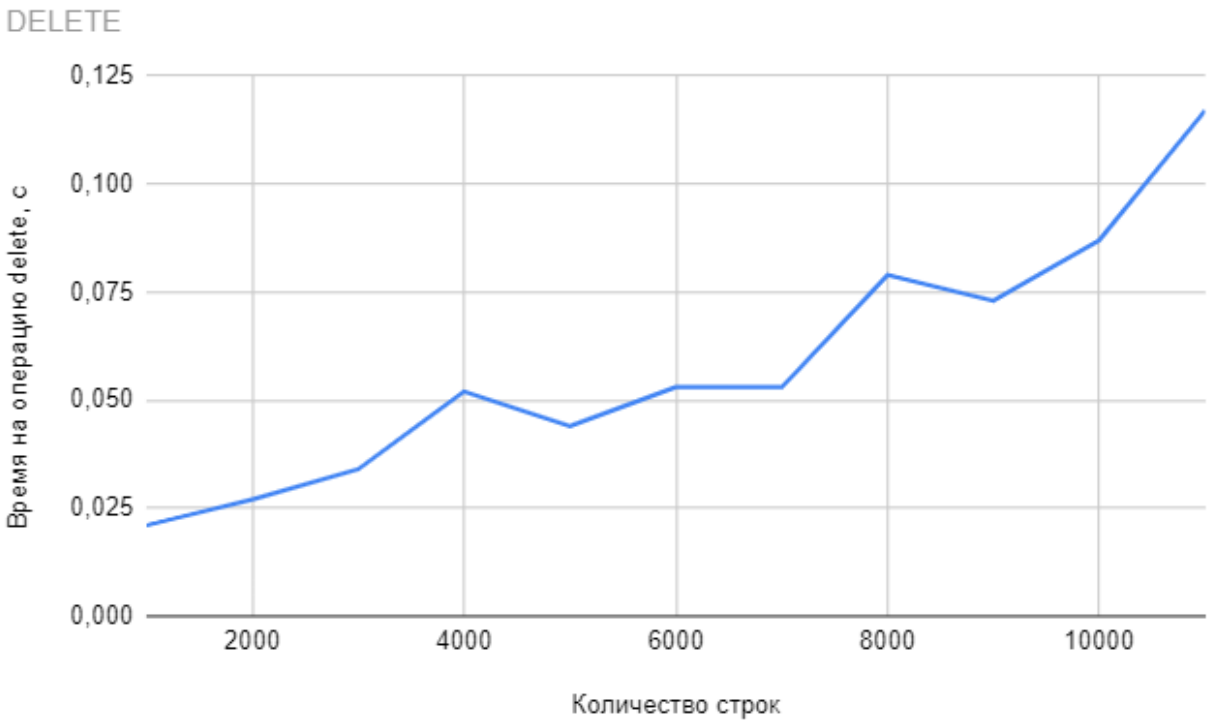
База данных содержится в файле, для нее создается заголовок, где храниться ее название, размер страниц, количество страниц, количество таблиц и номер последней страницы. База данных разделена на страницы одинакового размера. У страниц есть шапка, где также храниться служебная информация: ее номер, информация о ее заполненности, информация о таблице, которой она принадлежит. Страницы пронумерованы для перемещения по ним. При переполнении страницы создается новая. Каждая страницы принадлежит только одной таблице. Для создания таблицы необходимо создать ее схема, где хранятся данные о ее колонках. У каждой таблицы должна быть схема и заголовок. Заголовок каждой таблицы содержит информацию: о базе данных, о схеме, названии таблицы, номера ее первой и последней станицы. Дальше для добавления данных мы создаем строку и основываясь на схеме таблицы заполняем ее.

Графики:

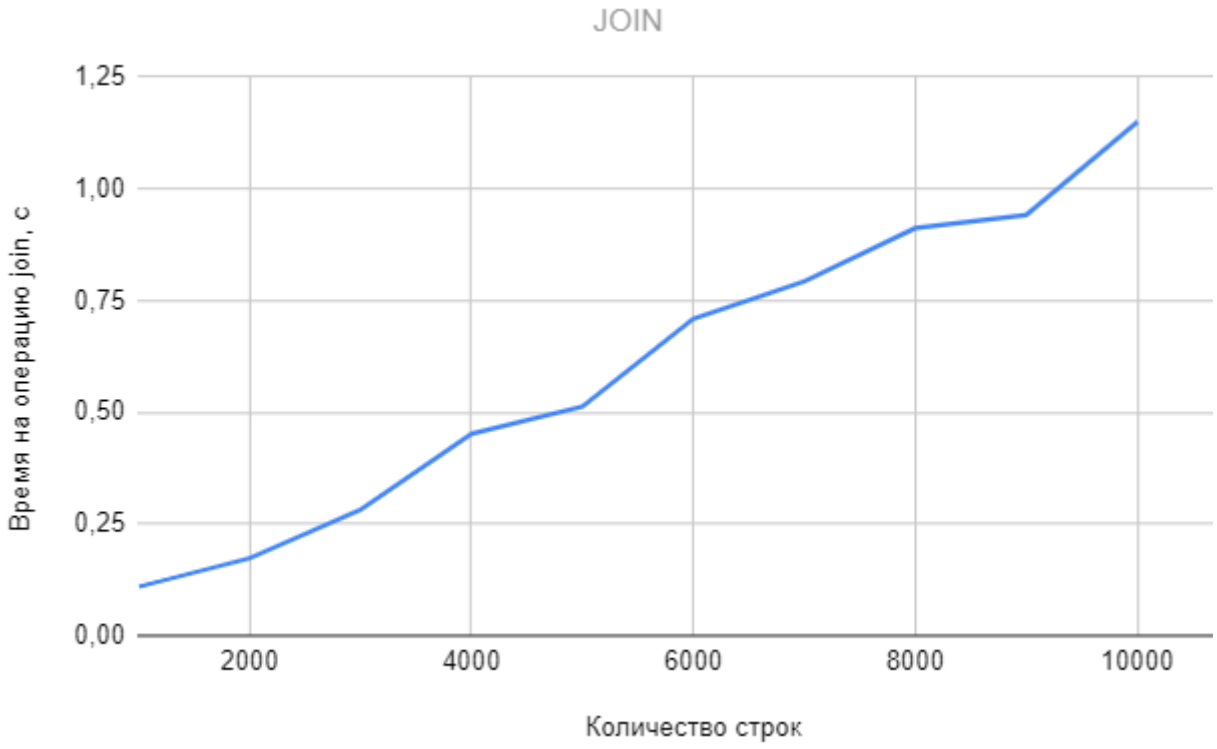
Можно увидеть, что разброс есть небольшой разброс, но он появляется скорее из-за особенности системы, а там все точки находят возле одной прямой. Поэтому можно сказать что операция вставки выполняется за O(1).

Здесь же видно, что операция выборки прямо пропорционально количеству строк, поэтому она выполняется за O(n)





Видны точки в которых происходит удаление данных или обновление, там происходит скачек по времени



Запуск

Выбираем нужный нам Makefile, если у вас linux то просто выполняем make.

Если у вас windows, то необходимо переименовать Makefile.linux -> Makefile.

Далее выполняем для запуска: make execute;
Для отчистки: make clean

Вывод
Был разработан модуль, который позволяет хранить данные внутри одного файла в формате реляционных таблиц. Поддерживающий операции вставки, обновления, выборки и удаления данных.