LOAD DATA INTO QLIKVIEW

Summary

Foreword

This document was first the content of personal notes. It has become bigger and bigger: I just hope it is not too big. This document aims to summarize some questions we all had one day with the LOAD statement.

It contains some very basic of the LOAD (Chapter 1) and also some less basic (Chapters 2 and 3). I give many links to the community because some topics are widely covered by the community and I know that I did not go too much into details for each topic (this document has already 50 pages!!)

This doc is NOT complete: it will NOT replace the reference manual or any book guiding you from bottom to the sky of wisdom. But it should help many people learning the LOAD statement in QlikView.

Enjoy …And if you really enjoy, please leave a message. It will show me that the time spent to write this document was worth to do.

The version of this document is 1.1. Personal Edition QlikView 11.20 SR2 has been used.

1 Basic LOAD statements

This chapter covers basic load statements. If you worked already with QlikView, you will certainly not learn much in it.

You may load data from several data sources:

- SQL database through ODBC, OLE DB, specific API connectors
- Text files: CSV, linear
- Excel files: 2003 (XLS) or 2010 (XLSX)
- QVD files: specific files that are created and read only by QlikView
- QVW files: other applications we want the data again
- This working application: we want to reuse the data previously loaded
- Hard-coded data: data we enter directly in the script
- HTML, XML …

It is possible to mix them into QV: the users will be able to analyze data coming from Excel, CSV files and also from several SQL databases …

For most of these sources, it is possible to load:

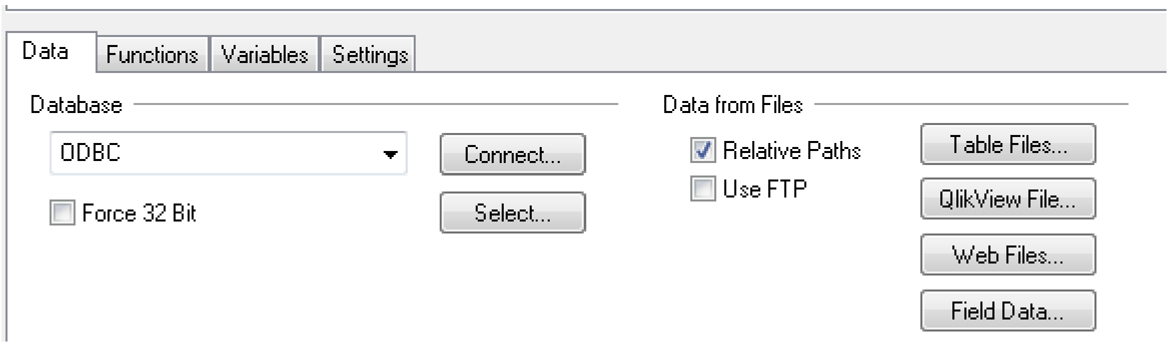
- Only some columns by naming them
- Only some rows with the WHERE statement

For each column, we may:

- Load the data as it is
- Load the data and rename it (AS 'NewName')
- Use functions to transform the data of the source

Before doing any script, it is a good habit to:

- Copy/paste other part of code
- Use the script generator to generate at least the beginning of the command:



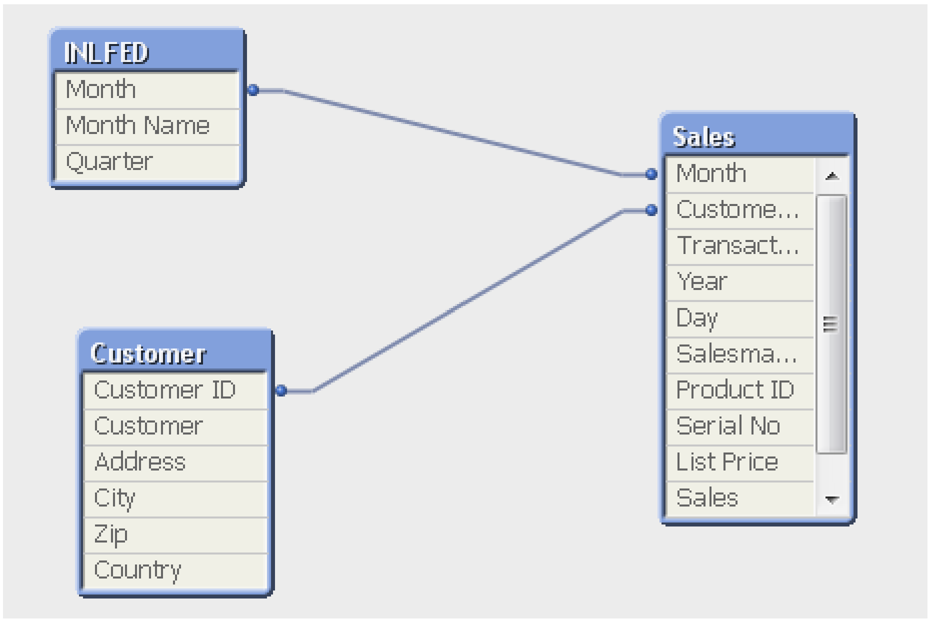
1.1 The sources

Most common syntax but that does not apply to every source:

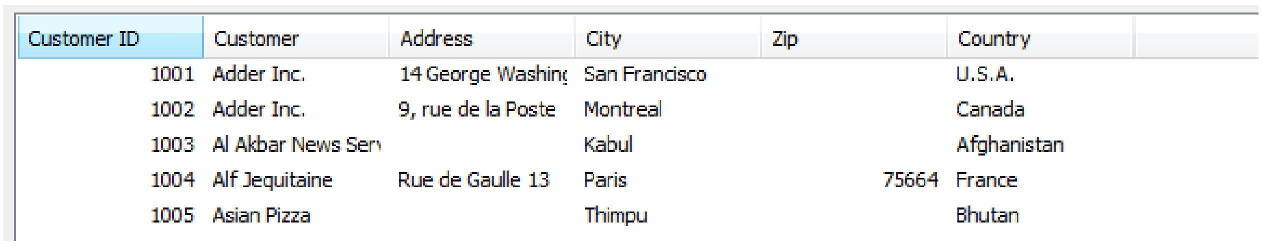
```
TableName:
LOAD [Distinct]
Fieldname1 [as newFieldName1] [, FieldNameN [as newFieldnameN] …]
FROM ASource
[WHERE condition]
[ORDER BY SortOrder];
```

You need to name just before the load statement the table you want to populate. The different tables you create and populate will form your data model.

After a load, you can easily check the data model: **File / Table Viewer** (Ctrl + T is a valid short key in the GUI)



See how it looks like. You may check the data by clicking on the table right click and choose **Preview**:



Customer ID	Customer	Address	City	Zip	Country
1001	Adder Inc.	14 George Washing	San Francisco		U.S.A.
1002	Adder Inc.	9, rue de la Poste	Montreal		Canada
1003	Al Akbar News Serv		Kabul		Afghanistan
1004	Alf Jequitaine	Rue de Gaulle 13	Paris	75664	France
1005	Asian Pizza		Thimpu		Bhutan

Only the first 1,000 rows will be displayed. It should be enough for a first quality check.

1.1.1 SQL

1) You need to open a connection to your database, something like:

```
CONNECT TO [Provider=OraOLEDB.Oracle.1;Persist Security Info=False;);
```

2) You need to do your LOAD command with just the field names. End your command with the semi-colon (;)
3) Write your SQL command

Note that Qlikteck recommends you to hide your connection statement if the script contains a password. Use either the hidden script **(File / Create Hidden Script)** or a script external file $(Include=filename).

Example:

```
MyTable:
LOAD if(isnum(xx), A, B) as Field1, yy*2 as Field2, Field3/Field4 as Field6, Field5;
SQL SELECT xx, yy, zz as Field3, Field4, Field5
FROM MytableName
WHERE Field5 > 0;
```

As you may see, the LOAD statement can rename and modify the SQL fields with QlikView functions. The SQL command creates such a temporary table with column names that can be reworked by the LOAD. As you see, you can rename the fields in SQL and/or in the Qv script.

The simplest commands if you load all the columns of Table1 without renaming them:

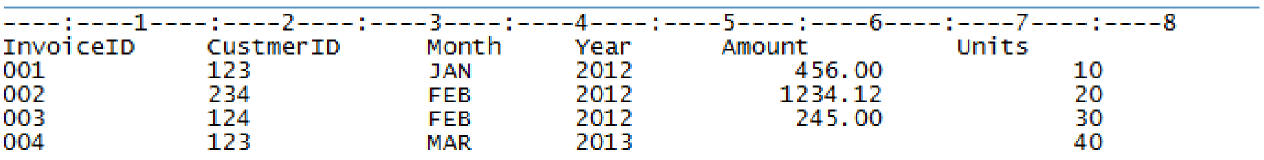
```
MyTable:
LOAD *;
SQL SELECT * FROM TABLE1;
```

When using a SQL database, the most part of the job of filtering, transforming the data is done in SQL. The most important part of time needed to load the data can be due to the SQL database.

You may use the **disconnect** command at the end of the loaded tables (this command is optional: QV disconnect the previous connection at the end of the script or at the next **connect** command)

1.1.2 Column TEXT files

Let's say that you need to load such a file:



```
----:----1----:----2----:----3----:----4----:----5----:----6----:----7----:----8
InvoiceID     CustmerID      Month     Year      Amount          Units
001           123            JAN       2012           456.00             10
002           234            FEB       2012          1234.12             20
003           124            FEB       2012           245.00             30
004           123            MAR       2013                              40
```

You may do:

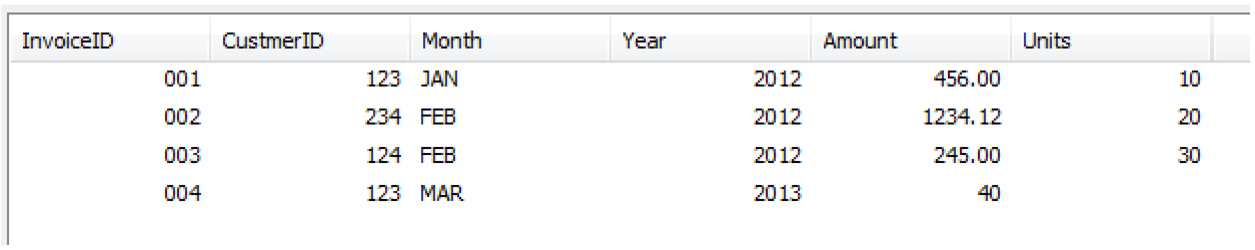
```
Table4:
LOAD *
FROM
[TXT_File.txt]
(txt, codepage is 1252, embedded labels, delimiter is spaces, msq, header is 1 lines);
```

`delimiter is spaces` : because of the blanks separating the fields

`header is 1 lines:` skip first line, because of the line beginning with `----:----1----:----2`

`embedded labels:` because the headings of the column is inside the file, and that line should be read by the script. If there is no line of label: `no labels.`

This code is OK until there is a complete blank field. Here the 40 units in the 4th line moves left to the Amount variable:



InvoiceID	CustmerID	Month	Year	Amount	Units
001	123	JAN	2012	456.00	10
002	234	FEB	2012	1234.12	20
003	124	FEB	2012	245.00	30
004	123	MAR	2013	40	

If you can get blank fields, you need to describe where to find the columns. The same file is read by the following command:

```
Table5:
LOAD @1:10 as InvoiceID,
     @15:25 as CustomerID,
     @30:39 as Month,
     @40:49 as year,
     @50:60 as Amount,
     @66:75 as units
FROM
[TXT_File.txt]
(ansi, fix, no labels, header is 2 lines);
```

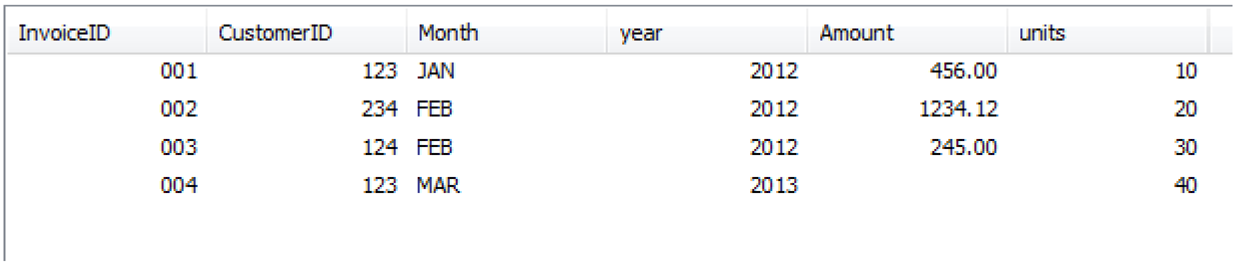
`@StartCol:EndCol` : where to find the information

`As NewFieldName`: this time, you need to rename the field names

`Header is 2 lines`: to not read the first 2 lines. If you have 1 line: `header is 1 line`. If you have no header (default), this become `header is 0`

`No labels:` there is no line of label for the columns to get read. If the first line to get read contains the names of the column: `embedded labels`

And we get the right result:

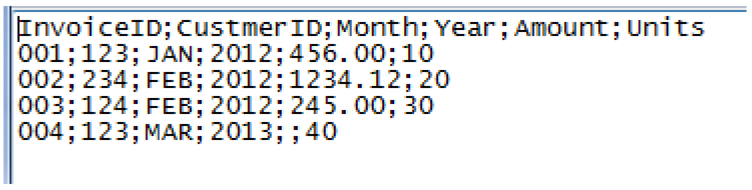


InvoiceID	CustomerID	Month	year	Amount	units
001	123	JAN	2012	456.00	10
002	234	FEB	2012	1234.12	20
003	124	FEB	2012	245.00	30
004	123	MAR	2013		40

1.1.3 CSV

The CSV file are also TEXT file but the fields are separated with a … separator that is most of the time a comma, a semi-colon, a tab

Let's use the same data as the one in previous section:



```
InvoiceID;CustmerID;Month;Year;Amount;Units
001;123;JAN;2012;456.00;10
002;234;FEB;2012;1234.12;20
003;124;FEB;2012;245.00;30
004;123;MAR;2013;;40
```

To read it:

```
Table6:
LOAD InvoiceID,
     CustmerID,
     Month,
     Year,
     Amount,
     Units
FROM
[CSV_File.csv]
(txt, codepage is 1252, embedded labels, delimiter is ';', msq);
```

We could have done more simply because we take all fields as they are, and the labels are in the file:

```
Table6:
LOAD *
FROM
[CSV_File.csv]
(txt, codepage is 1252, embedded labels, header is 0, delimiter is ';', msq);
```

In a production environment, it is a good habit 1) to name the columns we want to read: we avoid the star (*) therefore 2) to be in a certain way independent of the file : if a column is added, do we want to get it automatically in the model?

The delimiter is `'\t'` for tabulation. Remember that the delimiter was `spaces` for the TXT file that we read in the previous section.

If there is no column heading (`no labels`), use the generic @number to name the column of the file:

```
LOAD @1 as InvoiceID,
     @2 as CustmerID,
     @3 as Month,
     @4 as Year,
     @5 as Amount,
     @6 as Units
FROM
[CSV_File.csv]
(txt, codepage is 1252, no labels, header is 1 line, delimiter is ';', msq);
```

`header is 1 line:` skip the first line

`no labels`: we use the @n instead

`msq:` quotation mark

If you use the labels (`embedded labels)` but want to change some, you MUST use the labels of the file (@n is then not allowed):

```
LOAD InvoiceID,
     CustmerID,
     Month,
     Year,
     Amount as Euro_Sales,
     Units as Units_Sales
FROM
[CSV_File.csv]
(txt, codepage is 1252, embedded labels, delimiter is ';', msq);
```

You may change the order of the fields in the future table:

```
LOAD Month,
     Year,
     InvoiceID,
     CustmerID,
     Amount as Euro_Sales,
     Units as Units_Sales
FROM
[CSV_File.csv]
(txt, codepage is 1252, embedded labels, delimiter is ';', msq);
```

1.1.4 XLS files

Excel 2003:

The syntax is the same as the CSV file. You use @n to name generically the column when there is no label for the columns. You may rename only some columns; you may reorder the columns if you wish.

Only the parameters describing the file are changing:

```
Table7:
LOAD *
FROM
[XLS_File.xls]
(biff, embedded labels, table is [MySheet$]);
```

`biff`: XLS files
`ooxml`: XLSX files
`Table is [SheetName$]:` name of the sheet followed by the **$** sign for the XLS files only (`biff`). When you use the $ sign, you must enclose the text into square brackets [ ].

Excel 2010:

We use the letters (A, B, C …) instead of @n to name generically the column when there is no label for the columns. You may rename only some columns; you may reorder the columns if you wish.

There is no $ sign to name the sheet.

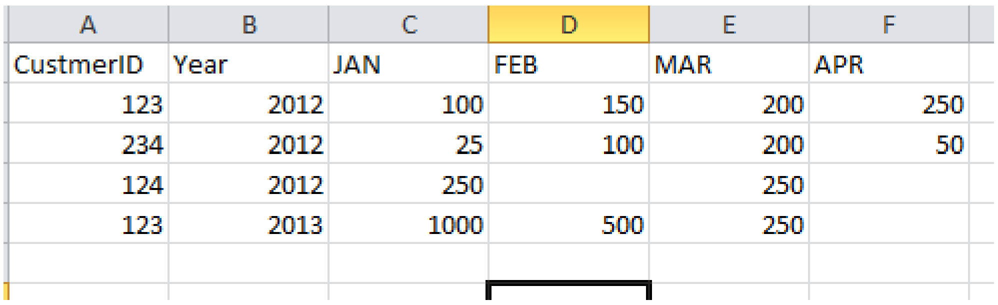
The type of file is `ooxml`.

```
Table8:
LOAD A as CustomerID,
     B as Date
FROM
[XLSX_File.xlsx]
(ooxml, embedded labels, table is MySheet);
```

	XLS	XLSX
Type	`biff`	`ooxml`
Sheet	Add the $ to the sheet name and therefore the brackets	Use brackets only if special characters
Generic column name	@1, @2, @3 …	A, B, C …

1.1.5 Cross table files

Here, instead of having the variables in columns, we have another dimension like periods. See example:



A	B	C	D	E	F
CustmerID	Year	JAN	FEB	MAR	APR
123	2012	100	150	200	250
234	2012	25	100	200	50
124	2012	250		250	
123	2013	1000	500	250	

Here above, we see that:

- We have two dimensions CustmerID and Year
- The Month dimension is in Across
- The data begins at column 3 because there are 2 dimensions
- That is XLS 2003

We must precede the LOAD statement with the **CrossTable** prefix:

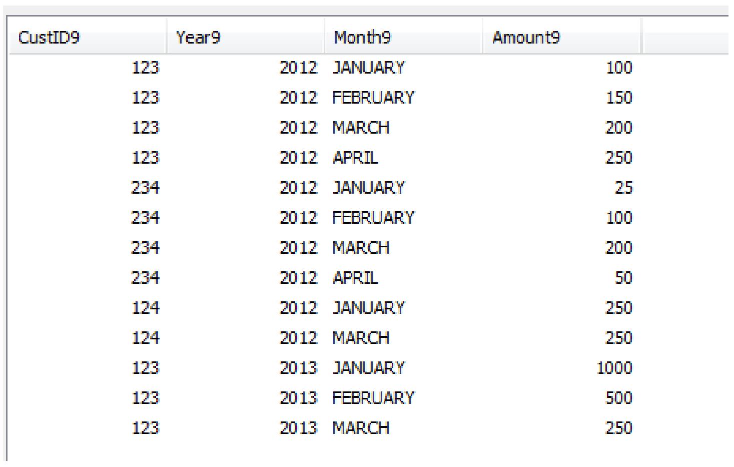
```
Syntax: CrossTable(Across Dimension, Variable Name, Number of dimensions preceeding
data)
```

```
Table9:
CrossTable(Month9, Amount9, 2)
LOAD CustmerID as CustID9,
     Year as Year9,
     JAN as JANUARY,
     FEB as FEBRUARY,
     MAR as MARCH,
     APR as APRIL
FROM
[Cross_File.xls]
(biff, embedded labels, table is [Cross_File$]);
```

⇨ Month9 and Year9 are the 2 dimensions, Amount9 is the measure

Please, notice that:

- We can name the dimension and the variable as we want, if there are special character like blank or $, we must enclose the name between brackets
- We can rename also the values of the dimension in ACROSS.



CustID9	Year9	Month9	Amount9
123	2012	JANUARY	100
123	2012	FEBRUARY	150
123	2012	MARCH	200
123	2012	APRIL	250
234	2012	JANUARY	25
234	2012	FEBRUARY	100
234	2012	MARCH	200
234	2012	APRIL	50
124	2012	JANUARY	250
124	2012	MARCH	250
123	2013	JANUARY	1000
123	2013	FEBRUARY	500
123	2013	MARCH	250

1.1.6 QVD files

QVD is a specific native format created and read only by QlikView (other BI software will not be able to read them in order to populate their own database). **These QVD files contain only ONE table (see the syntax to create them)**. We use these files mainly when:

- We need to read a lot of data quickly (this solution is very fast, it can be optimized)
- The data stored in the file was quite complicated to compute : instead of duplicating the code, we store the data that will reused completely or partially
- We do incremental reloads: part of the data comes from standard sources like data warehouses or XLS files, but historical data come from QVD files

To create a QVD file, we can export the entire table or only some fields, but all rows:

```
STORE Mytable INTO MyQVDFile.qvd;
STORE Field1 [as NewField1], Field2 [as NewField2] … FROM Mytable INTO MyQVDFile.qvd;
```

To read a QVD file:

```
LOAD * FROM MyQVDFile.qvd (qvd);
LOAD Field1, Field2 as NewField2 FROM MyQVDFile.qvd (qvd);
```

You may also, if you know the content of the QVD file, load only some columns by naming them or only some fields (WHERE clause studied later):

LOAD ID, Field1 as newField1 From MyQVDFile.qvd (qvd) WHERE ID > 10;

The load of a QVD can be either optimized or not-optimized. The reference manual talks about super fast mode or just fast mode (standard mode). By using the **WHERE** clause, a forced concatenation, a **Join**, the QVD load becomes un-optimized (standard). In other words, it would be faster to store the right rows into a QVD, and fetch them later on.

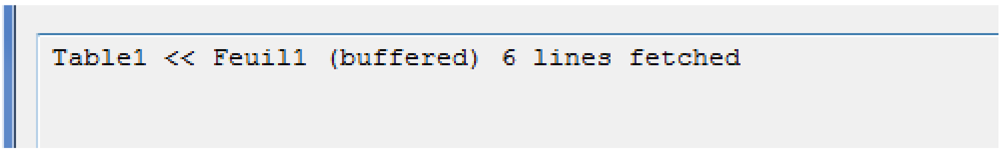
There is a trick explained by dathu.qv with the **Exists**() function. See http://community.qlikview.com/docs/DOC-5706

However the read of a QVD file is always faster than any other source.

These files can contain a lot of historical data as a data warehouse. You cannot protect your files (and hence your data) through a password: you have to implement the security settings through the Operating System.

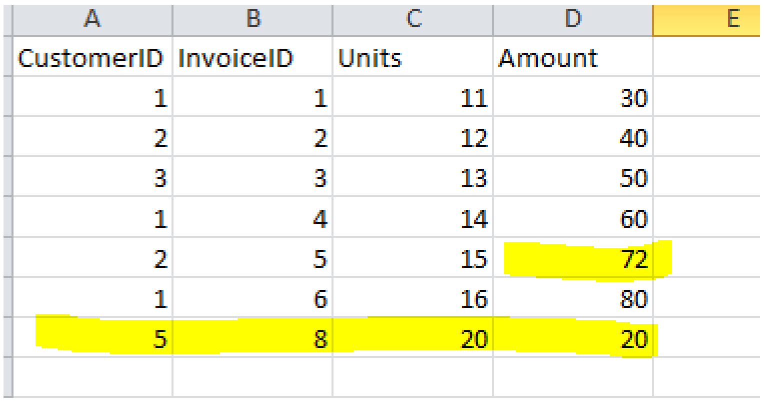
QVD files are also used by QlikView when we use the **BUFFER** keyword before the **LOAD** statement. QV creates and reads a QVD file created by it's own: it decides afterwards to read QVD again or the file instead.

```
Table1:
BUFFER
LOAD CustomerID,
     InvoiceID,
     Amount,
     Units,
     (CustomerID < 3 and InvoiceID < 3) as Flag
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil1);
```



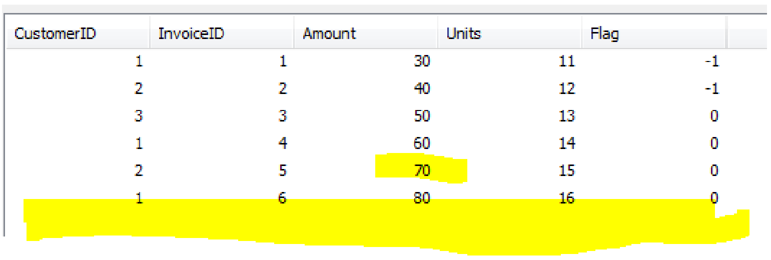
```
Table1 << Feuil1 (buffered) 6 lines fetched
```

This technique can be dangerous especially if the lines of the files may change. QVD temp file would contain the old data and be reread instead of those of the file. See below: 1 line has been changed, 1 line has been added:



A	B	C	D	E
CustomerID	InvoiceID	Units	Amount	
1	1	11	30	
2	2	12	40	
3	3	13	50	
1	4	14	60	
2	5	15	72	
1	6	16	80	
5	8	20	20	

But neither the line has been changed, nor the new line added:



CustomerID	InvoiceID	Amount	Units	Flag
1	1	30	11	-1
2	2	40	12	-1
3	3	50	13	0
1	4	60	14	0
2	5	70	15	0
1	6	80	16	0

With Excel or SQL databases, it seems that the new lines are not guessed and therefore not read at all. In case of a text file (for example CSV), it seems that QlikView guesses the number of the lines (compared to the ones of the QVD) and reads the last lines. Therefore, in any case, with that technique, the lines should NOT change.

When the file is modified, the QVD file must be deleted or the **BUFFER** keyword removed. The QVD path may be determined with the menu **Settings / User Preferences / Tab Location**. But the name is computed with a HashCode. It can be difficult to say which QVD replaces which specific file.

A freeware to load QVD files, see what is inside without importing them …: http://community.qlikview.com/docs/DOC-3401

There are many functions to determine the content of these QVD files. The file name is given as an argument. If the file does not exist, the function returns null.

QVDCreateTime('QVD File'): string, the creation date of the QVD file.

QVDNoOfFields('QVD File'): integer, the number of fields contained by the file.

QVDNoOfRecords('QVD File'): integer, the number of rows contained by the file.

QVDTableName('QVD File'): string, the table name used to populate the QVD file.

QVDFieldName('QVD File', Number): string, the field name at the given position. If the number is too high and therefore the field does not exist, the function returns null.

Example:

```
let vNoOfFields = QvdNoOfFields('C:\ Qlikview\Tutorial\essai.qvd');
```

See section 2.2 about variables.

1.1.7 QVW files

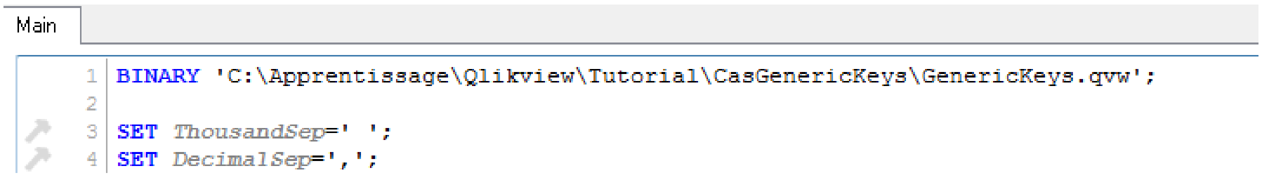
We have also the possibility to load all the data (and therefore the model) that is stored into another application (QVW). We call that a binary load because the used command is BINARY.

That way, you can share (but hide) a single script with several applications. Modifying that single script will have an impact on all others derived from it.

The BINARY statement must be written in the first line, before any SET or LOAD statements. You cannot even test the presence of the file (the **if** line would be written before the BINARY one). As a consequence, you can have only one **BINARY** load (you cannot have two lines at the first place!). The BINARY load is complete: you cannot load only few tables or only part of some tables. You will need to use other techniques to reduce the scope if you want to remove some data.

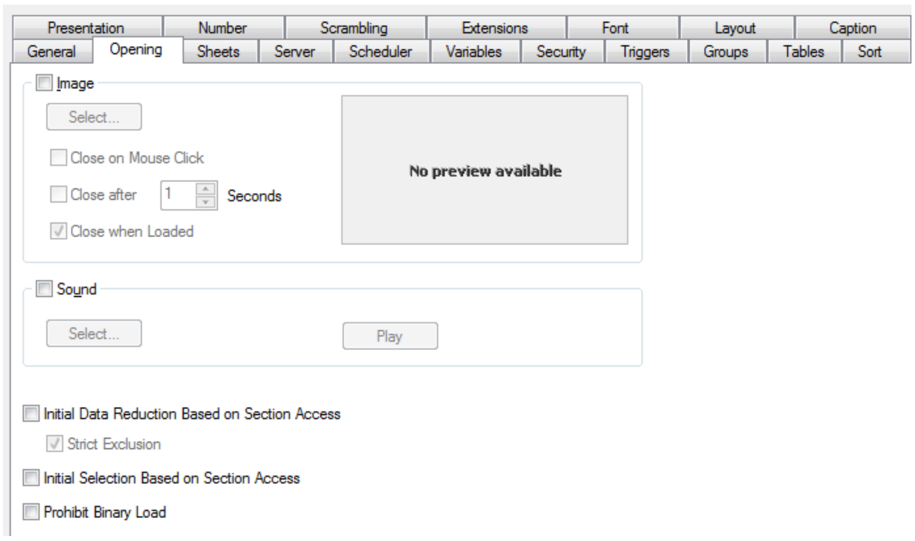
If the source QVW contains restriction access, the target will also get them according to the last person who entered the source QVW.

```
Syntax: BINARY 'Path\FileName.qvw';
```



```
BINARY 'C:\Apprentissage\Qlikview\Tutorial\CasGenericKeys\GenericKeys.qvw';

SET ThousandSep=' ';
SET DecimalSep=',';
```

Note that you need to allow this possibility: Menu **Settings / Document Properties /** tab **Opening** / Uncheck the field **Prohibit Binary Load**.


Presentation | Number | Scrambling | Extensions | Font | Layout | Caption
General | Opening | Sheets | Server | Scheduler | Variables | Security | Triggers | Groups | Tables | Sort

☐ Image
Select...
☐ Close on Mouse Click
☐ Close after 1 Seconds
☑ Close when Loaded
No preview available

☐ Sound
Select... Play

☐ Initial Data Reduction Based on Section Access
☑ Strict Exclusion
☐ Initial Selection Based on Section Access
☐ Prohibit Binary Load

1.1.8 An already loaded file / In-memory load

As we saw, a file (except qvw) is loaded into a single table. But a table can also be populated with other tables: QV call it the in-memory load because the data is already in the RAM (all the data of table is the RAM).

We use the in-memory load when we want to transfer partial or entire data from one table to another, reordered or not.

The syntax is very similar to the one reading a file: the **RESIDENT TableName** replaces the **FROM FileName**.

Syntax:

```
TableName:
LOAD [Distinct]
Fieldname1 [as newFieldName1] [, FieldNameN [as newFieldnameN] …]
RESIDENT Table1
[WHERE condition]
[ORDER BY SortOrder];
```

Please notice that you can:

- Transfer only some fields (**WHERE**) and some columns by naming them
- Rename the columns (**AS**)
- Add some fields also by using functions to create flags, counts …
- Can reorder the fields (**ORDER BY**): this clause is available only for the in-memory loads

⚠ **ORDER BY** is only available with the **RESIDENT** keyword. You cannot reorder rows when reading an external file (xls, csv, qvd …)

1.1.9 Hard-coded data

We may also enter some data directly in the script. This possibility is useful:

- For test purposes
- When data will not change

The basic syntax is:

```
LOAD * INLINE [
Field1, Field2, Field3
Value11, Value21, Value31
Value12, Value22, Value32
…
];
```

Please notice that:

- You may enter as many fields as you want
- The field names of the table or the star (*), the functions are before INLINE and the brackets ([]): the names are the one of the target table
- The field names are also just after the open bracket ([) first line: the names are the one of the data source
- The fields and values are separated with a comma (,) but the end of line has no comma

You may:

- Replace the [] with double quotes (").
- Use a function to format the fields before the INLINE statement
- Rename the input fields
- Use several times a single input

Example:

```
Temp:
Load
Field1,
Date(Date#(Field2, 'DD-MM-YYYY')) as Field2,
Field3,,
Field4
Inline [
Field1,Field2,Field3,Field4
A, 01-01-2013, 200, 100
B, 01-01-2013, 100,300
C, 01-01-2013, 300,400
A, 02-02-2013, 0,500
];
```

As you see with this example, the field names appear twice. We use a function to format the date but the other 3 fields are not modified.

Please notice that:

- The names of the columns of the table (Col1 …) and those of the source (NomCol1 …) are not necessarily the same: you can rename all or some of them
- The number of columns may also be different: an input field may be used several times to compute several columns

```
Table3B:
LOAD NomCol1 as Col1, NomCol2 as Col2, NomCol3 as Col3, NomCol2 &'-' & NomCol3 as Col4
INLINE
[NomCol1, NomCol2, NomCol3
L1C1,L1C2,L1C3
L2C1,L2C2,L2C3
];
```

Well most people (including me) would write:

```
Table3B:
LOAD *, Col2 &'-' & Col3 as Col4 INLINE
[Col1, Col2, Col3
L1C1,L1C2,L1C3
L2C1,L2C2,L2C3
];
```

1.2 WHERE clause

This is like SQL: if you want to load only some fields into the table, you may include a WHERE clause.

Syntax:

```
LOAD … FROM/RESIDENT …
WHERE boolean_condition
GROUP BY … ORDER BY ….
```

The boolean expression may be composed of one or several boolean expressions. The field names used in the **WHERE** clause are the one of the source (therefore, before renaming).

```
Temp_Table1:
  LOAD InvoiceID as ID2,
       CustomerID as Number2,
       Month,
       Year,
       Amount as Montant
  FROM 'XLSX_File.xlsx'
  (ooxml, embedded labels, table is MySheet)
  WHERE InvoiceID > 1 AND NOT isnull(Amount)
  ;
```

As you can see, InvoiceID and Amount are the names of the source. The **WHERE** clause does not know the final names of the table. That will be the case also if you use generic column names:

For column text files:

```
Temp1:
LOAD rowno() as LineNr,
     @36:80 as LightFileName
FROM
[aaa[1]].txt]
(ansi, fix, no labels, header is 2 lines)
WHERE @23:25 <> 'DIR'
```

For Excel 2010 files, the generic column names are A, B …:

```
table1:
LOAD A as ID,
     B as Sales
FROM
Number.xlsx
(ooxml, no labels, header is 1 line, table is Feuil1)
```

```
WHERE B > 20
;
```

For Excel 2003 files, the generic columns are @1, @2 …:

```
table1:
LOAD @1 as InvoiceID,
     @2 as CustmerID,
     @3 as Month,
     @4 as Year,
     @5 as Amount,
     @6 as Units
FROM
XLS_File.xls
(biff, no labels, header is 1 lines, table is [MySheet$])
WHERE @4 = 2013 AND @3 <> 'DEC';
```

Take care: QV is case sensitive. Amount is not the same as AMOUnt and AMOUNT.

1.3 GROUP BY statement

This is like SQL: if you want to load aggregate some data into the table, you may use the **GROUP BY** statement. It implies that you use some aggregation functions like **sum**(), **count**(), **max**(), **min**() for the data fields you load.

The fields (or dimensions) after the GROUP BY clause MUST be present in the LOAD part and will belong to the table.

```
LOAD …, Dimension1, Dimension2, … FROM …
GROUP BY Dimension1, Dimension2 …
ORDER BY ….
```

Imagine we want to get the data already loaded in Temp_Table1 and get them summed by year and Customer only:

```
Sum_Table2:
LOAD Number2 as CustomerID,
     Year,
     sum(Montant) as MontantByYear
RESIDENT Temp_Table1
GROUP BY Number2, Year;
```

As the **WHERE** clause, the fields used in the **GROUP BY** clause are the names of the source (see Number2 field).

You can use many functions with the GROUP BY clause that you know from SQL or not:

- **Sum**: sum
- **Avg**: average
- **Count**: number of times appearing
- **Max**: maximum of a range
- **Min**: minimum of a range
- **FirstSortedValue**: first value found among several depending on a sort order given in parameter
- **FirstValue, LastValue**: first or last value in load order of expression over a number of records as defined by a **group by** clause
- Statistical funtions like **chi2test_p, TTest_t, TTestw_t** …
- …

1.4 ORDER BY statement

This is like SQL: if you want to sort the fields into the table, you may use the **ORDER BY** statement.

This **ORDER BY** clause:

- Can only be used with in-memory load: it is NOT possible to use it when reading directly a file.
- Is used when the sort of the table is important: you need to use specific functions like **previous**() that are row dependant.

```
Sum_Table2:
LOAD Number2 as CustomerID,
     Year,
     sum(Montant) as MontantByYear,
     sum(Units) as UnitsByYear
RESIDENT Temp_Table1
GROUP BY Number2, Year
ORDER BY Number2, Year
;
```

As in WHERE and GROUP BY clauses, you use the name of the source (here Number2) before renaming.

2 Intermediate loads

2.1 Delete fields and tables

We need sometimes delete tables we have created during the script. The ORDER statement may be used only if the data already is in QlikView: resident load (see section 1.4). You will need to load into a temp table and drop it when it is not needed anymore.

DROP TABLE Name1[, Name2 [,Name3 …]];

DROP TABLES Name1[, Name2 [,Name3 …]];

The command DROP lets you also delete one or several fields:

DROP FIELD name1[, name2 [, name3 ….]] ;

DROP FIELDS name1[, name2 [, name3 ….]] ;

As you may notice, the keyword **FIELD** or **TABLE** can set in plural (**FIELDS**, **TABLES**) even if you have to delete one single table or field.

2.2 Local variables

The local variable may replace some parts of the LOAD syntax. So you can write more generic code, but sometimes less clear.

A variable can replace
- `Paths (to easily move from development to production environment)`
- `Table to populate`
- `File or Table used as source (Resident load)`
- `Sheet`
- `Field to rename`
- `WHERE clause`

Because this code is more generic, you can introduce it into:

- A sub
- A loop

A local variable may be set with two commands: **SET** or **LET**. **SET** will store the literal content (the formula), **LET** will store the interpreted content (perhaps after the formula has been interpreted). Many times, when you want to set a value, there is no difference between the 2 because that is a fixed value, not a formula.

The local variables will be seen by the application at the end of the script. If you want to suppress them, you need to set them to null before the end.

```
set vAddition1= 1+2;
let vAddition2= 1+2;
```
vAddition2 will contain 3 (the interpreted content) while vAddition1 will contain 1+2 (the literal one)

```
set vToday1= Today();
let vToday2= Today();
```

vToday2 contains the date while vToday1 contains the formula that returns the date of the day. If the two variables are displayed 1 day after the script, they will not contain the same date. vToday2 stores the date of the script execution while vToday1 contains the formula that returns the date when calling it.

Take care that the local variables that you define and use in the script will still exist at the end of the script. You can therefore use them in the GUI objects. If you want to drop them, you just need to reset them to null before the end of the script.

```
let vToday2 = null();
```

To use the content of the variable, we use the $ sign and the parenthesis: $(*VariableName*). If the "future" content should be included into quotes, for any literal string, you should also set the $(xxx) between quotes. See the file name to get read:

```
let vPath = 'C:\Qlikview\Tutorial\';
Dim:
LOAD ClientID,
     ZIP,
     City,
     Country
FROM
'$(vPath)102714.xlsx'
(ooxml, embedded labels, table is Dim);
let vPath = Null() ;
```

As usual, the code is interpreted by QlikView, rewritten and executed. It means that you can use it into SQL commands: the code will be "rewritten" with the content of the variable before being sent to the SQL database.

```
set vFile= 'XLSX_File.xlsx';
set vSheet = 'MySheet';
set vMonth= 'Month';
set vYear = 'Year';
set vYear2 = 'TotalYear';
set vTest= 'Units > 20 AND InvoiceID > 3';
set vTable ='Temp_Table1';
set vSum1 ='Amount';
set vSum2 ='sum(Units)';

$(vTable):
  LOAD InvoiceID as ID2,
       CustomerID as Number2,
       $(vMonth),
       $(vYear),
       $(vSum1) as Montant,
       Units
  FROM $(vFile)
  (ooxml, embedded labels, table is $(vSheet))
  WHERE $(vTest)  ;

Sum_Table2:
LOAD Number2 as CustomerID,
     $(vYear) as $(vYear2),
     sum($(vSum1)) as MontantByYear,
     $(vSum2) as UnitsByYear
RESIDENT $(vTable)
GROUP BY Number2, $(vYear)
ORDER BY $(vYear), Number2;
```

As you notice, QlikView rewrites the command to get executed when it meets a $(*variable*). The content of the variable MUST therefore respect the future syntax of what is expected by QlikView.

We could put the parameters (ooxml …) into a variable and use it:

We could imagine also write the whole command into a variable that will be executed later:

```
let vCommand = 'LOAD InvoiceID as ID2,  CustomerID as Number2, $(vMonth), $(vYear),
Amount as Montant, Units FROM  $(vFile) $(vParam) WHERE Units < 40 ;' ;
$(vCommand);
```

To read the files under a given directory, for example:

```
for each file in FileList('*.xlsx')
  Temp_Table1:
  load A as ID,
       B as Number
  FROM $(file)
  (ooxml, no labels, table is Feuil1);
next
```

We can get this result also without variables. QlikView will iterate through an implicit loop:

```
  Temp_Table1:
  load A as ID,
       B as Number
  FROM ‘*.xlsx’
  (ooxml, no labels, table is Feuil1);
```

If we want to read only some particular files:

```
for each file in ‘A.xls’, ‘B.xls’, ‘C.xls’
```

See section 3.5 if you want to populate a local variable with the content of a table.

2.3 Populate a single table with several sources

QlikView is an associative BI tool. **The links (or associations) between the tables are done automatically by the names of the fields**. If two tables share a field with the same name, they will be linked through the common field. You cannot unlink them except if you rename one of the fields so that the two tables do not share any field.

Because **the link (or association) is automatic**, there is no way with QlikView to link several tables by using specific fields with different names. The values in the 2 fields must also be identical to be associated!

If these two tables share several fields, QlikView will link the two tables through a key that is called synthetic. This key is like a new generated field “summarizing” the different fields: the tables will be linked though this synthetic key. Because the key may be harmful to the response time, QlikTech encourage you to create your own key. We will see that point later.

2.3.1 Concatenete / Noconcatenete

When you load data from any source, this data will be stored into a table (QV calls it logical table). This table has one or several columns.

If you intend to load another source with exactly the same column names, all the data of the second source will be appended to the first table even though you name a second table. So you will get one table populated with the two sources. QlikView has performed an automatic concatenation.

If you do not want to concatenate, you have to explicitly:

- Name another table (if no name, the table will be named as the input sheet)
- Use the NOCONCATENATE keyword before the LOAD

```
Table2:
NoConcatenate
LOAD InvoiceID,
     CustomerID,
     Month,
     Year,
     Amount,
     Units
FROM
XLSX_File.xlsx
(ooxml, embedded labels, table is MySheet2);
```

So but what if you load only some common fields? The tables are not concatenated: the loads are populating different tables.

⚠ So remember that:

- if all the fields are common, the CONCATENATE is the default that you can overwrite with NOCONCATENETE.
- If, only some fields are in common, the NOCONCATENATE is the default that you can overwrite with CONTACATENETE (the missing columns will be populated with Null values).

2.3.2 Outer / Inner / Left / Right JOIN

With **Concatenete** and **NoConcatenete**, we add rows to the previous table or to a new table. The lines are added at the end of the table.

Sometimes, we have info coming from different sources, with different columns names, and we would like to link them. For example:

File1: CustomerID, InvoiceID, Units, Amount

File2: CustomerID, InvoiceID, DateDelivery

TargetTable: CustomerID, InvoiceID, Units, Amount, DateDelivery

We will do the link with the **JOIN** statement. QlikView will perform the link with the common fields. But if one field (or a key in several fields) is not in a file, what happens: should it be added or not? Do the previous rows remain in the table or are they removed?

⚠ QlikView joins tables, not fields.

We will load one file after the other. The table already loaded in QV is considered to be at the left place, the file to get loaded is considered to be at the right place.

There are 4 types of **JOIN**:

- **INNER JOIN** (default): the common keys must belong to the table (LEFT) and the file (RIGHT). It is an intersection between the two sources.
- **OUTER JOIN**: the common keys may belong to the table (LEFT) or the file (RIGHT). It is an union between the two sources.
- **LEFT JOIN**: the common keys may belong to the table (LEFT). Only the rows belonging to the table will be kept.
- **RIGHT JOIN**: the common keys may belong to the file (RIGHT). Only the rows belonging to the file will be kept.

Imagine the two sources:

File1:

CustomerID	InvoiceID	Units	Amount
1	1	10	100
2	2	20	150
3	3	30	200
1	4	10	150
2	5	30	100
1	6	30	100

File2:

InvoiceID	DateDelivery
1	01/07/2013
2	02/07/2013
3	03/07/2013
4	10/07/2013
7	20/07/2013

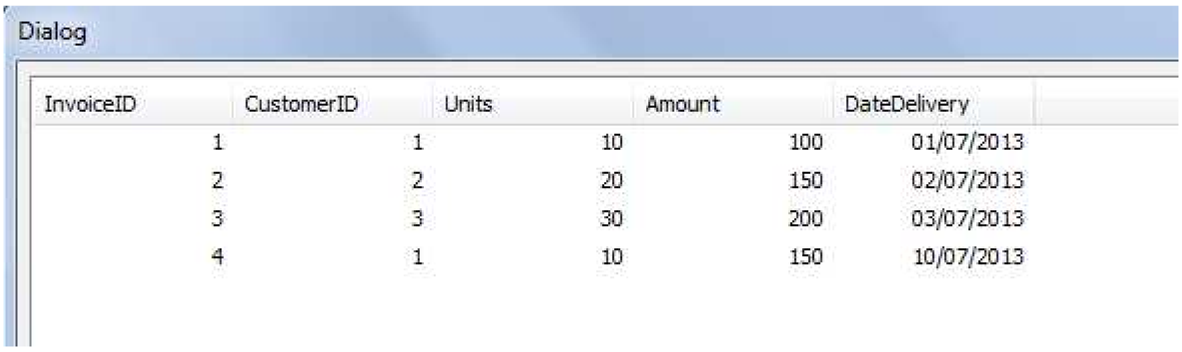
As you may notice, the InvoiceID 1, 2, 3 and 4 belong to the two sources. The InvoiceID 5 and 6 belong to the Source1 only, while the Invoice7 belongs to the Source2 only.

First of all, we load the first file in a table.

```
Targeted_Table:
LOAD CustomerID,
     InvoiceID,
     Units,
     Amount
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil1);
```

Case No1: we only want the InvoiceID present in both files. It must be an **INNER JOIN**.

```
INNER JOIN (Targeted_Table)
LOAD InvoiceID,
     DateDelivery
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil2);
```

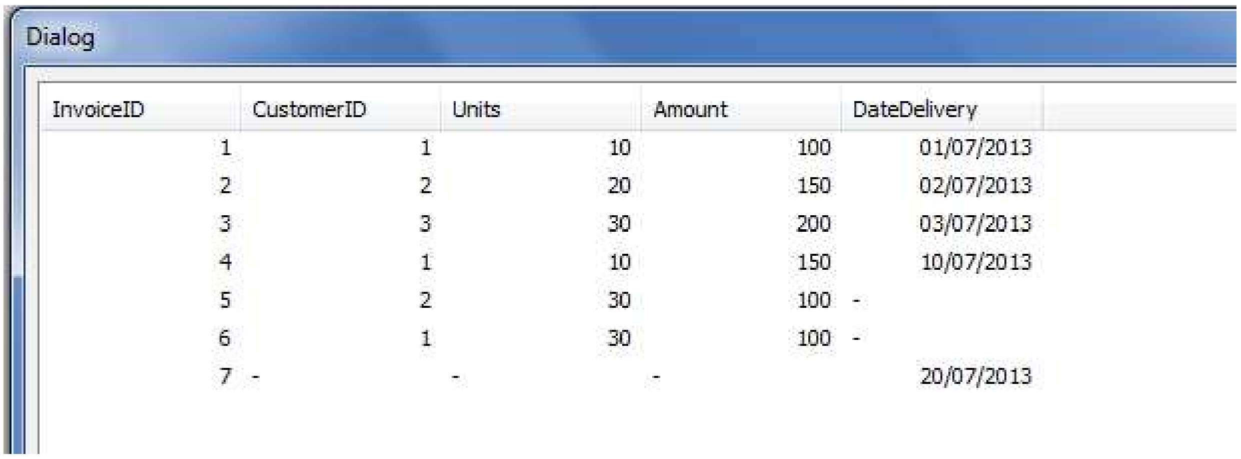

Dialog

InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013

The DateDelivery field has been added. Only the common InvoiceID are in the joined table. The Invoices 5 and 6 belonging only to File1, and Invoice 7 belonging only to File2 have disappeared.

Case No2: we want all InvoiceID, present in one of the files, even incomplete. It will be an **OUTER JOIN**.

```
OUTER JOIN (Targeted_Table)
LOAD InvoiceID,
     DateDelivery
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil2);
```

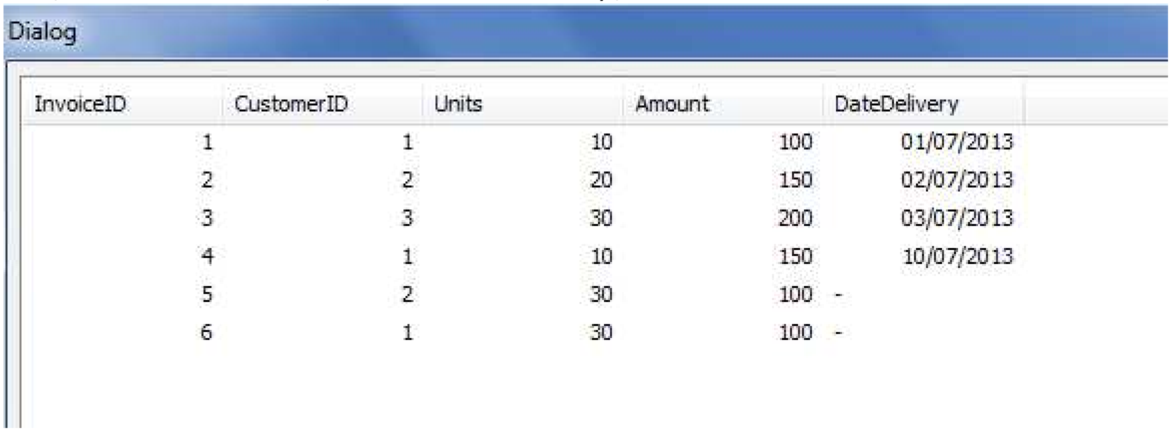


InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013
5	2	30	100	-
6	1	30	100	-
7	-	-	-	20/07/2013

The DateDelivery field has been added. All Invoices are in the table even incomplete (because belonging to only one source). The missing fields are NULL.

Case No3: We want only the InvoiceID that are in the Source1. This source has already been loaded: it is considered as **LEFT**. It will be a **LEFT JOIN**.

```
LEFT JOIN (Targeted_Table)
LOAD InvoiceID,
     DateDelivery
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil2);
```

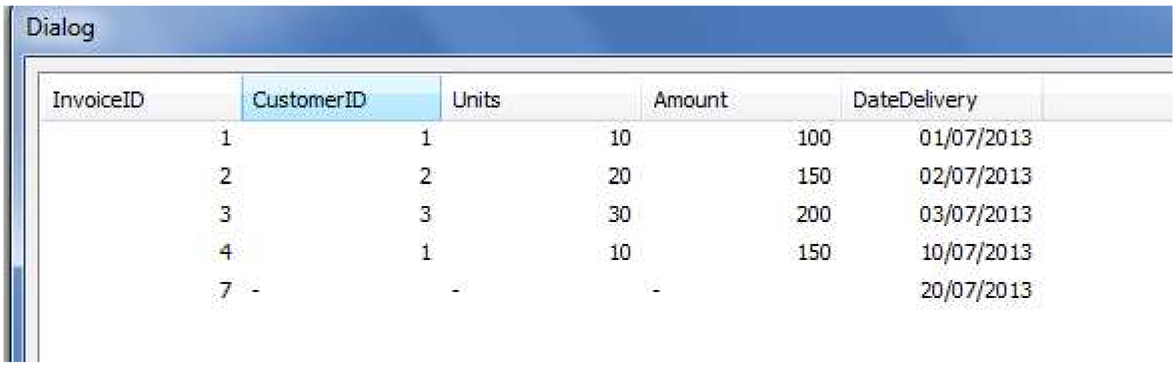


InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013
5	2	30	100	-
6	1	30	100	-

The missing fields are NULL. The InvoiceID 7 belonging only to the Source2 has disappeared. The invoices 5 and 6 are not present in the source 2 file: the DateDelivery field is then NULL.

Case No4: We want only the InvoiceID that are in the Source2. Source1 is the source has already been loaded: it is considered as LEFT. The File that has not been loaded is considered as RIGHT. It will be a **RIGHT JOIN**.

```
RIGHT JOIN (Targeted_Table)
LOAD InvoiceID,
     DateDelivery
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil2);
```



InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013
7	-	-	-	20/07/2013

The missing fields are NULL. The InvoiceID 5 and 6 belonging only to the Source1 have disappeared.

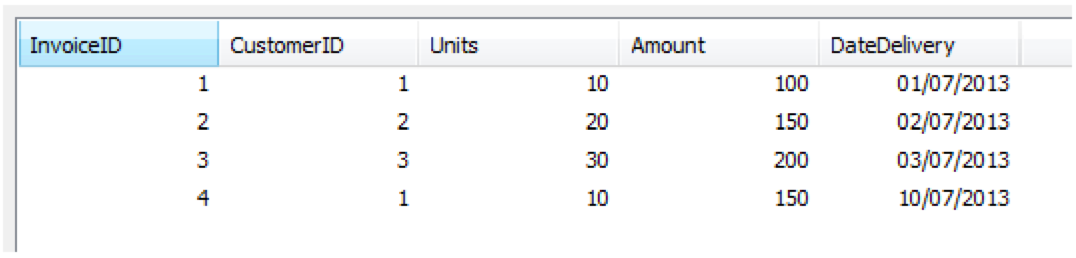
If you have several fields in common, the common key is guessed on the several common fields:

Imagine the File2 is the following one:

CustomerID	InvoiceID	DateDelivery
1	1	01/07/2013
2	2	02/07/2013
3	3	03/07/2013
1	4	10/07/2013
2	1	20/07/2013

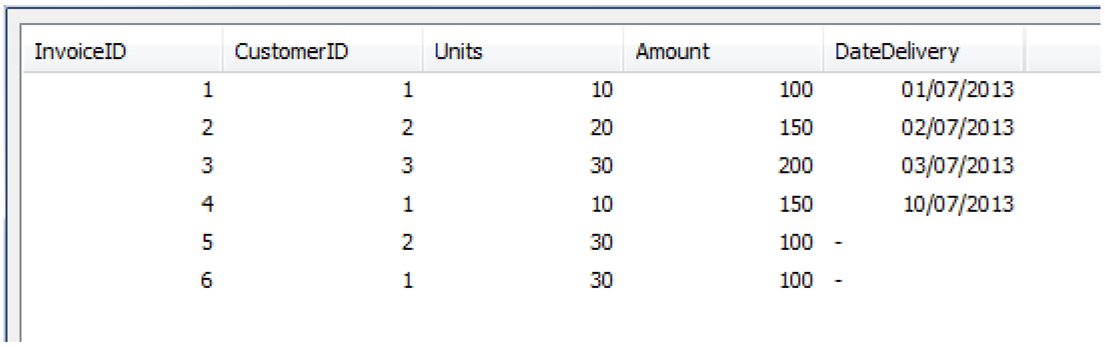
The first 4 lines can be joined because the File2 shares the same CustomerID and InvoiceID. The 5th key (CustomerID=2 and InvoiceID=1) does not exist in File1 even if the CustomerID 2 exists, and InvoiceID 1 also exists:

If we perform an **INNER JOIN**:



InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013

If we a perform **LEFT JOIN**:



InvoiceID	CustomerID	Units	Amount	DateDelivery
1	1	10	100	01/07/2013
2	2	20	150	02/07/2013
3	3	30	200	03/07/2013
4	1	10	150	10/07/2013
5	2	30	100	-
6	1	30	100	-

If you have no common fields, it is still possible to perform a **JOIN**. However, the number of lines resulting from this **JOIN** will be the multiplication of the number of lines of both tables. If other words, this operation will be RAM consuming if the two tables are quite big..

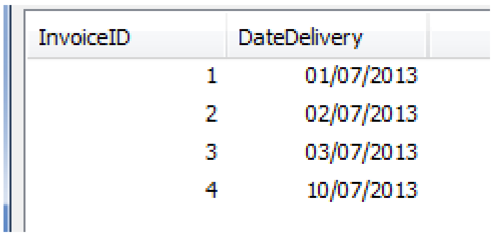
Further resources on **JOIN** and **Lookups**: see the excellent post (as usual) written by HIC: http://community.qlikview.com/docs/DOC-3412

2.4 KEEP

We will load the data into a separate table. But we will use the data contained in another table to do the selection.

I reuse the same 2 first files as the 2.2 section. Because we load data into a separate table, we need to name it first. Because we do the **KEEP** using another table, we need to name it also.

```
Targeted_Table2:
INNER KEEP (Targeted_Table)
LOAD InvoiceID,
     DateDelivery
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil2);
```



InvoiceID	DateDelivery
1	01/07/2013
2	02/07/2013
3	03/07/2013
4	10/07/2013

There are 2 types of KEEP: **INNER**, **LEFT**:

- **INNER KEEP**: the fields used to the selection belongs to the 2 sources
- **LEFT KEEP**: the fields used to the selection belongs to the LEFT source, it means the table

In fact, there is a 3d possibility. The **RIGHT KEEP** where the lines of the being-read file are kept! I must admit that I do not understand well the concept of that KEEP.

2.5 The synthetic key

This key appears when several fields of one table are linked to the same several fields of another table.

The literature says we MUST remove them. But I must admit that I am not so sure. See two posts by John Witherspoon and Henric Cronström (HIC):

http://community.qlikview.com/message/10279#10279

http://community.qlikview.com/blogs/qlikviewdesignblog/2013/04/16/synthetic-keys

To remove these keys, you can:

- Merge the different fields into a single one (Field1 & Field2 & Field3 as NewField), for each table
- Use the **AutoNumber** () functions so QlikView creates a number associated to each couple (or n-field), see the section 3.6.2.

2.6 Incremental load

This type of load is designed to save time in case of huge loads by splitting data between QVD and the normal source files. Instead of loading and computing millions of rows at each update from the normal source, you would prefer perhaps just load the new rows or the ones that have been updated.

Your application would keep old data that has not changed (because you know that it cannot change), and you would load only some data (the new ones because you know they are new). You must determine what is new, what cannot be changed. Most of the time, historical data after a given moment will NOT change anymore while the data of the last few days may change.

The steps for an incremental load are the following:

- Get data from the place where the not-changing data is stored: it can be a QVD file. Get the end date of the data contained in this file
- Elaborate the test that will be applied to retrieve data from the big file sources, something like WHERE date_big_source > Date_found_in QVD_file - NrDaysOfXXX
- Update, if needed the QVD file

If the incremental load is a good load in terms of rapidity, you also must write a script to get a complete LOAD (if the QVD does not exist for example) or at least a less partial load.

See a complete program: http://community.qlikview.com/message/12730#12730

The main advantage of that type of load is the rapidity of the script (remember that the QVD load may be optimized) compared to a SQL source that may take “some” time to respond.

You may also imagine storing QVD files that will be shared across several applications. In that case, you will certainly have an application that will read the “millions” of rows of the source, compute them and store them the way the different applications will use them.

2.7 Partial Reload

A partial load is a load that will complete a previous normal load. That is not the same as an incremental one.

First you do a complete load. Second you (often the user) do a partial load in order to get the last data that have been updated since the complete load. This technique avoids performing a complete load several times especially if very few data are updated several times, and the complete load takes too much time to get done by a user.

⚠ As usual, we should take care of this possibility. When we do a complete load, QlikView deletes all previous data and reload the data according to the script. In case of partial reload, QlikView does not delete the data but it will add the new data according to the script: therefore, **you must be sure that you do not load any data that had already been loaded during the complete load**.

⚠ If you intend to restrict access data, you cannot use any Partial Reloads.

Some keywords or functions will help us distinguish a partial to a complete load.

Only: a qualifier denoting that the command is to be done only during partial loads. The command will therefore be disregarded during normal (complete) loads.

IsPartialReload(): boolean function returning TRUE (-1) if it is a partial reload.

ONLY LOAD …. : the data will be read only in case of partial reload, disregarded during complete load

REPLACE ONLY LOAD …. The replace keyword forces QlikView to drop the table before reading the file. In the case of partial reload, QV drops the table and load the data. In case of complete load, QV disregards the whole command (no table dropped, no data read)

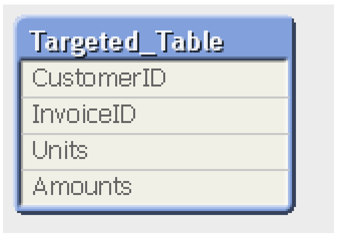
Further resources in the community: http://community.qlikview.com/docs/DOC-5312

2.8 The Preload technique

This technique consists of having several LOAD command in a row to populate one table. If you have a field populated with very complicated and nested functions, and this field should be used again to test the values of several fields, you certainly would prefer give an alias to the complicated and nested functions and reuse this alias to avoid to rewrite again and again all the nested stuff.

Take care: **the preload command is AFTER the load command that populates the table**.

```
Targeted_Table:
LOAD CustomerID,
     InvoiceID,
     if(Flag, Units*10, Units) as Units,
     if(Flag, Amount*2, Amount) as Amounts
;
LOAD CustomerID,
     InvoiceID,
     InvoiceID as NotUsedID,
     Amount,
     Units,
    (CustomerID < 3 and InvoiceID < 3) as Flag
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil1);
```



⚠ As you can see, **the PRELOAD is BELOW the real LOAD**. The table is populated with the Amounts field done by the first LOAD. The first LOAD has no explicit source (no FROM, no RESIDENT): the implicit LOAD is the one following.

In the PRELOAD, you can:

- create flags
- read some fields and rename them
- have complex functions that will be named as a field, and be reused in the LOAD: this field may (or not) belong to the final table (see the flag that is not a field of the Targeted_Table)

The LOAD takes the names of the PRELOAD “temp” table and can reuse these fields (the flag), rename them also (Amount into Amounts for example) and of course load only some of them (Field NotUsedID).

3 Others loads

3.1 The look-up tables

When reading the source, we may want to replace a key field by a more appropriate field: this replacement cannot be computed by a function. Each key value has its own replacement value that we have, most of the time, in another file.

To avoid many **JOIN** and difficult code, it is easy to:

- Populate a lookup table
- Use it when populating the more important table

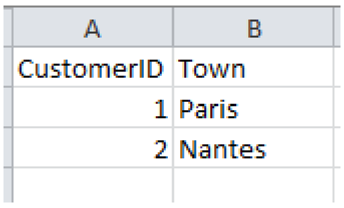
These look-up tables have at least 2 columns:

- One key
- One field to replace the key: a description, a group, a unit per key like a cost ….

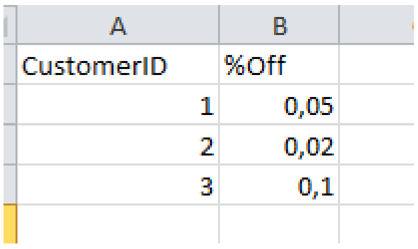
If we want to use **ApplyMap**() function that is very efficient, the table should contain only 2 columns. In all cases, each key value has only ONE replacement value (it cannot vary on a 3d field like time).

3.1.1 Change the content of a field

We want to add information about Customers: the Town and % of rebate they should get to compute the total amount. As you may see, the Town is not populated for each Customer.



A	B
CustomerID	Town
1	Paris
2	Nantes



A	B
CustomerID	%Off
1	0,05
2	0,02
3	0,1

ApplyMap()

```
Syntax: ApplyMap('TableOfLookup', KeyField [, Default if key is not found])
```

First we load a 2-field table of lookup with the **MAPPING** prefix:

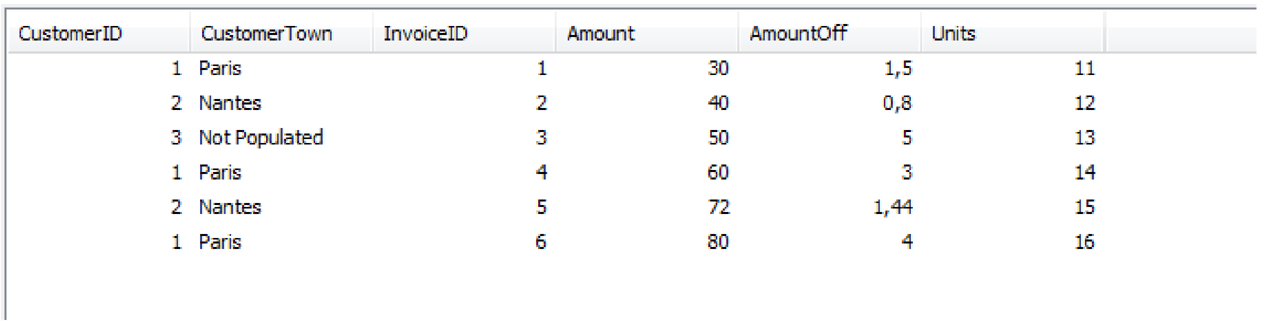
```
MAP_Table:
MAPPING LOAD
     CustomerID,
     %Off
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil4);
```

```
MAP_Table_Town:
MAPPING LOAD
     CustomerID,
     Town
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil5);
```

The **MAPPING** tables are temporary and can be used in the **ApplyMap**() function. No need to drop them at the end of the script. They do not belong to the model.

Secondly we use the tables to modify the content of the field with the **ApplyMap**() function:

```
Table1:
LOAD CustomerID,
     ApplyMap('MAP_Table_Town', CustomerID, 'Not Populated') as CustomerTown,
     InvoiceID,
     Amount,
     Amount * ApplyMap('MAP_Table_Off', CustomerID, 0) as AmountOff,
     Units
FROM
Customer.xlsx
(ooxml, embedded labels, table is Feuil1);
```



CustomerID	CustomerTown	InvoiceID	Amount	AmountOff	Units
1	Paris	1	30	1,5	11
2	Nantes	2	40	0,8	12
3	Not Populated	3	50	5	13
1	Paris	4	60	3	14
2	Nantes	5	72	1,44	15
1	Paris	6	80	4	16

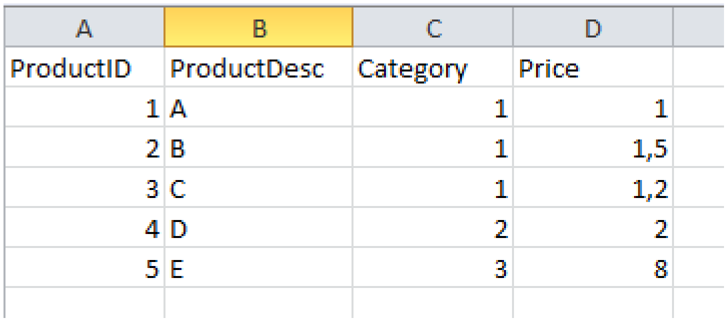
Not Populated is the default if the CustomerID is not found in the MAPPING table. We could also imagine a default like : “Not populated for Customer “ & CustomerID, so that the user see immediately which CustomerID has not been populated.

Lookup()

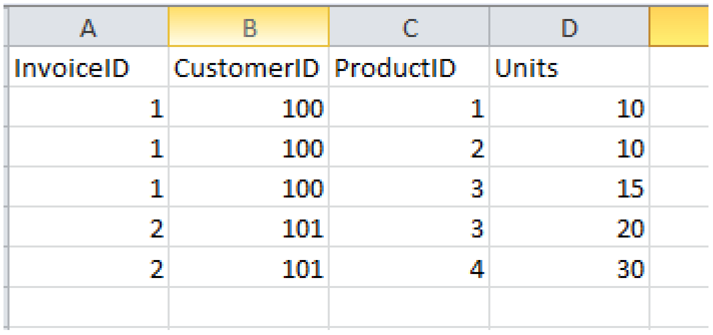
It is a similar function as **ApplyMap**() but the number of fields may be greater than 2 in the source table: so, you can use all the tables you want.

ApplyMap() is faster than **Lookup**(): it is why it is more widely used.

```
Syntax: Lookup(‘FieldName’, ‘KeyFieldName’, KeyFieldValue [, ‘TableName’])
```



A	B	C	D
ProductID	ProductDesc	Category	Price
1	A	1	1
2	B	1	1,5
3	C	1	1,2
4	D	2	2
5	E	3	8

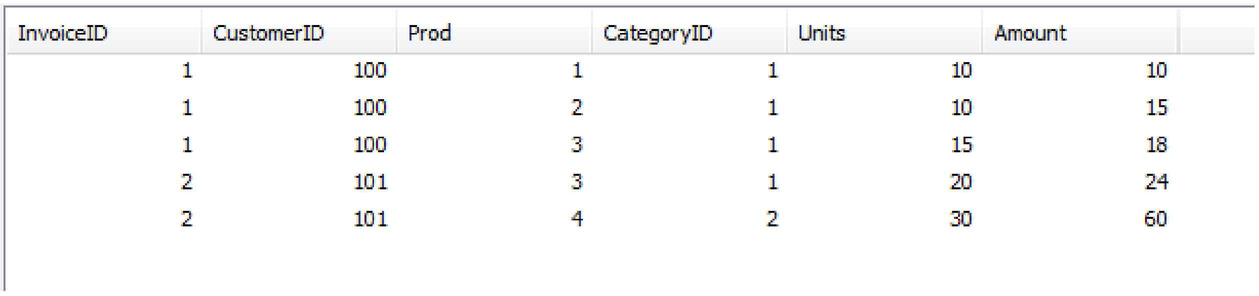


A	B	C	D
InvoiceID	CustomerID	ProductID	Units
1	100	1	10
1	100	2	10
1	100	3	15
2	101	3	20
2	101	4	30

```
Product_List:
LOAD ProductID,
     ProductDesc,
     Category,
     Price
FROM
Product.xlsx
(ooxml, embedded labels, table is Feuil1);

Data:
LOAD InvoiceID,
     CustomerID,
     ProductID as Prod,
     Lookup('Category', 'ProductID', ProductID, 'Product_List') as CategoryID,
     Units,
     Units * Lookup('Price', 'ProductID', ProductID, 'Product_List') as Amount
FROM
Product.xlsx
(ooxml, embedded labels, table is Feuil2);
```

Please, notice that:

- The source table (Product_List) is loaded without any prefix: it is a “normal” table belonging to the model
- The **Lookup()** function uses the source name of the field (ProductID), before renaming.



InvoiceID	CustomerID	Prod	CategoryID	Units	Amount
1	100	1	1	10	10
1	100	2	1	10	15
1	100	3	1	15	18
2	101	3	1	20	24
2	101	4	2	30	60

Further resources on JOIN and Lookups: see the excellent post (as usual) written by HIC:
http://community.qlikview.com/docs/DOC-3412

MapSubString()

This function replaces part of the string found in one table by the result of another table. **ApplyMap**() or **Lookup**() do a comparison based on the complete field: that is not the case with **MapSubString**().

3.1.2 Other uses of mapping tables

The mapping tables let you:

- Modify the content of one field with the content of the other, as we already saw it in previous section
- Rename tables, fields
- Add comments to tables or fields
- Add tags to fields

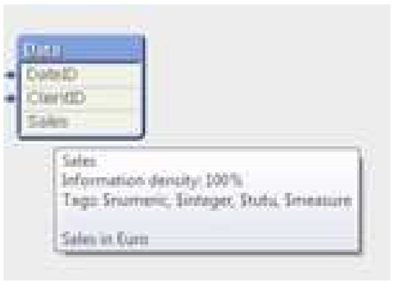
The process is divided in three steps:

1. Load the mapping table with the appropriate information: a) if you want to rename fields, the 1st column must contain the current field names and the 2nd column the new names, b) if you want to add comments to tables, the 1st column will contain the current table names and the 2nd column the comments you want to apply
2. Use the appropriate command
   a. Add comments to fields: **Comment fields using** MapTable
   b. Add comments to tables: **Comment tables using** MapTable
   c. Rename fields: **Rename fields using** MapTable
   d. Rename tables: **Rename tables using** MapTable
   e. Add tags to fields: **Tag fields using** MapTable

Example:

FieldName	Comments
ClientID	ID of each Client
ZIP	ZIP code of headquarter of the client
City	City of headquarter of the client
Country	Country of headquarter of the client
DateID	Date of order
Sales	Sales in Euro
Data	Data coming from the OSA database

```
Map_Comment:
Mapping LOAD
     FieldName,
     Comments
FROM
[MyFile.xlsx]
(ooxml, embedded labels, table is Comments);

COMMENT Fields using Map_Comment;
COMMENT Tables using Map_Comment;
```



Data
DateID
ClientID
Sales

Sales
Information density: 100%
Tags: $numeric, $integer, [illegible], $measure

Sales in Euro

Note that the lines that would not be recognized (here, because they are not valid fields) are just ignored: they are no error message. So, we can use the same mapping tables to add comments to fields and tables.

3.2 Loops

3.2.1 Classical Loops

As seen in section 2.2, we can use:

- For each *var* in *liste* [….] next
- Do / Loop

```
for each file in 'A.xls', 'B.xls', 'C.xls'
  LOAD […] FROM $(file) […]
Next
```

The condition in the do loop may be set several ways:

- Do while condition [….] loop
- Do until condition […] loop
- Do [….] loop while condition
- Do […] loop until condition

The condition is a boolean that becomes false by the code between the commands **do** and **loop**. A counter is a typical example.

```
let counter=1;
do while counter < 10
 […]
 let counter=counter+1;
loop
```

A loop may also be terminated if the **exit do** command is encountered. QlikView will execute then the command just after the **loop** command.

3.2.2 IterNo()

IterNo() is an iterative integer function that starts at 1. It is used with the **WHILE** statement to populate the table: the LOAD will be repeated for each line of the source while the WHILE statement is true.

We can populate a calendar table with the **IterNo()** function to generate all the rows (the dates) between the first and the last dates:

```
Calendar:
Load
  date(BeginMonth+IterNo()-1) as Date,
  month(BeginMonth+Iterno()-1) as Month,
  Year(BeginMonth+Iterno()-1) as Year
Resident Temp
WHILE BeginMonth+Iterno()-1<= EndingMonth;
```

For each row of the temp table, we will generate all the dates between the beginning and the end of the month contained in the fields BeginMonth and EndingMonth.

Because **IterNo**() starts at 1, the use of it will often include a -1.

You will find many examples in the community and especially in the posts by HIC. One of them concerns Generating missing data: http://community.qlikview.com/docs/DOC-3786

3.2.3 Subfield()

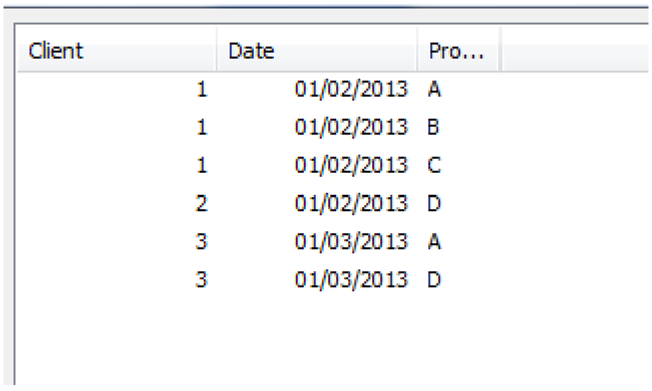
Subfield() is a function that will duplicate the lines by splitting the different fields contained in a value of a field.

Client	Date	Product
1	01/02/2013	A;B;C
2	01/02/2013	D
3	01/03/2013	A;D

In this file, we want to get 3 lines of the Client1 (because 3 products A, B and C), 1 line for Client 2, 2 lines for Client 3 (products A and D).

```
Sales:
LOAD Client,
     Date,
     subfield(Product, ';') as Product
FROM
SubField.xlsx
(ooxml, embedded labels, table is Feuil1);
```

For the result:



Client	Date	Pro...
1	01/02/2013	A
1	01/02/2013	B
1	01/02/2013	C
2	01/02/2013	D
3	01/03/2013	A
3	01/03/2013	D

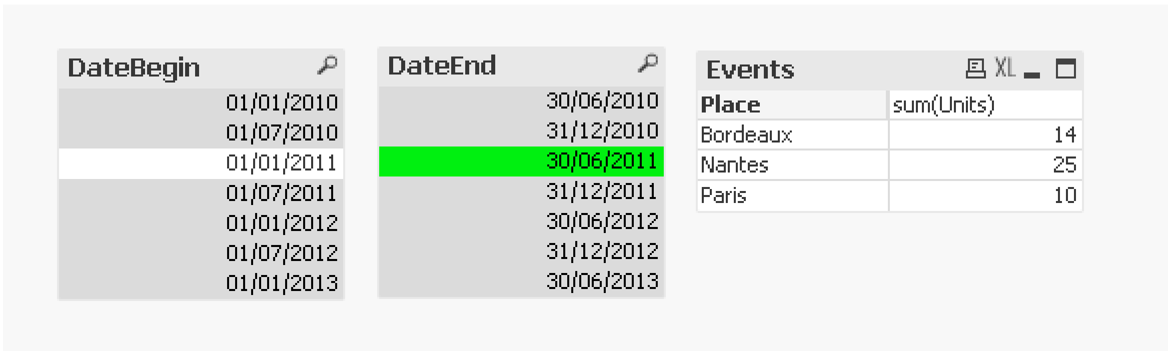
If we use the **subfield**() function on several fields, we will get the Cartesian product of the two fields. For example, if the Field1 (here Date) contains 2 values and the Field2 (here Product) contains 3 values, we would get 6 lines at the end for the Client.

You can get more information with the following post: http://community.qlikview.com/docs/DOC-4657

3.2.4 IntervalMatch

We use the **IntervalMatch** prefix when we need to match a numeric value (or date) to intervals. Afterwards, we may use these intervals as a dimension.

For example, I want to see the number spectacles by town between two dates DateBegin and DateEnd. Here the intervals are 6 month long, but they could be changing by year.



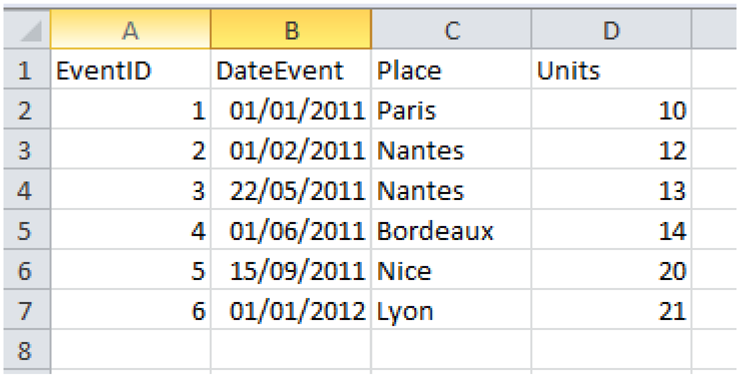
DateBegin
01/01/2010
01/07/2010
01/01/2011
01/07/2011
01/01/2012
01/07/2012
01/01/2013

DateEnd
30/06/2010
31/12/2010
30/06/2011
31/12/2011
30/06/2012
31/12/2012
30/06/2013

Events

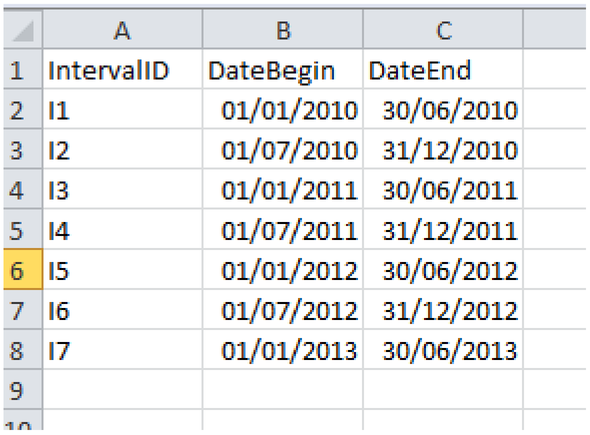
Place	sum(Units)
Bordeaux	14
Nantes	25
Paris	10

But all the spectacles have a specific date (French dates are coded DD/MM/YYYY):



	A	B	C	D
1	EventID	DateEvent	Place	Units
2	1	01/01/2011	Paris	10
3	2	01/02/2011	Nantes	12
4	3	22/05/2011	Nantes	13
5	4	01/06/2011	Bordeaux	14
6	5	15/09/2011	Nice	20
7	6	01/01/2012	Lyon	21
8				

Basically, we need to create a dimension with two boundaries:



	A	B	C
1	IntervalID	DateBegin	DateEnd
2	I1	01/01/2010	30/06/2010
3	I2	01/07/2010	31/12/2010
4	I3	01/01/2011	30/06/2011
5	I4	01/07/2011	31/12/2011
6	I5	01/01/2012	30/06/2012
7	I6	01/07/2012	31/12/2012
8	I7	01/01/2013	30/06/2013
9			

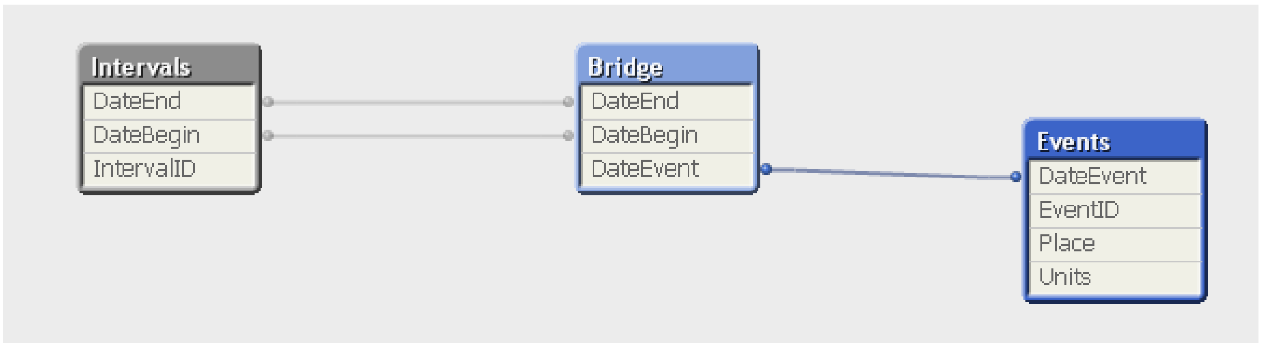
IntervalMatch will then link the single date to this interval.

The way to perform this:

1) Load the two tables: the discrete field, the intervals
2) Create a Bridge table with IntervalMatch(Discrete Field) prefix

```
//The discrete fields
Events:
LOAD EventID,
     DateEvent,
     Place,
     Units
FROM
Intervals.xlsx
```

```
(ooxml, embedded labels, table is Feuil1);

// the optional dimension IntervalID and the two boundaries
Intervals:
LOAD IntervalID,
     DateBegin,
     DateEnd
FROM
Intervals.xlsx
(ooxml, embedded labels, table is Feuil2);

//the links made by QlikView: 2 boundaries only
Bridge:
IntervalMatch (DateEvent)
LOAD DISTINCT DateBegin, DateEnd
Resident Intervals;
```



Further resources; see the complete and excellent post by Henric Cronström: http://community.qlikview.com/docs/DOC-4310

3.3 Read several files under a directory

We have seen several examples that may guide you. You can also find resources in the community:

http://community.qlikview.com/docs/DOC-1306

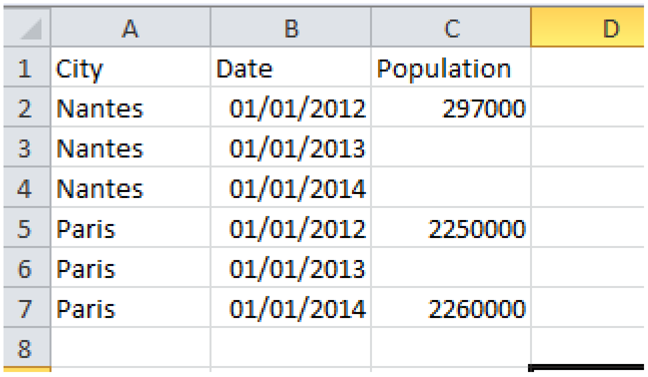
http://community.qlikview.com/thread/89440

Read several sheets of a Excel file

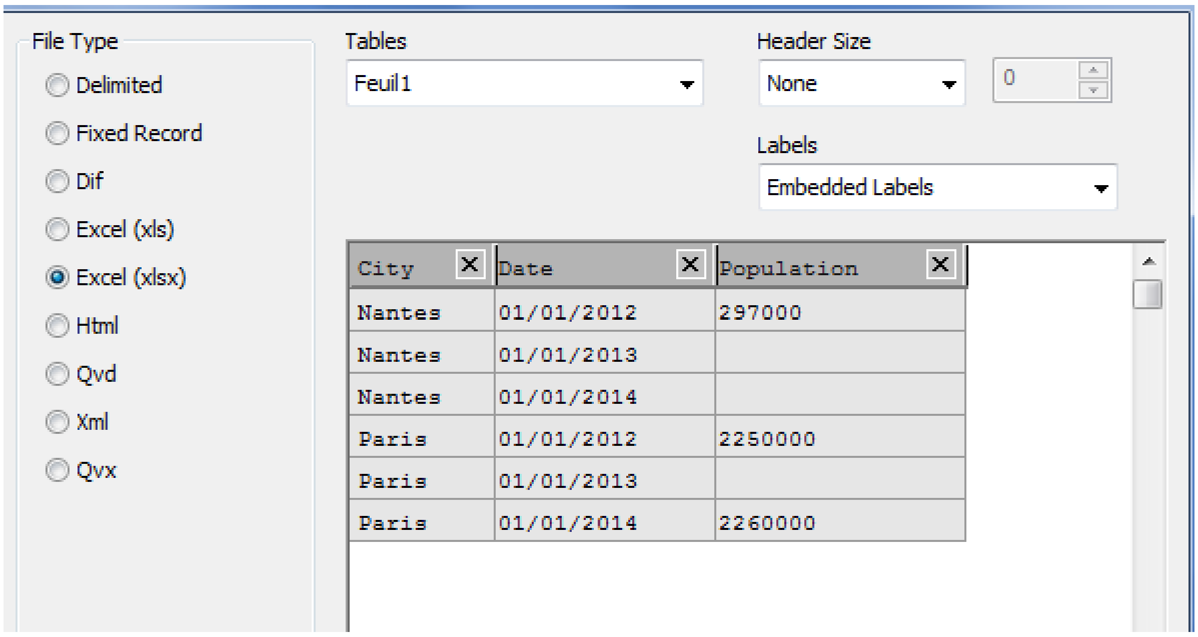
http://community.qlikview.com/docs/DOC-4452

3.4 Reuse information of a previous line

The population is not entered for each year. If it is empty, we want to take the population of the previous year. You may use the GUI to replace null values with above lines:



	A	B	C	D
1	City	Date	Population	
2	Nantes	01/01/2012	297000	
3	Nantes	01/01/2013		
4	Nantes	01/01/2014		
5	Paris	01/01/2012	2250000	
6	Paris	01/01/2013		
7	Paris	01/01/2014	2260000	
8				



File Type: ( ) Delimited ( ) Fixed Record ( ) Dif ( ) Excel (xls) [✓ Excel (xlsx)] ( ) Html ( ) Qvd ( ) Xml ( ) Qvx

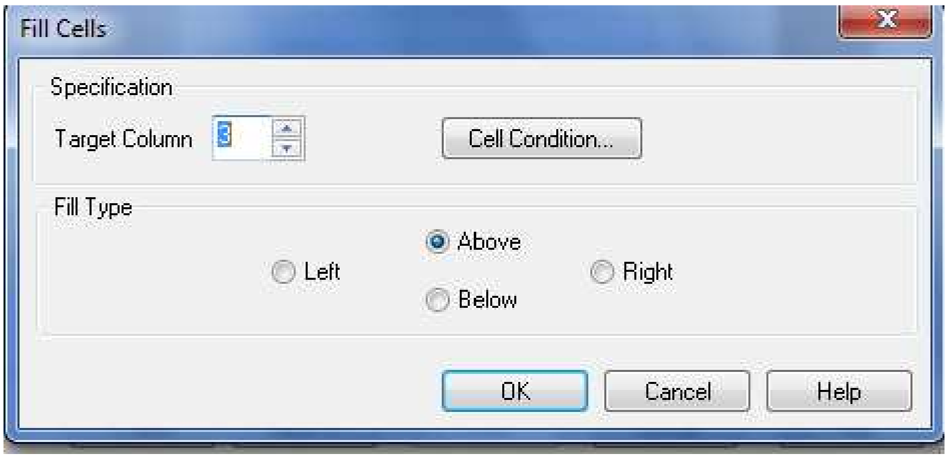
Tables: Feuil1

Header Size: None 0

Labels: Embedded Labels

City	Date	Population
Nantes	01/01/2012	297000
Nantes	01/01/2013	
Nantes	01/01/2014	
Paris	01/01/2012	2250000
Paris	01/01/2013	
Paris	01/01/2014	2260000

Click Next, Enable Transformation Step, Click the Fill button and the Cell Condition:



Fill Cells

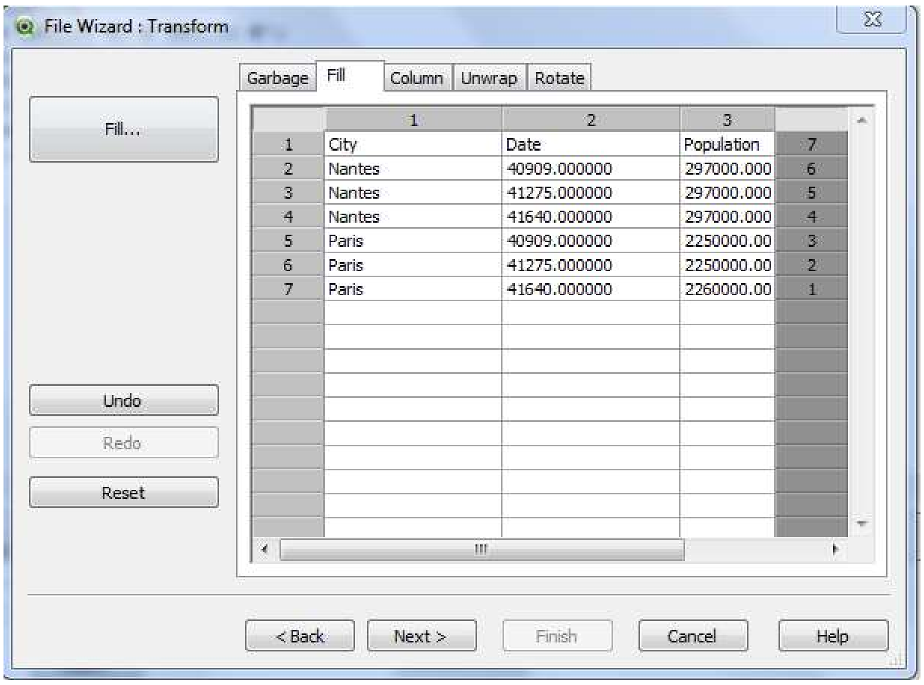
Specification

Target Column 3

Cell Condition...

Fill Type: ( ) Left [✓ Above] ( ) Below ( ) Right

OK Cancel Help



	1	2	3	
1	City	Date	Population	7
2	Nantes	40909.000000	297000.000	6
3	Nantes	41275.000000	297000.000	5
4	Nantes	41640.000000	297000.000	4
5	Paris	40909.000000	2250000.00	3
6	Paris	41275.000000	2250000.00	2
7	Paris	41640.000000	2260000.00	1

It will generate the following line:

```
LOAD City,
     Date,
     Population
FROM
Population.xlsx
(ooxml, embedded labels, table is Feuil1, filters(
Replace(3, top, StrCnd(null))
));
```

This assumes that you may replace the data of the 3d column of the current line with the data of the previous line: there is no test of validity (that the city is the good one for example).

This wizard is very useful and we do not think of it most of the time. For example, if you want to read one column among two with the same names (2 columns have the same title), you can also use the wizard to generate such a script:

```
LOAD [….]

FROM [C:\Qlik examples\test.csv]

(txt, codepage is 1252, embedded labels, delimiter is ',', msq,

filters(Remove(Col, Pos(Top, 7))));
```

See the answer made by SALTO for this thread: http://community.qlikview.com/thread/103203

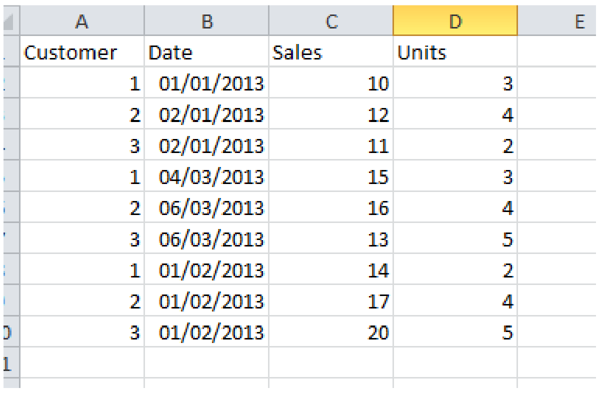
Most of the time, we use several functions on an **ordered** file or table; if the file is not ordered, load it first into a temp table and then reorder it by loading it into a new table:

- **Peek**('FieldName' [, rowNumber [,'TableName]]): returns the value of the field name. The row number may be positive (index 0) or negative (-1 means the last record read: it is the default). The table name is by default the table being populated. You will use the **peek**() function to get a value for a populated table. The Field Name may be in construction through the **as** keyword.
- **Previous**(FieldName): returns the last record of the field name that was populated. Returns **Null** for the first line. The field name must exist (you cannot use it if the field is in construction with the **as** keyword)
- **If**(boolean condition, value if true[, value if false])

- **Rangesum**(value1, value2 [, value3 …]): sum the different values, if one of the value is null, **rangesum** will NOT return null. When you want to cumulate the values with the data of previous line, **peek**() or **previous**() will return **null** for the first line: if you just add current line with the data of the previous one, the sum will be null, except if you use **rangesum**().

Examples:

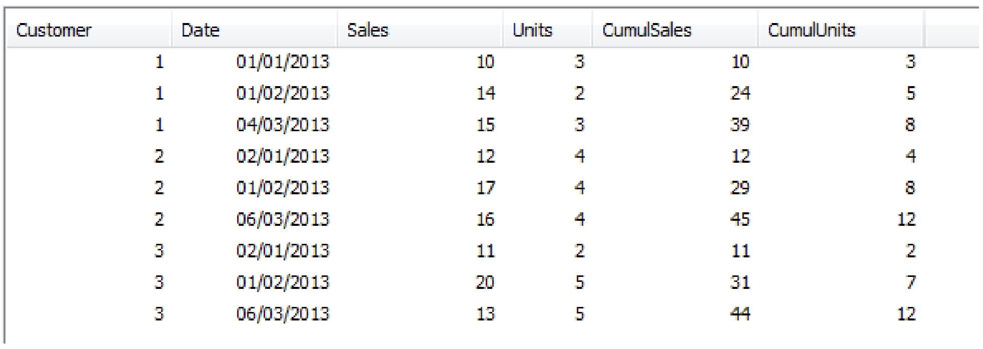
To cumulate data with the previous one only if the customer is the same:



Customer	Date	Sales	Units
1	01/01/2013	10	3
2	02/01/2013	12	4
3	02/01/2013	11	2
1	04/03/2013	15	3
2	06/03/2013	16	4
3	06/03/2013	13	5
1	01/02/2013	14	2
2	01/02/2013	17	4
3	01/02/2013	20	5

Notice that the rows are not ordered: we will need to load the file first, and order the rows later:

```
Temp:
LOAD Customer,
     Date,
     Sales,
     Units
FROM
CumulativeSales.xlsx
(ooxml, embedded labels, table is Feuil1);

Sales:
NoConcatenate
LOAD Customer, Date, Sales, Units,
if (previous(Customer)=Customer, Rangesum(Sales, peek('CumulSales')), Sales ) as
CumulSales,
if (previous(Customer)=Customer, Rangesum(Units, Peek('CumulUnits')), Units ) as
CumulUnits
Resident Temp
ORDER BY Customer, Date;

drop table Temp;
```



Customer	Date	Sales	Units	CumulSales	CumulUnits
1	01/01/2013	10	3	10	3
1	01/02/2013	14	2	24	5
1	04/03/2013	15	3	39	8
2	02/01/2013	12	4	12	4
2	01/02/2013	17	4	29	8
2	06/03/2013	16	4	45	12
3	02/01/2013	11	2	11	2
3	01/02/2013	20	5	31	7
3	06/03/2013	13	5	44	12

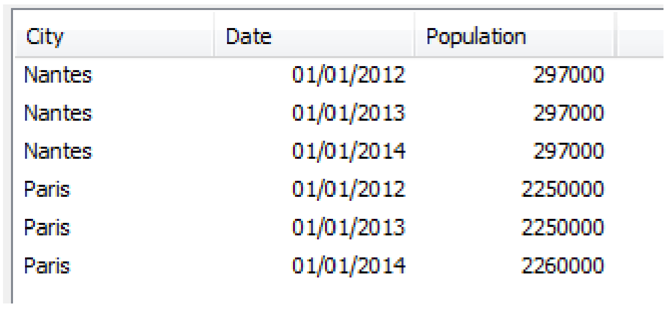
The **Rangesum**() function should be preferred than a simple **+** addition, in case some lines are null.

In this example, the **previous**() function could not be used to cumulate data because the used field is renamed (as CumulSales).

To get the data of the last populated line (not only the previous one) like in our first file where the Population in Nantes had only 1 data for the first year:

```
Population:
LOAD City,
     Date,
     if(isnull(Population), peek('Population', Counter), Population) as Population;
LOAD City,
     Date,
     Population,
     if(isnull(Population), rangesum(peek('Counter'),-1), 0) as Counter
FROM
Population.xlsx
(ooxml, embedded labels, table is Feuil1);
```

Please, notice that we use the technique of the preload (see section 2.8). We create in the second LOAD command (but it is the first to be executed in that case) a counter that stores the number of lines needed to “go up” the table to find the population: if the population is null, we need to go up one line more (the counter of the previous line minus 1). If the population is “populated”, we will read this line: therefore the counter must be 0.



City	Date	Population
Nantes	01/01/2012	297000
Nantes	01/01/2013	297000
Nantes	01/01/2014	297000
Paris	01/01/2012	2250000
Paris	01/01/2013	2250000
Paris	01/01/2014	2260000

Other posts in the community:

- when use **Peek**() vs **Previous**(). See the post by Charles BANNON: http://community.qlikview.com/docs/DOC-4073
- using recursively **Peek**() and **Previous**() by Steve RIMAR : http://community.qlikview.com/docs/DOC-1304

3.5 Populate a local variable with a value of a table

The **peek**() function will also be used when we want to fetch data that is already stored into a table:

```
Syntax: Peek(‘FieldName’, Row, ‘TableName’)
```

The row number starts at 0 and ends at **NoOfRows**(‘TableName’)-1 to get the last line of the table. The TableName is optional during the LOAD statement (QlikView assumes the table name is the one being loaded), but here you MUST enter the name.

Example:

We loaded the dates into a temp table in order to get the min and max date. Of course, this table was ordered by date to be sure that the first line is the oldest one, and the last line the newest one.

```
vMinDate = num(peek('Date', 0, 'Temp_Table'));
vMaxDate = num(peek('Date', NoOfRows('Temp_Table')-1, 'Temp_Table2'));
```

NoOfRows() is a function that returns the number of lines of the table passed as argument.

3.6 Number the loaded lines

It can be useful to number the lines of a table to:

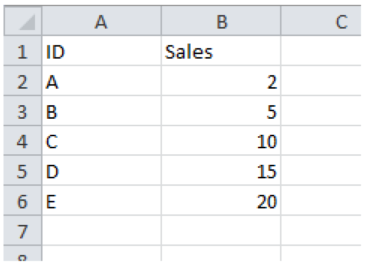
- Easy the order of the table
- Link different tables based on an integer (it will be easier and quicker for QlikView to operate that link)

3.6.1 RowNo() and RecNo()

You can use these functions:

- **RowNo**(): numerates the lines of the table
- **RecNo**(): numerates the lines of the loaded file. If all the lines are loaded in one shot, **recno**() is equivalent to **rowno**(). If you read several files to populate a table, take care that **Recno**() starts again at 1.

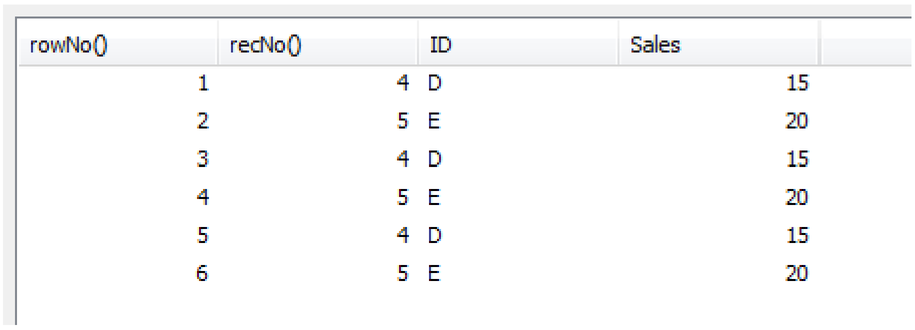
We want to load only part of this file, and just to explain the result of a loop, we load it three times:



	A	B	C
1	ID	Sales	
2	A	2	
3	B	5	
4	C	10	
5	D	15	
6	E	20	
7			

```
let counter = 1;
do
table1:
LOAD rowNo(),
     recNo(),
     ID,
     Sales
FROM
Number.xlsx
(ooxml, embedded labels, table is Feuil1)
Where Sales > 10
;
counter=counter+1;
loop until counter > 3
```

For this result:



rowNo()	recNo()	ID	Sales
1	4	D	15
2	5	E	20
3	4	D	15
4	5	E	20
5	4	D	15
6	5	E	20

As you may notice, **RowNo**() number the lines of the table (the returned numbers are different even if we read several times the same file) while **RecNo**() numbers the lines of the file, and therefore may repeat the same number (4 and 5 are repeated 3 times).

3.6.2 AutoNumber functions

The functions will return a unique number according to the expression(s) given in parameter. It can be a good option to use these functions to transform string ID into Integer ID that will be used as links. Like SQL, QlikView prefers integer links to string links.

You can use these functions:

- **Autonumber(**expression**):** returns an integer according to the expression given in argument
- **AutonumberHash128**(expressions), **AutonumberHash160**(expressions), **AutonumberHash256**(expressions): returns a number according to the values given in argument. This is useful when creating a link table. See below.

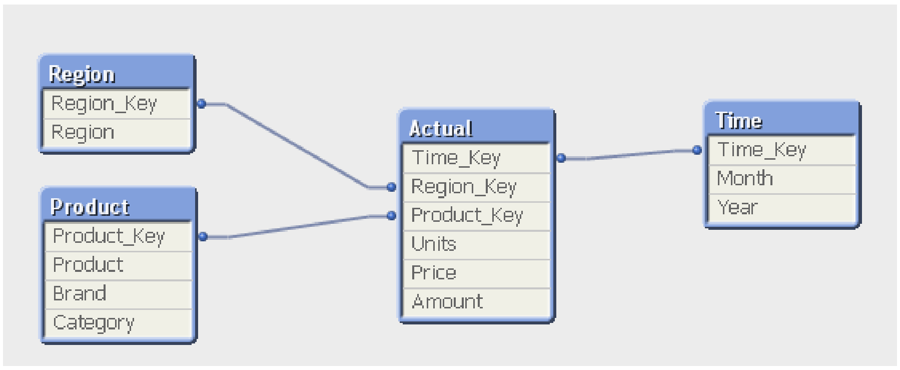
Imagine that we need to load different tables with different dimensionality:

- Actual Data: split by Time, Product and Region
- Price: split by Time and Product (same price everywhere for every client)
- Budget: split by Product and Region

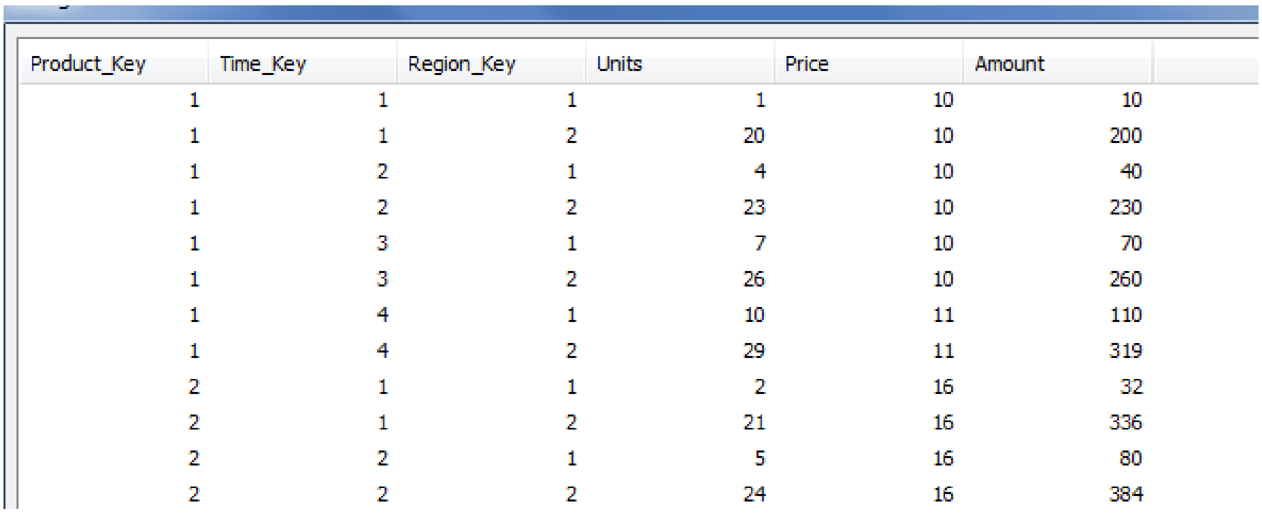
Because the Price means that the data is the same for all products, we can multiply the lines. We will do it with a **LEFT JOIN** and then compute the Sales in Euro:

```
// the sold units are split by Region, Product and Time
Temp_Actual:
LOAD Region_Key,
     Product_Key,
     Time_Key,
     Units
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Actual);

// The price is the same for ALL regions: it is split only by Time and Product
Temp_Actual:
LEFT JOIN
LOAD Product_Key,
     Time_Key,
     Price
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Price);

// To compute the Sales in Euro
Actual:
NoConcatenate
LOAD *,
     Units * Price as Amount
Resident Temp_Actual;
DROP table Temp_Actual;
```

I get this model before loading the budget:



And the price lines are multiplied:

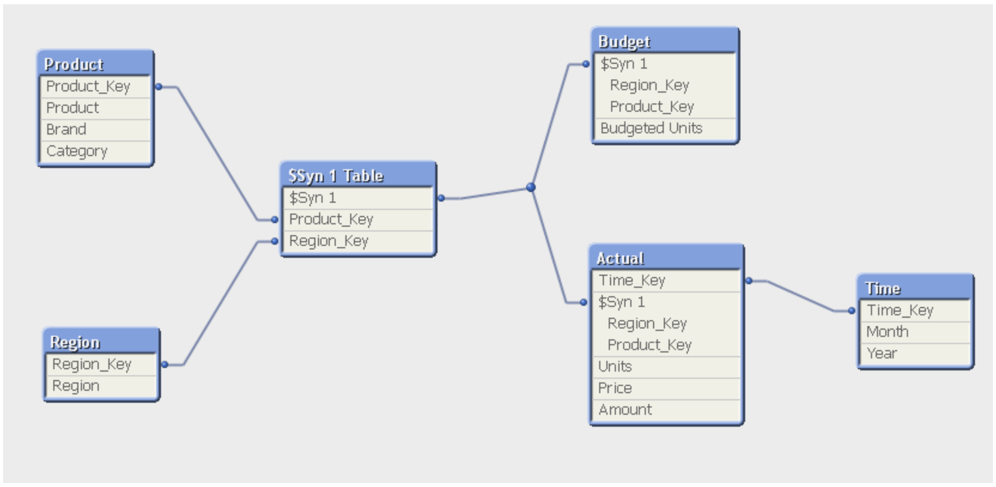


Product_Key	Time_Key	Region_Key	Units	Price	Amount
1	1	1	1	10	10
1	1	2	20	10	200
1	2	1	4	10	40
1	2	2	23	10	230
1	3	1	7	10	70
1	3	2	26	10	260
1	4	1	10	11	110
1	4	2	29	11	319
2	1	1	2	16	32
2	1	2	21	16	336
2	2	1	5	16	80
2	2	2	24	16	384

If we load the budget data:

```
Budget:
LOAD Region_Key,
     Product_Key,
     Units as [Budgeted Units]
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Budget);
```

We see that there are 2 common fields with the Actual table: Region_Key and Product_Key. We will get therefore a synthetic key:

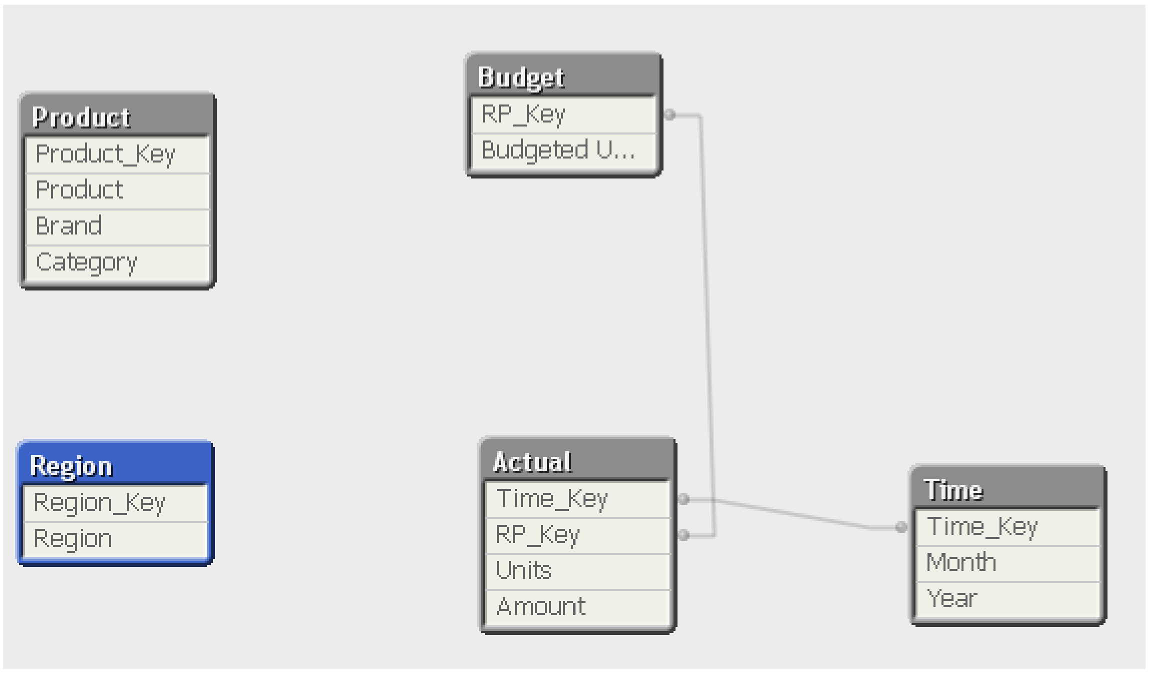


The synthetic key is composed of two fields: Product_Key and Region_Key. If there were only one common field between the tables, they would be linked naturally.

All the literature, the books, tells us to remove this synthetic key. But how? We need to merge the fields of the synthetic key in every table we find it. Something like:

```
Actual:
NoConcatenate
LOAD Region_Key & '_' & Product_Key as RP_Key,
     Time_Key,
     Units,
     Units * Price as Amount
Resident Temp_Actual;
DROP table Temp_Actual;

Budget:
LOAD Region_Key & '_' & Product_Key as RP_Key,
     Units as [Budgeted Units]
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Budget);
```

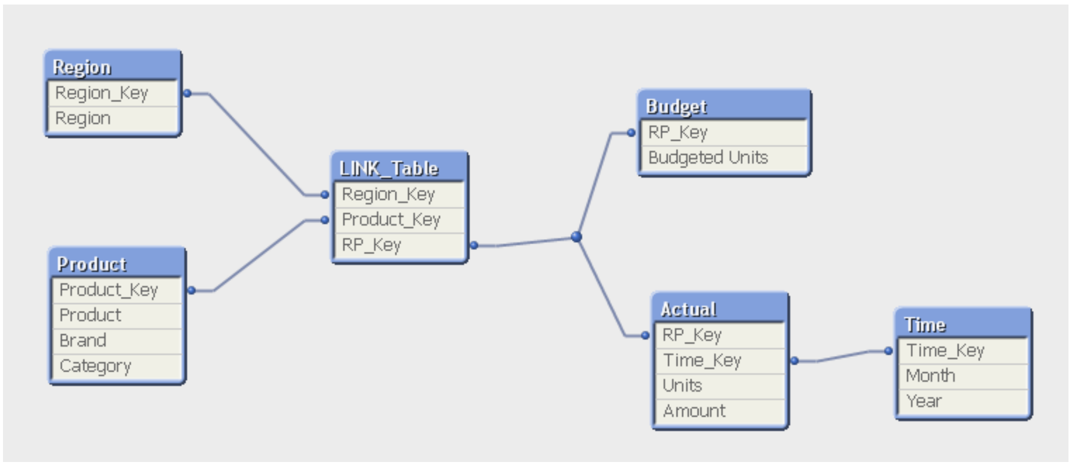
But if we forget the dimensions, there is something missing. The Actual field is not related any more to Product, Brand, Category:



We need to create an intermediate table that will do the link for each composite key to the different fields that compose it.

```
LINK_Table:
LOAD Region_Key & '_' & Product_Key as RP_Key,
     Region_Key,
     Product_Key
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Budget);
```

To get this result:



The links are made with strings, we can do the same with numbers: it will be more efficient.

The **AutoNumber**() functions generate a number according to the given parameter, and will return the same number if you pass the same parameters (in order to complete the link, we need the same number).

The **AutoNumber**() accepts only ONE text argument to compute the number. We need to concatenate the different fields.

Let's take the previous example above:

```
Budget:
LOAD AutoNumber(Region_Key & '_' & Product_Key) as RP_Key,
     Units as [Budgeted Units]
FROM
[Reel vs Budget.xlsx]
(ooxml, embedded labels, table is Budget);
```

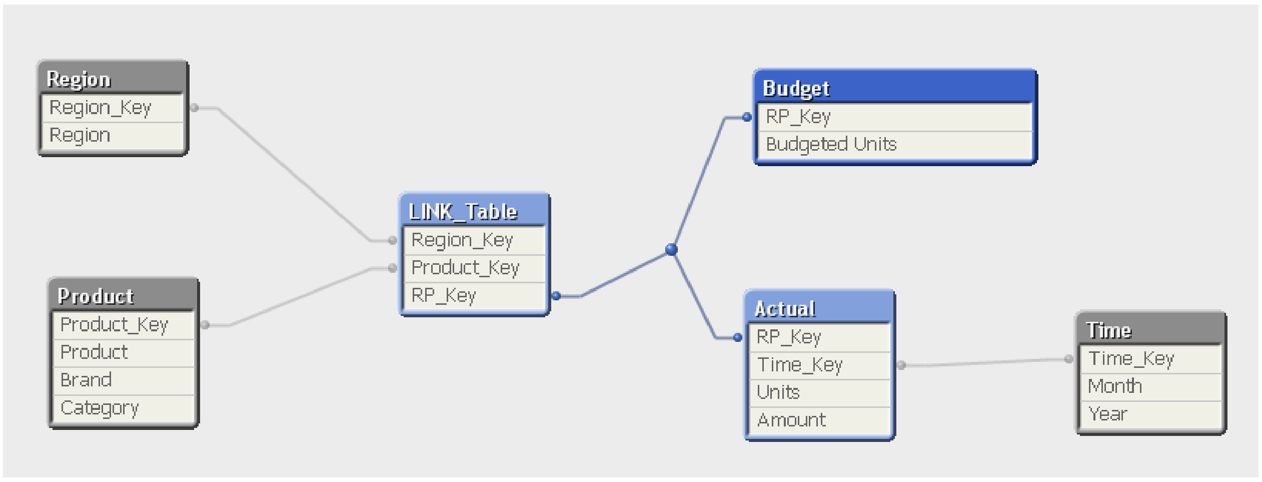
If you use **AutoNumber**() to number several fields, you will need to add a second argument to differentiate the counters:

```
AutoNumber(String to compute the number, Anther String to differentiate the fields)
```

We may use **AutoNumberhash128** (or 256) function that accepts several parameters:

Of course, the method applied should be the same for each composite key found in different tables.

You will get the same model:



But the RP_Key has become an integer.

3.7 Hierarchical load

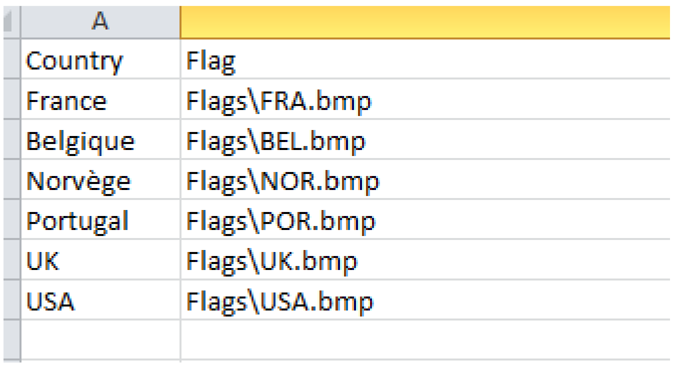
I already documented deeply this topic in a separate doc. You can find it at: http://community.qlikview.com/docs/DOC-4823

3.8 Image load

We want sometimes display some images depending on a selection made by the users, like a flag for one country, a picture of a product …

Several techniques may be used

3.8.1 Reference the external files



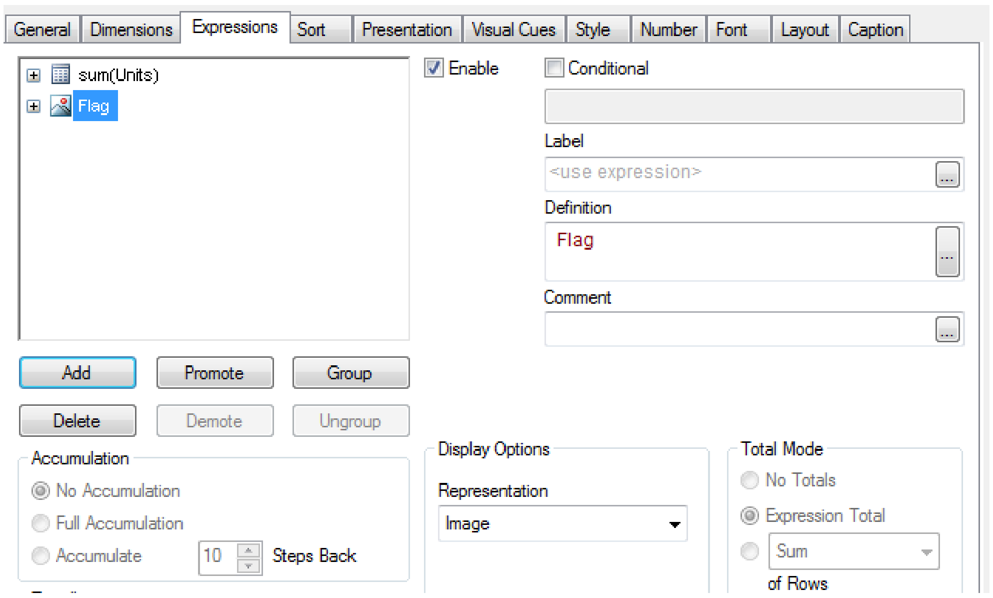
A	
Country	Flag
France	Flags\FRA.bmp
Belgique	Flags\BEL.bmp
Norvège	Flags\NOR.bmp
Portugal	Flags\POR.bmp
UK	Flags\UK.bmp
USA	Flags\USA.bmp

The Flag field is a path to a real file in the drive. This path may be absolute or relative.

```
Country:
LOAD Country,
     Flag
FROM
Country.xlsx
(ooxml, embedded labels, table is Flags);
```

As you notice, this field is a “normal” string field. The **representation** of the field will be **image**:

In a table in the GUI:


General | Dimensions | Expressions | Sort | Presentation | Visual Cues | Style | Number | Font | Layout | Caption

sum(Units)
Flag

[✓ Enable] Conditional

Label: <use expression>

Definition: Flag

Comment:

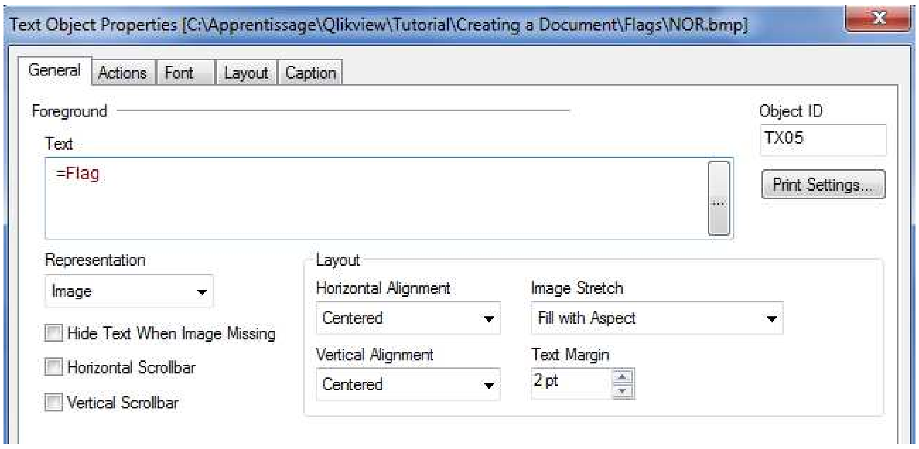
Add | Promote | Group | Delete | Demote | Ungroup

Accumulation: [✓ No Accumulation] Full Accumulation Accumulate 10 Steps Back

Display Options — Representation: Image

Total Mode: No Totals [✓ Expression Total] Sum of Rows

In a textbox:


Text Object Properties [C:\Apprentissage\Qlikview\Tutorial\Creating a Document\Flags\NOR.bmp]

General | Actions | Font | Layout | Caption

Foreground — Text: =Flag

Object ID: TX05

Print Settings...

Representation: Image

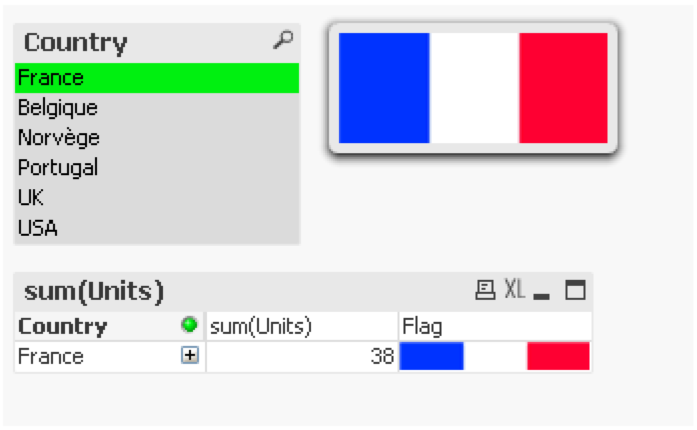
Hide Text When Image Missing

Horizontal Scrollbar

Vertical Scrollbar

Layout — Horizontal Alignment: Centered; Image Stretch: Fill with Aspect; Vertical Alignment: Centered; Text Margin: 2 pt

To get this result:



Country	sum(Units)	Flag
France	38	🟦⬜🟥

The textbox should return a flag only if one country is selected. You may write in the layout tab a conditional statement like: =isnull(only(Country))=0 , so that the textbox disappears when several countries are selected.

3.8.2 Store the external files into the application

If you want to store the image files into your application (qvw file), especially if you need to distribute it, you will need to use the **BUNDLE** prefix.

```
Country:
BUNDLE
LOAD Country,
     Flag
FROM
Country.xlsx
(ooxml, embedded labels, table is Flags);
```

The **LOAD** must be done in a 2 field-table (like the **Applymaps**). And we will create an expression starting with **qmem** to get the image file:

```
Syntax: ‘qmem//LookUpField/LookUpFieldValue’
```

Take care, in order to get the flag of the country that is selected in the ListBox, we do not reference the flag field but only the lookup field (that is why you need to create a 2-field table):

='qmem://Country/'&Country

The formula is a string concatenating the **qmem://**, the lookup field as an immutable string (here country) and the choice made by the user (also Country but outside the fixed string).

Please, notice that:

- You can use also the **qmem** way of referencing files without bundling the files into the application. Use only the **INFO** prefix if you want to keep the files outside the application.
- You can load and link other types of files like audio, video …

3.9 VB use during a load

You may use VB in your script by creating your own functions that will be called during the script (take care, it is very long to execute).

Take care:

- All VB code must be in VB scripts
- The VB code is far longer to execute than the standard script
- Create only functions with arguments

Example:

You want to increment a counter but this one should return to 0 if it is greater than 100:

In the VB Module:

```
Function Increment(Counter)

if isnull(Counter) then Counter = 0
if Counter > 100 then Counter = 0
Increment= Counter + 1
End Function
```

In the script, you can use directly the function:

```
vBorne=Increment(vBorne);
```

You may also use it in the **LOAD** command. Your VB function has become a true function like any other of QlikView. If we want to compute the number of Sundays between two dates:

```
LOAD IntervalID,
     DateBegin,
     DateEnd,
     NbOfSundays(DateBegin, DateEnd) as NbDeDimanches
FROM
intervals.csv
(txt, utf8, embedded labels, delimiter is ';', msq);
```

And the VB function is:

```
Function NbOfSundays(DateBegin, DateEnd)
Dim counter, d1, d2

'Basic check
if NOT isdate(DateBegin) or NOT isdate(DateEnd)  then
  NbOfSundays = 0
  Exit Function
End if

d1=datevalue(DateBegin)
d2=datevalue(DateEnd)

if d2 < d1 then
   NbOfSundays = 0
   Exit function
End if

'Set the first date to Sunday
NbOfSundays = 0
counter = 0
while counter < 7 AND weekday(d1+counter) <> 1 AND d1+counter <= d2
  counter=counter+1
wend

'Count the number of sundays by incrementing 7 days at each loop
if weekday(d1+counter) = 1 then
  NbOfSundays = 1
  counter=cunter+7
End if

While d1+counter <= d2
  counter=counter+7
  NbOfSundays  = NbOfSundays + 1
Wend

End Function
```

Take care that some VB functions are similar to the QlikView functions, but you need to use VB functions.

You can also create SUB routines in the script (but not functions) that will be called by the main process. These routines must be created **before** they are called, why not in a previous tab:

```
Sub Load_TBL(vBorne, vFile)
table1:
LOAD rowNo() as Line1,
     recNo() as Line2,
     ID,
     Sales
FROM
$(vFile)
(ooxml, embedded labels, table is Feuil1)
Where Sales > $(vBorne);
End Sub
```

And in the main tab (and **AFTER** the definition of the routine):

```
call Load_TBL(1, 'Number.xlsx');
```

Further useful resources on VB macros by Jagan Mohan: http://community.qlikview.com/docs/DOC-4870

3.10 Error handling

An error always may come, especially when we do not expect it to come. The error handling will perform specific actions in case of errors.

Errormode = 0, 1 (by default) or 2:

In case of errors, QlikView will either ignore the error and continue the script (Errormode=0), or will halt and prompt the user with the errormessage (Errormode = 1, this mode should be avoided in batch mode because QlikView will wait until someone presses the OK button), or will stop the script and trigger an error message (Errormode=2)

The following options are used in case of **Errormode** is set to 0.

ScriptErrors: the error code of the last executed line. 1 means that there was no error, 3 means that there was a syntax error, 8 that the file was not found etc. You should test this variable just after a command that may crash.

ScriptErrorDetails: contain more details especially when the load concerns OLE DB or ODBC.

ScriptErrorCount: number of errors during the script. This variable is reset to 0 at the beginning of the script.

ScriptErrorList: contains all ScriptErrors during the script.

3.11 Trigger a load

3.11.1 Batch

This part should be done on the server side. There are many options (see the reference manual)

For a desktop, you need to create a batch and set up a Windows task if you want it to be executed at night for example.

Syntax:

```
Qv.exe /options  “application.qvw file”
```

The options are:

/r: to perform a complete load

/rp: to perform a partial reload

/v: to set a variable name (if your variable also start with a v, you will get 2)

/l: to perform a complete load, but the application will remain open

/lp: to perform a partial load, but the application will remain open

Example to perform a complete load and setting the variable vBorne to 20::

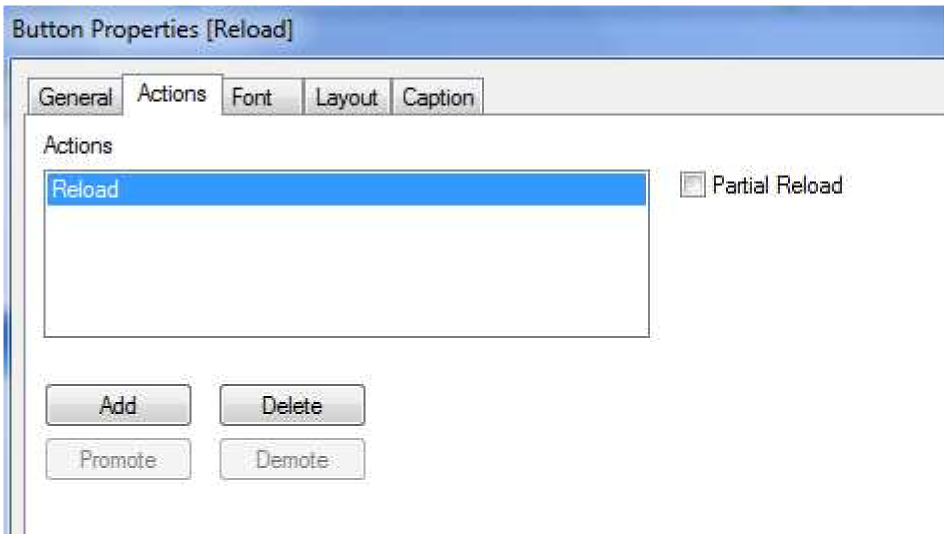
"C:\Program Files\QlikView\qv.exe" /r /vvBorne=20 C:\Apprentissage\Qlikview\Ecriture Documentation\Load_ENG\Number.qvw

To start this batch on a regular basis (every day, every Monday …), you may use a specific software or just the Task Scheduler of Windows.

Further resources: http://quickdevtips.blogspot.in/2012/06/qlikview-how-to-refresh-data.html

3.11.2 A button in the application

A simple button can also trigger the reload:



Button Properties [Reload]

General | Actions | Font | Layout | Caption

Actions

Reload

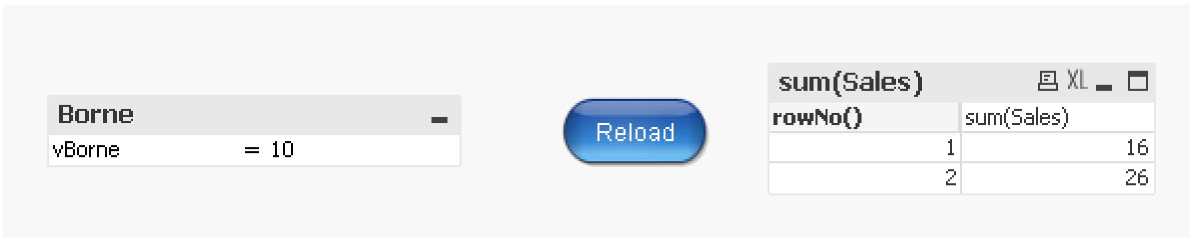
☐ Partial Reload

You can also indicate if the load is a partial reload.

You can also set a threshold in the GUI in order to read only some of the lines. Your code should use this or these variables, but also set a default in case the variable does not exist or is null:

```
let vBorne = alt(vBorne, 10);
table1:
LOAD rowNo(),
     recNo(),
     ID,
     Sales
FROM
Number.xlsx
(ooxml, embedded labels, table is Feuil1)
Where Sales > $(vBorne)
```

The **alt**() function returns the second parameter if the first one is null.

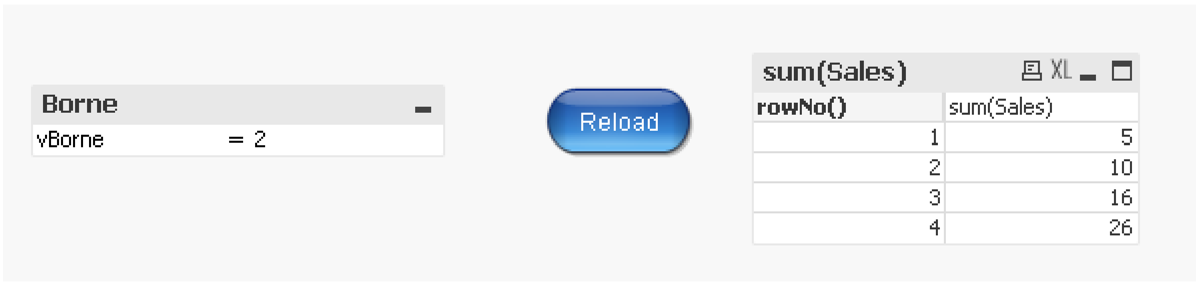


Borne

vBorne = 10

sum(Sales)

rowNo()	sum(Sales)
1	16
2	26



Borne

vBorne = 2

sum(Sales)

rowNo()	sum(Sales)
1	5
2	10
3	16
4	26

In case of batch, you really need to handle the errors that may come during the script.

4 Other useful functions and options when reading data

4.1 Qualify

The tables are linked through fields that have the same name. if, for any reason, you will have two fields with same name but you do not want QlikView to perform a link, you will have to rename that field (AS keyword).

You can use the AS keyword to rename these fields, but also QUALIFY that will prefix the fields with the name of the table.

```
Syntax: QUALIFY FieldList

Syntax: QUALIFY *  (for all fields)
```

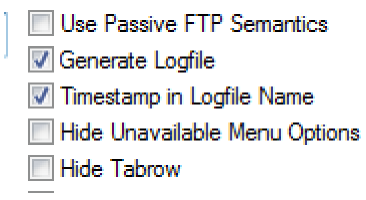
Each time QlikView will load these qualified fields, it will prefix them with the name of the table (Field1 will be renamed Table1.Field1). The tables cannot be linked anymore with that field because they cannot have the same name.

To remove that

```
Syntax: UNQUALIFY FieldList

Syntax: UNQUALIFY *  (for all fields)
```

4.2 Log and Trace

To generate a log file, you must activate one checkbox you will find by pressing in the GUI menu **Settings**/**Document Properties** and choosing the **General** tab:



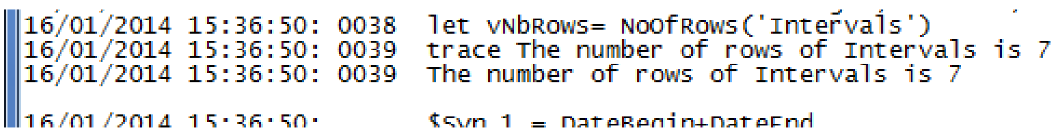
- [ ] Use Passive FTP Semantics
- [x] Generate Logfile
- [x] Timestamp in Logfile Name
- [ ] Hide Unavailable Menu Options
- [ ] Hide Tabrow

You may also add a timestamp to the log filename.

The command **Trace** will let you show some useful information about the load. **There is no quote**: to show the content of a variable, use the $ expansion.

```
Syntax: Trace  string
```

```
let vNbRows= NoOfRows('Intervals');
trace The number of rows of Intervals is $(vNbRows);
```



```
16/01/2014 15:36:50: 0038   let vNbRows= NoOfRows('Intervals')
16/01/2014 15:36:50: 0039   trace The number of rows of Intervals is 7
16/01/2014 15:36:50: 0039   The number of rows of Intervals is 7

16/01/2014 15:36:50:        $Syn 1 = DateBegin+DateEnd
```

4.3 File Information

We can use several functions get information about different type of files:

- **Filelist** (‘Directory’, search string): returns a list of the files
- **Filesize** (‘FileName’): returns the size in bytes of the file
- **Fileextension**(‘FileName’): returns the extension of the file
- **Filetime**(‘FileName’): returns the timestamp of the last modification of the file

We have already seen these functions in the section 1.1.6. If the file does not exist, the function returns null.

- **QVDCreateTime**(‘QVD File’): string, the creation date of the QVD file.
- **QVDNoOfFields**(‘QVD File’): integer, the number of fields contained by the file.
- **QVDNoOfRecords**(‘QVD File’): integer, the number of rows contained by the file.
- **QVDTableName**(‘QVD File’): string, the table name used to populate the QVD file.
- **QVDFieldName**(‘QVD File’, Number): string, the field name at the given position. If the number is too high and therefore the field does not exist, the function returns null.

4.4 Exporting data

You may export data during the script. This data may be reused in a later script to make the load quicker, for example the QVD.

You can export data from ONE table only at each time. If you need to export data from 2 tables, you will write 2 STORE commands.

```
Syntax: STORE [FieldList FROM] TableName INTO FileName (TypeOfExtract)
```

Where:

The **fieldlist** is an optional argument if you want to export only some fields of the table. Separate the different fields with a comma. Each field may be renamed with a AS keyword.

FileName: if it exists, it will be overwritten

TypeOfExtract: QVD (by default), TXT. The TXT file format is in fact a CSV file (fields separated with comma)

Examples:

```
STORE Intervals INTO 'C:\Qlikview\Ecriture Documentation\Load_ENG\Intervals.qvd';
STORE Intervals INTO 'Intervals2.csv' (txt);


STORE EventID, Units as Unités FROM Events INTO 'Events.csv' (txt);
```

You cannot reduce the scope of the data exported. If you want to have a WHERE clause in order to export part of the table, you will need to

1) Load another table (Resident LOAD) with the WHERE clause

2) Export that temp table
3) Drop that temp table

4.5 IsNull / Exist functions

Handling the null:

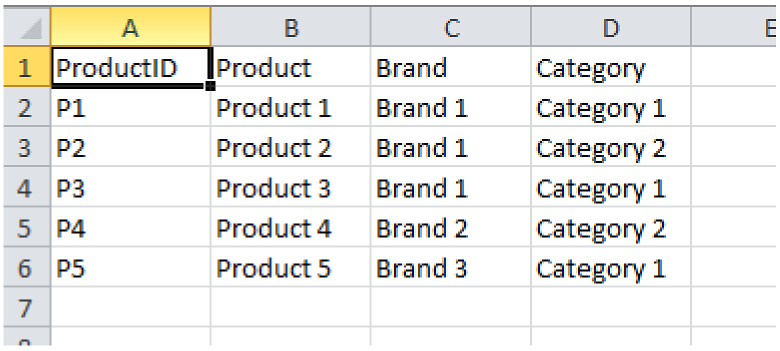
- **Isnull**(FieldValue): returns a boolean if the field value is null
- **NullInterpret** is a QlikView variable that will tell the **LOAD** how to handle some text as Null (used for text files):
  - `set NullInterpret= '<NA>';`

- **Null**(): returns null, may be used to render the code more explicit
- **if(bool expression, expression if true [, expression if false]):** returns the true or false expression according to the result of the boolean expression to get tested. The last expression is optional: **null**() is the default.

Check that the field contains a specific value

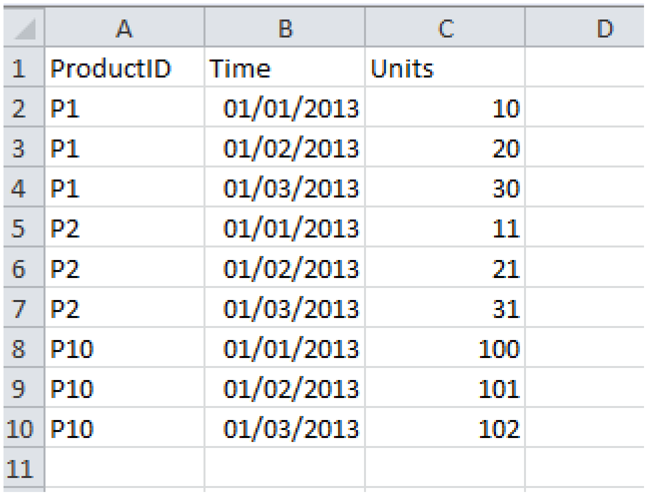
Exist(FieldName) returns true if the field contains the value

You may want to load only the values that belong already to the dimensions:



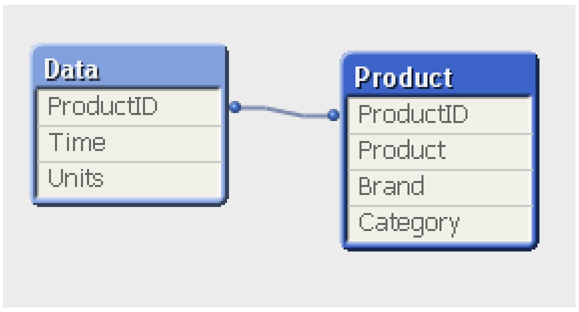
	A	B	C	D	E
1	ProductID	Product	Brand	Category	
2	P1	Product 1	Brand 1	Category 1	
3	P2	Product 2	Brand 1	Category 2	
4	P3	Product 3	Brand 1	Category 1	
5	P4	Product 4	Brand 2	Category 2	
6	P5	Product 5	Brand 3	Category 1	
7					

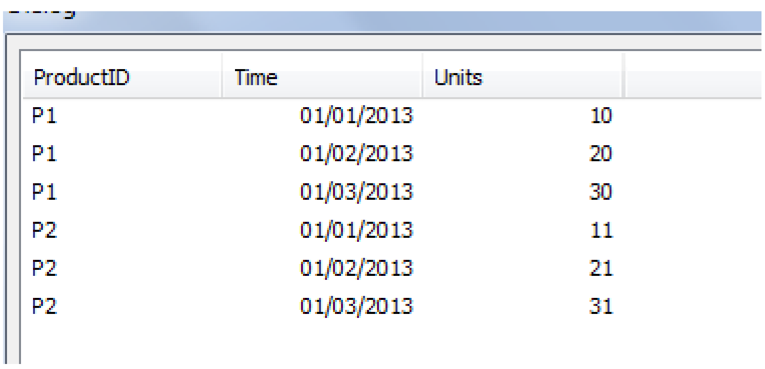
And the data may contain invalid ID for products:



	A	B	C	D
1	ProductID	Time	Units	
2	P1	01/01/2013	10	
3	P1	01/02/2013	20	
4	P1	01/03/2013	30	
5	P2	01/01/2013	11	
6	P2	01/02/2013	21	
7	P2	01/03/2013	31	
8	P10	01/01/2013	100	
9	P10	01/02/2013	101	
10	P10	01/03/2013	102	
11				

```
Product:
LOAD ProductID,
     Product,
     Brand,
     Category
FROM
Product2.xlsx
(ooxml, embedded labels, table is Dimension);

//you MUST load the dimension first to perform a valid test
Data:
LOAD ProductID,
     Time,
     Units
FROM
Product2.xlsx
(ooxml, embedded labels, table is Data)
Where exists(ProductID)
;
```





ProductID	Time	Units
P1	01/01/2013	10
P1	01/02/2013	20
P1	01/03/2013	30
P2	01/01/2013	11
P2	01/02/2013	21
P2	01/03/2013	31

4.6 Generate Random data

It can be useful to generate some random data to perform some tests. We will use:

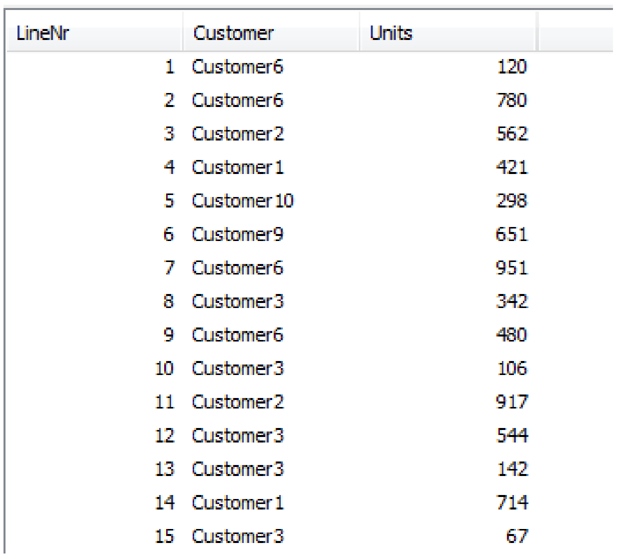
- **Rand**(): that returns a decimal number between 0 and 1, it will be most of the time multiplied with your own number
- **Autogenerate** n: a keyword that will loop n times and therefore generate n lines

For example:

```
Sales:
LOAD
 Recno() as LineNr,
 'Customer' & Ceil(Rand()*10) as Customer,
 Floor(Rand()*1000) as Units
 AutoGenerate 50000;
```

Floor() is a function that returns the integer part of the number.

Ceil() is a function that rounds the number but to the upper bound: 2.356 will be ceiled to 3.



LineNr	Customer	Units
1	Customer6	120
2	Customer6	780
3	Customer2	562
4	Customer1	421
5	Customer10	298
6	Customer9	651
7	Customer6	951
8	Customer3	342
9	Customer6	480
10	Customer3	106
11	Customer2	917
12	Customer3	544
13	Customer3	142
14	Customer1	714
15	Customer3	67

4.7 INPUTFIELD

You may want the user to enter some data (especially Budget, targeted values …) or some text. The text will be modified in the ListBoxes, the data in the tables.

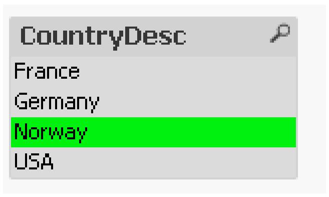
```
Syntax: INPUTFIELD Field1 [, Field 2 [, Field 3….]]
```

Notice:

- that you may use wild characters like “?” and “*”.
- That the text you want to edit MUST be unique. It is a text of a dimension, associated to a single key, not a text inside a DATA file for example.

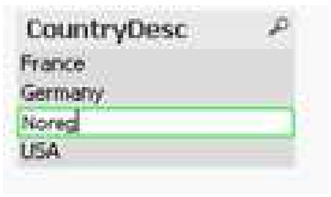
Example:

```
INPUTFIELD 'CountryD*', 'MonthDesc', 'Un??s', 'Commentaire';


TableCountry:
LOAD Country,
     CountryDesc
FROM
      InputCase.xlsx
      (ooxml, embedded labels, table is Feuil2);


TableMonth:
LOAD Month,
     MonthDesc
FROM
      InputCase.xlsx
      (ooxml, embedded labels, table is Feuil3);


Table1:
LOAD Country,
     Month,
     Units,
     Amount,
     Null() as Commentaire
FROM
      InputCase.xlsx
      (ooxml, embedded labels, table is Feuil1);
```



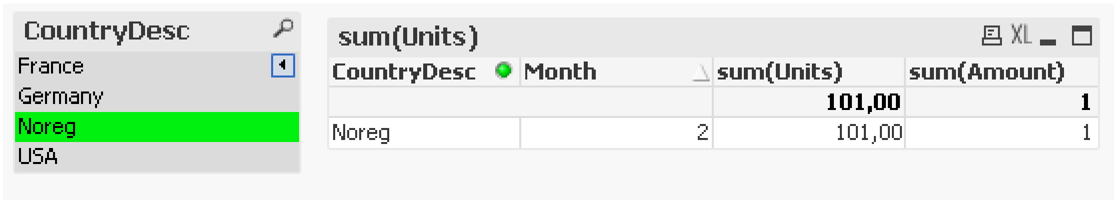
CountryDesc
France
Germany
Norway
USA

To the right of the value, appears a small arrow. By cliking on it, you will be able to edit the value.



CountryDesc
France
Germany
Noreg
USA

After having named the country Norway with one of the official languages of that country, I see the result appearing in the ListBox and also in the table:

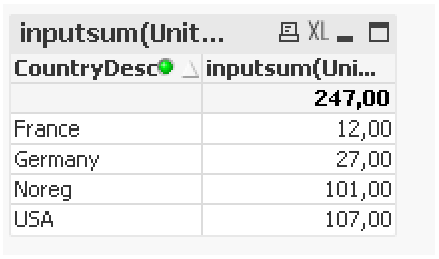


CountryDesc
France
Germany
Noreg
USA

sum(Units)

CountryDesc	Month	sum(Units)	sum(Amount)
		101,00	**1**
Noreg	2	101,00	1

Data in the table:



CountryDesc	inputsum(Uni...
	247,00
France	12,00
Germany	27,00
Noreg	101,00
USA	107,00

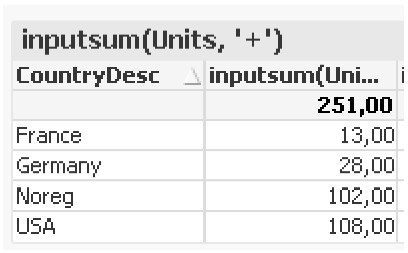
As you may see, we use the function **inputsum**() or **inputavg**() functions. These functions let you enter data at a higher level: this new data will be distributed to the children according to the second argument.

```
Syntax: inputsum(FieldName [, way to distribute new data to the children])
```

Example: inputsum(Units, '+')

Some ways to distribute data:

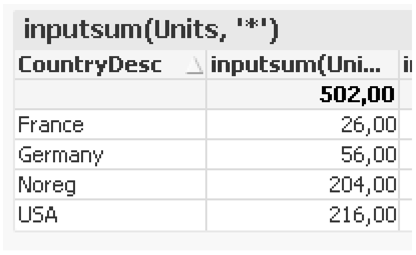
'+' : (default) the difference with the current data will be divided by the number of children. For example, in the total, if I enter 251 (difference of 4 with current 247), I will add 1 to each country (4 children)



inputsum(Units, '+')

CountryDesc	inputsum(Uni...
	251,00
France	13,00
Germany	28,00
Noreg	102,00
USA	108,00

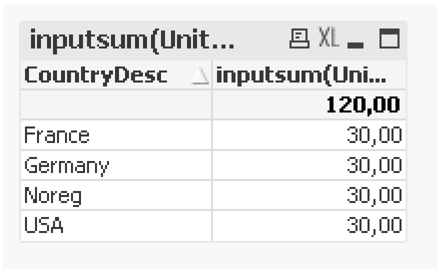
'*' : the difference with the current data will be proportionally divided by the number of children. In other words, if you add 10% to the total, you will add 10% to each of the children.



inputsum(Units, '*')

CountryDesc	inputsum(Uni...
	502,00
France	26,00
Germany	56,00
Noreg	204,00
USA	216,00

'=' : the same data will be repeated to all children (could be OK for a price, a percentage of sth). For example, here, with Units I enter 30 to the total that is duplicated to the children: but the total becomes 120.



CountryDesc	inputsum(Uni...
	120,00
France	30,00
Germany	30,00
Noreg	30,00
USA	30,00

See helpfile or reference manual for a complete explanation of the different possibilities.

When you reload the data, the data entered by the user remain !! If you want the original data from the source (file, SQL…), you need to remove the **INPUTFIELD** command.

5 Annex

I had the pleasure to write other documents that may be of interest:

LOAD Data into QlikView: http://community.qlikview.com/docs/DOC-5698

How to build a Hierarchy ListBox: http://community.qlikview.com/docs/DOC-4823

Set Analyses by example:

In English: http://community.qlikview.com/docs/DOC-4951

In French: http://community.qlikview.com/docs/DOC-4889

Simply create YTD, moving totals and comparison vs Year ago:

http://community.qlikview.com/docs/DOC-4821

Other links that you may find interesting:

Best practices for data modeling and scripting: http://community.qlikview.com/docs/DOC-4556

Documents by HIC: http://community.qlikview.com/people/hic/content