Relatorio do trabalho prático
(Processamento de Linguagens)

Grupo No. 8

David Carvalho Nº27973
Diogo Marques Nº.27972
Gonçalo Vidal Nº27984

Licenciatura em Engenharia Sistemas Informáticos

2ºano

Barcelos | 2025

Índice de Figuras

Índice

1. Introdução

Este relatório descreve o desenvolvimento do projeto "CQL Interpreter", realizado no âmbito da unidade curricular de Processamento de Linguagens da Licenciatura em Engenharia de Sistemas Informáticos, no Instituto Politécnico do Cávado e do Ave, durante o ano letivo 2024/2025.

O objetivo principal do projeto foi implementar um interpretador para a linguagem CQL (Comma Query Language), projetada para operar sobre ficheiros CSV. A aplicação foi desenvolvida em Python, utilizando a biblioteca PLY (Python Lex-Yacc), permitindo aos utilizadores realizar operações de consulta e manipulação de dados em ficheiros CSV através de comandos.

2. Arvore abstrata (AST)

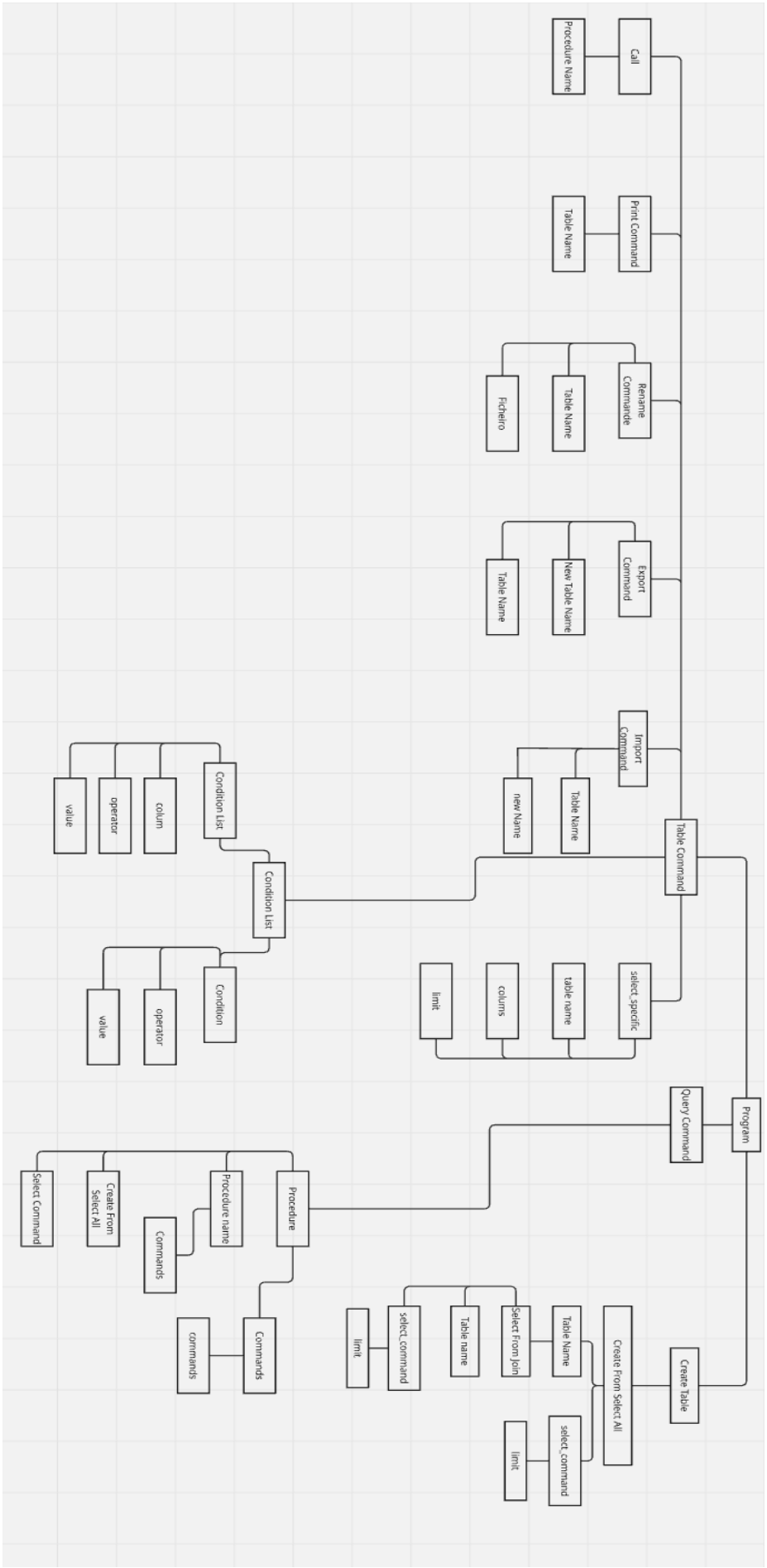

Figura 1- Arvore Abastrata

3. Desenvolvimento

O desenvolvimento da aplicação foi realizado em **Python**, utilizando a biblioteca **PLY** (Python Lex-Yacc).

3.1. Estrutura do projeto

- **lexer.py**: define o analisador léxico da linguagem CQL;
- **parser.py**: define a gramática sintática da linguagem e as ações semânticas associadas;
- **interpreter.py**: contém as funções que executam os comandos definidos pela linguagem;
- **filesCSV.py**: responsável pelo carregamento e exportação das tabelas;
- **cql_interpreter.py:** ponto de entrada da aplicação que liga os componentes lexer, parser e interpreter;
- **/test:** pasta com os testes da aplicação;
- **/data:** pasta com os ficheiros csv;
- **/input:** pasta com os ficheiros de entrada da aplicação.
- **/output:** pasta com os ficheiros de saída da aplicação;

3.2. Analisador Léxico (lexer.py)

O ficheiro ***lexer.py*** define os tokens da linguagem, como palavras-chave (SELECT, FROM, WHERE, IMPORT, etc.), operadores (=, <>, <, >, <=, >=), símbolos de pontuação (;, ,, (,)), identificadores e literais (strings e números).

Através da biblioteca PLY, foram criadas expressões regulares para cada tipo de token, permitindo identificar corretamente os elementos da linguagem.

3.3. Analisador Sintático (parser.py)

No ***parser.py***, é definida a gramática da linguagem CQL, em conjunto com as ações semânticas associadas a cada regra. A análise sintática constrói diretamente estruturas representativas das instruções, sob a forma de uma árvore de sintaxe abstrata (AST).

As regras suportam a totalidade dos comandos descritos no enunciado, incluindo:

- **Configuração e gestão de tabelas:**
  - IMPORT TABLE nome FROM "ficheiro.csv";
  - EXPORT TABLE nome AS "ficheiro.csv";
  - RENAME TABLE nome1 nome2;
  - DISCARD TABLE nome;
  - PRINT TABLE nome;
- **Consultas:**
  - SELECT * FROM tabela;
  - SELECT col FROM tabela;
  - SELECT * FROM tabela WHERE col > 5 AND col2 = 2;

  Para todas as consultas é possível limitar o numero de resultados utilizando “LIMIT X” no final do comando

- **Criação de tabelas:**
  - CREATE TABLE nome SELECT * FROM tabela;
  - CREATE TABLE nome SELECT * FROM tabela WHERE coluna > 1;
  - CREATE TABLE nome FROM tabela1 JOIN tabela2 USING(coluna)
- **Procedimentos:**
  - PROCEDURE nome DO ... END;
  - CALL nome;

As instruções reconhecidas são convertidas e depois processados pelo interpretador.

3.4. Interpretador (interpreter.py)

O interpreter.py é responsável pela execução das ações descritas na AST. Cada tipo de instrução tem uma função correspondente que interage com os dados em memória.

As operações implementadas incluem:

- Descartar, renomear e imprimir tabelas em memoria.
- Seleção de colunas específicas ou todas de uma tabela em memoria, com a possibilidade de limitar os resultados;
- Criação de novas tabelas em memória a partir de queries;
- Criação e execução de procedimentos.

A estrutura de dados usada para armazenar as tabelas e os procedimentos e foi um dicionário, em que cada tabela e procedimento é identificada por uma chave que é criada a partir do nome da tabela ou pelo nome do procedimento.

3.5. Gestor dos ficheiros CSV (filesCSV.py)

O módulo **filesCSV.py** é responsável pela importação, exportação de ficheiros CSV, funcionando como uma interface entre os dados em ficheiro e as estruturas de dados em memória utilizadas pela linguagem aplicação.

- **Importar tabelas:** No processo de importação, a aplicação lê o conteúdo de um ficheiro CSV e guarda o conteúdo num array. Durante esta leitura, o sistema ignora linhas que começam por #, tratadas como comentários, e interpreta corretamente os campos delimitados por aspas, permitindo, por exemplo, que valores contendo vírgulas internas sejam lidos como uma única célula.
- **Exportar tabelas:** No processo de exportação, os dados que se encontram em memória são escritos de volta num ficheiro CSV.

3.6. Ponto de entrada da aplicação (interpreter_cql.py)

O ficheiro interpreter_cql.py é o ponto de entrada da aplicação. Permite dois modos de operação:

- **Modo por ficheiro:** o utilizador fornece um ficheiro .fca com comandos CQL, que são lidos e processados sequencialmente;
- **Modo interativo:** o utilizador escreve comandos diretamente no terminal, que são processados imediatamente.

Em ambos os modos a aplicação carrega todos os procedimentos presentes no ficheiro *"procedures.fca"*.

Este módulo inicializa o interpretador, e controla a entrada/saída dos dados. Os ficheiros CSV são lidos da pasta data/, os comandos são carregados da pasta input/ e os resultados de exportações são escritos na pasta output/.

3.7. Testes

Foi criado um conjunto de testes organizados na pasta tests. Estes testes estão distribuídos por três ficheiros principais: test_lexer.py, test_parser.py e test_interpreter.py, e permitem verificar o comportamento do sistema em diferentes etapas do processo de interpretação da linguagem CQL.

Testes Léxicos (test_lexer.py)

Este conjunto de testes tem como objetivo garantir que o **analisador léxico** identifica corretamente os tokens da linguagem. São testados diversos elementos da sintaxe da CQL, incluindo:

- Palavras-chave (IMPORT, SELECT, TABLE, WHERE, etc.)
- Identificadores de tabelas e colunas
- Strings e literais numéricos
- Comentários (de uma ou várias linhas)
- Símbolos e operadores (=, <>, ;, *, etc.)

Os testes validam se o lexer, construído reconhece corretamente os tokens esperados para entradas típicas da linguagem, e se ignora adequadamente os espaços em branco e comentários. Este tipo de teste é essencial para garantir que a gramática funciona sobre uma base léxica estável.

Testes Sintáticos (test_parser.py)

Estes testes focam-se na **análise sintática**, assegurando que a estrutura das frases da linguagem é corretamente interpretada e que a **árvore de sintaxe abstrata (AST)** correspondente é gerada conforme o esperado.

Entre os casos de teste estão instruções completas como:

- IMPORT TABLE estacoes FROM "estacoes.csv";
- SELECT * FROM observacoes WHERE Temperatura > 22;
- CREATE TABLE resultado SELECT * FROM observacoes;

Os testes verificam se o parser, implementado com `ply.yacc`, aceita corretamente instruções válidas e rejeita instruções malformadas. Além disso, confirmam se a estrutura da AST resultante está coerente com a semântica da linguagem, permitindo que o interpretador execute ações apropriadas sobre os nós da árvore.

Testes do Interpretador (test_interpreter.py)

Neste ficheiro testam-se os comportamentos do **interpretador**, que executa as instruções da CQL a partir da AST gerada. Estes testes cobrem o ciclo completo: desde o parsing de um comando até à execução da operação correspondente.

Exemplos de funcionalidades testadas incluem:

- Importação e exportação de ficheiros CSV
- Execução de queries com filtros (`WHERE`) e limites (`LIMIT`)
- Criação de tabelas a partir de outras
- Impressão de resultados no terminal
- Manipulação correta de dados em memória

4. Bibliografia

- https://docs.python.org/3/
- https://www.dabeaz.com/ply/ply.html

5. Conclusão

O desenvolvimento deste projeto permitiu aplicar, de forma prática, os conceitos fundamentais abordados na unidade curricular de Processamento de Linguagens, como a construção de analisadores léxicos e sintáticos, a definição de árvores de sintaxe abstrata (AST) e a execução de ações semânticas associadas a uma linguagem específica.

Durante o desenvolvimento, trabalhámos em equipa para compreender como funciona o processo de análise de linguagens, desde o reconhecimento de comandos até à sua execução. A linguagem que desenvolvemos permite realizar operações úteis sobre dados tabulares, como consultas, importações, exportações e junções de tabelas.

Além disso, aprendemos a importância de testar cada componente da aplicação para garantir que tudo funciona corretamente. Com este projeto, conseguimos não só cumprir os objetivos propostos no enunciado, mas também ganhar uma maior compreensão prática sobre a forma como funcionam linguagens e interpretadores.

Consideramos que o trabalho foi positivo e enriquecedor, tanto a nível técnico como académico.