Script Syntax Definition

Paul Bakker	Robin Borst	Matthijs de Groot	Julian Hols
pbakker3	robinborst	mawdegroot	jthols
4326091	4291972	4171683	4247930

Jan van Rosmalen
javanrosmalen
4318609

June 24, 2015

Contents

1 Introduction

The program's operations are executed according to the specifications that are put into the script file. The script file contains the information on what tables the operations are executed, what the parameters are, etc. The script syntax are the rules that define how the script file must be formatted for the program to correctly parse the operations. The exact definition of the syntax is described in this document.

2 Common

There are several parts of the syntax that are not used on their own, but are instead the build stones that are used to create the syntax of more complex items. The parts of the syntax that are shared among multiple syntax elements are defined here.

2.1 ID

ID is a special type of text that is used to specify table names and field names. To create a straightforward definition of ID lets first define:

character class	matching characters
`[a-z]`	Any character from `a` to `z`
`[A-Z]`	Any character from `A` to `Z`
`[0-9]`	Any character from `0` to `9`
`[_]`	The character `_`
`*`	The character set before it, any number of times.

```
[a-z][A-Z][_] [a-z][A-Z][0-9][_]*
```

2.1.1 Examples

```
[table]
[atable0]
[a_table]
[a_table_0]
```

2.2 Table

The table syntax is used to specify a table to the various components of the operations. It is defined as the table name surrounded by `[` and `]`

```
[ ID ]
```

2.2.1 Examples

Say we have three tables, one table is named "table1", one table is called "imatable" and one table is called "is_a_table". We could then call these tables in the script as follows:

```
[table1]
[imatable]
[is_a_table]
```

2.3 Field

A field is defined similarly to a table but it also has a second pair of `[` and `]`'s containing the field name, separated from the table field with a dot `.`

```
[ ID ].[ ID ]
```

2.4 Pattern

A pattern is defined as a set of pattern descriptions. The patterns descriptions are concatenated to form a pattern.

```
pattern_description
pattern_description pattern_description
```

2.5 Pattern description

A pattern description can be used to define a pattern that has to occur or a pattern that may not occur.

```
{ count record_condition }
{! pattern_description }
```

2.6 Count

A count is used to specify how many times a said pattern must occur, it can either by any number of times > 1 (*) or a specific number of times.

```
*
number
```

2.7 Record Condition

A record condition specifies either a table, or a condition on a field.

```
table
field condition
```

2.8 Condition

A condition is defined as a compare operator and a value to compare to.

```
compare_operator value
```

2.9 Number

A number is defined as either a integer, a decimal or a decimal followed by an exponent.

```
[0-9]*
[0-9].[0-9]*
[0-9].[0-9]* e+[0-9]*
[0-9].[0-9]* e-[0-9]*
```

2.10 Compare Operator

The compare operator is defined as either one of the normal logical operators:

```
==
!=
>=
>
<=
<
```

2.11 Compute Operator

The compute operator is defined as one of the following operations.

```
AVG()
COUNT()
MAX()
MIN()
STDEV()
SUM()
```

2.12 Date

A date is specified in ISO format surrounded by `Date(` and `)`.

```
DATE(2000-10-10)
DATE(2010-10-15)
```

2.13 Text

A text is specified by a string surrounded by quotes.

```
"astring"
"somerandomstring"
"etc"
```

2.14 Value

A value can be either a date, a number or a text.

```
date
number
text
```

3 Operations

The operation commands are used to specify with which parameters the program must execute an operation. The parameters that can be specified for an operation are specified using the syntaxes described in the previous section.

3.1 Between Operation

The between operation calculates the time in hours between two events. The between operation requires a column where it should find the event, a column where it should find the date, a second column where it should find the date of the other column, a value of the first event and lastly a value for the event to find the time between. The two date columns can be the same column if both events use the same date column.

```
BETWEEN field field field value value
```

3.1.1 Example

```
BETWEEN event date second_date "eat" "sleep"
BETWEEN event date date 20 50
```

3.2 Chunking Operation

The chunking operation is used to split the dataset into several smaller subsets of the dataset in order to execute processing on the subsets. One could for example chunk the dataset on MONTH, which would create the possibility to execute operations on a per-month basis.

There are several chunk types available to the user, first there is the chunk type YEAR which retrieves one chunk for every year. There is a chunk type MONTH which retrieves one chunk for every month.

There is the chunk type DAY which retries one chunk for every day. Finally there is the special chunk type PHASE, it creates a chunk for every phase of the ADMIRE study. The PHASE chunk operation can be used to find differences in behaviour between a patient who has just been transplanted and a patient who had a transplant a long time ago.

```
YEAR
MONTH
DAY
PHASE
```

The chunking operation must specify a table to chunk, a chunk type to use and for chunk types texttttYEAR, `MONTH`, `DAY` a number to specify how many years, days or months must be represented as one chunk.

```
CHUNK field ON YEAR number
CHUNK field ON MONTH number
CHUNK field ON DAY number
CHUNK field ON PHASE 1
```

3.2.1 Example

```
CHUNK [table].[date] ON YEAR 1
CHUNK [table].[date] ON MONTH 1
CHUNK [table].[date] ON DAY 1
CHUNK [table].[date] ON DAY 7
CHUNK [table].[date] ON PHASE 1
```

3.3 Coding Operation

The coding operation is used to attach codes to records in the dataset that match a certain pattern. The pattern that must me matched can be specified using the syntax. A pattern is build up of a set of pattern descriptions that are concatenated together.

The coding operation requires a table to be specified, a pattern and finally a code label to attach to the found codes.

```
CODE table ON pattern AS text
```

3.3.1 Example

```
CODE [table] ON { * [table] } { 3 [table].[field] > 10 }
CODE [table] ON {! {* [table]} }
CODE [table] ON { 3 [table] } {! {* [table]} }
```

3.4 Combine Operation

The combine operation is used to combine two tables together in a SQL-join like fashion. It creates records that hold the columns of both tables excluding the column on which they are combined.

The combine operation requires a field, and a second field to combine with.

```
COMBINE field TO field
```

3.4.1 Example

```
COMBINE [firsttable].[id] TO [secondtable].[ID]
COMBINE [firsttable].[date] TO [secondtable].[date]
```

3.5 Compute Operation

The compute operation is used to perform a variety of calculations on a dataset. The compute operation can be used to calculate the average, the sum, the count, the standard deviation, the maximum and the minimum of a dataset.

The computation operation requires a table, a compute operator and a field to operate on.

```
COMPUTE table compute_operator field
```

3.5.1 Example

```
COMPUTE [table] AVG() [table].[numfield]
COMPUTE [table] COUNT() [table].[numfield]
COMPUTE [table] STDEV() [table].[numfield]
```

```
COMPUTE [table] MAX() [table].[numfield]
COMPUTE [table] MIN() [table].[numfield]
COMPUTE [table] SUM() [table].[numfield]
```

3.6 Connection Operation

The connection operation works similar to the combine operation with the main difference that where the combine operation actually combines the records, the connect operation only creates a new data table with all the records sorted on a specified column. The data entries from both tables will be mixed in the new table, sorted on the specified column.

The connection operation takes two arguments, the first argument is the field of the first table, the second argument is the field of the second table.

```
CONNECT field TO field
```

3.6.1 Example

```
CONNECT [firsttable].[ID] TO [secondtable].[ID]
CONNECT [firsttable].[date] TO [secondtable].[date]
```

3.7 Constraint Operation

The constraint operation is used to place a constraint on a field of a dataset. For example if you want only the records in a dataset that have a field called "field" above a value 10 you would use the constraint operation.

The constraint operation requires a field to place the constraint on, and a condition.

```
CONSTRAINT field condition
```

3.7.1 Example

```
CONSTRAINT [table].[field] > 10
CONSTRAINT [table].[field] == 35
```

3.8 For Each Chunk Operation

If the chunking operation is used to create separate chunks of the original dataset it can be useful to perform operations on each of the chunks individually. The for each chunk operation is used to perform any other operation on each chunk individually.

The for each chunk operation requires a table to find the chunks in, the chunk level, and a operation as specified in the other parts of this section. The chunk level must be specified because chunks can also have chunks, and those chunks can also have chunks. The chunking data structure is recursive and thus a level must be specified on which of the chunk levels to operate.

```
FOR EACH CHUNK table number operation
```

3.8.1 Example

```
FOR EACH CHUNK [table] 1 CONSTRAINT [table].[field] > 10
```

3.9 Lsa Operation

The Lag Sequential Analysis Operation outputs a table with one column containing the lag, and another column containing the frequency of that lag. Lag is measured in amount of records from the key to the target value. The first argument is the field to check values. Then the range of the lag analysis must be specified, e.g. -4 to 5. Finally the key value and target value are needed.

```
LSA field number number value value
```

3.9.1 Example

```
LSA [table].[measurement] -4 5 "creatinine" "bloodpressure"
```