

Санкт-Петербургский Национальный Исследовательский Университет
ИТМО

МФКТиУ, факультет ПИиКТ

**Лабораторная работа №1 по предмету
«Низкоуровневое программирование»**

Преподаватель: Кореньков Юрий Дмитриевич

Выполнил: Стефан Лабович

Группа: Р33102

Вариант: Реляционные таблицы

Санкт-Петербург, 2022

Цель работы:

Разработка модуля реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего реляционным таблицам.

Описание:

Данные которые хранятся в файле, хранятся в трех разных логических видов блоках.

- Database block (Блок Базы Данных)

- Schema block (Блок Схемы)

- Record block (Блок Записи)

В каждом из блоках хранятся метаданные Блока, которые содержат информацию о типе блока, его идентификатор, идентификатор следующего блока, и смещение в этом блоке.

В блоке Базы Данных хранятся метаданные, содержащие информацию о названию базы данных (название файла в котором данные хранятся), величине блока, количеству блоков в файле, идентификатор первого блока схемы, идентификатор последнего блока схемы и идентификатор первого свободного блока. В одном файле хранится одна база данных.

В блоке Схемы хранятся данные связанные с таблицами, валидность таблицы, название таблицы, идентификатор первого блока записи, идентификатор последнего блока записи, количество атрибутов таблицы, название, тип и величина каждого атрибута, количество записей таблицы.

В блоке Записи хранятся элементы данных соответствующей таблицы.

Структуры которые используются в модуле:

Структура БД:

```
struct database {
    char name[MAX_NAME_LENGTH];
    uint32_t block_size;
    uint32_t block_num;
    uint32_t schema_first_block_id;
    uint32_t schema_last_block_id;
    uint32_t first_clean_block;
    block* blk;
};
```

Структура блока:

```
struct block {
    enum block_type type;
    uint32_t id;
    uint32_t next_id;
    uint16_t offset;
};
```

Структура страницы:

```
struct page {
    uint16_t size;
    uint8_t* buffer;
    page* next;
};
```

Структура таблицы:

```
struct table{
    bool valid;
    char name[MAX_NAME_LENGTH];
    uint32_t block_id;
    uint32_t offset;
    uint32_t first_record_block_id;
    uint32_t last_record_block_id;
    uint32_t num_rows;
    uint8_t num_columns;
    column** columns;
};
```

Структура столбца:

```
struct column {
    char name[MAX_NAME_LENGTH];
    enum data_type type;
    uint8_t size;
};
```

Структура записи:

```
struct record {
    bool valid;
    uint8_t *data;
};
```

Вспомогательные структуры которые используются в запросах:

```
struct table_to_join {
    table* tb;
    uint8_t num_cols;
    char** view_cols;

    column* col_to_join;
};

struct column_to_update {
    char* col_name;
    enum data_type type;
    uint32_t size;
    union type_value* value;

    column_to_update* next;
};
```

```

struct condition {
    enum condition_type type;
    enum belongs_to_table belongs_to_table;

    condition_operand* left_operand;
    condition_operand* right_operand;
    enum relation relation;

    condition* next;
    enum next_relation next_relation;
};

struct condition_operand {
    enum data_type type;
    uint32_t size;
    union type_value* value;
};

union type_value {
    int32_t i;
    float f;
    bool b;
    char* s;
};

```

Публичный интерфейс:

Добавление, удаление и получение информации о таблицах:

```

void insert_table_to_schema(table* tb, database* db, FILE* f);
bool delete_table_from_schema(char* name, database* db, FILE* f);
table* get_table_from_schema(char* name, database* db, FILE* f);

```

Добавление, удаление, изменение и получение информации о элементах данных:

```

void insert_record_to_table(record* r, table* tb, database* db, FILE* f);

void update_records_in_table(column_to_update* col,
                             condition* cond,
                             table* tb,
                             database* db,
                             FILE* f);

void delete_records_from_table(condition* cond,
                               table* tb,
                               database* db,
                               FILE* f);

uint32_t select_records_from_table(uint32_t block_offset,
                                   char* buffer,
                                   uint32_t buff_sz,
                                   uint8_t num_cols,
                                   char** view_cols,
                                   condition* cond,
                                   table* tb,

```

```

        database* db,
        FILE* f)

void select_records_from_table_inner_join(uint32_t* left_block_off,
        uint32_t* right_block_off,
        char* buffer,
        uint32_t buff_sz,
        table_to_join* left,
        table_to_join* right,
        condition* cond,
        database* db,
        FILE* f)

```

Результаты тестов:

Вставка:

Для теста вставки, мы 100 раз засечем время вставки 10 000 элементов в одну и ту же таблицу.



Зависимость вставки не зависит от размера данных.

Выборка:

Для теста выборки, мы 100 раз засечем время выборки, и после каждой добавим 10 000 элементов в одну и ту же таблицу.



Зависимость линейная.

Удаление:

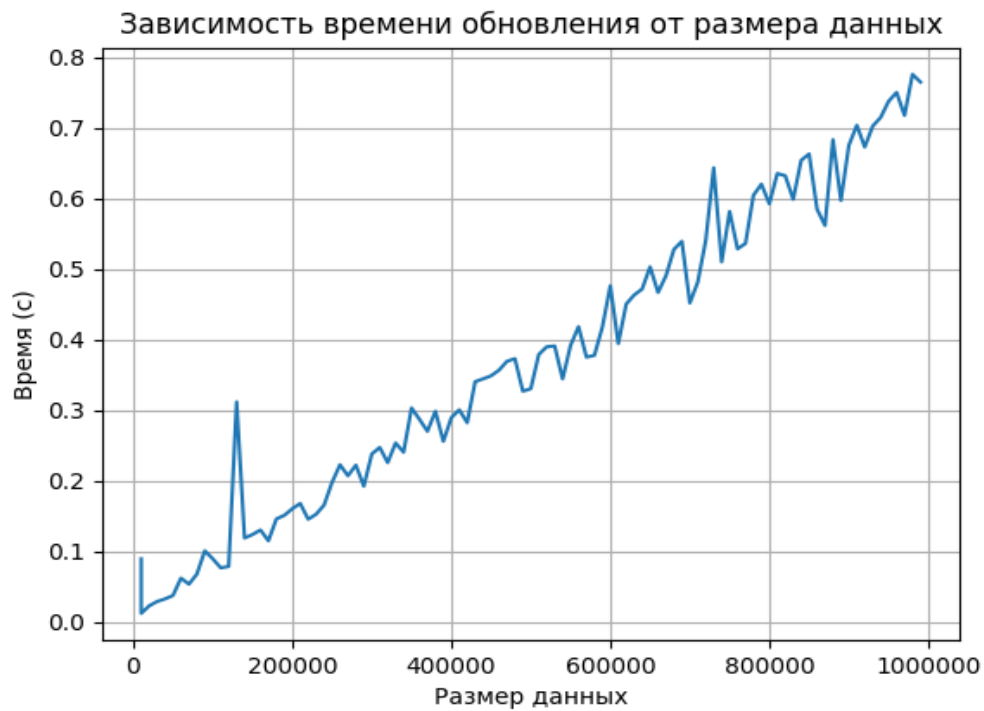
Для теста удаления, мы 100 раз засечем время удаления 5 000 элементов, и после каждого добавим 10 000 элементов в одну и ту же таблицу.



Зависимость линейная.

Обновление:

Для теста обновления, мы 100 раз засечем время обновления 5 000 элементов, и после каждого добавим 10 000 элементов в одну и ту же таблицу.



Зависимость линейная.

Слияние двух таблиц:

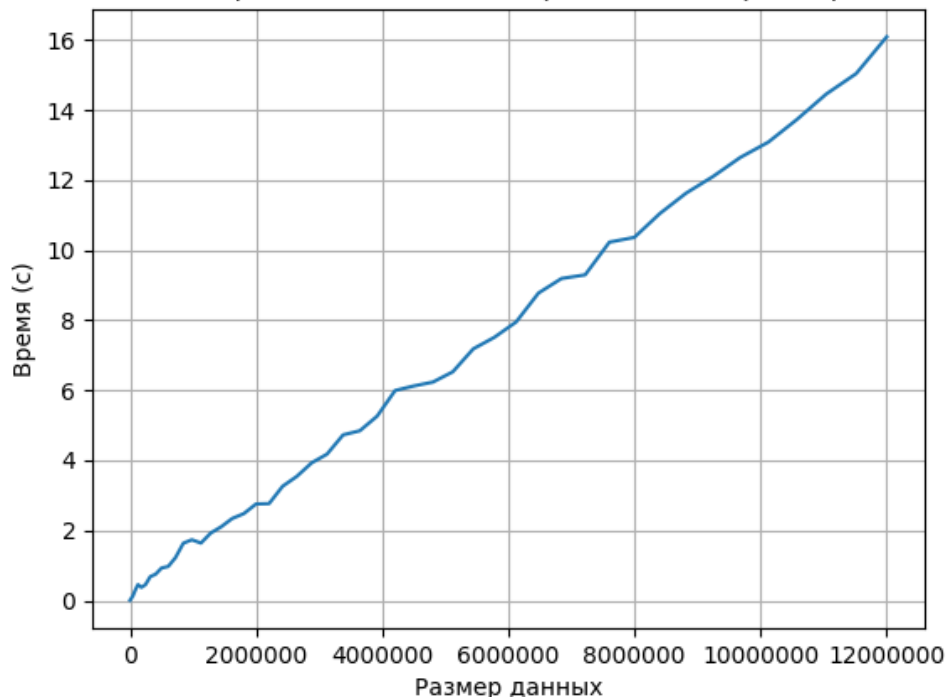
Для теста слияния, мы 20 раз засечем время слияния двух таблиц, будем увеличивать их размер за 100 и 50 элементов между измерениями.



Зависимость степенная.

Если построить график зависимости времени от произведения количества элементов данных первой таблицы и второй таблицы получаем следующий график.

Зависимость времени слияния от произведения размеров таблиц



Отсюда и доказательство что $O(n*m)$, где n размер первой, а m второй таблицы.

Использование файлового пространства:

Переиспользование файлового пространства

Создадим новый файл:

Добавим туда 100 000 записей:

-Величина файла: 4132KB

Удалим оттуда 100 000 записей:

-Величина файла: 4132KB

Добавим туда 100 000 записей:

-Величина файла: 4132KB

Ясно что файловое пространство переиспользуется.

Реальное использование файлового пространства

-Величина файла: 4132KB

-Количество записей в файле: 100 000

-Величина одной записи: 42B

-Величина метаданных блока: 16B

-Доступное для размещения элементов данных пространство блока: 4080B

-Количество блоков со злементами данных: $100000 * 42B / 4080KB = 1029.36 = 1030$

-1 Блок базы данных

-1 Блок схемы

-1 Первый свободный блок

-Количество всех блоков в файле:1033

-Величина всех блоков в файле в KB = $1033 * 4096B / 1024B = 4132KB$

Выводы:

Во время выполнения данной лабораторной работы, мне удалось создать модуль реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) соответствующий реляционным таблицам.