

**CQL**

**Trabalho Prático de Processamento de Linguagens**

**Grupo 06:**

**23016 – Dani Carvalho da Cruz**

**23279 – Hugo Ferreira Baptista**

**23010 – Hugo Tiago Mendes Cruz**

**Professor: Óscar Ribeiro**

**Licenciatura em Engenharia de Sistemas Informáticos**

Barcelos | maio, 2025

**Índice**

[1. Introdução 3](#_Toc198222854)

[2. Descrição da Linguagem CQL 4](#_Toc198222855)

[2.1. Tipos de Comandos 4](#_Toc198222856)

[A. Configuração de Tabelas de Dados 4](#_Toc198222857)

[B. Queries 4](#_Toc198222858)

[C. Criação de Novas Tabelas 5](#_Toc198222859)

[D. Procedimentos 5](#_Toc198222860)

[3. Especificação da Gramática (BNF) 6](#_Toc198222861)

[4. Reconhecedor Léxico (Lex) 7](#_Toc198222862)

[5. Reconhecedor Sintático (Yacc) 7](#_Toc198222863)

[6. Árvore de Sintaxe Abstrata (AST) 8](#_Toc198222864)

[7. Semântica e Execução 9](#_Toc198222865)

[8. Exemplos de Utilização 9](#_Toc198222866)

[9. Conclusão 10](#_Toc198222867)

[10. Referências 11](#_Toc198222868)

# Introdução

Este relatório descreve o desenvolvimento de um projeto prático realizado no âmbito da unidade curricular de **Processamento de Linguagens**, do curso de **Licenciatura em Engenharia de Sistemas Informáticos**, no ano letivo de **2024/2025**. O principal objetivo deste trabalho foi proporcionar aos estudantes a oportunidade de adquirir experiência prática na conceção e implementação de analisadores léxicos e sintáticos, bem como na definição de ações semânticas associadas a uma linguagem formal.

O desafio proposto consistiu na criação de uma linguagem de consulta a bases de dados, denominada **CQL (Comma Query Language)**, concebida para operar sobre ficheiros de texto no formato **CSV (Comma Separated Values)**. Esta linguagem permite efetuar operações de configuração, manipulação e consulta de tabelas de dados armazenadas em memória, assim como a criação de novas tabelas e definição de procedimentos reutilizáveis.

Para a concretização desta aplicação, foi utilizada a linguagem de programação **Python**, em conjunto com a biblioteca **PLY (Python Lex-Yacc)**, que disponibiliza as ferramentas necessárias para a implementação de reconhecedores léxicos e sintáticos, semelhantes aos conhecidos, **Lex** e **Yacc** em C.

O desenvolvimento do projeto seguiu as seguintes fases:

1. Definição da gramática concreta da linguagem CQL.
2. Implementação de um reconhecedor léxico capaz de identificar os tokens da linguagem.
3. Construção de um reconhecedor sintático, responsável por validar a estrutura das instruções.
4. Definição da **Árvore de Sintaxe Abstrata (AST)** e associadas ações semânticas.
5. Desenvolvimento de um mecanismo de execução das instruções a partir da AST, permitindo manipular dados em memória e interagir com ficheiros CSV.

Além disso, o projeto incluiu a implementação de funcionalidades para importar e exportar tabelas em formato CSV, realizar seleções condicionais, efetuar junções de tabelas e definir procedimentos reutilizáveis. O sistema contempla ainda o suporte a comentários de linha e de bloco, melhorando a legibilidade e manutenção do código-fonte escrito em CQL.

Este relatório encontra-se organizado de forma a acompanhar a sequência de desenvolvimento do trabalho, iniciando-se com a descrição da linguagem e da sua gramática, passando pela conceção dos reconhecedores, estruturação da árvore de sintaxe abstrata, definição da semântica e execução das instruções, até à apresentação de exemplos de utilização e considerações finais.

# Descrição da Linguagem CQL

A **CQL (Comma Query Language)** é uma linguagem de consulta desenhada especificamente para manipulação e análise de dados armazenados em ficheiros **CSV**. Inspirada em linguagens de consulta a bases de dados relacionais, como o **SQL**, a CQL permite aos utilizadores executar operações de importação, exportação, filtragem e junção de dados, bem como definir procedimentos que agrupam várias instruções.

# Tipos de Comandos

A linguagem CQL organiza-se em diferentes tipos de comandos, classificados de acordo com a sua funcionalidade

# Configuração de Tabelas de Dados

Estes comandos permitem a gestão de tabelas de dados em memória, incluindo a sua importação a partir de ficheiros CSV, exportação para ficheiros, alteração de identificadores e eliminação de dados.

IMPORT TABLE obs FROM "observacoes.csv";

EXPORT TABLE obs AS "obs.csv";

DISCARD TABLE obs;

RENAME TABLE obs observacoes;

PRINT TABLE observacoes;

# Queries

Permite a consulta de tabelas em memória, com possibilidade de seleção de colunas e de aplicação de condições.

SELECT \* FROM observacoes;

SELECT DataHoraObservacao,Id FROM observacoes;

SELECT \* FROM observacoes WHERE Temperatura > 22;

SELECT \* FROM observacoes WHERE Temperatura > 10 AND Radiacao > 100;

SELECT \* FROM observacoes LIMIT 2;

SELECT \* FROM observacoes WHERE Temperatura > 10 AND Radiacao > 100 LIMIT 1;

# Criação de Novas Tabelas

Permite a criação de novas tabelas em memória, quer a partir de queries, quer a partir da junção de outras tabelas.

CREATE TABLE quentes SELECT \* FROM observacoes WHERE Temperatura > 22;

CREATE TABLE quenteslimit SELECT \* FROM observacoes LIMIT 2;

CREATE TABLE completo FROM est JOIN observacoes USING(Id);

# Procedimentos

Permite agrupar várias instruções numa só entidade reutilizável, que pode ser chamada a qualquer momento.

PROCEDURE selecionar DO

SELECT \* FROM observacoes WHERE Temperatura > 22;

SELECT \* FROM observacoes WHERE Temperatura > 10 AND Radiacao > 100;

SELECT \* FROM observacoes LIMIT 2;

END

Executa o procedimento definido.

CALL selecionar;

Apagar o precedirmento defenido.

DELETE selecionar;

# Comentários

IMPORT TABLE obs FROM "observacoes.csv"; -- Comentario

SELECT \* FROM observacoes;

{-PROCEDURE selecionar DO

SELECT \* FROM observacoes WHERE Temperatura > 22;

SELECT \* FROM observacoes WHERE Temperatura > 10 AND Radiacao > 100;

SELECT \* FROM observacoes LIMIT 2;

END-}

# Especificação da Gramática (BNF)

**G = <T, N, S, P>**

**Terminais (T):** "IMPORT", "TABLE", "FROM", "AS", "DISCARD", "RENAME", "PRINT", "SELECT", "WHERE", "LIMIT", "CREATE", "JOIN", "USING", "DO", "END", "CALL", "DELETE", '\*', ',', ';', '(', ')', id, file, num, numdec, string, operador

**Não Terminais (N):** PROG, CMDLIST, CMD, CONF, QRS, SELEC, COLLIST, CONDLIST, COND, OPERADOR, VALOR, NEW, PROCS

**Axioma (S): PROG**

P1: PROG → CMDLIST

P2: CMDLIST → CMD

P3: | CMD CMDLIST

P4: CMD → CONF

P5: | QRS

P6: | NEW

P7: | PROCS

P8: CONF → "IMPORT" "TABLE" id "FROM" file ';'

P9: | "EXPORT" "TABLE" id "AS" file ';'

P10: | "DISCARD" "TABLE" id ';'

P11: | "RENAME" "TABLE" id id ';'

P12: | "PRINT" "TABLE" id ';'

P13: QRS → "SELECT" SELEC "FROM" id ';'

P14: | "SELECT" SELEC "FROM" id "WHERE" CONDLIST ';'

P15: | "SELECT" SELEC "FROM" id "LIMIT" num ';'

P16: | "SELECT" SELEC "FROM" id "WHERE" CONDLIST "LIMIT" num ';'

P17: SELEC → '\*'

P18: | COLLIST

P19: COLLIST → id

P20: | id ',' COLLIST

P21: CONDLIST → COND "AND" CONDLIST

P22: | COND

P23: COND → id OPERADOR VALOR

P24: OPERADOR → operador

P25: VALOR → numdec

P26: | string

P27: | num

P28: NEW → "CREATE" "TABLE" id QRS

P29: | "CREATE" "TABLE" id "FROM" id "JOIN" id "USING" '('id')' ';'

P30: PROCS → "PROCEDURE" id "DO" CMDLIST "END"

P31: | "CALL" id ';'

P32: | "DELETE" id ';'

CMDLIST: permite ter apenas um comando ou vários.

id: identificador de tabelas colunas e procedures. Faz a ponte para as palavras reservadas.

file: arquivo de entrada ou saída (ex: “observacoes.csv”, “data/observacoes.csv” ).

VALOR: permite ter valores decimais inteiros ou strings.

COND: Condição usada para os filtros

CONDLIST: permite ter uma coluna ou varias

# Reconhecedor Léxico (Lex)

**Como foi definido o lexer (tokens principais).**

* Exemplos de tokens: IMPORT, SELECT, num, id, string, comentários, etc.
* Desafios (ex: leitura de strings com vírgulas ou aspas).

# Reconhecedor Sintático (Yacc)

# Árvore de Sintaxe Abstrata (AST)

* Objetivo: representar comandos ignorando açúcar sintático.
* Estrutura de representação:{'op': 'SELECT', 'args': [...]} etc. Exemplos concretos: Um SELECT simples Um SELECT com WHERE + LIMIT
* Um procedimento Comenta como foi pensada a estrutura seq para comandos múltiplos.

# Semântica e Execução

* Como cada comando AST é executado: IMPORT, EXPORT, SELECT, JOIN, etc. Como se faz JOIN de duas tabelas.
* Como se aplicam filtros WHERE e limites.
* Como são armazenadas as tabelas em memória (dicionário)

# Exemplos de Utilização

* 2 a 3 trechos de código .cql.
* AST gerada.
* Resultado produzido no terminal.
* Breves comentários sobre cada um.

# Conclusão

# Referências