Университет ИТМО

Факультет ПИиКТ

Низкоуровневое программирование

Задание 2

Вариант 9

Выполнил: Сладков М. Ю.

Группа: P33301

Преподаватель: Кореньков Ю. Д.

Санкт-Петербург

2022

Содержание

Цель:

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:

1. Изучить средство синтаксического анализа GNU Bison
2. Изучить язык запросов Cypher
3. Описать грамматику подмножества Cypher для Bison
4. Описать структуры для представления запроса
5. Добавить к правилам продукций грамматики правила создания структур запроса
6. Проверить работоспособность, отладить

Описание работы

Программа реализована в виде статической библиотеки, включающей лексический анализатор, генерируемый flex и синтаксический анализатор, генерируемый bison. Работой анализаторов управляет драйвер. Он предоставляет пользователю библиотеки метод parse, который принимает текст на стандартный ввод. На выходе формируется AST, для которого есть метод printout.

Аспекты реализации

Описание узлов содержится в файле ast.hpp. Он включается в описание грамматики parser.y. Драйвер описан в файле driver.hpp. Парсер и лексер зависят от драйвера. Пользователю библиотеки достаточно включить driver.hpp, чтобы произвести парсинг и вывести дерево. Пример этого описан в файле demo/main.cpp

AST строится из объектов следующих классов (методы опущены):

```
class INode {
public:
    virtual ~INode() {}
    virtual void print(int level, std::ostream& out) const = 0;
};
```

```
class ExpressionNode : public INode {};

class ValueNode : public INode {};

class VariableValueNode : public ValueNode {
    std::string* VariableName;
    std::string* FieldName;
};

class StringLiteralNode : public ValueNode {
    std::string *Value;
};

class BoolLiteralNode : public ValueNode {
    bool Value;
};

class IntLiteralNode : public ValueNode {
    int Value;
};

class FloatLiteralNode : public ValueNode {
    float Value;
};

enum FilterCheckOperation {
    GREATER,
    GREATER_OR_EQUAL,
    LESS,
    LESS_OR_EQUAL,
    EQUAL,
    CONTAINS
};

class FilterNode : public INode {
    ValueNode *RHS;
    ValueNode *LHS;
    FilterCheckOperation Operation;
};

class LogicalExpressionNode : public INode {};

class FilterByPassNode : public LogicalExpressionNode {
    FilterNode *Wrapped;
};

class NotOperationNode : public LogicalExpressionNode {
    LogicalExpressionNode *Operand;
};

class BinaryLogicalOperationNode : public LogicalExpressionNode {
protected:
    LogicalExpressionNode *RHS;
    LogicalExpressionNode *LHS;
};
```

```
class AndOperationNode : public BinaryLogicalOperationNode {
};

class OrOperationNode : public BinaryLogicalOperationNode {
};

class AttributeListNode : public INode {
    std::unordered_map<std::string, ValueNode*> AttrList;
};

class VariableMatchNode : public INode {
protected:
    std::string *VariableName;
    std::string *SchemeName;
};

class VariablePatternMatchNode : public VariableMatchNode {
    AttributeListNode *Pattern;
};

class VariableFilterMatchNode : public VariableMatchNode {
    PredicateNode* Predicate;
};

enum RelationDirection {
    FORWARD,
    REVERSE,
    ANY
};

class RelationMatchNode : public INode {
    std::string *VariableName;
    std::string *RelationName;
    RelationDirection Direction;
};

class MatchExpressionNode : public ExpressionNode {
    VariableMatchNode *LeftNode;
    VariableMatchNode *RightNode;
    RelationMatchNode *Relation;
};

class ReturnExpressionNode : public ExpressionNode {
    std::vector<ValueNode*> Values;
};

class SetExpressionNode : public ExpressionNode {
    VariableValueNode *Dest;
    ValueNode *Src;
};

class DeleteExpressionNode : public ExpressionNode {
    std::string* VariableName;
};
```

```
class CreateExpressionNode : public ExpressionNode {
    VariableMatchNode *LeftNode;
    VariableMatchNode *RightNode;
    RelationMatchNode *Relation;
};

class RequestNode : public INode {
    std::vector<ExpressionNode*> Expressions;
};
```

Результаты

Примеры запросов

1. Базовая выборка

   ```
   MATCH (m:Movie)<-[d:DIRECTED]-(directors)
   RETURN directors.name
   ```

   ```
   Request: {
       Match expression: {
           Left node: Any variable match: {
               Variable name: m
               Scheme name: Movie
           }
           Relation: Relation match: {
               Variable name: d
               Relation: DIRECTED
               Direction: REVERSE
           }
           Right node: Any variable match: {
               Variable name: directors
               Scheme name:
           }
       }
       Return Expression: {
           Variable value node: {
               Variable name: directors
               Field name: name
           }
       }
   }
   ```

2. Выборка с условием на число

```
MATCH (m:Movie WHERE m.year < 1990)
RETURN m
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: m
            Scheme name: Movie
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: m
                            Field name: year
                        }
                        Operation: LESS
                        Right part: Int Literal node: {
                            Value: 1990
                        }
                    }
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: m
            Field name:
        }
    }
}
```

3. Выборка с условием на подстроку

```
MATCH (m:Movie WHERE m.name contains "007")<-[d:DIRECTED]-(directors)
RETURN directors.name, directors.oscars
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: m
            Scheme name: Movie
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: m
                            Field name: name
                        }
                        Operation: CONTAINS
                        Right part: String Literal node: {
                            Value: "007"
                        }
                    }
                }
            }
        }
        Relation: Relation match: {
            Variable name: d
            Relation: DIRECTED
            Direction: REVERSE
        }
        Right node: Any variable match: {
            Variable name: directors
            Scheme name:
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: directors
            Field name: name
        }
        Variable value node: {
            Variable name: directors
            Field name: oscars
        }
    }
}
```

4. **Выборка по шаблону с несколькими атрибутами**

```
MATCH (m:Movie {year: 1990, genre:”comedy"})<-[d:DIRECTED]-(directors)
RETURN directors.name, directors.oscars
```

```
Request: {
    Match expression: {
        Left node: Variable pattern match: {
            Variable name: m
            Scheme name: Movie
            Pattern: Attribute List: {
                {
                    Attribute name: year
                    Value: Int Literal node: {
                        Value: 1990
                    }
                },
                {
                    Attribute name: genre
                    Value: String Literal node: {
                        Value: "comedy"
                    }
                },
            }
        }
        Relation: Relation match: {
            Variable name: d
            Relation: DIRECTED
            Direction: REVERSE
        }
        Right node: Any variable match: {
            Variable name: directors
            Scheme name:
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: directors
            Field name: name
        }
        Variable value node: {
            Variable name: directors
            Field name: oscars
        }
    }
}
```

5. **Выборка по логической комбинации фильтров**

```
MATCH (p:Person WHERE name contains "re" AND NOT age == 14)
RETURN p
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: p
            Scheme name: Person
            Filter: Predicate: {
                expr: AND: {
                    Left operand: Filter bypass: {
                        Wrapped filter: Filter node: {
                            Left part: Variable value node: {
                                Variable name: name
                                Field name:
                            }
                            Operation: CONTAINS
                            Right part: String Literal node: {
                                Value: "re"
                            }
                        }
                    }
                    Right operand: NOT: {
                        Operand: Filter bypass: {
                            Wrapped filter: Filter node: {
                                Left part: Variable value node: {
                                    Variable name: age
                                    Field name:
                                }
                                Operation: EQUAL
                                Right part: Int Literal node: {
                                    Value: 14
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: p
            Field name:
        }
    }
}
```

6. **Вставка**

```
MATCH (p:Person WHERE age == 14)
CREATE (pw:PotentialWorker {name: p.name, age: p.age, education:
"None"})
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: p
            Scheme name: Person
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: age
                            Field name:
                        }
                        Operation: EQUAL
                        Right part: Int Literal node: {
                            Value: 14
                        }
                    }
                }
            }
        }
    }
    Create expression: {
        Left node: Variable pattern match: {
            Variable name: pw
            Scheme name: PotentialWorker
            Pattern: Attribute List: {
                {
                    Attribute name: name
                    Value: Variable value node: {
                        Variable name: p
                        Field name: name
                    }
                },
                {
                    Attribute name: age
                    Value: Variable value node: {
                        Variable name: p
                        Field name: age
                    }
                },
                {
                    Attribute name: education
                    Value: String Literal node: {
                        Value: "None"
                    }
                },
            }
        }
    }
}
```

7. **Удаление**

```
MATCH (p:Worker WHERE fired == true)
DELETE p
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: p
            Scheme name: Worker
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: fired
                            Field name:
                        }
                        Operation: EQUAL
                        Right part: Bool Literal node: {
                            Value: 1
                        }
                    }
                }
            }
        }
    }
    Delete Expresssion {
        Variable: p
    }
}
```

8. **Обновление**

```
MATCH (p:Worker WHERE fired == true)-[r:FRIEND]->(o:Worker)
SET o.sus = true
RETURN o
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: p
            Scheme name: Worker
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: fired
                            Field name:
                        }
                        Operation: EQUAL
                        Right part: Bool Literal node: {
                            Value: 1
                        }
                    }
                }
            }
        }
        Relation: Relation match: {
            Variable name: r
            Relation: FRIEND
            Direction: FORWARD
        }
        Right node: Any variable match: {
            Variable name: o
            Scheme name: Worker
        }
    }
    Set Expression: {
        Destination: Variable value node: {
            Variable name: o
            Field name: sus
        }
        Source: Bool Literal node: {
            Value: 1
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: o
            Field name:
        }
    }
}
```

Выводы

Выполнены цель и задачи лабораторной работы. Изучен язык Cypher, средства лексического и синтаксического анализа Bison и Flex. Реализован модуль разбора подмножества языка Cypher и вывода синтаксического дерева.