



Факультет программной инженерии и компьютерной техники  
Направление подготовки: Информатика и вычислительная техника  
Дисциплина «Низкоуровневое программирование»

## Лабораторная работа №2

Вариант 5

Выполнил:

*Богатов А. С.*

*P33302*

Преподаватель

*Кореньков Ю.Д.*

г. Санкт-Петербург, 2023 г.

### Задание:

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.

Языка запросов по варианту - AQL.

### Используемые средства:

- Flex - токенизация
- Bison - парсинг

### Разработанный модуль:

В токенайзере для Flex предусмотрены возможные термы в AQL для базовых операций по заданию, а также обработка ошибочных запросов.

%%

```
"for"      {return FOR;}
"insert"    {return INSERT;}
"update"    {return UPDATE;}
"remove"    {return REMOVE;}
"in"        {return IN;}
"filter"    {return FILTER;}
"with"      {return WITH;}
"return"    {return RETURN;}

"create"    {return CREATE;}
"drop"      {return DROP;}

"&&"       {yyval.logic_op_type = 1; return AND;}
"||"        {yyval.logic_op_type = 2; return OR;}

">"        {yyval.cmp_op_type = 1; return CMP;}
">="       {yyval.cmp_op_type = 2; return CMP;}
"<"        {yyval.cmp_op_type = 3; return CMP;}
"<="       {yyval.cmp_op_type = 4; return CMP;}
"=="       {yyval.cmp_op_type = 5; return CMP;}
"!="       {yyval.cmp_op_type = 6; return CMP;}

"string"    {yyval.type = 1; return TYPE;}
"int"       {yyval.type = 2; return TYPE;}
"float"     {yyval.type = 3; return TYPE;}
```

```

"bool"      {yyval.type = 4; return TYPE;}
"true"      {yyval.boolean_value = 1; return BOOL;}
"false"     {yyval.boolean_value = 0; return BOOL;}

"("         {return START_PARENTHESIS;}
")"         {return CLOSE_PARENTHESIS;}
"{"         {return START_BRACKET;}
"}"         {return CLOSE_BRACKET;}
";"         {return SEMICOLON;}
":"         {return COLON;}
"."         {return DOT;}
","         {return COMMA;}
"\"         {return QUOTE;}

{word}      {
    sscanf(yytext, "%s", yyval.string);
    return (STR);
}
{int}       {
    yyval.number_value = atoi(yytext);
    return (INT);
}
{float}      {
    yyval.float_num_value = atof(yytext);
    return (FLOAT);
}

[ \t]       {
}
[\n]        {
}
.           {
    print_error(yytext);
    return (UNEXPECTED_TOKEN);
}

%%

```

На основе результатов токенизации и парсинга запроса обращаемся в основной модуль, который создает структуры узла синтаксического дерева.

```

struct ast_node {
    ast_node_type type;
    fields fields_one;

```

```
fields fields_two;
ast_node* first;
ast_node* second;
ast_node* third;
};
```

Узел содержит указание на тип операции/объекта запроса, для корректного отображения в текстовый вид, ссылки на поля, относящиеся к запросу, а также ссылки на другие узлы - имеем три ссылки как максимально поддерживаемое число под узлов (используется для отображения дополнений к основному запросу - соединения таблиц, фильтров и т.п.).

Функции по созданию узлов вызываются парсером, сгенерированным Bison:

```
%%

ast_root:
|   ast_root query SEMICOLON { output($2); printf("$> "); }
;

terminal:
|   TYPE { $$ = new_type($1); }
|   INT { $$ = new_number($1); }
|   FLOAT { $$ = new_float_number($1); }
|   BOOL { $$ = new_bool($1); }
|   QUOTE string_list QUOTE { $$ = new_string($2, NULL); }
;

query:
|   select_query
|   insert_query
|   delete_query
|   drop_query
|   update_query
|   create_query
;

select_query:
|   select_simple
|   select_with_filter
|   select_join_simple
|   select_join_with_filter
```

```

;
select_simple:
|   FOR column_name IN STR RETURN STR { $$ = new_select($4, NULL,
NULL, NULL, NULL); }
;
select_with_filter:
|   FOR column_name IN STR FILTER filter_condition RETURN STR { $$
= new_select($4, $6, NULL, NULL, NULL); }
;

select_join_simple:
|   FOR column_name IN STR FOR column_name IN STR RETURN STR ","
STR { $$ = new_select($4, NULL, $8, $6, $2); }
;

select_join_with_filter:
|   FOR column_name IN STR FOR column_name IN STR FILTER
filter_condition RETURN STR "," STR { $$ = new_select($4, $10, $8,
$6, $2); }
;

update_query:
|   update_simple
|   update_with_filter
;
update_simple:
|   FOR column_name IN STR UPDATE STR WITH "{" values_list "}" IN
STR { $$ = new_update($4, NULL, $9); }
;
update_with_filter:
|   FOR column_name IN STR FILTER filter_condition UPDATE STR WITH
 "{" values_list "}" IN STR { $$ = new_update($4, $6, $11); }
;

delete_query:
|   delete_simple
|   delete_with_filter
;
delete_simple:
|   FOR column_name IN STR REMOVE STR IN STR { $$ = new_delete($4,
NULL); }
;

```

```

delete_with_filter:
|   FOR column_name IN STR FILTER filter_condition REMOVE STR IN
STR { $$ = new_delete($4, $6); }
;

insert_query:
|   INSERT "{" values_list "}" IN STR { $$ = new_insert($6, $3); }
;

create_query:
|   CREATE STR "{" values_list "}" { $$ = new_create($2, $4); }
;

drop_query:
|   DROP STR { $$ = new_drop($2); }
;

values_list:
|   values_list "," values_pair { $$ = new_list($3, $1); }
|   values_pair { $$ = new_list($1, NULL); }
;

values_pair:
|   STR ":" terminal { $$ = new_pair($1, $3); }
;

filter_condition:
|   filter_condition "&&" filter_condition { $$ = new_where($2,
$1, $3); }
|   filter_condition "||" filter_condition { $$ = new_where($2,
$1, $3); }
|   "(" filter_condition ")" { $$ = $2; }
|   logic_statement
;

logic_statement:
|   column CMP terminal { $$ = new_compare($2, $1, $3); }
|   terminal CMP column { $$ = new_compare(switch_cmp_mode($2),
$1, $3); }
|   column CMP column { $$ = new_compare($2, $1, $3); }
;

```

```

column:
|   STR DOT STR { $$ = new_name($1, $3); }
;

column_name:
|   STR { $$ = new_name(NULL, $1); }
;

string_list:
|   string_list STR { $$ = new_string($1, $2); }
|   STR { $$ = new_string(NULL, $1); }
;

%left OR;
%left AND;

%%

```

Используем %left для указания ассоциативности в логических запросах.

Арифметические операторы не поддерживаются.

Модули string\_utils.c и output.c отвечают за перевод структуры узла AST в текстовый вид и вывод результата пользователю.

### Примеры:

Простая выборка: FOR a IN table\_a RETURN a;

```

FOR {
    KEYS:
        KEY {
            row_alias: a
        }
    TABLE: table_a
    JOIN: NULL
    FOR: NULL
    FILTER: NULL
}

```

Выборка с условием: FOR a IN table\_a FILTER a.b == "just what i need" || a.c >= 1 && a.c < 5 return a;

```

FOR {
  KEYS:
    KEY {
      row_alias: a
    }
  TABLE: table_a
  JOIN: NULL
  FOR: NULL
  FILTER:
    WHERE {
      COMPARE {
        KEY {
          row_alias: a
          column_name: b
        }
        EQUAL
        STRING { just what i need }
      }
      GREATER
      WHERE {
        COMPARE {
          COLUMN {
            table: a
            column_name: c
          }
          LESS
          NUMBER { 1 }
        }
        ||
        COMPARE {
          COLUMN {
            table: a
            column_name: c
          }
          LESS_OR_EQUAL
          NUMBER { 5 }
        }
      }
    }
  }
}

```



Выборка с соединением таблиц: FOR a IN table\_a FOR b IN table\_b RETURN a, b;

```
FOR {
  KEYS:
    KEY {
      row_alias: a
    }
  TABLE: table_a
  JOIN: table_b
  FOR:
    KEY {
      row_alias: b
    }
  FILTER: NULL
}
```

Выборка с соединением таблиц и условием: FOR a IN table\_a FOR b IN table\_b FILTER a.bool\_field == true && (a.text\_field == "text" || b.number\_field > 1 || b.float\_field < 1.1) RETURN a, b;

```
FOR {
  KEYS:
    KEY {
      row_alias: a
    }
  TABLE: table_a
  JOIN: table_b
  FOR:
    KEY {
      row_alias: b
    }
  FILTER:
    WHERE {
      COMPARE {
        KEY {
          row_alias: a
          column_name: bool_field
        }
        EQUAL
        BOOLEAN { TRUE }
      }
    }
}
```

```

||
WHERE {
  WHERE {
    COMPARE {
      KEY {
        row_alias: a
        column_name: text_field
      }
      EQUAL
      STRING { text }
    }
    GREATER
    COMPARE {
      KEY {
        row_alias: b
        column_name: number_field
      }
      GREATER_OR_EQUAL
      NUMBER { 1 }
    }
  }
  GREATER
  COMPARE {
    KEY {
      row_alias: b
      column_name: float_field
    }
    LESS_OR_EQUAL
    FLOAT { 1.100000 }
  }
}
}
}
}

```

Обновление строк: FOR a IN table\_a UPDATE a WITH { field\_one: "FIELD ONE" } IN table\_a;

```

UPDATE {
  TABLE: table_a
  VALUE_LIST:
    VALUE_LIST {
      PAIR {

```

```

        column_name: field_one
        STRING { FIELD ONE }
    };
}
CONDITION: NULL
}

```

Обновление строк с условием: FOR a IN table\_a FILTER a.b == "just what i need"  
 UPDATE a WITH { field\_one: "FIELD ONE" } IN table\_a;

```

UPDATE {
    TABLE: table_a
    VALUE_LIST:
        VALUE_LIST {
            PAIR {
                column_name: field_one
                STRING { FIELD ONE }
            };
        }
    CONDITION:
        COMPARE {
            KEY {
                row_alias: a
                column_name: b
            }
            EQUAL
            STRING { just what i need }
        }
}

```

Удаление строк: FOR a IN table\_a REMOVE a IN table\_a;

```

REMOVE {
    TABLE: table_a
    KEYS:
        KEY {
            row_alias: a
        }
    CONDITION: NULL
}

```

Удаление строк с условием: FOR a IN table\_a FILTER a.b == "just what i need"  
REMOVE a IN table\_a;

```
REMOVE {
  TABLE: table_a
  KEYS:
    KEY {
      row_alias: a
    }
  CONDITION:
    COMPARE {
      KEY {
        row_alias: a
        column_name: b
      }
      EQUAL
      STRING { just what i need }
    }
}
```

Добавление строк: INSERT { field\_one: "FIELD ONE", field\_two: "FIELD TWO" } IN table\_a;

```
INSERT {
  name: table_a
  VALUE_LIST {
    PAIR {
      column_name: field_two
      STRING { FIELD TWO }
    };
    PAIR {
      column_name: field_one
      STRING { FIELD ONE }
    };
  }
}
```

Создание таблицы: CREATE table\_a { field\_one: INT, active: STRING, else: BOOL };

```
CREATE {
  NAME: table_a
  VALUE_LIST {
```

```
PAIR {  
    column_name: else  
    TYPE { BOOLEAN }  
};  
PAIR {  
    column_name: active  
    TYPE { STRING }  
};  
PAIR {  
    column_name: field_one  
    TYPE { NUMBER }  
};  
}  
}
```

Удаление таблицы: DROP table\_a;

```
DROP {  
    table: table_a  
}
```

### Запуск:

Требования:

- OC Windows/\*NIX
- Make
- Flex
- Bison (version 3.8+)

Как запустить:

1. Выбрать Makefile для вашей ОС
2. make all - собрать и запустить модуль

### Выводы:

Были изучены принципы работы с средствами синтаксического анализа Flex и Bison, базовый синтаксис языка запросов AQL и написан модуль, позволяющий разобрать базовые запросы на данном языке.