Федеральное государственное автономное образовательное учреждение высшего образования "Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная №2
по дисциплине
'Низкоуровневое программирование'

Выполнил Студент группы P33102
Лапин Алексей Александрович
Преподаватель:
Кореньков Юрий Дмитриевич

г. Санкт-Петербург
2023г.

Содержание

1 Цель:

Выданный вариант - 5 (AQL - ArangoDb Query Language)

Использовать средство синтаксического анализа по выбору, реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных. Должна быть обеспечена возможность описания команд создания, выборки, модификации и удаления элементов данных.

2 Задачи:

1. Изучить выбранное средство синтаксического анализа
   - Средство должно поддерживать программный интерфейс совместимый с языком C
   - Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка
   - Средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек
   - Средство может быть реализовано с нуля, в этом случае оно должно быть основано на обобщённом алгоритме, управляемом спецификацией
2. Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа
   (a) При необходимости добавления новых конструкций в язык, добавить нужные синтаксические конструкции в спецификацию (например, сравнения в GraphQL)
   (b) Язык запросов должен поддерживать возможность описания следующих конструкций: порождение нового элемента данных, выборка, обновление и удаление существующих элементов данных по условию
      - Условия
        – На равенство и неравенство для чисел, строк и булевских значений
        – На строгие и нестрогие сравнения для чисел
        – Существование подстроки
      - Логическую комбинацию произвольного количества условий и булевских значений
      - В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
      - Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
      - Поддержка арифметических операций и конкатенации строк не обязательна
   (c) Разрешается разработать свой язык запросов с нуля, в этом случае необходимо показать отличие основных конструкций от остальных вариантов (за исключением типичных выражений типа инфиксных операторов сравнения)
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов

(a) Программный интерфейс модуля должен принимать строку с текстом запроса и возвращать структуру, описывающую дерево разбора запроса или сообщение о синтаксической ошибке

(b) Результат работы модуля должен содержать иерархическое представление условий и других выражений, логически представляющие собой иерархически организованные данные, даже если на уровне средства синтаксического анализа для их разбора было использовано линейное представление

4. Реализовать тестовую программу для демонстрации работоспособности созданного модуля, принимающую на стандартный ввод текст запроса и выводящую на стандартный вывод результирующее дерево разбора или сообщение об ошибке

5. Результаты тестирования представить в виде отчёта, в который включить:

   (a) В части 3 привести описание структур данных, представляющих результат разбора запроса

   (b) В части 4 описать, какая дополнительная обработка потребовалась для результата разбора, представляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля

   (c) В части 5 привести примеры запросов для всех возможностей из п.2.b и результирующий вывод тестовой программы, оценить использование разработанным модулем оперативной памяти

2.1 Выполнение

Для создания лексического анализатора использовался **flex**. Для создания синтаксического анализатора использовался **bison**. Для завершения запроса надо вывести символ EOF (Ctrl+D в терминале).

2.2 Структуры данных

```
  /* Типы вершин AST */
  enum ntype
/* Абстрактное дерево */

/* Кеширование */
typedef struct caching{
    file_t file;
    size_t size, used, max_used, capacity;
    uint32_t* usage_count;
    time_t* last_used;
    void** cached_page_ptr;
    char* flags;
} caching_t;

/* Переиспользование страниц */
typedef struct pager{
    caching_t ch;
    int64_t deleted_pages; // index of parray page with deleted pages
} pager_t;

```

```c
/* Связанные страницы */
typedef struct linked_page{
    int64_t next_page;
    int64_t page_index;
    int64_t mem_start;
} linked_page_t;

/* Пул страниц */

/* Чанк */
typedef struct chunk {
    linked_page_t lp_header;
    int64_t page_index;
    int64_t capacity;
    int64_t num_of_free_blocks;
    int64_t num_of_used_blocks;
    int64_t next;
    int64_t prev_page;
    int64_t next_page;
} chunk_t;

/* Менеджер пула */
typedef struct page_pool {
    linked_page_t lp_header;
    int64_t current_idx;
    int64_t head;
    int64_t tail;
    int64_t block_size;
    int64_t wait; // parray index
} page_pool_t;

/* Связанные блоки */
typedef struct linked_block{
    chblix_t next_block;
    chblix_t prev_block;
    chblix_t chblix;
    char flag;
    int64_t mem_start;
} linked_block_t;

  /* Поле схемы */
typedef struct field{
  linked_block_t lb_header;
  char name[MAX_NAME_LENGTH];
  datatype_t type;
  uint64_t size;
  uint64_t offset;
} field_t;

/* Схема таблицы */
typedef struct schema{
  page_pool_t ppl_header;
  int64_t slot_size;
```

```
} schema_t;

/* Таблица */
typedef struct table {
    page_pool_t ppl_header;
    int64_t schidx; //schema index
    char name[MAX_NAME_LENGTH];
} table_t;

/* Типы данных */
typedef enum datatype {DT_UNKNOWN = -1,
                       DT_INT = 0,
                       DT_FLOAT = 1,
                       DT_VARCHAR = 2,
                       DT_CHAR = 3,
                       DT_BOOL = 4} datatype_t;
```

3 Описание работы:

3.1 Публичный интерфейс:

3.1.1 Таблица

```

/**
* @brief        Initialize table and add it to the metatable
* @param[in]    db: pointer to db
* @param[in]    name: name of the table
* @param[in]    schema: pointer to schema
* @return       index of the table on success, TABLE_FAIL on failure
*/
table_t* tab_init(db_t* db, const char* name, schema_t* schema);
/**
 * @brief        Get row by value in column
 * @param[in]    db: pointer to db
 * @param[in]    table: pointer to the table
 * @param[in]    schema: pointer to the schema
 * @param[in]    field: pointer to the field
 * @param[in]    value: pointer to the value
 * @param[in]    type: type of the value
 * @return       chblix_t of row on success, CHBLIX_FAIL on failure
 */
chblix_t tab_get_row(db_t* db,
                     table_t* table,
                     schema_t* schema,
                     field_t* field,
                     void* value,
                     datatype_t type);
/**
* @brief        Print table
* @param[in]    db: pointer to db
* @param[in]    table: index of the table
```

```
30 * @param[in]   schema: pointer to the schema
31 */
32 void tab_print(db_t* db,
33              table_t* table,
34              schema_t* schema);
35 /**
36 * @brief       Inner join two tables
37 * @param[in]   db: pointer to db
38 * @param[in]   left: pointer to the left table
39 * @param[in]   left_schema: pointer to the schema of the left table
40 * @param[in]   right: pointer to the right table
41 * @param[in]   right_schema: pointer to the schema of the right table
42 * @param[in]   join_field_left: join field of the left table
43 * @param[in]   join_field_right: join field of the right table
44 * @param[in]   name: name of the new table
45 * @return      pointer to the new table on success, NULL on failure
46 */
47 int64_t tab_join(
48         db_t* db,
49         int64_t leftidx,
50         int64_t rightidx,
51         const char* join_field_left,
52         const char* join_field_right,
53         const char* name);
54 /**
55 * @brief       Select row form table on condition
56 * @param[in]   db: pointer to db
57 * @param[in]   sel_table: pointer to table from which the selection
      is made
58 * @param[in]   sel_schema: pointer to schema of the table from which
      the selection is made
59 * @param[in]   select_field: the field by which the selection is
      performed
60 * @param[in]   name: name of new table that will be created
61 * @param[in]   condition: comparison condition
62 * @param[in]   value: value to compare with
63 * @param[in]   type: the type of value to compare with
64 * @return      pointer to new table on success, NULL on failure
65 */
66 table_t* tab_select_op(db_t* db,
67         table_t* sel_table,
68         schema_t* sel_schema,
69         field_t* select_field,
70         const char* name,
71         condition_t condition,
72         void* value,
73         datatype_t type);
74 /**
75 * @brief       Drop a table
76 * @param[in]   db: pointer to db
77 * @param[in]   table: pointer of the table
78 * @return      PPL_SUCCESS on success, PPL_FAIL on failure
79 */
```

```
80 int tab_drop(db_t* db, table_t* table);
81
82 /**
83  * @brief       Update row in table
84  * @param[in]   db: pointer to db
85  * @param[in]   table: pointer to table
86  * @param[in]   schema: pointer to schema
87  * @param[in]   field: pointer to field
88  * @param[in]   condition: comparison condition
89  * @param[in]   value: value to compare with
90  * @param[in]   type: the type of value to compare with
91  * @param[in]   row: pointer to new row which will replace the old
      one
92  * @return
93  */
94 int tab_update_row_op(db_t* db,
95                       table_t* table,
96                       schema_t* schema,
97                       field_t* field,
98                       condition_t condition,
99                       void* value,
100                      datatype_t type,
101                      void* row);
102
103 /**
104  * @brief       Update element in table
105  * @param[in]   db: pointer to db
106  * @param[in]   tablix: index of the table
107  * @param[in]   element: element to write
108  * @param[in]   field_name: name of the element field
109  * @param[in]   field_comp: name of the field compare with
110  * @param[in]   condition: comparison condition
111  * @param[in]   value: value to compare with
112  * @param[in]   type: the type of value to compare with
113  * @return      TABLE_SUCCESS on success, TABLE_FAIL on failure
114  */
115
116 int tab_update_element_op(db_t* db,
117                           int64_t tablix,
118                           void* element,
119                           const char* field_name,
120                           const char* field_comp,
121                           condition_t condition,
122                           void* value,
123                           datatype_t type);
124 /**
125 * @brief       Delete row from table
126 * @param[in]   db: pointer to db
127 * @param[in]   table: pointer to table
128 * @param[in]   schema: pointer to schema
129 * @param[in]   field_comp: pointer to field to compare
130 * @param[in]   condition: comparison condition
131 * @param[in]   value: value to compare with
```

```
132 * @return      TABLE_SUCCESS on success, TABLE_FAIL on failure
133 */
134 int tab_delete_op(db_t* db,
135                   table_t* table,
136                   schema_t* schema,
137                   field_t* comp,
138                   condition_t condition,
139                   void* value);
140 /**
141 * @brief       Insert a row
142 * @param[in]   table: pointer to table
143 * @param[in]   schema: pointer to schema
144 * @param[in]   src: source
145 * @return      chblix_t of row on success, CHBLIX_FAIL on failure
146 */
147
148 chblix_t tab_insert(table_t* table, schema_t* schema, void* src)
149
150 /** @brief       Create a table on a subset of fields
151 *  @param[in]   db: pointer to db
152 *  @param[in]   table: pointer to table
153 *  @param[in]   schema: pointer to schema
154 *  @param[in]   fields: pointer to fields
155 *  @param[in]   num_of_fields: number of fields
156 *  @param[in]   name: name of the new table
157 *  @return      pointer to new table on success, NULL on failure
158 */
159
160 table_t* tab_projection(db_t* db,
161                         table_t* table,
162                         schema_t* schema,
163                         field_t* fields,
164                         int64_t num_of_fields,
165                         const char* name);
166
167 /**
168 * @brief       For each element specific column in a table
169 * @param[in]   table: pointer to the table
170 * @param[in]   chunk: chunk
171 * @param[in]   chblix: chblix of the row
172 * @param[in]   element: pointer to the element, must be allocated
    before calling this macro
173 * @param[in]   field: pointer to field of the element
174 */
175
176 #define tab_for_each_element(table, chunk, chblix, element, field)
177 /**
178  * @brief       For each element specific column in a table
179  * @param[in]   table: pointer to the table
180  * @param[in]   chunk: chunk
181  * @param[in]   chblix: chblix of the row
182  * @param[in]   row: pointer to the row, must be allocated before
     calling this macro
```

```
 * @param[in]   schema: pointer to the schema
 */

#define tab_for_each_row(table, chunk, chblix, row, schema)
```

3.1.2 Схема

```
  /**
   * @brief       Delete a schema
   * @param[in]   schidx: index of the schema
   * @return      SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
   */

 #define sch_delete(schidx)

 /**
  * @brief      Load field
  * @param[in]  schidx: index of the schema
  * @param[in]  fieldix: index of the field
  * @param[out] field: pointer to the field
  * @return     LB_SUCCESS on success, LB_FAIL on failure
  */

#define sch_field_load(schidx, fieldix, field)

/**
 * @brief      Update field
 * @param[in]  schidx: index of the schema
 * @param[in]  fieldix: index of the field
 * @param[out] field: pointer to the field
 * @return     LB_SUCCESS on success, LB_FAIL on failure
 */

#define sch_field_update(schidx, fieldix, field)

#define sch_add_int_field(schema, name) sch_add_field((schema),
   name, DT_INT, sizeof(int64_t))
#define sch_add_char_field(schema, name, size)
   sch_add_field((schema), name, DT_CHAR, size)
#define sch_add_varchar_field(schema, name) sch_add_field((schema),
   name, DT_VARCHAR, sizeof(vch_ticket_t))
#define sch_add_float_field(schema, name) sch_add_field((schema),
   name, DT_FLOAT, sizeof(float))
#define sch_add_bool_field(schema, name) sch_add_field((schema),
   name, DT_BOOL, sizeof(bool))

/**
 * @brief       For each field in a schema
 * @param[in]   sch: pointer to the schema
 * @param[in]   chunk: chunk
 * @param[in]   field: pointer to the field
 * @param[in]   chblix: chblix of the row
 * @param[in]   schidx: index of the schema
```

```
 */

#define sch_for_each(sch,chunk, field, chblix, schidx)
/**
 * @brief       Initialize a schema
 * @return      pointer to schema on success, NULL on failure
 */
void* sch_init(void);
/**
 * @brief       Add a field
 * @param[in]   schema: pointer to schema
 * @param[in]   name: name of the field
 * @param[in]   type: type of the field
 * @param[in]   size: size of the type
 * @return      SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
 */
int sch_add_field(schema_t* schema, const char* name, datatype_t
   type, int64_t size);
/**
 * @brief       Get a field
 * @param[in]   schema: pointer to the schema
 * @param[in]   name: name of the field
 * @param[out]  field: pointer to destination field
 * @return      SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
 */
int sch_get_field(schema_t* schema, const char* name, field_t*
   field);
/**
 * @brief       Delete a field
 * @warning     This function delete field, but dont touch offsets
    in other fields in schema.
 * @param[in]   schema: pointer to schema
 * @param[in]   name: name of the field
 * @return      SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
 */
int sch_delete_field(schema_t* schema, const char* name);
```

3.1.3 База данных

```
/**
 * @brief       Initialize database
 * @param[in]   filename: name of the file
 * @return      pointer to database on success, NULL on failure
 */
 void* db_init(const char* filename);
 /**
 * @brief       Close database
 * @return      DB_SUCCESS on success, DB_FAIL on failure
 */
int db_close(void);
/**
 * @brief       Drop database
 * @return      DB_SUCCESS on success, DB_FAIL on failure
```

```
15  */
16 int db_drop(void);
```

3.2 Модули:

Директория core - содержит модули, строящие абстракции над работой с файлом, вводом-выводом и т.п.

1. file - осуществляет базовую работу с файлом, при использовании одной страницы.
2. caching - кеширует страницы, освобождает их по мере заполнения оперативной памяти.
3. pager - хранит очередь страниц, готовых для переиспользования.
4. liked_pages - реализация связанных страниц.
5. page_pool - реализация пула страниц.
6. linked_blocks - реализация связанных блоков.

Директория backend - содержит модули, реализовывающие работу с реляционной базой данных.

1. schema - реализация операций с схемой таблиц
2. table_base - базовые операции с таблицами (нужны для других модулей)
3. matatab - таблица с метаданными (название и индекс страницы) других таблиц.
4. varchar_mgr - пул хранящий строки произвольной длинны и выдающий номерки для их поиска в нем.
5. db - модуль выполняющий операции на старте (инициализация файла, метатаблицы, varchar_mgr), и в завершении работы программы (закрытие файла, удаление файла). Также хранит индекс метатаблицы и тп.
6. data-types - поддерживаемые типы данных.
7. comparator - сравнение поддерживаемых типов.

4 Аспекты реализации:

1. Данные хранятся в страницах стандартного размера для операционной системы (обычно 4 кб).
2. Странницы кешируются и освобождаются по необходимости. Для этого создана map ключем в которой является номер страницы, а значением указатель на область памяти, статус-флаги, последние время использования, количество использования. Освобождение страницы из конца файла удаляет страницу. Освобождение страницы из середины файла ставит её в очередь не переиспользование при следующем запросе на аллокацию.

3. Чтобы поддерживать возможность добавление единиц данных больше размера страницы, страницы могут расширяться путем связывания linkedlist с новыми страницами. Для обращения и удаления этих страниц используется номер первой входящей в список страницы.

4. Чтобы выдавать элементы(строки) за O(1) на страницах построен pool. Pool состоит из chunks размером в одну связную страницу, chunks связаны между собой двусвязным linkedlist. При отдаче всех блоков из chunk мы переставляем указатель на следующий. Когда в chunk освобождается блок, то он ставится в очередь на использование, когда в текущем chunk закончится место. Если в chunk освобождены все строки, то страница полностью удаляется.

5. Чтобы поддерживать возможность записывать в блоки данные произвольной длинны, блоки могут расшириться связываясь с помощью linkedlist.

6. Схема таблицы построена поверх пула страниц, а поля являются блоками этого пула.

7. Таблицы строятся поверх своих пулов страниц, строки являются блоками пула.

5 Выводы:

Как видно из графика вставка работает за O(1), скочки на графике связаны скорее всего с аллокацией дополнительных страниц, как для хранения строк, так и для хранения очередей.

Выборка работает за O(n), так как мы проходим по всем строкам. Есть одна точка, которая сильно отличается от остальных, считаю её выбросом.

Удаление работает за O(n), так как мы проходим по всем строкам.

Обновление работает за O(n), так как мы проходим по всем строкам.

Размер файла растет линейно, так как мы добавляем новые страницы.

Использование оперативной памяти O(1), так как мы загружаем страницы, когда нам они нужны и освобождаем, когда уже нет.

Что я узнал, чему научился:

1. Написал свою первую большую программу на C, больше прогрузился в язык, в работу с ним.

2. Погрузился в то как работают базы данных изнури, написал свою.

3. Узнал новые алгоритмы и структуры данных, научился сам их реализовывать.

4. Научился работать с файлами, работать с памятью, работать с потоками ввода-вывода.

5. Научился писать переносимые приложения под разные операционные системы: Linux, Windows, MacOS.