УНИВЕРСИТЕТ ИТМО

Факультет ПИИКТ

Отчёт по лабораторной работе №1 по дисциплине «Низкоуровневое програмирование»

Выполнил:
Чулаков К. Ф.

Группа: **P33102**

Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург 2023 г.

Задание:

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

Вариант:

Вариант: 1 (Документное дерево)

Особенности реализации:

Основная идея:

Выделение блоков фиксированного размера (PAGE_SIZE) для схем, документов и строк и поддержание их связанности.

Структуры данных:

```
typedef enum {DOUBLE, STRING, BOOLEAN, INT32} ELEMENT_TYPE;


typedef struct {
    ELEMENT_TYPE e_type;
    string* e_name;
} field;


typedef struct {
    uint32_t field_count;
    field* fields;
} schema;


typedef struct {
    field* e_field;
    union {
        int32_t int_data;
        double double_data;
```

```
        bool bool_data;
        string* string_data;
        string_part* string_split;
    };
} element;
struct document {
    uint32_t parentPage;
    uint32_t childPage;
    uint32_t prevBrotherPage;
    uint32_t nextBrotherPage;
    uint32_t prevCollectionDocument;
    uint32_t nextCollectionDocument;
    uint32_t collectionPage;

    struct document_data {
        uint32_t nextPage;
        uint32_t count;
        element* elements;
        document* nextDoc;
    } data;
};
```

Структуры запросов:

```
typedef struct {
    collection* col;
} create_schema_query;

typedef struct {
```

```
    string* collection;
} delete_schema_query;

typedef struct {
    string* collection;
} get_schema_query;

typedef struct {
    uint32_t parent_id;

} parent_ref;

typedef struct {
    parent_ref* parent; // nullable
    string* collection;
    document* doc;
} insert_query;

typedef struct {
    string* collection; // nullable
    complex_filter* filters; // only top level checks
} find_query;

typedef struct {
    find_query* find;
    element* elements;
} update_query;

//_________________result_________________
typedef enum {
```

```
    SCHEMA_RESULT_TYPE,
    SCHEMA_ARRAY_RESULT_TYPE,
    DOCUMENT_RESULT_TYPE,
    DOCUMENT_LIST_RESULT_TYPE,
    COLLECTION_RESULT_TYPE
} query_result_data_type;

typedef struct document_list document_list;
struct document_list{
    document* currDoc;
    uint32_t pageId;
    document_list* nxt;
};

typedef struct {
    query_result_data_type type;
    uint32_t pageId;
    union {
        schema* schema1;
        schema* schemas;
        element* element1;
        document_list* documents;
        collection* col;
    };
} query_result_data;

typedef enum {BOOL_RESULT_TYPE, ERR_RESULT_TYPE, DATA_RESULT_TYPE}
query_result_type;

typedef struct {
```

```
    query_result_type type;
    union {
        bool ok;
        string* err;
        query_result_data* data;
    };
} query_result;

typedef enum {
    CMP_EQ = 0,
    CMP_NEQ = 1,
    CMP_GT = 2,
    CMP_GTE = 3,
    CMP_LT = 4,
    CMP_LTE = 5,
    CMP_REGEX = 6
} CMP_TYPE;
typedef struct element_filter element_filter;
struct element_filter {
    CMP_TYPE type;
    element* el;
    element_filter* nxt;
};

typedef enum {
    OR_OPERATOR = 0,
    AND_OPERATOR
} OP_TYPE;
typedef struct operator_filter operator_filter;
typedef enum { ELEMENT_FILTER, OPERATOR_FILTER } FILTER_TYPE;
```

```
typedef struct complex_filter complex_filter;
struct operator_filter {
    complex_filter* flt1;
    OP_TYPE type;
    complex_filter* flt2;
};
struct complex_filter {
  union {
      operator_filter* op_filter;
      element_filter* el_filter;
  };
  FILTER_TYPE type;
};
```

Сборка:

cmake -B build && cd build && make

./test — запуск unit тестов (покрыты 90% функций)

CI сборки на Win11 https://github.com/kamilchulakov/llp_labs/blob/master/.github/workflows/makefile.yml

Пример unit теста:

```
void test_collection_update(db_handler* db) {
    print_running_test("test_collection_update");


    find_query find = {string_of("hex"), NULL};
    find.filters = malloc(sizeof(complex_filter));
    find.filters->type = ELEMENT_FILTER;
    find.filters->el_filter = create_element_filter(CMP_EQ, create_element(STRING, "big string"));
    find.filters->el_filter->el->string_data = string_of("small string");
    update_query query = {&find, create_element(STRING, "big string")};
```

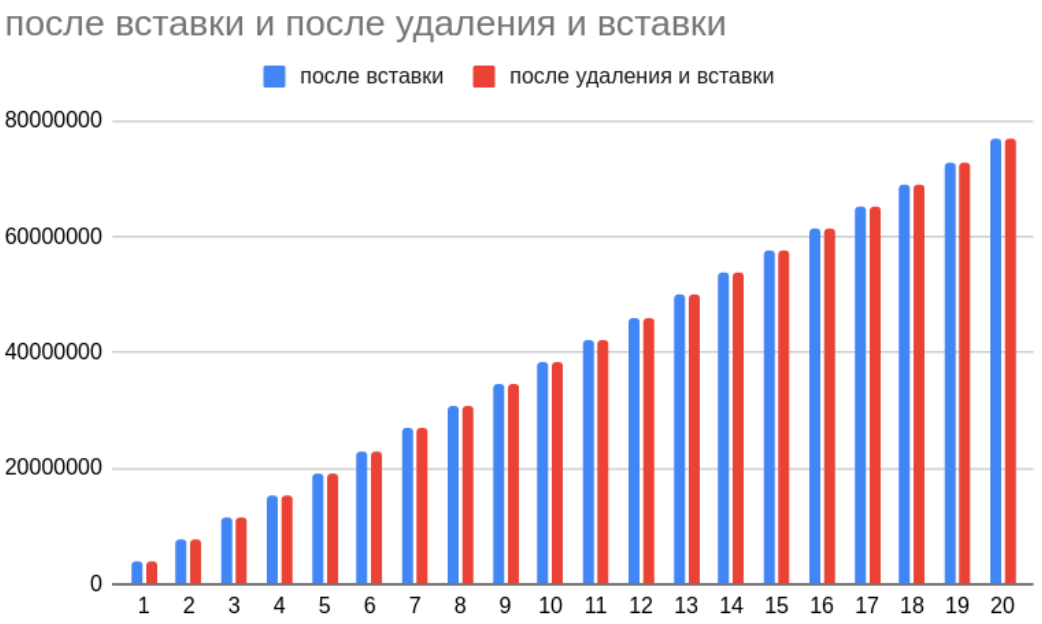
```
    query.elements->string_data = string_of("not small string");


    assert(db->pagerData->pageIdSeq == 12);
    assert(db->pagerData->lastStringPage == 11);
    assert(db->pagerData->firstFreeStringPageId == -1);
    assert_true(collection_update(db, &query).ok);
    element* el = create_element(STRING, "big string");
    el->string_data = string_of("not small string");
    assert_element_equals(get_document(db, 10)->data.elements, el);
    assert(db->pagerData->pageIdSeq == 12);
    assert(db->pagerData->firstFreeStringPageId == -1);
    assert(db->pagerData->lastStringPage == 11); // reused 11
    assert(get_page(db, 11)->prevPageId == 9);
    assert(get_page(db, 9)->nextPageId == 11);
}
```

Результаты тестирование:

Ветка с для тестирования - bench

На каждом шаге происходит вставка 5000 документов (4 поля, 1 — большая строка). Вставка работает такое же время O(1), а выборка (без учета отношений) с каждым шагом становится дольше, зависимость линейная => O(n).



На каждом шаге i происходит вставка 5000 документов, удаление всех элементов данных и вставка (i+1)*5000 документов. По графику видно, что выделенная память переиспользуется => размер файла данных пропорционален количеству фактически размещённых элементов данных.

Вывод:

Во время выполнения лабораторной работы получил понимание принципов организации хранения баз данных.