Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia

Trabalho Prático de Processamento de Linguagens

Autores:
Diogo Machado nº26042
Enrique Rodrigues nº28602
José Alves nº279967

Docente: Óscar Ribeiro
Data: Maio de 2025

Abstract

Este artigo descreve o desenvolvimento de um interpretador para a linguagem CQL (Cassandra Query Language), implementado em Python. O interpretador é capaz de processar comandos CQL, realizar análise léxica e sintática, e executar operações básicas de manipulação de dados. O projeto demonstra a aplicação prática de conceitos de compiladores e processamento de linguagens, incluindo análise léxica, análise sintática, e geração de árvores de sintaxe abstrata (AST). São apresentados exemplos práticos de código e visualizações da AST para demonstrar o funcionamento do interpretador.

1 Introdução

A linguagem CQL (Cassandra Query Language) é uma linguagem de consulta semelhante ao SQL, especificamente projetada para trabalhar com o Apache Cassandra. Este projeto visa desenvolver um interpretador que possa processar e executar comandos CQL básicos, demonstrando os princípios fundamentais de processamento de linguagens e compiladores. A implementação foi realizada em Python, uma linguagem que oferece excelentes ferramentas para processamento de linguagens e análise de texto.

2 Metodologia

O interpretador foi desenvolvido em Python, utilizando as seguintes bibliotecas principais:

- PLY (Python Lex-Yacc) para análise léxica e sintática
- Graphviz para visualização da árvore de sintaxe abstrata
- PrettyTable para formatação de resultados em formato tabular

A estrutura do projeto foi organizada da seguinte forma:

3 Implementação

3.1 Análise Léxica

A análise léxica é realizada através de expressões regulares que identificam os diferentes tokens da linguagem CQL. O analisador léxico é implementado utilizando a biblioteca PLY, que permite definir tokens através de expressões regulares. Segue um exemplo da definição de alguns tokens:

```
# Tokens
tokens = (
    'CREATE', 'TABLE', 'INSERT', 'INTO', 'SELECT', 'FROM',
    'WHERE', 'DELETE', 'DROP', 'IDENTIFIER', 'NUMBER',
    'STRING', 'COMMA', 'SEMICOLON'
)

# Regras para tokens
```

```
t_CREATE = r'CREATE'
t_TABLE = r'TABLE'
t_INSERT = r'INSERT'
t_INTO = r'INTO'
t_SELECT = r'SELECT'
t_FROM = r'FROM'
t_WHERE = r'WHERE'
t_DELETE = r'DELETE'
t_DROP = r'DROP'
t_COMMA = r','
t_SEMICOLON = r';'
```

3.2 Análise Sintática

A análise sintática é implementada usando uma gramática que define a estrutura válida dos comandos CQL. A gramática suporta os seguintes tipos de comandos:

- CREATE TABLE
- INSERT INTO
- SELECT
- DELETE
- DROP TABLE

Exemplo da definição de algumas regras gramaticais:

```
def p_create_table(p):
    '''create_table : CREATE TABLE IDENTIFIER LPAREN column_defs RPAREN'''
    p[0] = ('CREATE_TABLE', p[3], p[5])

def p_column_defs(p):
    '''column_defs : column_def
                   | column_defs COMMA column_def'''
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[0] = p[1] + [p[3]]
```

3.3 Árvore de Sintaxe Abstrata (AST)

A AST é gerada durante a análise sintática e representa a estrutura hierárquica do comando. Cada nó da árvore representa uma operação ou elemento do comando, facilitando sua interpretação e execução. A visualização da AST é realizada utilizando a biblioteca Graphviz.

Exemplo de um comando CQL e sua representação em AST:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name TEXT,
    email TEXT
);
```

A AST gerada para este comando é visualizada como uma árvore onde:

- O nó raiz representa o comando CREATE TABLE
- Os nós filhos representam o nome da tabela e as definições das colunas
- Cada definição de coluna contém o nome, tipo e restrições

4 Resultados

O interpretador foi testado com diversos comandos CQL, demonstrando sua capacidade de:

- Processar comandos sintaticamente corretos
- Gerar visualizações da AST
- Executar operações básicas de manipulação de dados
- Apresentar resultados em formato tabular

Exemplo de execução de um comando SELECT:

```
SELECT * FROM users WHERE id = 1;
```

Resultado apresentado em formato tabular:

```
+----+--------+------------------+
| id |  name  |      email       |
+----+--------+------------------+
| 1  | João   | joao@email.com   |
+----+--------+------------------+
```

5 Conclusão

O desenvolvimento deste interpretador CQL demonstrou a aplicação prática de conceitos fundamentais de compiladores e processamento de linguagens. O projeto pode ser expandido para suportar mais funcionalidades da linguagem CQL, tais como:

- Suporte a índices secundários
- Implementação de transações
- Otimização de consultas
- Suporte a tipos de dados mais complexos

A implementação atual serve como uma base sólida para futuras expansões e melhorias, demonstrando a viabilidade de desenvolver um interpretador completo para a linguagem CQL utilizando Python e suas bibliotecas de processamento de linguagens.