Low level programming Lab 2

Вариант 5 - AQL

Цели

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.

Задачи

- Выбор и изучение средства синтаксического анализа.
- Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа.
- Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов
- 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
- Результаты тестирования представить в виде отчёта.

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)

Build

```
make
```

Описание работы

Программа реализована в виде модуля: запрашивает у пользователя строку на ввод и обертку над Ast деревом для возвращения результаты. Код возврата — int. Не нуль — все плохо.

Пример использования:

```
> FOR x IN data
        FILTER x.name == "first";
node_type: for
variable: x
table: data
actions:
  action:
    node_type: filter
    predicate:
      node_type: condition
      Operation: ==
      Left:
        node_type: constant
        type: reference
        value: x.name
      Right:
        node_type: constant
        type: bool
        value: true
```

Аспекты реализации

Структура модуля:

- `lexer.l` — файл лексера (flex)
- `parse.y` — файл парсера (bison)
- `ast.cpp` `ast.h` — реализация узлов дерева запроса

Типы узлов:

```
enum NodeType {
    FOR_NODE,
    ACTION_NODE,
    FILTER_NODE,
    RETURN_NODE,
    UPDATE_NODE,
    REMOVE_NODE,
    INSERT_NODE,
    MAP_NODE,
    MAP_ENTRY_NODE,
    CONDITION_NODE,
    CONDITION_UNION_NODE,
    CONSTANT_NODE
};
```

Типы логических операторов:

```
enum LogicalOp { AND, OR };
```

Типы операций:

```
enum ConstantOperation { EQ, NEQ, GT, LT, GTE, LTE, LIKE };
```

Типы данных:

```
enum DataType { INT, FLOAT, STRING, BOOL, REF };
```

Примеры запросов

Basic select:

```
> FOR x IN data
        FILTER x.name == "first";
node_type: for
variable: x
table: data
actions:
  action:
    node_type: filter
    predicate:
      node_type: condition
      Operation: ==
      Left:
        node_type: constant
        type: reference
        value: x.name
      Right:
        node_type: constant
        type: bool
        value: true
```

Select with join:

```
> FOR x IN data
        FOR y IN another
                FILTER x.id == y.id
                RETURN { "name": x.name, "id": 5, "num": y.num };
node_type: for
variable: x
table: data
actions:
  action:
    node_type: for
    variable: y
    table: another
    actions:
      action:
        node_type: filter
        predicate:
          node_type: condition
          Operation: ==
          Left:
            node_type: constant
            type: reference
            value: x.id
          Right:
            node_type: constant
            type: reference
            value: y.id
      action:
        node_type: return
        node_type: map
        entries:
          entry:
            node_type: map_entry
            key: "num"
            value:
              node_type: constant
              type: reference
              value: y.num
          entry:
            node_type: map_entry
            key: "id"
            value:
              node_type: constant
              type: int
              value: 5
          entry:
            node_type: map_entry
            key: "name"
            value:
              node_type: constant
              type: reference
              value: x.name
```

Conditional select:

```
> FOR x IN data
        FILTER x.name LIKE "Ste" && x.age < 21
        RETURN x;
node_type: for
variable: x
table: data
actions:
  action:
    node_type: filter
    predicate:
      node_type: condition_union
      Operation: and
      Left:
        node_type: condition
        Operation: like
        Left:
          node_type: constant
          type: reference
          value: x.name
        Right:
          node_type: constant
          type: bool
          value: true
      Right:
        node_type: condition
        Operation: <
        Left:
          node_type: constant
          type: reference
          value: x.age
        Right:
          node_type: constant
          type: int
          value: 21
  action:
    node_type: return
    node_type: constant
    type: reference
    value: x
```

Update:

```
> FOR x IN data
        UPDATE x WITH { "name": "Abuba" } IN data;
node_type: for
variable: x
table: data
actions:
  action:
    node_type: update
      variable: x
      table: data
      node_type: map
      entries:
        entry:
          node_type: map_entry
          key: "name"
          value:
            node_type: constant
            type: bool
            value: tru
```

Delete:

```
> FOR x IN data
        REMOVE x IN data;
node_type: for
variable: x
table: data
actions:
  action:
    node_type: remove
      variable: x
      table: data
```

Insert:

```
 INSERT { "name": "ASDF", "age": 23 } INTO data;
node_type: insert
table: data
  node_type: map
  entries:
    entry:
      node_type: map_entry
      key: "age"
      value:
        node_type: constant
        type: int
        value: 23
    entry:
      node_type: map_entry
      key: "name"
      value:
        node_type: constant
        type: bool
        value: true
```

Create:

```
 > CREATE TABLE data { "id": int, "name": string, "salary": float };
node_type: create_table
table: data
fields:
  node_type: map
  entries:
    entry:
      node_type: map_entry
      key: "salary"
      value:
        node_type: constant
        type: reference
        value: float
    entry:
      node_type: map_entry
      key: "name"
      value:
        node_type: constant
        type: reference
        value: string
    entry:
      node_type: map_entry
      key: "id"
      value:
        node_type: constant
        type: reference
        value: int
```

Drop:

```
> DROP TABLE data;
node_type: drop_table
table: data
```