федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №1

по дисциплине «**Низкоуровневое программирование**»

Вариант 11

Автор: Кулаков Н. В.

Факультет: ПИиКТ

Группа: P33312

Преподаватель: Кореньков Ю.Д.

Санкт-Петербург 2022

Цели:

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

-rw-r--r-- 1 nikit nikit 11G дек 14 21:14 .db-very-large.bin

Задачи:

1) Спроектировать структуры данных для представления информации в оперативной памяти.

2) Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним.

3) Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:

- Добавление, удаление и получение информации о элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей
- Добавление нового элемента данных определённого вида
- Выборка набора элементов данных с учётом заданных условий и отношений со смежными элементами данных (по свойствам/полями/атрибутам и логическим связям соответственно).
- Обновление элементов данных, соответствующих заданным условиям.
- Удаление элементов данных, соответствующих заданным условиям.

4) Реализовать тестовую программу для демонстрации работоспособности решения.

5) Результаты тестирования по п.4 представить в составе отчёта. Включить графики на основе тестов, демонстрирующие амортизированные показатели ресурсоёмкости по п. 4.

Описание работы:

Программа представляет из себя библиотеку, предоставляющую возможность написания своих собственных запросов. Перейду к реализованным возможностям.

Создание, открытие, закрытие, удаление бд:

```c
#pragma once

struct dbms *dbms_create(const char *fname);
struct dbms *dbms_open(const char *fname);
void dbms_close(struct dbms **dbms_ptr);
void dbms_remove(struct dbms **dbms_ptr);
```

Создание, удаление таблицы на уровне dto; добавление колонок, подсчет колонок в таблице; ограничение на колонки (is_null, is_unique не работают в плане попыток записи в бд, однако используются на уровне фильтров запросов, об этом далее):

```c
enum dto_table_column_type {
    DTO_COLUMN_INT32 = 0,
    DTO_COLUMN_DOUBLE,
    DTO_COLUMN_STRING,
    DTO_COLUMN_BOOL
};

typedef struct dto_table_column_limits {
    bool is_null;
    bool is_unique;
} dto_table_column_limits;

struct dto_table *dto_table_construct(const char *name);
void dto_table_destruct(struct dto_table **table_ptr);
void dto_table_add_column(struct dto_table *table, const char *name,
                          const enum dto_table_column_type type,
                          const struct dto_table_column_limits lims);
int dto_table_column_cnt(const struct dto_table *table);
```

Создание, удаление таблиц, проверка существования таблицы:

```c

bool table_exists(struct dbms *dbms, const char *name);
bool table_create(struct dbms *dbms, struct dto_table *table);
bool table_drop(struct dbms *dbms, struct dto_table *table);
```

Создание списка строк на уровне dto, добавление строк в этот список (копируются указатели, а не сами данные):

```c

struct dto_row_list dto_row_list_construct();
void dto_row_list_destruct(struct dto_row_list *lst);
void dto_row_list_append(struct dto_row_list *lst, const void *row[]);
```

Вставка списка строк в бд (оптимизирована вставка сразу нескольких строк без перезагрузки блока):

```
int row_list_insert(struct dbms *dbms, const char *table_name,
                    struct dto_row_list *list);
```

Сейчас перейдем к запросам на select, update, delete. Сам запрос (plan) строится из нод, каждая из которых выполняет свою функцию. Ниже перечислены основные типы plan_nodes. Для формирования запросы мы оборачиваем ноды.

```
struct column_value {
  const char *column_name;
  const void *column_value;
};

struct plan_source *plan_source_construct(const void *table_name, struct dbms *dbms);

struct plan_select *plan_select_construct_move(void *parent_void,
                                               const char *table_name);

struct plan_update *plan_update_construct_move(void *parent_void, size_t size,
                                               struct column_value *arr);

struct plan_delete *plan_delete_construct_move(void *parent_void);

struct plan_cross_join *plan_cross_join_construct_move(void *parent_left,
                                                       void *parent_right);

struct plan_filter *plan_filter_construct_move(void *parent, void *filt);
```

Запросы пишутся в объектом lua стиле, ниже пример (можно было маллокнуть base, сделать типо приватный интерфейс для пользователя, однако не хочется лишний раз маллокать). Пользователь может вызывать функции из примера ниже. Можно было разрешить запускать эти функции только для терминальных нод, но думаю ничего страшного нет, если я разрешил для всех).

```
// plan_source {{{
struct plan_source {
  struct plan base;

  struct dbms *dbms;
  // iterator
  struct tp_iter *iter;

  INHERIT const struct plan_table_info *(*get_info)(void *self, size_t *size);
  INHERIT struct tp_tuple **(*get)(void *self);
  INHERIT bool (*end)(void *self);

  OVERRIDE bool (*next)(void *self);
  OVERRIDE void (*destruct)(void *self_ptr);
};
```

Вызов выглядит так:

```
// TEST select (passed)
{
  printf("Selection: \n");
  struct plan_select *select_table1 = plan_select_construct_move(
      plan_source_construct("table1", dbms), "temp-table1");

  size_t arr_size;
  const struct plan_table_info *table_info =
      select_table1->get_info(select_table1, &arr_size);

  select_table1->start(select_table1);

  while (!select_table1->end(select_table1)) {
    struct tp_tuple **tpt = select_table1->get(select_table1);
    print_table_tuple(tpt[0], table_info[0].dpt, table_info[0].col_info, dbms);

    select_table1->next(select_table1);
  }
  select_table1->destruct(select_table1);
}
```

Фильтры для запросов (where в sql) реализуются аналогичным образом. Пример абстрактного фильтра и const (константа колонки). Это приватных хедер, вызывается через plan_filter:

```
// fast {{{
struct fast {
  enum fast_type type;
  // result of calc
  void *res;
  enum table_column_type res_type;

  void (*compile)(void *self, size_t pti_size, struct plan_table_info *info_arr);
  void (*destruct)(void *self);

  void *(*calc)(void *self);// void * - returned result
  void (*pass)(void *self, const struct tp_tuple **tuple_arr);
};
// }}}

// fast_const {{{
struct fast_const {
  struct fast base;

  struct dbms *dbms;
};
```

Пользователю предоставляется возможность создавать и комбинировать фильтры, соответственно:

```c
struct fast_const *fast_const_construct(enum dto_table_column_type col_type,
                                        const void *value, struct dbms *dbms);

struct fast_column *fast_column_construct(const char *table_name,
                                          const char *column_name, struct dbms *dbms);

struct fast_unop *fast_unop_construct(void *parent, struct fast_unop_func *fuf,
                                      struct dbms *dbms);

struct fast_binop *fast_binop_construct(void *p_left, void *p_right,
                                        struct fast_binop_func *fbf, struct dbms *dbms);
```

fast_unop_func, fast_binop_func задаются разработчиком, так как в них осуществляетя доступ struct fast. Ниже представлены функции, которые реализованы, но можно конечно реализовать и больше:

```c
#define EXT_UNOP(name) extern struct fast_unop_func name
#define EXT_BINOP(name) extern struct fast_binop_func name

struct fast_unop;
struct fast_binop;

// fast_unop {{{
struct fast_unop_func {
  void (*func)(struct fast_unop *self, void *arg);
  enum table_column_type ret_type;
};

EXT_UNOP(BOOL_NOT);
// }}}

// fast_binop {{{
struct fast_binop_func {
  void (*func)(struct fast_binop *self, void *arg1, void *arg2);
  enum table_column_type ret_type;
};

EXT_BINOP(DOUBLE_LARGER);
EXT_BINOP(DOUBLE_EQUALS);

EXT_BINOP(INT32_EQUALS);
EXT_BINOP(INT32_LARGER);

EXT_BINOP(BOOL_OR);
EXT_BINOP(BOOL_AND);

EXT_BINOP(STRING_EQUALS);
```

Примеры группировки через фильтры и написание полноценного запроса, конкретнее update:

```
struct plan_source *so1 = plan_source_construct("table1", dbms);
// struct plan_source *so2 = plan_source_construct("table2", dbms);
// struct plan_source *so3 = plan_source_construct("table1", dbms);
// struct plan_cross_join *j1 = plan_cross_join_construct_move(so1, so2);
// struct plan_cross_join *j2 = plan_cross_join_construct_move(j1, so3);

struct fast_column *fc_name = fast_column_construct("table1", "name", dbms);

struct fast_const *fc_nikit =
    fast_const_construct(COLUMN_TYPE_STRING, "nikita", dbms);

struct fast_column *fc_weight = fast_column_construct("table1", "weight", dbms);
struct fast_binop *fb1 =
    fast_binop_construct(fc_name, fc_nikit, &STRING_EQUALS, dbms);

const double min_val = 80;
struct fast_const *fc_weight80 =
    fast_const_construct(COLUMN_TYPE_DOUBLE, &min_val, dbms);

struct fast_binop *fc_max_wight =
    fast_binop_construct(fc_weight, fc_weight80, &DOUBLE_LARGER, dbms);

struct fast_binop *fc_and =
    fast_binop_construct(fc_max_wight, fb1, &BOOL_AND, dbms);

// struct plan_filter *f = plan_filter_construct_move(j1, fb1);
struct column_value *cols = malloc(sizeof(struct column_value));
cols[0].column_name = "name";
cols[0].column_value = "perestaralsiya";

struct plan_filter *fi = plan_filter_construct_move(so1, fc_and);
struct plan_update *se = plan_update_construct_move(fi, 1, cols);

size_t ti_size;
struct plan_table_info ti = se->get_info(se, &ti_size)[0];

se->start(se);
while (!se->end(se)) {
  tp_tuple *tpt = se->get(se)[0];
  print_table_tuple(tpt, ti.dpt, ti.col_info, dbms);
  se->next(se);
}
se->destruct(se);
```

запрос аналогичен sql:

update from table1 set table1.name = «perestaralsiya» where table1.name = «nikita» and table1.weight > 80;

print_table_tuple — функция в util/printers.h

Другие примеры запросов и основных функций можно найти в директории tests (тесты запускаются через gtest), а в app/ для select, update, delete и фильтров.

Аспекты реализации:

влфоылаофыа

Результаты:

dfjasfkjsafjsaf

Выводы:

jdjsafjasfjsajf