

**Licenciatura em Engenharia de sistemas Informáticos**  
**Processamento de Linguagens**

# **Relatório do Trabalho Prático**

Gonçalo Figueiredo 26747

Hugo Azevedo 25431

Tiago Castro 25434

Professor: Óscar Ribeiro

Maio de 2025

# Índice

Índice de figuras .....	3
Introdução .....	4
Visão geral .....	5
Componentes principais.....	5
Analizador Léxico.....	6
Tokens definidos.....	6
Comentários e espaços.....	6
Como funciona.....	7
Analizador Sintático e AST .....	8
Construção da AST .....	8
Modulo de Avaliação.....	9
Execução do Programa.....	10
Exemplos de Uso e Testes .....	11
Dados de exemplo .....	11
Scripts de demonstração .....	11
Resultados.....	12
Conclusão .....	14
Bibliografia.....	15

## Índice de figuras

Figura 1 - Execução em modo interativo .....	12
Figura 2 - Início da execução do ficheiro exemplo.cql.....	13
Figura 3 - Fim da execução do ficheiro exemplo.cql.....	13

# Introdução

Este trabalho teve como objetivo criar um interpretador para a linguagem CQL (Comma Query Language), que foi pensada para trabalhar com dados guardados em ficheiros CSV. Esta linguagem é usada para importar e exportar tabelas, aplicar filtros aos dados, fazer junções entre diferentes tabelas, criar novas estruturas e renomeá-las, além de definir procedimentos que podem ser reutilizados.

A implementação foi feita em Python e dividida em quatro partes principais. Começa com o ficheiro *cql\_lexer.py*, que trata da parte léxica, ou seja, identifica os elementos da linguagem como nomes, símbolos e palavras reservadas. Depois vem o *cql\_grammar.py*, que analisa a estrutura dos comandos e constrói uma árvore que representa o seu significado. O *cql\_eval.py* é responsável por interpretar essa árvore e executar as ações necessárias. Por fim, o *main.py* funciona como a interface principal, onde se podem correr scripts ou dar comandos diretamente.

Durante o desenvolvimento, tentamos manter o código limpo e organizado para facilitar alterações e testes. O interpretador consegue tanto correr comandos isolados como processar ficheiros inteiros com várias instruções CQL.

## Visão geral

O projeto foi desenvolvido com uma estrutura modular, dividida em vários ficheiros, cada um com uma responsabilidade clara.

Além dos módulos principais do interpretador, o projeto inclui também uma estrutura de pastas para ajudar a separar os diferentes tipos de ficheiros usados. Na pasta *data* encontram-se os ficheiros .csv, que são os dados a serem importados e analisados. A pasta *input* guarda os ficheiros .cql, que contêm scripts com comandos escritos em CQL, servindo como exemplos de utilização. Já a pasta *saida* está reservada para guardar os ficheiros exportados durante a execução, ou seja, os resultados das operações feitas com a linguagem.

No centro de tudo está a linguagem CQL, que funciona como uma mini-linguagem de consulta para trabalhar com dados em CSV. Para que o interpretador consiga entender e executar os comandos, o sistema passa por várias etapas típicas de um compilador: começa por analisar o texto (análise léxica e sintática), depois estrutura a informação numa árvore (AST), e por fim executa as ações pedidas com base nessa representação.

## Componentes principais

*cql\_lexer.py* – Trata da parte léxica, ou seja, é o responsável por dividir os comandos em unidades (tokens), como palavras-chave, identificadores, operadores, etc.

*cql\_grammar.py* – Aqui fica definida a gramática da linguagem. Também é neste módulo que se constrói a árvore que representa a lógica dos comandos (AST).

*cql\_eval.py* – Este é o “cérebro” que interpreta a AST e executa as ações pedidas: desde carregar ficheiros até fazer junções de tabelas, aplicar filtros e mais.

*main.py* – Serve de ponto de entrada. Permite correr scripts com comandos CQL ou usar o modo interativo, onde os comandos são dados um a um.

## Analizador Léxico

O analisador léxico é o primeiro passo do interpretador. A sua função principal é ler o código escrito em CQL e transformá-lo numa sequência de unidades menores chamadas tokens. Estes tokens representam coisas como nomes de tabelas, operadores, números, palavras-chave e outros símbolos que fazem parte da linguagem. Para esta tarefa, foi usado o PLY (Python Lex-Yacc), que facilita bastante a criação de léxicos e parsers em Python. No ficheiro `cql_lexer.py`, definem-se todos os tokens possíveis, bem como as expressões que permitem reconhecê-los.

### Tokens definidos

Os tokens usados cobrem tudo o que é necessário para interpretar instruções CQL. Alguns exemplos são:

- Palavras-chave: `import`, `export`, `select`, `from`, `where`, `procedure`, entre outras.
- Identificadores: nomes de tabelas ou colunas.
- Literais: strings (entre aspas), números inteiros e decimais.
- Símbolos especiais: como `=`, `<>`, `>`, `<`, `>=`, `<=`, `,,`, `;`, `(` e `)`.

O código também garante que palavras como `import` ou `select` sejam reconhecidas como palavras reservadas e não apenas como nomes normais.

### Comentários e espaços

O lexer ignora automaticamente comentários, tanto os de linha única (que começam com `--`) como os de várias linhas (entre `{-` e `-}`). Também ignora espaços em branco e quebras de linha, já que não são relevantes para a estrutura da linguagem.

## Como funciona

Quando o utilizador escreve uma instrução, o lexer lê o texto e identifica cada parte com base nas regras definidas. Por exemplo, ao encontrar a palavra `IMPORT`, sabe que se trata de uma instrução de importação. Se vir um nome como `estacoes` a seguir, assume que é um identificador, e assim por diante. Isto permite ao parser, na fase seguinte, saber exatamente com o que está a lidar.

No final, o analisador léxico fornece ao sistema uma lista de tokens já prontos para serem organizados e interpretados. Esta etapa é fundamental porque qualquer erro aqui pode impedir que o resto do processo funcione corretamente.

## Analizador Sintático e AST

Depois da fase léxica, o interpretador entra na etapa de análise sintática, também conhecida como parsing. Esta fase verifica se a sequência de tokens obtida do analisador léxico segue as regras gramaticais da linguagem CQL. Ou seja, determina se a estrutura dos comandos está correta do ponto de vista sintático.

A análise sintática foi implementada com recurso à biblioteca PLY (Python Lex-Yacc), mais concretamente ao módulo yacc. No ficheiro *cql\_grammar.py*, definem-se as regras da gramática da linguagem CQL, onde cada regra corresponde a uma construção válida, como a definição de uma tabela, a execução de um SELECT, ou a criação de um procedimento.

### Construção da AST

Durante a análise, não só se valida a estrutura, como também se constrói uma Árvore de Sintaxe Abstrata (AST). Esta árvore representa a estrutura lógica dos comandos, de forma hierárquica, sem os detalhes sintáticos desnecessários (como parêntesis ou pontos e vírgulas). Cada nó da árvore corresponde a um tipo de operação, como:

- ImportTableNode – representa um comando de importação de tabela.
- SelectNode – representa uma consulta com SELECT, possivelmente com cláusulas WHERE, LIMIT e JOIN.
- ProcedureNode – representa a definição de um procedimento reutilizável.
- ExportTableNode, PrintTableNode, entre outros.

Cada um destes nós é representado por uma classe Python, e contém os dados relevantes para a operação, como o nome da tabela, colunas a selecionar, condições de filtragem, etc.



## Modulo de Avaliação

O módulo *cql\_eval.py* é responsável por interpretar e executar os comandos representados pela árvore de sintaxe abstrata (AST). É nesta fase que o sistema realiza efetivamente as ações pedidas pelo utilizador, como importar tabelas, fazer seleções, criar novas estruturas ou guardar resultados.

Este módulo está organizado em torno de duas classes principais:

**Memory**, que funciona como um espaço de armazenamento temporário para guardar tabelas e procedimentos definidos durante a execução;

**Evaluator**, que percorre a AST e executa cada instrução, chamando métodos específicos consoante o tipo de nó, por exemplo, `visit_ImportTableNode`, `visit_SelectNode`, `visit_ProcedureNode`, entre outros.

A avaliação dos comandos inclui funcionalidades como:

- Leitura de ficheiros CSV para importar dados;
- Escrita de ficheiros para exportação de resultados;
- Aplicação de filtros com condições (WHERE);
- Limitação de resultados (LIMIT);
- Junções entre tabelas com base numa coluna comum (JOIN);
- Impressão de resultados formatados em forma de tabela.

Este módulo também permite definir e invocar procedimentos (blocos de comandos reutilizáveis), o que torna o uso da linguagem mais flexível e organizado.

Em resumo, o *cql\_eval.py* é o componente que dá “vida” ao interpretador, executando o que foi definido nas fases anteriores e produzindo os resultados esperados com base nos dados fornecidos.

## Execução do Programa

O ficheiro *main.py* serve como ponto de entrada do projeto. É através dele que o interpretador é lançado e que o utilizador pode interagir com o sistema. Este módulo permite dois modos de funcionamento: executar scripts completos ou utilizar o modo interativo, onde se escrevem os comandos um a um.

Quando o programa é iniciado, o sistema verifica se foi passado algum argumento na linha de comando. Se for um ficheiro .csv, ele é importado e mostrado automaticamente. Se for um ficheiro de texto com comandos (como .cql), o interpretador lê e executa todas as instruções nele contidas. Caso não haja ficheiros, o utilizador entra no modo interativo (REPL), onde pode escrever comandos CQL diretamente no terminal.

O *main.py* também fornece alguns comandos especiais como :help para mostrar ajuda e :quit para sair. Além disso, garante que cada comando termina com ponto e vírgula, facilitando o uso da linguagem e prevenindo erros.

Este módulo junta todas as peças do sistema sendo elas lexer, parser e eval e também que permite que o utilizador interaja com o interpretador de forma simples e intuitiva.

## Exemplos de Uso e Testes

Para garantir o bom funcionamento do interpretador CQL, foram realizados dois tipos de testes: um em modo interativo, e outro em modo de execução por ficheiro de entrada.

### Dados de exemplo

Os ficheiros `estacoes.csv`, `observacoes.csv`, `jogadores.csv` e `estatisticas.csv` guardados na pasta `data`, servem como base para testar as instruções de importação, filtragem, junção e exportação. Estes ficheiros contêm dados estruturados com cabeçalhos, tal como exigido pela linguagem.

### Scripts de demonstração

Na pasta `input` encontra-se o ficheiro *exemplo.cql* e *exemplo2.cql*, que contém um conjunto de instruções CQL organizadas por blocos. Este script demonstra o funcionamento das principais operações da linguagem, como:

- Importar tabelas com `IMPORT TABLE`;
- Ver dados com `PRINT TABLE` ;
- Executar consultas com `SELECT` (com ou sem `WHERE` e `LIMIT`);
- Criar tabelas novas com base em filtros ou `JOINS`;
- Renomear ou descartar tabelas;
- Exportar resultados para ficheiros `.csv`;
- Definir e chamar procedimentos reutilizáveis;

Este ficheiro pode ser executado diretamente, permitindo testar o sistema de forma automatizada.

## Resultados

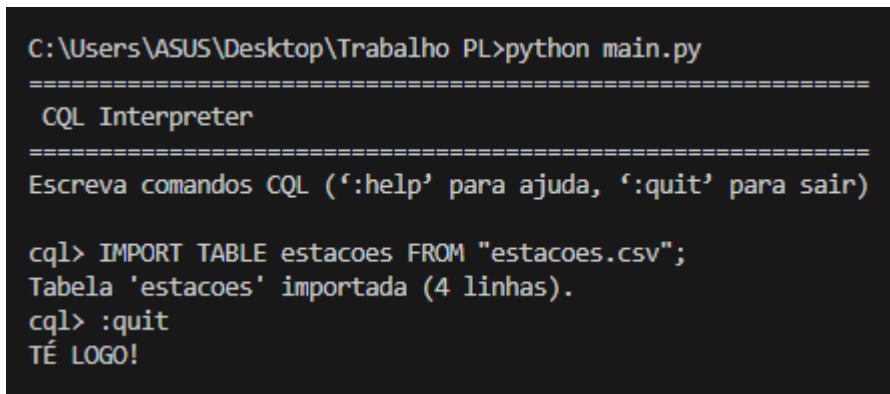
### Teste 1 – Execução em Modo Interativo

Neste teste, o interpretador foi iniciado com o seguinte comando: `python main.py`

Este modo permite escrever comandos CQL diretamente no terminal. Foram testadas instruções como:

```
IMPORT TABLE observacoes FROM "estacoes.csv";  
  
SELECT * FROM observacoes WHERE Temperatura > 22;  
  
EXPORT TABLE estacoes AS "est.csv";  
  
RENAME TABLE estacoes est;  
  
DISCARD TABLE est;
```

O sistema respondeu corretamente a todos os comandos, imprimindo os resultados, exportando ficheiros e manipulando tabelas conforme o esperado.



```
C:\Users\ASUS\Desktop\Trabalho PL>python main.py  
=====  
CQL Interpreter  
=====  
Escreva comandos CQL (':help' para ajuda, ':quit' para sair)  
  
cql> IMPORT TABLE estacoes FROM "estacoes.csv";  
Tabela 'estacoes' importada (4 linhas).  
cql> :quit  
TÉ LOGO!
```

Figura 1 - Execução em modo interativo

## Teste 2 – Execução com Ficheiro de Entrada

O segundo teste consistiu em correr o interpretador com um ficheiro de script .cql, que contém múltiplas instruções CQL encadeadas: `python main.py input/exemplo.cql`

O ficheiro *exemplo.cql* inclui blocos para importação, visualização, filtragem, criação de tabelas, procedimentos e exportação de dados.

Durante o teste, o interpretador leu e executou todas as instruções do ficheiro sem erros, produzindo os resultados esperados no terminal e nos ficheiros de saída.

```
C:\Users\ASUS\Desktop\Trabalho PL>python main.py input/exemplo.cql
Tabela 'estacoes' importada (4 linhas).
Tabela 'observacoes' importada (4 linhas).

Tabela: estacoes
-----
Id | Local | Coordenadas
-----
E1 | Terras de Bouro/Barral (CIM) | [-8.31808611,41.70225278]
E2 | Graciosa / Serra das Fontes (DROTRH) | [-28.0038,39.0672]
E3 | Olhão, EPP0 | [-7.821,37.033]
E4 | Setúbal, Areias | [-8.89066111,38.54846667]
-----
Total: 4 linhas
```

Figura 2 - Início da execução do ficheiro exemplo.cql

```
Tabela 'mais_quentes' criada (3 linhas).
Procedimento 'atualizar_observacoes' guardado.
Tabela 'completo' criada (4 linhas).

Tabela: completo
-----
Id | Local | Coordenadas | IntensidadeVentoKM | Temperatura | Radiacao | DirecaoVento | IntensidadeVento | Humidade | DataHoraObservacao
-----
E1 | Terras de Bouro/Barral (CIM) | [-8.31808611,41.70225278] | 2.5 | 23.2 | 133.2 | NE | 0.7 | 58.0 | 2025-04-10T19:00
E2 | Graciosa / Serra das Fontes (DROTRH) | [-28.0038,39.0672] | 15.1 | 12.5 | 679.6 | E | 0.0 | 4.2 | 2025-04-10T19:00
E3 | Olhão, EPP0 | [-7.821,37.033] | 4.0 | 16.4 | 0.0 | NE | 0.0 | 1.1 | 2025-04-10T19:00
E4 | Setúbal, Areias | [-8.89066111,38.54846667] | 3.6 | 16.8 | 1.6 | SW | 0.0 | 1.0 | 2025-04-10T19:00
-----
Total: 4 linhas

Descartada.
Descartada.
Descartada.
Descartada.
```

Figura 3 - Fim da execução do ficheiro exemplo.cql

## Conclusão

Este projeto permitiu desenvolver um interpretador para a linguagem CQL, cobrindo todas as fases essenciais de um sistema deste tipo: análise léxica, análise sintática, construção da AST, interpretação e execução. A implementação em Python, com o apoio da biblioteca PLY, revelou-se adequada para este tipo de trabalho e facilitou a criação de uma linguagem simples, mas com funcionalidades úteis para manipulação de dados em ficheiros CSV.

Ao longo do desenvolvimento, foi possível aplicar e consolidar conhecimentos sobre gramáticas, estruturas de dados, e modularização de código. A organização em ficheiros separados, com responsabilidades bem definidas, contribuiu para a clareza e manutenção do sistema. O suporte a comandos interativos, execução de scripts, operações de junção, criação de tabelas e reutilização de procedimentos mostra a flexibilidade da solução.

No final, o interpretador demonstrou ser eficaz e estável, permitindo ao utilizador explorar dados de forma prática e intuitiva através de uma linguagem própria. O trabalho cumpriu os objetivos propostos e deixou espaço para melhorias futuras, como o suporte a novos operadores, mais tipos de filtros ou até interfaces gráficas para facilitar ainda mais a interação com os dados.

## Bibliografia

- Conteúdo fornecido no Moodle pelo Professor Óscar Ribeiro
- <https://www.youtube.com>