



Trabalho Prático

Processamento de Linguagens

Grupo No. 9

No.27963 – Hugo Especial
No.27966 – Paulo Gonçalves
No.27969 – Marco Cardoso

Licenciatura em Engenharia Sistemas Informáticos

2ºano

Barcelos | maio, 2025

Lista de Abreviaturas e Siglas

SQL - Structured Query Language

CQL - CSV Query Language

PLY - Python Lex-Yacc

CSV - Comma Separated Values

Índice de Figuras

Figura 1 - Expressões Regulares Simples.....	7
Figura 2 - Defenição das Palavras Reservadas	7
Figura 3 - Exemplo de uma Regra Gramatical	9
Figura 4 - Função execute_command	10
Figura 5 - Função evaluate_condition	11

Conteúdo

1. Introdução	5
2. Desenvolvimento	6
2.1. Como foi desenvolvido	6
2.2. Analisador Léxico	7
2.3. Analisador Sintático	9
2.4. Analisador Semântico	10
3. Conclusão	13
4. Bibliografia	14

1. Introdução

Este trabalho tem como objetivo o desenvolvimento de uma nova linguagem de programação inspirada no SQL, que será designada por CQL. O propósito principal da CQL é permitir a realização de consultas e operações sobre ficheiros no formato .csv, de forma simples, intuitiva e semelhante ao funcionamento de bases de dados tradicionais.

No âmbito deste projeto, será necessário aplicar vários conceitos abordados ao longo das aulas da unidade curricular de Processamento de Linguagens, como a definição formal da gramática da linguagem (através de expressões regulares e gramáticas livres de contexto), a construção de analisadores léxicos e sintáticos, a interpretação de comandos e a execução de operações sobre dados estruturados.

2. Desenvolvimento

2.1. Como foi desenvolvido

Para o desenvolvimento deste trabalho foram utilizados três analisadores/reconhecedores: léxico, sintático e semântico.

O analisador léxico tem como função identificar os elementos básicos da linguagem, como palavras-chave, identificadores, números e símbolos, convertendo a entrada em uma sequência de *tokens*.

O analisador sintático verifica se a estrutura desses *tokens* segue as regras gramaticais definidas, garantindo que a sequência formada seja válida do ponto de vista da linguagem.

Por fim, o analisador semântico assegura que os elementos utilizados tenham significado no contexto do sistema, como a existência de uma tabela, a validade dos nomes de colunas e a coerência das condições impostas.

Estes três componentes trabalham de forma integrada para interpretar corretamente os comandos fornecidos, detetando e reportando erros quando necessário.

2.2. Analisador Léxico

O analisador léxico, desenvolvido com a biblioteca PLY, tem como principal objetivo identificar e classificar os elementos básicos de uma linguagem de comandos SQL-like. Ele reconhece palavras reservadas como “SELECT”, “IMPORT”, “CREATE”, “WHERE”, entre outras, bem como símbolos e operadores como vírgulas, parênteses, operadores de comparação (=, <>, <=, etc.), e ainda valores literais como números e *strings*.

```
# Expressões regulares simples
t_SEMICOLON = r';'
t_COMMA = r','
t_STAR = r'\*'
t_EQUAL = r'='
t_NOTEQUAL = r'<>'
t_LESS = r'<'
t_GREATER = r'>'
t_LESS_EQUAL = r'<='
t_GREATER_EQUAL = r'>='
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_ignore = ' \t'

t_NUMBER = r'\d+(\.\d+)?'
```

Figura 2 - Expressões Regulares Simples

```
# Definindo palavras reservadas
reserved = {
    'IMPORT': 'IMPORT',
    'CREATE': 'CREATE',
    'EXPORT': 'EXPORT',
    'TABLE': 'TABLE',
    'FROM': 'FROM',
    'SELECT': 'SELECT',
    'AS': 'AS',
    'DISCARD': 'DISCARD',
    'RENAME': 'RENAME',
    'PRINT': 'PRINT',
    'LIMIT': 'LIMIT',
    'WHERE': 'WHERE',
    'AND': 'AND',
    'JOIN': 'JOIN',
    'USING': 'USING',
    'PROCEDURE': 'PROCEDURE',
    'DO': 'DO',
    'CALL': 'CALL',
}
```

Figura 1 - Definição das Palavras Reservadas

Esses elementos são convertidos em tokens, que representam as unidades mínimas significativas da linguagem, facilitando a posterior análise estrutural do código. O analisador também distingue palavras reservadas de identificadores comuns, garantindo que os comandos da linguagem sejam interpretados corretamente.

Além da identificação dos tokens, o analisador léxico trata outros aspetos fundamentais, como a ignorância de espaços em branco, tabulações e comentários (de linha e de bloco), para que não interfiram na análise. Também é capaz de detetar e reportar caracteres “ilegais”, permitindo a continuidade da análise ao ignorá-los de forma controlada. Este processo de tokenização é essencial para transformar o código de entrada em uma forma estruturada e compreensível por um analisador sintático, funcionando como a primeira etapa do processo de interpretação de comandos em sistemas de gestão de dados.

2.3. Analisador Sintático

O analisador sintático, ou *parser*, construído com o módulo *ply.yacc*, tem como objetivo interpretar a sequência de *tokens* gerada pelo analisador léxico e validar se a estrutura sintática da linguagem está correta. Cada função que começa com “p_” representa uma regra gramatical que descreve uma estrutura válida da linguagem, como os comandos “SELECT”, “CREATE TABLE”, “IMPORT”, entre outros.

```
def p_select_command(p):
    """
    command : SELECT star_columns FROM IDENTIFIER WHERE condition_list SEMICOLON
             | SELECT column_list FROM IDENTIFIER WHERE condition_list SEMICOLON
             | SELECT star_columns FROM IDENTIFIER SEMICOLON
             | SELECT column_list FROM IDENTIFIER SEMICOLON
    """

    if len(p) == 8:
        if isinstance(p[2], list):
            p[0] = ('SELECT_COLUMNS_WHERE', p[4], p[2], p[6])
        else:
            p[0] = ('SELECT_ALL_WHERE', p[4], p[6])
    else:
        if isinstance(p[2], list):
            p[0] = ('SELECT_COLUMNS', p[4], p[2])
        else:
            p[0] = ('SELECT_ALL', p[4], p[6])
    else:
        if isinstance(p[2], list):
            p[0] = ('SELECT_COLUMNS', p[4], p[2])
        else:
            p[0] = ('SELECT_ALL', p[4])
```

Figura 3 - Exemplo de uma Regra Gramatical

Essas regras são definidas diretamente dentro das funções por meio de cadeias de texto que indicam a estrutura esperada da instrução. Quando uma dessas regras é satisfeita, a função atribui a “p[0]” um conjunto com os dados extraídos da instrução.

Por exemplo, a regra “p_select_command” reconhece instruções do tipo “SELECT ... FROM ... WHERE ...” e permite diferentes variações com ou sem condições. O código analisa quantos elementos existem na instrução e decide o tipo de comando correspondente (como “SELECT_ALL”, “SELECT_COLUMNS_WHERE”, etc.). Esta abordagem torna o *parser* flexível,

capaz de lidar com comandos complexos com cláusulas como “WHERE”, “LIMIT” ou até “JOIN”.

No final, o *parser* é inicializado com “yacc.yacc()” e pode ser utilizado com a função “parse_sql(sql)”, que processa uma *string* de entrada e retorna a estrutura sintática correspondente.

2.4. Analisador Semântico

Complementando o funcionamento do *parser*, o “semantic.py”, que é responsável por interpretar e executar os comandos analisados, aplica a lógica necessária sobre os dados armazenados. A função de execução, “execute_command”, recebe os comandos já processados pelo *parser* e determina qual operação deve ser realizada com base no tipo de instrução, que é identificado por palavras-chave como “IMPORT”, “SELECT”, “EXPORT”, “CREATE TABLE”, entre outras.

```
def execute_command(parsed):
    if parsed is None:
        # Linha vazia ou comentário ignorado
        return "Parse inválido ou comando vazio."

    if not parsed or len(parsed) < 1:
        print("[ERRO] Comando inválido.")
        return "Parse inválido ou comando vazio."

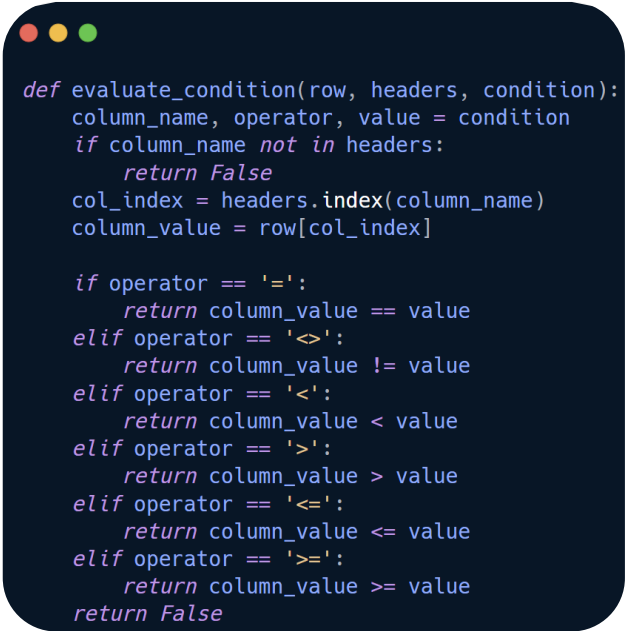
    cmd_type = parsed[0]

    if cmd_type == 'IMPORT' and len(parsed) >= 3:
        _, table_name, file_path = parsed
        if process_import(table_name, file_path):
            print(f"Tabela '{table_name}' importada com sucesso!")

    elif cmd_type == 'SELECT_ALL' and len(parsed) >= 2:
        table_name = parsed[1]
        table = database.get_table(table_name)
```

Figura 4 - Função execute_command

Para operações de importação e exportação de dados em formato CSV, as funções “process_import” e “process_export” permitem carregar ficheiros para tabelas internas ou exportar essas tabelas para o nosso computador, garantindo que os dados e os cabeçalhos estão consistentes. No caso dos comandos “SELECT”, o sistema permite selecionar todas as colunas ou colunas específicas, aplicar condições (WHERE) e limites de linhas (LIMIT). As condições são avaliadas pela função “evaluate_condition”, que compara valores com operadores como ‘=’, ‘<>’, ‘<’, ‘>’ e afins.



```
def evaluate_condition(row, headers, condition):
    column_name, operator, value = condition
    if column_name not in headers:
        return False
    col_index = headers.index(column_name)
    column_value = row[col_index]

    if operator == '=':
        return column_value == value
    elif operator == '<>':
        return column_value != value
    elif operator == '<':
        return column_value < value
    elif operator == '>':
        return column_value > value
    elif operator == '<=':
        return column_value <= value
    elif operator == '>=':
        return column_value >= value
    return False
```

Figura 5 - Função evaluate_condition

Também é possível modificar o estado das tabelas com os comandos “DISCARD”, que limpa os dados da tabela, ou “RENAME”, que altera o nome de uma tabela existente. Além disso, comandos mais avançados como “CREATE_TABLE” e “CREATE_TABLE_JOIN” permitem criar tabelas a partir da seleção filtrada de uma tabela existente, ou realizar uma junção entre duas tabelas com base numa chave comum, respeitando os princípios básicos de uma operação “JOIN”.

Uma funcionalidade adicional é a definição de procedimentos reutilizáveis através dos comandos “PROCEDURE” e “CALL” disponíveis no ficheiro “parser.py”. Estes permitem guardar uma sequência de comandos sob um nome e executá-los posteriormente quantas vezes forem necessárias. Estes procedimentos são armazenados num objeto interno e executados sequencialmente quando chamados, proporcionando reutilização ao sistema. Assim, este analisador atua como o motor de execução dos comandos analisados, garantindo que a lógica da linguagem é refletida diretamente nas operações sobre os dados.

3. Conclusão

Este projeto mostra como é possível criar uma linguagem de consultas, própria para manipular dados de forma simples. Com comandos parecidos com SQL, pode importar, consultar, exportar dados e até criar procedimentos automáticos. Com este projeto adquirimos uma compreensão mais profunda sobre a construção de linguagens formais, desenvolvendo desde o analisador léxico e sintático até a execução de comandos personalizados, além de fortalecer habilidades em estrutura de dados, lógica de programação e organização de sistemas de manipulação de dados.

4. Bibliografia

<https://www.dabeaz.com/ply/ply.html> Ply [Artigo]. - [s.l.] : David Beazley, 2025.