Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Информатика и вычислительная техника
Дисциплина «Низкоуровневое программирование»

Отчет

По лабораторной работе №2

Вариант 6 (Gremlin)

Выполнил:
Кузнецов Н. Д.
Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург, 2023 г.

Цель: реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных.

Задачи:

- Изучить выбранное средство 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

Описание работы:

Разработанная программа принимает на вход один запрос языка GremlinQL (его версия разработанная для выполнения данной лабораторной работы), разбирает его в виде дерево и выводит в виде текста с отступами для удобочитаемости.

Было принято решение разработать средство синтаксического анализа самостоятельно, основываясь на спецификации языка запросов GremlinQL.

Особенности языка GremlinQL:

- Запрос имеет функциональный вид, представляет из себя конвейерный вызов методов объекта графа.
  Пример: `gremlin> g.V().has('age',outside(20,30)).values('age')`
- Непосредственно запрос не описывает поиск по атрибутам напрямую (хоть и поддерживает такую возможность), а описывает выборку окрестностей вершин (включая фильтрацию).

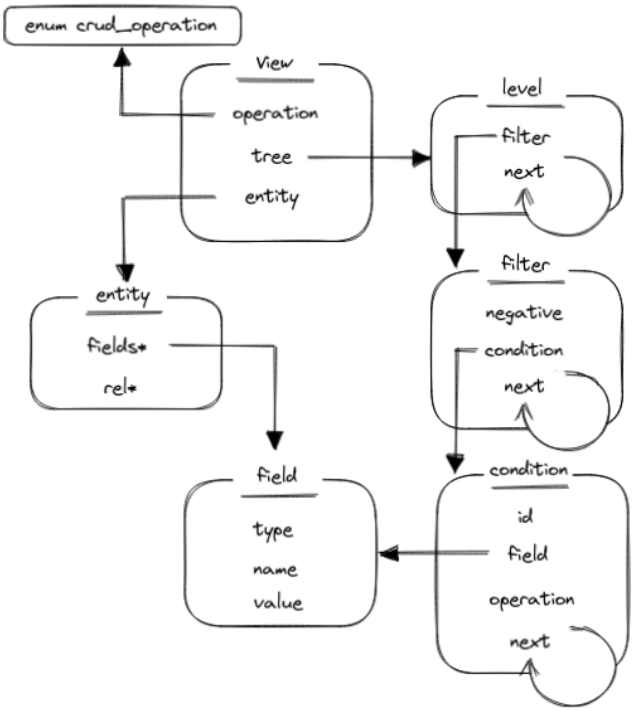
  --- Цитата из документации ---
  `out(string...)`: Move to the outgoing adjacent vertices given the edge labels.

Для реализации средства было принято решение использовать неявно обобщённый алгоритм разбора строки с применением машины состоянии. Точками перехода между состояниями приняты проход по ключевым символам: точка, скобки, кавычки (для чтения строки).

Аспекты синтаксиса и используемые части синтаксиса GremlinQL описаны в пункте 3.

Аспекты реализации:

Описание разработанной непрозрачной структуры (см файл structure.h):



Для выполнения задания язык GremlinQL был преобразован.

Запрос начинается со строки g,V которая подразумевает выборку вершин, в языке имеется разные конструкции для выборки прочих элементов графов, но в реализации они не присутствуют, потому что, учитывая модель из Модуля 1, они были бы избыточны.

Далее через точку вызываются методы поиска, метод CRUD, метод заполнения данных.

Методы поиска:

- .as(<id>) - поиск по id
- .has(<name>,<value>,<operation>) - поиск по фильтру
- .out() - переход к соседям вершин текущей выборки
- .or() - функция связка для того, чтобы можно было искать по нескольким фильтрам через операцию дизъюнкции
- not() - инвертирует текущий фильтр

Методы CRUD:

- .new() - новый элемент
- .update() - изменение элементов
- .get() - поиск элемента
- .remove() - удаление элемента

Методы заполнения данных:

- .value(<name>,<value>) - добавляет поле в сущность
- .rel(<id>) - добавить связь

Для реализации было принято решение не использовать библиотек синтаксического анализа текста (в виду того, что удалось упростить синтаксис языка до банальной последовательности термов), поэтому парсер последовательно анализирует поток символов и фиксирует состояния в структуре при достижении символа точки или конца строки

Результаты:

```
g.V.as(12).out().has(name,"Nikita",=).or().not().as(321).new().value(age,15).rel(12)

--- OPERATION ---
 OPERATION IS NEW

--- ENTITY ---
FIELDS:
        age: 15
RELATIONS:
        12

--- FILTERS ---
        LEVEL 0:
                FILTER 0:
                        It is negative
                        CONDITION 0:
                                CONDITION BY ID 321
                FILTER 1:
                        CONDITION 0:
                                OPERATION IS =:
                                name: Nikita
        LEVEL 1:
                FILTER 0:
                        CONDITION 0:
                                CONDITION BY ID 12
---------------------------------
View ram: 550
Full ram: 550
```

Как можно заметить, программа использует оперативную память только для хранение целевой структуры.
Примеры запросов:

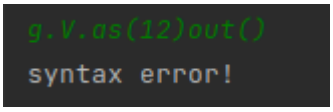
```
g.V.as(12).out().has(name,"Nikita",=).or().not().as(321).new().value(age,15).rel(12)
```

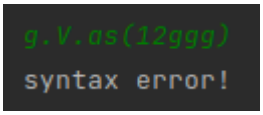
Создать запись с age=15, связанную со всеми соседями вершины 12, у которых name=”Nikita” или !id=321 + связанную с вершиной 12

```
g.V.as(12).out().has(name,"Nikita",=).or().as(321).get()
```

Найти всех соседей вершины 12 с именем Никита или id=321

Пример некорректного запроса:





Выводы:

- Был реализован модуль производящий синтаксический анализ и разбор запроса GremlinQL (его измененной версии для упрощения запросов).
- Изначально планировалось использовать YACC & LEX для разбора запроса, но удалось упростить язык для разработанной модели, поэтому было принято решение разбирать запрос на основании потока символов.
- Для хранения неопределенных по размеру данных проще оказалось удобнее использовать непрозрачные типы данных (поле tree внутри view).