Университет ИТМО, мегафакультет КТиУ

Лабораторная работа №1

по low-level programming

Вариант «Реляционная бд»

Студент: Ингликова С.

Группа: P33312

Преподаватель: Кореньков Ю.

г. Санкт-Петербург

2022 г

Задание

Создать бд в одном файле, поддерживающую реляционную модель данных и отвечающую заданным критериям производительности.

Описание решения

Разработанная бд внутри устроена как связный список таблиц, каждая из которых представляет собой связный список записей (строк/рядов). Подобная структура позволяет соблюсти требования к производительности.

Страничная организация отсутствует.

У файла обязательно должен быть заголовок, где в числе прочего содержатся данные о фрагментации.

```
struct File_Header {
        uint16_t signature;
        uint16_t tables_number;
        uint32_t gap_sz;
        uint32_t first_gap_offset;
        uint32_t last_gap_offset;
        uint32_t first_table_offset;
        uint32_t last_table_offset;
};
```

Метаданные о таблице хранятся в формате: | Table_Header | table_name_str | Column_Header 1| column_name_str_1 | ...

Таким образом, они переменного размера (их размер зависи от названия таблицы и столбцов).

```
struct Table_Header {
        uint16_t columns_number;
        uint16_t table_metadata_size; // with column metadata
        uint32_t first_row_offset;
        uint32_t last_row_offset;
        uint32_t next_table_header_offset;
        uint32_t prev_table_header_offset;
        struct String_Metadata table_name_metadata;
};

struct Column_Header {
        enum DB_Data_Type data_type;
        struct String_Metadata column_name_metadata;
};
```

У каждого ряда (строки/записи) также есть свой заголовок, необходимый для поддержания двусвязного списка.

```
struct Row_Header {
        uint32_t next_row_header_offset;
        uint32_t prev_row_header_offset;
        uint16_t row_size;
};
```

Инсерт представляет итерирование по списку таблиц в поисках нужной (константа, тк число таблиц в сравнении с числом записей пренебрежимо мало и меняется редко) и добавление записи в конец двусвязнонго списка (константа).

При удалении записи превращаются в гэпы и размер пустого пространства в заголовке файла увеличивается. При обновлении записи та либо остается на прежнем месте, либо перемещается в конец (при увеличении ее размера).

Существует аналог операции вакууминга, допустающий гибкую настройку на уровне сеанса (работы приложения). Во-первых, пользовательможет задать максимальные значения фрагментации в процентном и абсолютном выражении. Команды, по результатам выполнения которых может образовываться пустое пространство (update, delete), принимают параметр, отвечающий за нормализацию.

```
enum Normalization {
        NORMALIZATION_RESTRICT,
        NORMALIZATION_ALLOW,
        NORMALIZATION_DO_FORCE // not looking at gap sz and rate
};
```

NORMALIZATION_RESTRICT - запрет нормализации файла при исполнении конкретной команды. Поскольку нормализация представляет собой перезапись файла, она замедляет исполнение конкретной команды, хотя в дальнейшем позитивно сказывается на производительности (обеспечивая максимально возможное попадание в буфер). Но если надо, чтобы update/delete исполнилась как можно быстрее, стоит нормализацию запретить. NORMALIZATION_ALLOW - после команды файл будет

перезаписан, если пороговые значения фрагментации достигнуты. NORMALIZATION_DO_FORCE - нормализация будет выполнена вне зависимости от показателей фрагментации.

Помимо команд нормализацию возможно провести при закрытии файла.

Чтение рядов осуществляется в буфер, исходный размер которого равен 4096 байт, однако может быть увеличен, если запись в буфер не помещается.

При сильных показателях фрагментации операции над данными выполняются все еще линейно, но за большее абсолютное время (тк снижено число попаданий в буфер). Поэтому рекомендуется иногда выполнять нормализацию и заранее настроить ее барьерные значения в зависимости от своих нужд.

Трансляция данных между форматами из оперативной памяти и внутренними форматами файла осуществляется функциями из файла format_translators.c, а непосредственной работой с файлом занимаетс file_handler.c (внезапно).

При операциях выборки, удаления, изменения к данным могут применяться фильтры (условия), которые могут быть составными, их разбор и применение осуществляется модулем filters.c.

Поддерживается исполнение джоин селектов неограниченной вложенности, которое осуществляется посредством рекурсивных вызовов и формирования древовидной структуры, состоящей из фрагментов строк и привязанной к Table_Chain_Result_Set. При этом фильтры применяются при работе с конкретной таблицей. Возможно (и рекомендовано) вводить ограничение на число выбираемых строк.

В решении используются максимально общие и примитивные инструменты языка С, что обеспечивает его кроссплатформенность.

Расход оперативной памяти, очевидно, не превышает фиксированного значения из-за того, что данные читаются в буфер, который может быть расширен лишь при работе со сверхгиганскими строками таблиц. А при исполнении селект запросов можно использовать ограничения на число строк, то есть регулировать потребление памяти.

Стуктуры данных

Представление данных

- Примитивные элементы данных:

```
struct String {
	uint32_t hash;
	uint16_t length;
	char* value;
};

enum DB_Data_Type {
	INT,
	FLOAT,
	BOOL,
	STRING
};

enum Boolean {
	FALSE,
	TRUE
};

union Data {
	struct String db_string;
	enum Boolean db_boolean;
	int32_t db_integer;
	float db_float;
};

struct Schema_Internals_Value {
	enum DB_Data_Type data_type;
	union Data value;
};

struct Data_Row_Node {
	struct String column_name;
	struct Schema_Internals_Value value;
	struct Data_Row_Node* next_node;
};
```

- Условия:

```
enum Condition_Relation {
	EQUALS,
	LESS,
	BIGGER,
	NOT_EQUALS
};

enum Condition_Chain_Relation {
	AND,
	OR
};

struct Simple_Condition {
	struct String column_name;
	enum Condition_Relation relation;
```

```
        struct Schema_Internals_Value right_part;
};

struct Condition;

struct Complex_Condition {
        struct Condition* left;
        struct Condition* right;
        enum Condition_Chain_Relation relation;
};

union Condition_Union {
        struct Complex_Condition complex_condition;
        struct Simple_Condition simple_condition;
};

struct Condition {
        uint8_t is_simple;
        union Condition_Union condition;
};
```

- Склеенные таблицы для селект запросов:

```
struct Join_Condition {
        uint32_t related_table_index;
        struct String related_table_column_name;
        struct String current_table_column_name;
};

struct Joined_Table {
        int32_t number_of_joined_tables;
        struct String* table_names;
        struct Join_Condition* join_conditions; // length =
number_of_joined_tables - 1
};
```

Запросы

- Структуры и функции для работы с файлом:

```
struct File_Handle {
        FILE* file;
        char* filename;
        float critical_gap_rate;
        uint32_t critical_gap_sz;
};

struct File_Handle* file_open_or_create(char* filename);
struct File_Handle* file_open_or_create_with_gap_rate(char* filename,
float critical_gap_rate);
```

```
struct File_Handle* file_open_or_create_with_gap_sz(char* filename,
uint32_t critical_gap_sz);
struct File_Handle* file_open_or_create_with_gap_rate_and_sz(char*
filename, float critical_gap_rate, uint32_t critical_gap_sz);
void file_close(struct File_Handle* f_handle, uint8_t normalize);
```

- Структуры и функции для работы со схемой данных:

```

struct Table_Schema {
	uint32_t number_of_columns;
	uint32_t column_array_size; // to calc if there is free space or
not
	struct Column_Info_Block* column_info;
};

struct Table_Schema table_schema_init();
int8_t table_schema_expand(struct Table_Schema* schema, char* column_name,
enum DB_Data_Type data_type);
int8_t table_create(struct File_Handle* f_handle, char* table_name, struct
Table_Schema schema);
int8_t table_delete(struct File_Handle* f_handle, char* table_name, enum
Normalization normalization);
```

- Структуры и функции запросов, работающие с элементами данных:

```

struct Joined_Table_Select {
	struct Joined_Table joined_table;
	uint32_t* number_of_columns_from_each_table;
	struct String** column_names;
	struct Condition** conditions;
};

struct Single_Table_Select {
	struct String table_name;
	uint32_t number_of_columns;
	struct String* column_names;
	struct Condition* condition;
};

union Select_Union {
	struct Single_Table_Select single_table_select;
	struct Joined_Table_Select joined_table_select;
};

struct Select {
	uint8_t is_single_table_select;
	union Select_Union query_details;
};

struct Update {
	struct String table_name;
	struct Data_Row_Node* new_data;
	struct Condition* condition;
```

```
};

struct Delete {
	struct String table_name;
	struct Condition* condition;
};

struct Insert {
	struct String table_name;
	struct Data_Row_Node* new_data; //one row
};

int32_t process_insert(struct File_Handle* f_handle, struct Insert
insert_command);
int32_t process_update(struct File_Handle* f_handle, struct Update
update_command, enum Normalization normalization);
int32_t process_delete(struct File_Handle* f_handle, struct Delete
delete_command, enum Normalization normalization);
struct Table_Chain_Result_Set* process_select_with_row_num(struct
File_Handle* f_handle, struct Select select_command, uint32_t
max_row_num);
struct Table_Chain_Result_Set* process_select(struct File_Handle*
f_handle, struct Select select_command);
struct Table_Chain_Result_Set* result_set_get_next(struct File_Handle*
f_handle, struct Table_Chain_Result_Set* result_set);
```

- Результат селекта:

```

struct Table_Chain_Result_Set {
	uint32_t rows_num;
	uint8_t probably_has_next;
	int32_t number_of_joined_tables;
	struct String* table_names;
	struct Join_Condition* join_conditions;
	uint32_t* cursor_offsets;
	struct Condition** conditions_on_single_tables;
	struct Table_Handle* tab_handles;
	void** table_metadata_buffers;
	struct Table_Row_Lists_Bunch* rows_chain;
	int32_t* number_of_selected_columns;
	struct String** column_names;

};
```

- Узел древовидной структуры, в которой хранятся результаты селекта:

```

struct Table_Row_Lists_Bunch {
	uint32_t local_rows_num;
	void* row_lists_buffer;
	uint32_t* row_starts_in_buffer;
	struct Table_Row_Lists_Bunch** row_tails;
};
```

Результаты тестирования

Адекватное сохранение таблиц и структуры

Создание модели из нескольких таблиц и заполнение ее сущностями можно посмотреть в файле tests.c.

В следующих функциях:

```

```
void prepare_short_test_schema(struct File_Handle* fh);
void test_select_on_short_test_schema(struct File_Handle* fh);
```

Тест, по сути, всего один, но зато какой! Аналог на sql и результат приведены ниже. В скобочках указаны названия таблиц, из которых взяты столбцы.

```

```
**---task---**

select all data about actual students having LLP in their curriculum:

**---sql---**

```
select * from student

join group on student.group_number = group.number

join curriculum on curriculum.code = group.curriculum_code

join cur_sub_relation on curriculum.code =
cur_sub_relation.curriculum_code

join subject on subject.code = cur_sub_relation.subject_code

where (subject.code == 'LLP')

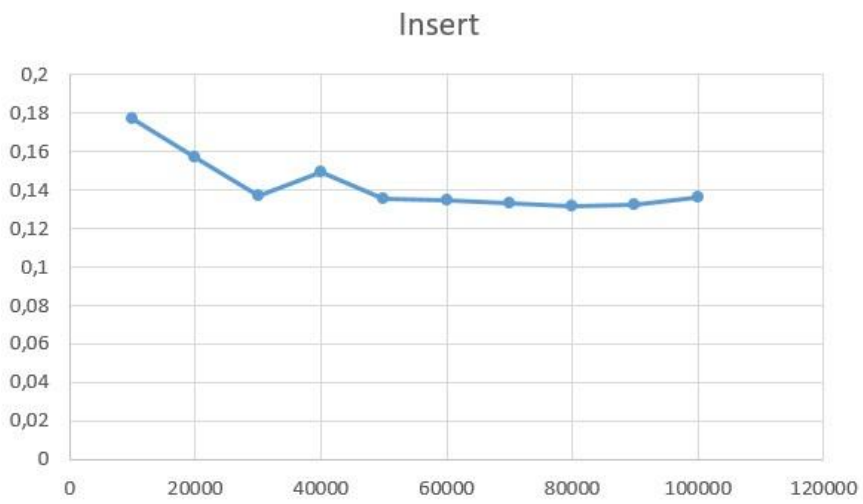
and (student.actual == TRUE)
```

JOINED TABLE

```
name(student)   group_number(student)   actual(student)
number(group)   year(group)   curriculum_code(group)
code(curriculum)   name(curriculum)
curriculum_code(cur_sub_relation)   subject_code(cur_sub_relation)
year(cur_sub_relation)   code(subject)   name(subject)

Inglikova   P33312   1   P33312   3   SEaCE_1   SEaCE_1   Software
engineering and... - 1   SEaCE_1   LLP   3   LLP   Low lvl programming

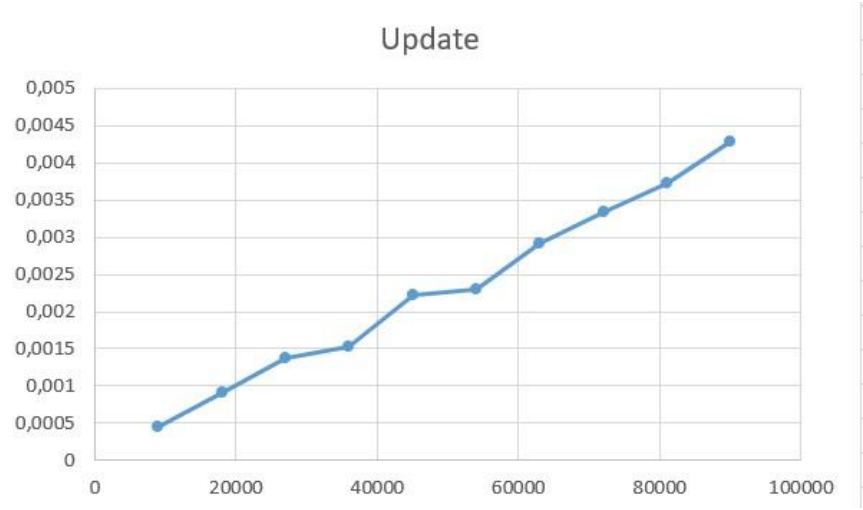
Erehinsky   P33312   1   P33312   3   SEaCE_1   SEaCE_1   Software
engineering and... - 1   SEaCE_1   LLP   3   LLP   Low lvl programming
```

```

# Производительность

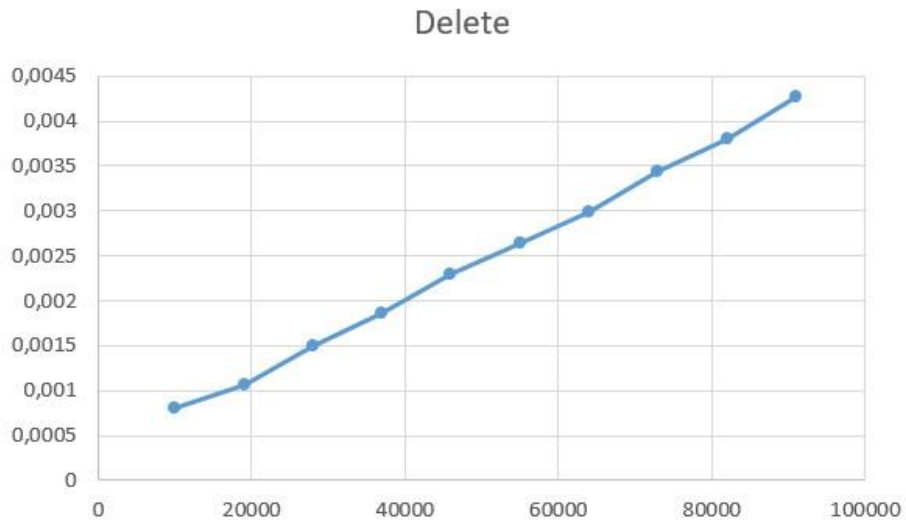
- Insert



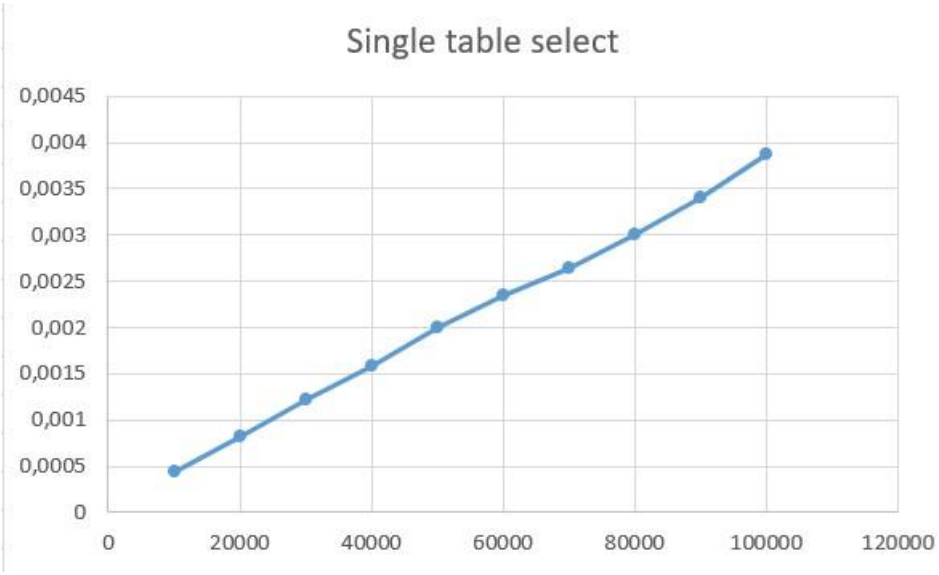

- Update



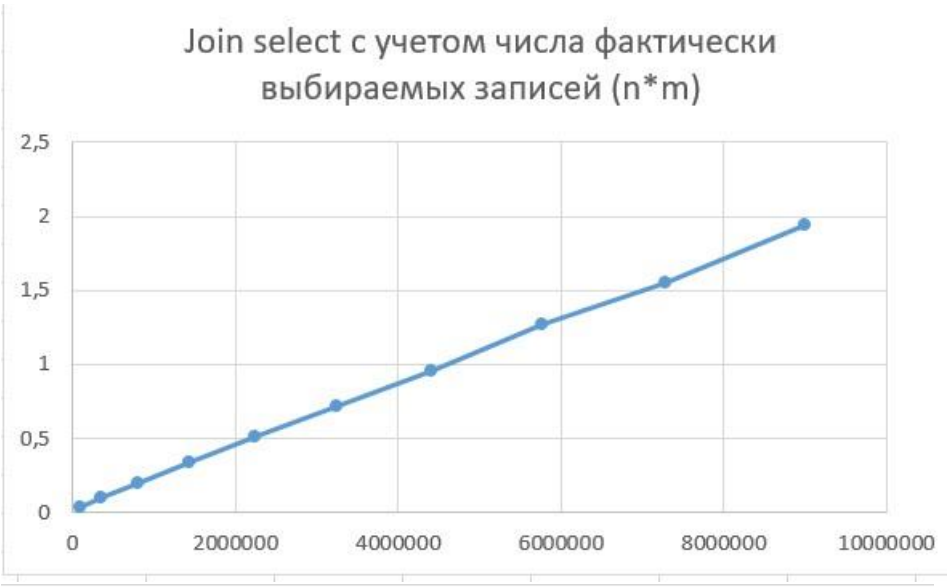

- Delete



- Single table select

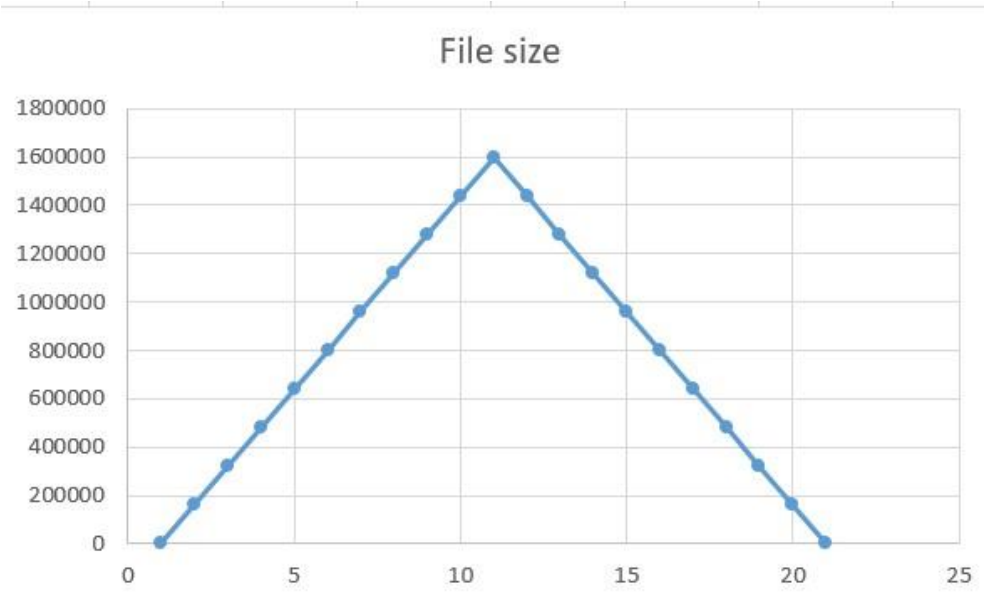



- Join select




## Размер файла

Сначала десятикратное добавление 10тыс записей, затем удаление 10тыс на каждой итерации с помощью подбора специфических условий. (При настроенных заранее параметрах максимально допустимой фрагментации и параметре NORMALIZATION_ALLOW в delete).




## Вывод

Берегите себя и своих близких от нервного срыва!