QLBasic

Programming Language Manual

1. *Description*

QLBasic (Query Language Basic) is a domain-specific programming language for querying CSV documents. Its simplicity and concept are based on SQL, so every user who is familiar enough with this kind of language should not have trouble using QLBasic.

2. *Syntax*

This programming language is made up of a list of variables and a list of constraints. Variables are first and separated by commas. Constants are mixed with variables and are within square brackets.

After that is the constraints list, which is encompassed with curly brackets and each constraint is separated by a semicolon. There are two types of constraints - a table constraint and an equality constraint. The first one is made up of the name of the csv file to be read (without the .csv extension) and a list of variables in regular brackets. With this constraint, each variable in the brackets is assigned a column from the file in the order they are written in.

The other type of constraint has two uses either as a way to join two tables or to check whether two variables are equal. Variables must begin with lowercase letters and tables (csv files) - with capital letters.

2.1. Some examples

As to make things more clear, here is an example file called Books.csv consisting of 3 fields - the name of the book, year of publication and the author:

```
Don  Quixote  , 1605 , Miguel de Cervantes
Pinocchio , 1883 , Carlo Collodi
Lord of the rings , 1954 , J.R.R. Tolkien
Alice's  Adventures  in  Wonderland  , 1865 , Lewis Carroll
```

Using QLBasic we would like to perform queries on this data in order to extract information :

```
book  {
        Books (book , year , author ) ;
}
```

All the values from the first column of Books.csv will be the output of this query.

As for equality constraints, here is an example file called Countries.csv, consisting of 2 columns for the capital city and the country name:

```
Panama  , Panama
Sofia , Bulgaria
San Marino , San Marino
Greece , Athens
```

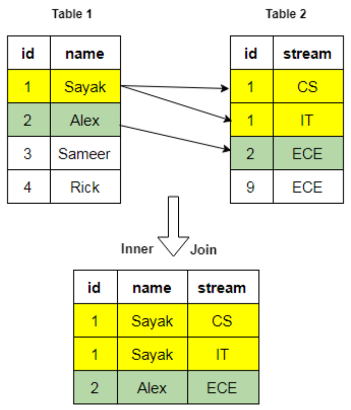
Now, we would like to extract only those countries whose capital city's name is the same as theirs. We write the following program:

```
first ,   second  {
       Countries ( first  ,    second ) ;
        first   =   second ;
}
```

We can see that the third line of the code demonstrates a new constraint on the output. The output of the program will have two columns and only rows, where the variables are equal, will be included in the output of the program.

Using joins, data from several tables can be combined in a single query:

```
name ,   stream  {
       Staff ( id,  stream ) ;
       People ( name ,   id ) ;
}
```



id	name
1	Sayak
2	Alex
3	Sameer
4	Rick

id	stream
1	CS
1	IT
2	ECE
9	ECE

id	name	stream
1	Sayak	CS
1	Sayak	IT
2	Alex	ECE

Here is an example of using constants:

```
name , [0] {
        People (name) ;
}
```

The output would be the name with a 0.

3. Error reporting

There are a couple of errors that may occur. Here are some examples of the error reporting messages:

3.1. Lexical error

```
Interpreter.hs: lexical error at line 1, column 3
CallStack (from HasCallStack):
  error, called at templates\wrappers.hs:349:35 in main:Lexer
```

An error resulting from a lexical analysis.

3.2. Parse error

```
Interpreter.hs:ParseError-Line 1, Column 8-Near'{'
CallStack (from HasCallStack):
  error, called at Main.hs:22:22 in main:Main
```

An error resulting from parsing the tokens.

3.3. Unconstrained variable

```
Interpreter.hs:Variable unconstrained:z
CallStack (from HasCallStack):
  error, called at .\Interpreter.hs:71:24 in main:Interpreter
```

An error resulting from referring to a variable that has not been bound to a column of a table.