HaSQL Query Language

Dimitar Tsvetkov, ddt1u19@soton.ac.uk
Gerasim Tsonev, gt2u19@soton.ac.uk
Kaloyan Spirov, kcs1u19@soton.ac.uk

1. Introduction

HaSQL is a conjunctive-filtering query language for CSV files allowing for programming freedom and the creation of a wide variety of queries. Its programs are written in CQL files and upon runtime, HaSQL supports the output of errors in the terminal. The overall design syntax of HaSQL is inspired by SQLite and Haskell.

2. User Manual

In this part, we will provide all the needed information about all the syntax statements of HaSQL's queries and how they affect the output of the program.The overall structure of every query in HaSQL takes the following form:

show (*Columns* **) where** *Requirements*

Lets see an example:

```
show (a1,a3)
where "A"(a1,a2,a3)
& a1 = a2
```

Here we will output columns *a1* and *a3* concatenated one after another in order of declaration, which are taken from the CSV file named *A.csv* that has 3 columns. Also having "*& a1 = a2*" as a statement will filter and output only the results from the rows where *a1* is equal to *a2*.
Whether you decide to separate the statements or operations on different lines does not matter for HaSQL.The same query can also be written like this:

```
show (a1,a3) where "A"(a1,a2,a3) & a1 = a2
```

Keywords:

show (c1,c2,c3)

Every cql query must contain a show() statement at the beginning, stating which of the column names will be output in the end result. The column names you enter in show() must be valid declared variables later when declaring CSV Tables.

For example:

```
show (b1,b2)
where "B"(b1,b2)
```

Will output the two columns b1 and b2 of csv file B but:

```
show (b1,something)
where "B"(b1,b2)
```

Will output an error, because column variable "something" has not been declared anywhere.

where

After the “where” keyword we specify the files we are reading, which we separate with ‘&’. We write the file name without the .csv extension in quotation marks. In parentheses, we list the variables divided by commas.

&

Simply means “and” and is a concatenating unit symbol for all the operations and assignments. It is left-associative, so the operation on the left of & will be the first to be evaluated.

Strings “Example”

All string statements are surrounded with double separated commas “ ”
This design decision allows us to have any mixture of strings containing capital or small letters and numbers.

Requirements:

“Name”(x1,x2, _ , x4) - CSV Table Declaration

This is our way of declaring CSV tables. The above statement will search for the Name.csv file in the directory of the interpreter. We have given 4 fields, separated with commas so it will assume that the CSV file also has 4 fields, separated with 3 commas for each line.The query requirement will read the whole table Name.csv and save each column in order of declaring as the given variable.
Having the “_” instead of a variable is an option to ease the declaration process if you do not care to give a name for the column and you are not going to use it.

Multiple CSV Table declaration

You can declare multiple csv tables one after another or even the same table and they will be ***conjoined*** with each other in order of declaration, for instance:

show (c1,c3)
where “A”(a1,a2) & “B”(b1,b2) & “c”(c1,c2,c3)

if () then () else ()

Our “*if then else*” statement contains 3 parts.First, inside the brackets after the *if* you will have constraints that will check if all of them are true for each row in the current table.
If the current row being checked complies with the constraints, the operation statements for the *then()* will be evaluated, otherwise, the operation statements in the *else()* part will be the ones evaluated.
Inside the *if* block you can have different constraints concatenated with an ‘*&*’.
Inside the *then* and *else* blocks you can have operations concatenated with an ‘*&*’.
Example structure:

if(a1=”Car” & a1!=a2)
then(a1<-“Tesla”) else(a1<-”Not Car” & a2<-a1)

This will check if a row’s column *a1* variable is “*Car*” and if *a1* is not equal to *a2*.Then if those are true, *a1* will become “*Tesla*” on the current row, otherwise, a1 will became “Not car” and *a2* will become the new value of a1(Not Car).

if () then ()

Similar to the above *if () then () else ()* statement, but it does not evaluate any operations when the *if()* constraints evaluate to false

Operations - Are also a type of requirement and are split into two categories

- **Constraints**
  - **var = var** - Constraint for two column variable values to be equal;
  - **var != var** - Constraint for a column variable values to be different to one another;
  - **var = "String"** - Constraint for a column variable to be equal to the given string;
  - **var != "String"** - Constraint for a column variable to be different of the given string;

- **Assignments**
  - **var <- var** - The first variable takes the value to the second one;
  - **var <- String** - The variable takes the value of the string;

Comments:

You can have line comments by adding a double dash(--) to a line. Everything after the double dash on that row will be considered as a comment. The comments are similar to those in Haskell.

```
show(a1)
from "A"(a1) -- This is a comment
```

3. Extras

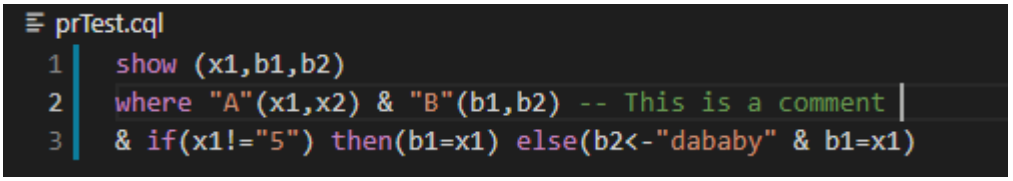
Error Output

Whenever an invalid syntax is detected , the lexer will inform you of the specific position of the potential error, as well as the type of it. If you have an undeclared variable in the code, you will receive an error stating which variable is undeclared.

```
Problem with Lexing : Parse error at line:column 3:2 With Variable p4
```

```
Problem with Lexing : Variable notStated1 not found!
```

Syntax Highlighting for Visual Studio Code



We have included a Syntax highlighter extension for HaSQL written with TextMate.
To add it to VS Code, copy the folder "cql-hasql-syntax-highlighter" to your VS extension folder.