Practical work report
Language Processing


# Text Query Language

**Teacher**
Alberto Simões

**Student**
20807 -> Bruno Aurélio Coelho Dantas

January 13, 2022

LESI

## Summary

The present report, is meant to document the academic practical work.
The practical work has two themes to choose from. The theme that was chosen to carry out in this practical work was the theme **B**.

This theme consists of the implementation of a variant of **SQL** but simpler that works in **CSV** files.

## Contents

# 1 Introduction

As mentioned in the summary, this project is an academic project that is part of the language programming discipline in the second year.

The pproject consists on applying the content taught on the subject´s class, where the student should mainly, be alle to create the **regular expressions** that adapts to read the **csv** files and the commands of the **TQL (Text Query Language),** also a **grammar** to check if the token entered by the user is valid in the TQL language, and a function to validate the command and execute it to present the result

This work was presented to us by professor Alberto Simões , in the 2021/2022 school year and the statement is present in the work folder

# 2 TQL (Text Query Language)

## 2.1 Identify the keywords

In SQL language we have a group of keywords to create the Query and each one has a specific function

## 2.2 List of keywords and their functions

- LOAD -> Command to Read and save data from CSV
- DISCARD -> Command to delete the table and its data
- SAVE -> Command to save the table on CSV
- SHOW -> Command to show a table
- CREATE -> Command to Create a new table
- TABLE -> Command to select the table
- FROM -> Command to select the table
- WHERE -> Command to add one condition
- AND -> Command to add one more condition
- JOIN -> Command to join two tables
- USING -> The column and value to join the two tables
- SELECT -> To be displayed or compare the values
- column-name -> Name of the Column
- table-name -> Name of the table where the data is
- Number -> A Number to be compared in the database
- string -> A text to be compared in the database

- AS -> Name of the file in which the data will be saved
- LIMIT -> Limit the number of rows that will be displayed

### 2.3 Some of the possible queries

Table Handling

1. **LOAD TABLE table-name FROM "file-name.csv";**
2. **DISCARD TABLE table-name;**
3. **SAVE TABLE table-name AS "file-name.csv";**
4. **SHOW TABLE table-name;**

Execution of Queries

1. **SELECT * FROM table-name;**
2. **SELECT * FROM table-name LIMIT number;**
3. **SELECT column-name, column-name; FROM table-name;**
4. **SELECT * FROM table-name JOIN table-name USING(column-name = string);**

5. **SELECT * FROM table-name; WHERE column-name = strung;**
6. **SELECT * FROM table-name; WHERE column-name > number LIMIT 5;**
7. **SELECT * FROM table-name; WHERE column-name > number AND column-name = string LIMIT 5;**

Creation of new Tables

1. **CREATE TABLE table-name FROM SELECT * FROM table-name WHERE column-name > number;**
2. **CREATE TABLE table-name FROM table-name JOIN table-name USING(column-name = string);**

# 3 Regular expression

## 3.1 TQl (Text Query language)

The first step was to take all the keywords in the languages and transform them into tokens.

In the second step we will create a regular expression for the [column-name, table-name] = **var**, and one for **strings** and one for **numbers**

- **t_str** -> Regular expression to identify the command
  **exemple: (LOAD|SELECT|FROM|CREATE)**
- **t_string** -> Regular expression to identify the strings that are enclosed in quotes
  **exemple : ("Hello"|"Roger")**
- **t_nr** -> Regular expression to identify integers and floating numbers
  **exemple : (0|1.5)**
- **t_var** -> Regular expression to identify all words that are not keywords and m are in quotes that can have upper and lower case letters as an underscore
  **exemple : (collun_name|table_name)**

In addition to the tokens we also have the literary ones that are the operators **Exemple:** '>' | '<' | '=' | '*' | ';' | ',' | '(' | ')'

### 3.2 CSV file

To create the lexer to read CSV files we have to pay attention to some points.

- First row always have the table header
- If the line starts with a hashtag that line and a comment then it can be ignored.
- Column values are separated by commas.
- To have commas in the middle of the value this value has to be inside quotes

1. **t_COMMENTS** -> Will read from a hashtag until it finds a newline.
2. **t_QUANTATION** -> Will read from one quote to another quote
3. **t_COMMA** -> Read the values that are between commas

To save the file data, a data class was created. Inside the data class we will manipulate a dictionary.

Manipulate dictionary

https://www.overleaf.com/project/61df87a178a8669c9259b716
Example = {TableNmae: {Key: [Values], key2: [Values], ... }} Check the existence of **"tableName"** and **"keys"**

and if they don't exist, create them

The data will be saved in your own key

Example: Ficheiro CSV "people.csv"

id,firstname,lastname
100,Christal,Fosque
101,Ilse,Zachary
102,Audrie,Hathaway
103,Letizia,Revkah

On dictionary:
dicionary = { "people": {
"id": [100, 101, 102, 103],
"frisname": [ "firstname, "ChristalIlse", "Audrie", "Letizia"
],
"lastname": Fosque"", "Zachary", "Hathaway", "Revkah"[]
} }

# 4 TCL Grammar

The grammar created for this project can evaluate one or more commands of query.
Within grammar we have 2 main operations

**Commands** -> Checks if the query entered by the user is valid
For the creation of the grammar it was decided by keywords in the program was to evaluate the tokens in order if there is any token that is not in the grammar or because the command does not exist or is not in the correct order it returns an error.

**Function** -> In this part of the grammar we will have two ways, one that will be to create a new function and the other will be to call an existing function.
In this part we are going to create functions, in which we are going to have two important points which consist of a variable and a list of commands
In the second part it will be where the CALL of the function will be evaluated.

## 5 Query evaluate and run from command

In this part of the program we are going to evaluate the tree generated by the grammar and we are going to transform it(s) again in command(s) that we can call the function and execute it.

### 5.1 Evaluate

In this function we will evaluate the type of **"ast"** that was returned.
If it is a list type, it will call this function again, If it is a dictionary type, it will call the function evaluate Command, if not return error

### 5.2 Evaluate Command

In this function we will evaluate if it is a command or function. if it is an **"ast"** function it will be sent to the Evaluate Operator function, if it is a function type ast will be sent to the Create function, if not return error

### 5.3 Evaluate Operator

In this function we will evaluate the operator types. We are going to stay in this function until we find an operator called '**args**' which is equal to '**;**' or '**END'**

### 5.4 Create Function

In this we will evaluate if there is already a function with the name that was inserted after the program will call the Evaluate Operator to evaluate the list of commands

Example of **ast**

{'Command': 'LOAD', 'args': {'TABLE': 'produtos', árgs':{'FROM': {'str': 'produtos2.csv'}, 'args': ';'}}}

{'Command': 'LOAD', 'TABLE': 'produtos', 'FROM': {'str': 'produtos2.csv'}, 'end': ';'}]

{'Command': 'SHOW', 'args': {'TABLE': 'produtos', 'args': ';'}}

Example of **AST evaluated**

{'Command': 'LOAD', 'TABLE': 'produtos', 'FROM':{'str': 'produtos2.csv'}, 'end': ';'}}

{'Command': 'SHOW', 'TABLE': 'produtos', 'end': ';'}