ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки: Информатика и вычислительная техника

Дисциплина «Низкоуровневое программирование»

Лабораторная работа №1

Вариант 2

Выполнил:

Богатов А. С.

P33302

Преподаватель

Кореньков Ю.Д.

г. Санкт-Петербург, 2023 г.

Задание:

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

Разработанный модуль:

Итоговый модуль разделен на следующие подмодули:

- database - работа с объектом БД, виртуальными страницами БД и запросами
- type_ops - работа с разными типами данных
- file_ops - реализация запросов, чтения и записи в файл, работы с страницами внутри файла
- schema - реализация схем БД
- relation - работа с данными в отношениях БД

Database: Создание структур базы данных, заголовка базы и заголовка страницы внутри базы

```
struct database_header {
    char name[MAX_DATABASE_NAME_LENGTH];
    struct database* database;
    uint32_t relation_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 page_header {
    uint16_t remaining_space;
    uint32_t page_number;
    bool is_dirty;
    uint32_t write_ptr;
    char relation_name[MAX_RELATION_NAME_LENGTH];
    uint32_t real_number;
    uint32_t next_page_number;
};
```

Relation: Отношение в БД также имеет заголовок, содержащий даннные для поиска отношения и данных в нем

```
struct relation_header {
    char name[MAX_NAME_LENGTH];
    struct database* database;
    struct relation* relation;
    uint32_t page_count;
    uint32_t real_number;
    bool is_available;
    struct schema schema;
    uint32_t page_number_first;
    uint32_t last_page_number;
};

struct relation {
    struct relation_header* relation_header;
    struct schema* schema;
};
```

Наполнение базы данными:

```
struct row* first_row = row_create(relation); //создание строки

attribute_add(first_row, "varchar", VARCHAR, (void*) &varchar);
//заполнение значения колонок

row_insert(first_row); //вставка строки в отношение

struct query* select_query = query_make(SELECT, relation, column,
value, -1); //создание и исполнение запроса
query_execute(select_query, true); // можно указывать необходимо
ли выводить выбранные строки, использовано для адекватного вывода
при тестировании
```

Пример вывода (SELECT ... JOIN):

```
1 placeholder foreign1
2 placeholder foreign1
3 placeholder foreign1
```

```
4 placeholder foreign1
5 placeholder foreign1
6 placeholder foreign1
7 placeholder foreign1
8 placeholder foreign1
9 placeholder foreign1
10 placeholder foreign1
Joined 10 rows
```

База данных хранится в файле, для БД создается заголовок, содержащий служебную информацию для навигации по БД. БД разделена на страницы (page), в файл также условно поделен на страницы (real_page). При создании страницы для нее также создается заголовок со служебной информацией - наполнение страницы, оставшийся размер, указатель на начало свободной части. Page относятся к отношениям в БД, real_page относятся к самим БД. Для создания отношения используются схемы - схема содержит информацию об атрибутах отношения и строках. Страницы пронумерованы для перемещения по ним, информация о номерах страниц хранится в заголовках нужной нам в тот или иной момент сущности. При нехватке места на странице, будет создана новая.

```
enum file_op_status relation_page_write(FILE *file, struct
page_header* page_header, struct schema* schema) {
    fseek(file, (page_header->page_number - 1) *
DEFAULT_PAGE_SIZE_BYTES, SEEK_SET);

    uint16_t size = schema->count * sizeof(struct column);

    page_header->remaining_space -= sizeof(struct page_header) +
sizeof(uint16_t) + size;
    page_header->write_ptr += sizeof(struct page_header) +
sizeof(uint16_t) + size;

    if (fwrite(page_header, sizeof(struct page_header), 1, file)
!= 1) {
        return ERROR;
    }

    struct column* columns = malloc(size);

    struct column* current = schema->start;
```

```
    for (size_t i = 0; i < schema->count; i++){
        columns[i] = *current;
        current = current->next;
    }

    fseek(file, (page_header->page_number - 1) *
DEFAULT_PAGE_SIZE_BYTES + sizeof(struct page_header), SEEK_SET);
    if (fwrite(&size, sizeof(uint16_t), 1, file) != 1) {
        return ERROR;
    }
    fseek(file, (page_header->page_number - 1) *
DEFAULT_PAGE_SIZE_BYTES + sizeof(struct page_header) +
sizeof(uint16_t), SEEK_SET);
    if (fwrite(columns, size, 1, file) != 1) {
        return ERROR;
    }
    free(columns);
    free(page_header);
    return OK;
}
```

```
enum file_op_status row_write_to_page(FILE *file, uint32_t
page_number, struct row* row) {

    struct page_header* new = NULL;
    uint32_t length =
row->relation->relation_header->schema.length;
    uint32_t size = sizeof(struct row_header) + length;
    fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES,
SEEK_SET);
    struct page_header* page_header = malloc(sizeof(struct
page_header));
    if (fread(page_header, sizeof(struct page_header), 1, file) ==
1) {

        if (!is_enough_space(page_header, size)) {

            new = page_add(row->relation->relation_header,
row->relation->relation_header->database->database_header);

            fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES
```

```
+ sizeof(struct page_header), SEEK_SET);
            uint16_t columns_size;
            fread(&columns_size, sizeof(uint16_t), 1, file);

            struct column* columns = malloc(columns_size);

            fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES
+ sizeof(struct page_header) + sizeof(uint16_t), SEEK_SET);
            fread(columns, columns_size, 1, file);

            page_number = new->page_number;
            page_header = new;

            page_header->remaining_space -= sizeof(struct
page_header) + sizeof(uint16_t) + columns_size;
            page_header->write_ptr += sizeof(struct page_header) +
sizeof(uint16_t) + columns_size;

            fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES,
SEEK_SET);

            fwrite(page_header, sizeof(struct page_header), 1,
file);
            fwrite(&columns_size, sizeof(uint16_t), 1, file);
            fwrite(columns, columns_size, 1, file);

            free(columns);
        }

        fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES +
page_header->write_ptr, SEEK_SET);

        if (fwrite(row->row_header, sizeof(struct row_header), 1,
file) == 1) {
            fseek(file, (page_number-1) * DEFAULT_PAGE_SIZE_BYTES
+ page_header->write_ptr + sizeof(struct row_header), SEEK_SET);
            if (fwrite(row->data, length, 1, file) == 1) {

                page_header->write_ptr += sizeof(struct
row_header) + length;
                page_header->remaining_space -= sizeof(struct
row_header) + length;
```

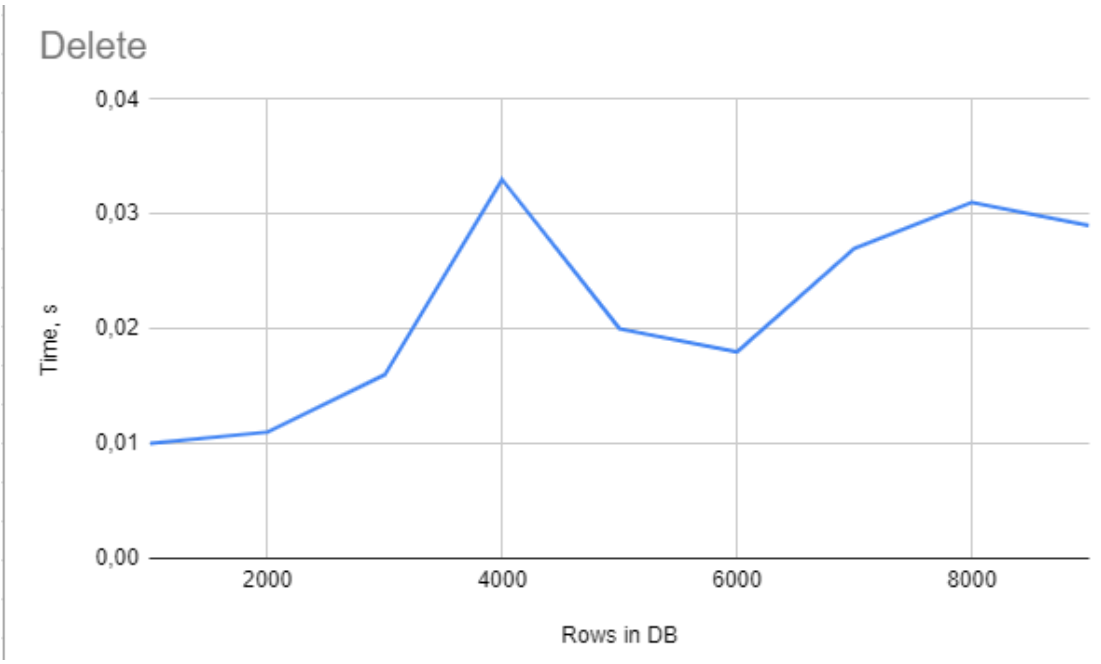
```
                fseek(file, (page_number-1) *
DEFAULT_PAGE_SIZE_BYTES, SEEK_SET);

                if (fwrite(page_header, sizeof(struct
page_header), 1, file) != 1) {
                    if (new) free(new);
                    free(page_header);
                    return ERROR;
                }
                if (new) {
                    free(new);
                }

                return OK;
            }
        }
        if (new) {
            free(new);
        }
        free(page_header);
        return ERROR;
    } else {
        if (new) {
            free(new);
        }
        free(page_header);
        return ERROR;
    }
}
```

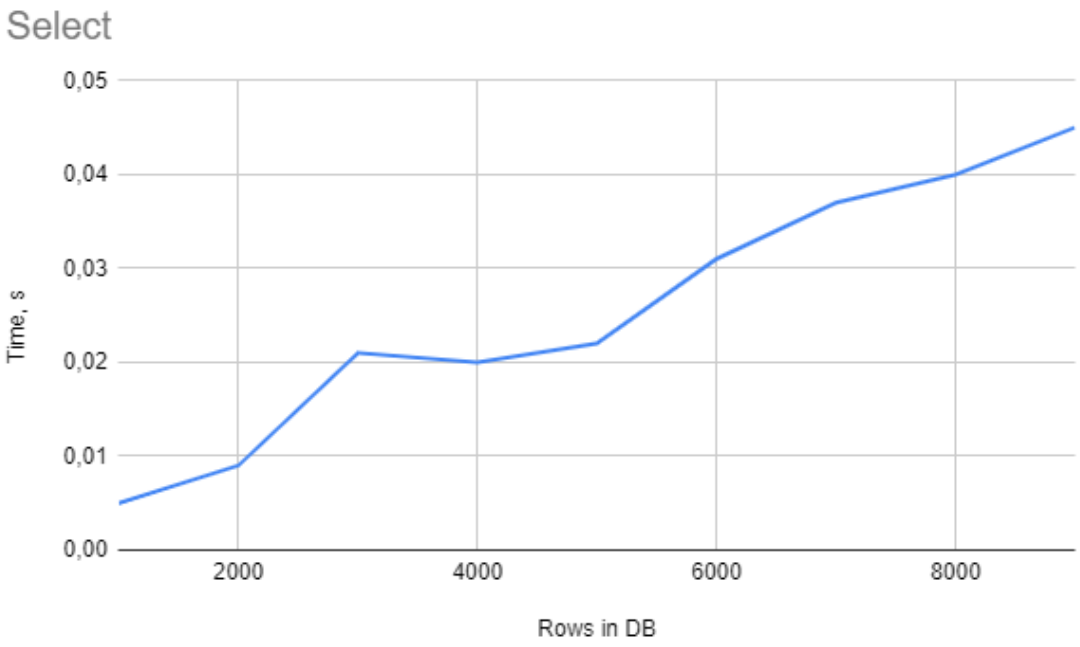
Тестирование:

В модуле main представлены следующие тесты:

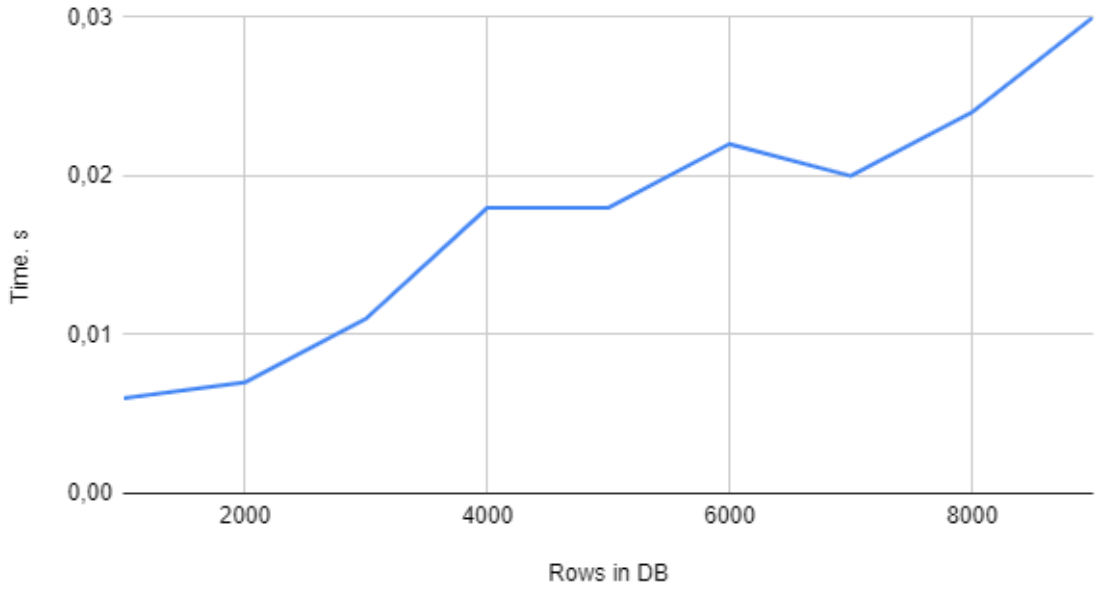
- size_test - проверяет работу с большим количеством данных в БД
- save_test - проверяет создание БД, сохранение в файл и последующее повторное открытие
- test_insert/test_select/test_delete/test_update/test_join - проверка работы соответствующих запросов к нашей БД

Графики:

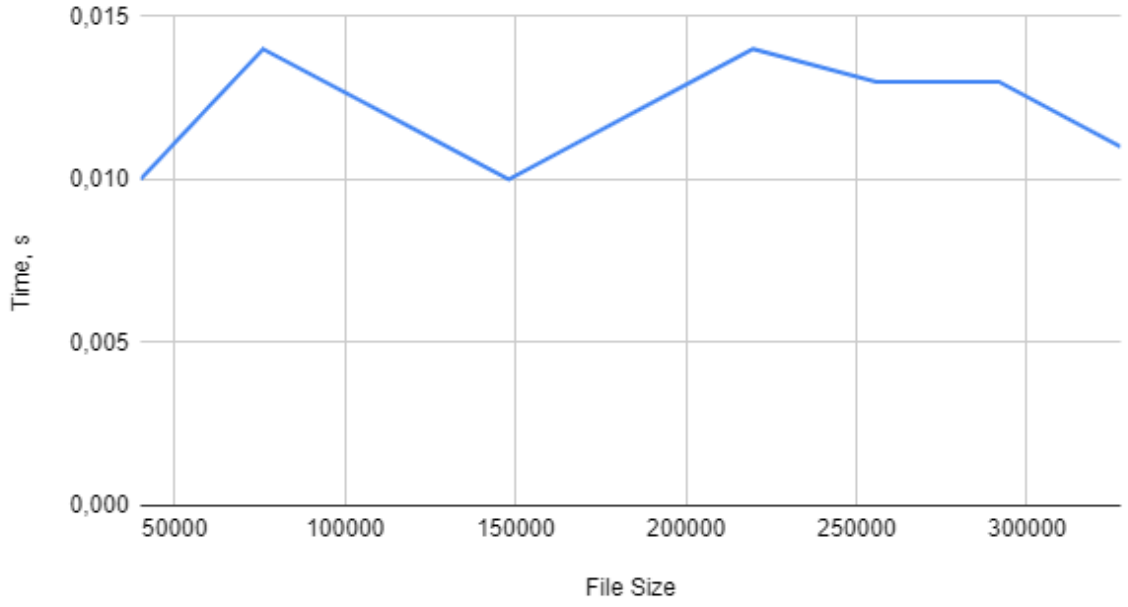

Delete
0,04
0,03
0,02
0,01
0,00
Time, s
2000
4000
6000
8000
Rows in DB

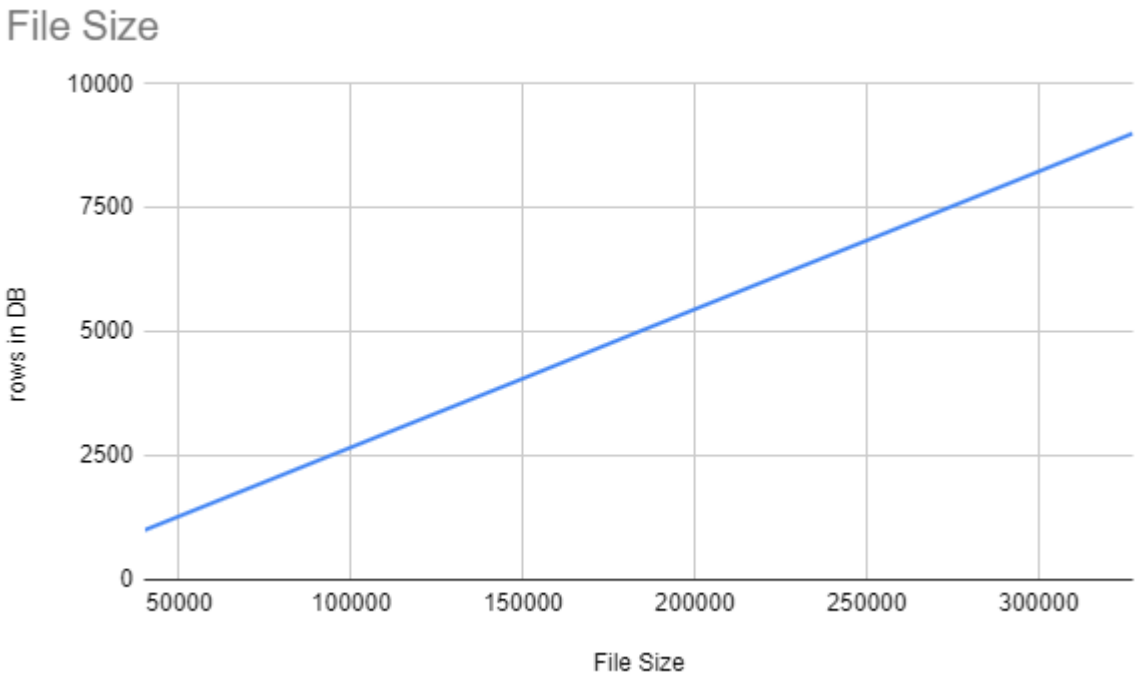

Select
0,05
0,04
0,03
0,02
0,01
0,00
Time, s
2000
4000
6000
8000
Rows in DB

Update



Insert





Запуск:

Требования:

- ОС Windows/*NIX
- Make

Как запустить:

1. Выбрать Makefile для вашей ОС
2. make all - собрать модуль
3. ./main - для *nix
4. .\main.exe - для Windows

Выводы:

Созданный модуль позволяет хранит в файле реляционную базу данных и использовать SQL DML для исполнения запросов к нашей БД.