QlikView

NULL and Nothing

QlikView Technical Brief

June 2012, HIC

www.qlikview.com

QlikView

Contents

Introduction

In databases and in QlikView, nothingness is represented by the concept of NULL, i.e. a field value to show that there is no value assigned to the field in this record. Strictly speaking, NULL is not a value – it is a lack of value. Although I know this well, I sometimes call it "NULL value" anyway.

NULLs have certain basic properties:

- All QlikView fields and all data types in a SQL database are NULL-able.
- In SQL, NULL does not have a data type. In QlikView, this corresponds to the fact that NULLs are not dual, i.e. they do not have both a string representation and a numeric representation.
- NULLs propagate. If you use a NULL in an expression, it will not cause an error. Rather, it will propagate through the expression and yield a result which often – but not always – is NULL.
- NULLs cannot be used as key value to join or link tables.
- NULLs are neither visible nor clickable in QlikView, unless you make them visible and clickable using a method described below. This means that NULLs are not selectable in QlikView.

A consequence of the above is that if you select all values in list box, you will get a different result than if you select no values. In the first case you have *all values selected* and in the second case you have *all values possible*, but you may still get different results in the two cases.

The reason for this is that the first case excludes records with NULLs, which could mean that you exclude real values in other fields that potentially are used in calculations.

But there are also other types of nothingness…

Different types of Nothingness

NULL is not the only way to describe nothingness in QlikView. There are many other cases of nothingness, some which should not be confused with NULL:

Numerical zero

If a field has a numerical zero in it – 0 – then this of course represents a numerical nothingness, but it is certainly not the same as NULL. The field ***has*** a value and is hence not NULL. The IsNull() function will return FALSE and the record will be included in the calculation of both Avg() and Count().

Empty strings

A string that looks empty is another candidate for nothingness, but also this is different from NULL. It could be an empty string, i.e. a string with the length zero, or it could be a string containing whitespace. In any case, such a string ***is a string*** and not a NULL. The IsNull() function will return FALSE and the record will be included in the calculation of Count() but not in Sum() and Avg() since it is not numeric.

Note that there are several type of characters for whitespace: soft blanks (Chr(32)), hard blanks (Chr(160)) and the tab character (Chr(09)). Only soft blanks will be removed by the trim functions.

True NULLs

True NULLs are cases when an ***existing*** record in the data model has a field where the value is missing and it hence is marked as NULL. These NULLs have all the properties described in the introduction.

There are several ways that true NULLs can get into the QlikView data model:

- Databases often contain NULLs and if you use ODBC or OLE DB to load data into QlikView, you will most likely get NULLs in your application.
- If you have joins or concatenations in the QlikView script, the missing values will be converted to NULLs. (SQL “JOIN” or “UNION” and QlikView “Join” or “Concatenate”).
- Some functions, most notably Null(), return NULL and if these functions are used in the QlikView script, you will most likely get NULLs in your application.

A text file cannot contain any NULLs in itself, so unless you load it using functions that can generate NULLs, or the table is part of a concatenate or a join operation, QlikView will not have any NULLs in the table.

The IsNull() function will return TRUE for NULLs and the record will not be included in the calculation of any aggregation function, except NullCount().

Missing values – Type one: Different cardinality of key field

In QlikView there is also the concept of missing values. These are cases where the entire record is missing for a specific field value or combination of field values. Hence, there is no specific cell in the table that can be marked as NULL.

The first type of missing value is the case when a record is missing in one table (in the data model) for a specific field value existing in another table. This happens when the key field has different sets of distinct values – different cardinality – in two different data tables. To get this type of missing values, you must thus have at least two data tables.

For example, if you have a customer table and an order table, it could happen that a specific customer has not placed any orders yet. I.e. the customer is represented in the customer table but not in the order table. So the fields in the order table lack values for this customer.

Customers

CompanyName	CustomerID
Bólido Comidas preparadas	BOLID
GROSELLA-Restaurante	GROSR
Paris spécialités	PARIS

Orders

CustomerID	OrderID	OrderDate
BOLID	10326	10/7/1993
BOLID	10801	12/26/1994
BOLID	10970	3/21/1995
GROSR	10268	7/27/1993
GROSR	10785	12/15/1994

In the example in the picture, the customer with the CustomerID “PARIS” is not represented in the Orders table and values for OrderID and OrderDate are missing for this specific customer.

If you in such a situation create a chart that uses fields from both tables, the chart will for calculation purposes generate an internal virtual table – the combination of the two data tables. In this virtual table, a missing field value will be treated as NULL and the IsNull() function will return TRUE. The record will not be included in the calculation of any aggregation function using this field, except NullCount().

CustomerID	Count (OrderID)	NullCount (OrderID)
BOLID	3	0
GROSR	2	0
PARIS	0	1

Missing values – Type two: Cross table with incomplete Cartesian product

The second type of missing value occurs if you have a cross table in QlikView, i.e. a pivot table with at least one vertical dimension and at least one horizontal dimension. Then you might get a situation where a specific combination of dimensional values is not represented in the data but still has a cell in the pivot table. Both dimensional fields may even exist in the same data model table. Hence, you can potentially get this type of missing values with just one data model table and two dimensional fields.

Data

Year	Quarter	Amount
2011	Q1	120000
2011	Q2	110000
2011	Q3	130000
2011	Q4	190000
2012	Q1	125000
2012	Q2	140000

Sum(Amount)

Year	Quarter	Q1	Q2	Q3	Q4
2011		120000	110000	130000	190000
2012		125000	140000	-	-

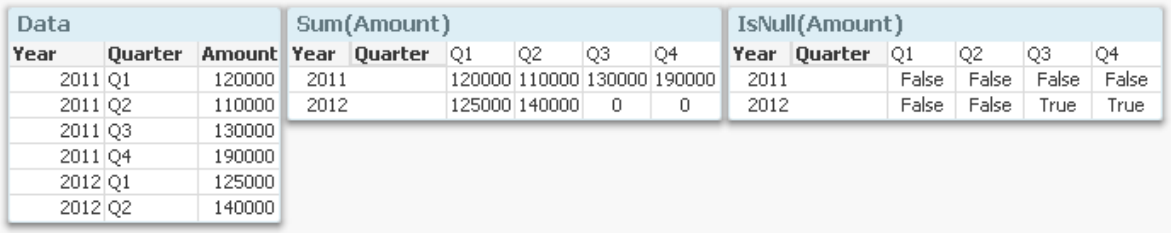
IsNull(Amount)

Year	Quarter	Q1	Q2	Q3	Q4
2011		False	False	False	False
2012		False	False	-	-

This type of missing value is also treated as a NULL when possible. It is however not always calculated. An example for this situation is if you have a data table with amounts per quarter and the table also holds a field Year.

In the example shown in the picture, there are amounts for all quarters in 2011 but not for Q3 and Q4 2012. When the pivot table is calculated, an algorithm loops over all records in the database. Since there are no data records for the last two quarters, the consequence is that the expression is never calculated for these cells (middle and rightmost tables). So basically it looks as if IsNull() has returned NULL, when it in fact has not been calculated at all.

The solution to this problem is to turn on a chart setting that populates missing cells (Chart Properties -> Presentation -> Populate missing cells). Then an internal virtual table will be generated and missing values will be populated with NULLs.



Data

Year	Quarter	Amount
2011	Q1	120000
2011	Q2	110000
2011	Q3	130000
2011	Q4	190000
2012	Q1	125000
2012	Q2	140000

Sum(Amount)

Year	Quarter	Q1	Q2	Q3	Q4
2011		120000	110000	130000	190000
2012		125000	140000	0	0

IsNull(Amount)

Year	Quarter	Q1	Q2	Q3	Q4
2011		False	False	False	False
2012		False	False	True	True

Kleenean logic

Before we look at how NULLs propagate through expressions, we need to look at Kleenean logic. (From Stephen Cole Kleene, a US mathematician.)

Most people know well what Boolean logic is: It is the algebra you use when you want to deal with concepts that are TRUE or FALSE. Using Boolean logic, you can set up truth tables to define how to propagate values through an expression. Below you see the Boolean truth tables for the OR and AND operators.

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

However, SQL databases and QlikView do not use Boolean logic. Instead, they use Kleenean logic, also called three-valued logic or ternary logic. The reason for this is that although Boolean fields are defined as having ***two*** possible values, they have in reality ***three*** possible states: TRUE, FALSE or NULL. And then you need to define how to propagate a NULL. Below you find the truth tables for the same operators, but using Kleenean logic.

OR	TRUE	MAYBE	FALSE
TRUE	TRUE	TRUE	TRUE
MAYBE	TRUE	MAYBE	MAYBE
FALSE	TRUE	MAYBE	FALSE

AND	TRUE	MAYBE	FALSE
TRUE	TRUE	MAYBE	FALSE
MAYBE	MAYBE	MAYBE	FALSE
FALSE	FALSE	FALSE	FALSE

In these tables, the third value is referred to as MAYBE, but it could just as well be denoted as “Undefined”, “Unknown”, “Indeterminate” or NULL.

Kleenean logic is a tool that helps us define how NULL should be propagated through different operators and functions. The two truth tables above (MAYBE replaced with NULL) define how the OR and AND operators work in QlikView.

The tables above show that [NULL OR TRUE] evaluates to TRUE, whereas [NULL AND TRUE] evaluates to NULL.

NULL propagation

NULL propagates in all expressions. This means that the calculation will be made although some operands or function parameters are NULL. The result is often NULL, but sometimes not. Here follows the main rules for how NULL propagates.

- Any arithmetic operation having NULL on one side will evaluate to NULL. If you use the Range functions instead, you can get an expression that disregards the NULL. Examples:
  - Null() + 5 evaluates to NULL
  - RangeSum(Null(), 5) evaluates to 5
- Any comparison having NULL on one side will evaluate to NULL. Use the function IsNull(Field) instead. Examples:
  - Field = Null() evaluates to NULL
  - IsNull(Field) evaluates to TRUE or FALSE depending on the field value
- Most numeric functions return NULL if the parameter is NULL. The Range functions are an exception. Examples:
  - Sqrt(Null()) returns NULL
  - Sqrt(RangeSum(Null())) returns 0
- String concatenation works fine if one of the operands is NULL. Examples:
  - 'xyz' & Null() evaluates to 'xyz'.
  - '' & Null() evaluates to an empty string and not to NULL

- Most string functions return NULL if the parameter is NULL. If you want an empty string instead, you can use a string concatenation inside the string function. Examples:
    - Left(Null(), 1) evaluates to NULL
    - Left('' & Null(), 1) evaluates to an empty string
    - Len(Null()) evaluates to 0, which means that NULL has zero length
    - Index(Null(), 'x') evaluates to 0, which means that the string 'x' could not be found in the first parameter. Hence the index function works normally also with NULL.
- If the condition (the first parameter) of an If() function evaluates to NULL, the ELSE expression (the third parameter) will be returned. I.e. the If() function does not necessarily return NULL just because the condition is NULL.
- An If() function where only two parameters are used will use Null() as ELSE expression.
- Aggregation functions are normally not affected by NULLs, i.e. they do not use the record with the NULL in the calculation. Except NullCount() which counts the records having NULL.
- The Only() aggregation function returns NULL if there are no or several possible values of the parameter. It returns the parameter value if there is only one possible value. This includes the case when there are several records with only one possible value, but at the same time there are some records with NULL.
- The NullCount() aggregation function returns the number of records with NULLs. When used in charts, it returns 1 also for missing values of type one (although there are no records in data) and 0 for missing values of type two.

Some tips

How to convert empty fields or blank strings to NULL?

If you have blanks or a specific symbol in a text file and you want to convert these to NULLs, you can use the QlikView variable NullInterpret:

```
Set NullInterpret = ;
```

This will convert all blank fields in a text file to NULLs. Unfortunately this does not work for empty cells in an Excel spreadsheet. Here you will need to have a different value of NullInterpret:

```
Set NullInterpret = '';
```

You can also set a specific field to NULL using the following construction. If the condition is not fulfilled, the If() function will return NULL.

```
If( Len(Trim(Field))>0, Field )
```

How to display NULLs?

NULLs are not displayed. However, there are a couple of ways that you can assign values to NULLs to make these visible.

First, if you load data from ODBC and want to display these, you can use the QlikView variable NullDisplay:

```
Set NullDisplay = '<NULL>';
```

But if your data come from another source, e.g. text files, you can use the statement NullAsValue together with a definition of a string that should replace the NULLs. Subsequent Load statements will convert the NULLs to this string.

```
NullAsValue *;

Set NullValue = '<NULL>';
```

Instead of the wild card, you can specify in which fields this should be done.

There are however several things to be careful of when using NullAsValue. The first one is when you load tables using optimized QVD loads. An optimized load will not change the data within the QVD and will hence not convert the NULLs. You will need to force QlikView to load the data unoptimized. A simple where-clause will do the trick, but I prefer to write code that is more explicit; so I remember why I did it when I look at the script a year later. My suggestion is hence:

```
Load If( Len( Trim( Field ) ) > 0, Field, '$(NullValue)' ) as NewField …
```

Another problem is that NULLs created through the Join and Concatenate prefixes will only be partially converted. The only way to make these values visible is to run a second pass through the table, using Load resident after the join or the concatenate has been performed. Example:

```
Temp:
LOAD    ...  FROM A;
Concatenate
LOAD    ...  FROM B;

Data:
NoConcatenate
LOAD * FROM Temp;
Drop Table Temp;
```

How to test for NULLs and missing values?

You cannot use the equal sign to test if a field value is NULL. You need to use the IsNull() function:

```
IsNull( Field )
```

However, this function will return FALSE for empty strings and often you want to treat empty strings and strings only containing spaces (soft blanks, Chr(32)) the same way as you treat NULLs. Then you could instead use the test

```
Len( Trim( Field ) ) = 0
```

This expression first removes leading and trailing soft blanks, then calculates its length. Soft blanks, empty strings, NULLs and missing values will all have zero length in this test.

Sometimes you want to extend the test to include also field values with any type of whitespace, i.e. also hard blanks (Chr(160)) and tabs (Chr(09)). Then you should use the following test:

```
Len( Purgechar( Field, Chr(09) & Chr(32) & Chr(160) ) ) = 0
```

A test for missing values is almost the same. All the above functions work for these also. In addition, when you test for missing values in the script, you can use the Exists() function and compare the content of the current table with what you previously have loaded. If you do not have a proper field to compare with, you can always create a temporary field that is used for this comparison only.

How to search and select NULLs or missing values?

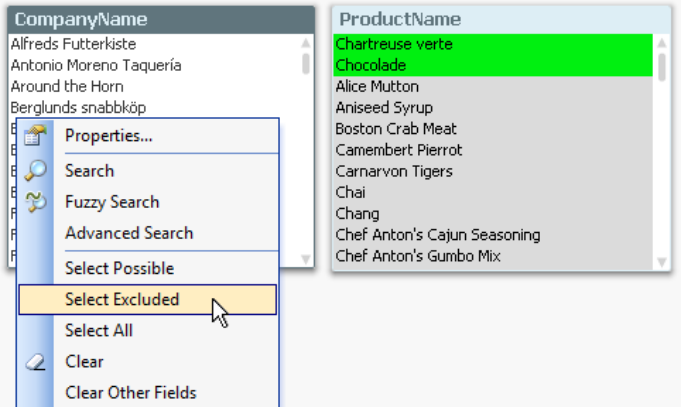
NULLs cannot be selected explicitly, so to find the records with NULLs, the selection must always be made in another field. For example, if you have a list of people and you know that some of the records lack phone numbers, i.e. the phone field is sometimes NULL, you need to select the unique person identifiers for which the phone number is NULL.

Using “Select Excluded”

The easiest way to find values that lack an attribute is to make a selection in two steps:

- First select *all* phone numbers (right-click the phone numbers and choose Select All, or search using a star as a wildcard: “*”).
- Then select the *excluded* person identifiers (right-click the person identifiers and choose Select Excluded).

This method works for both NULLs and missing values of type one. It can also be used for similar questions like “Which customers have *not* bought a certain product?” Just select the product and then select the excluded customers.



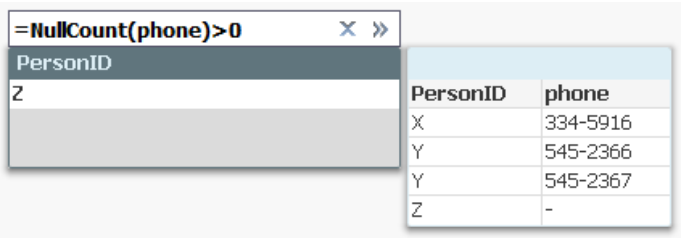
If your users make such a selection often, it may be a good idea to put the second step, the selection of excluded values, in a button. Label it clearly and use the action “Select Excluded”. This simplifies a lot for the user.

Using advanced search

Another approach is to make an advanced search. Then the user can explicitly search for a person having records that lack phone numbers. Mark the list box with the person identifier and enter the following search expression:

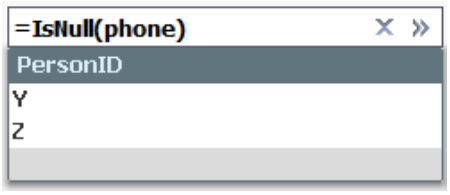
'=NullCount(phone)>0'

This will explicitly find people that have one or several records with NULL in the phone field. In the example in the picture, the search result correctly displays only the person “Z”. The real data is shown in the table box to the right and looking at it, you clearly see that only Z has NULL as phone number.



This method works for both NULLs and missing values of type one.

One problem with this approach is that it is not obvious which expression to use. Most people would probably use '=IsNull(phone)'. However, in my view *this expression is not correct*: it sometimes yields the wrong result. This can clearly be seen in the example in the picture below. The search result displays two people; “Y” and “Z” when only “Z” has NULL as phone number:



To understand this, you must first understand that there potentially may exist several phone numbers per person. Hence, QlikView must use an aggregation function to evaluate the expression. In fact, in advanced searches, QlikView *always* uses some aggregation function. If there is no explicit aggregation function in the expression, QlikView assumes the use of the Only() function. Thus, QlikView uses '=IsNull(Only(phone))' in the search.

The Only() function returns NULL for two different cases: First, when there is no value, and secondly, when there are several possible values.

Hence, if there is only one record per person, this expression will correctly return TRUE for people that have no phone number and FALSE for those who have a phone number. But for people that have several records, the expression may return an unwanted value: If a person has two different phone numbers, the expression will return TRUE, which is not what you want.

Further, if a person has two records, one with NULL and the other with a phone number, the Only() function will return the phone number and the expression will return FALSE. Hence, this search string will not find such a person, although you probably want it to.

Unfortunately, there is a widespread misconception that the search string '=IsNull(Field)' should be used as advanced search. My recommendation is instead to use the NullCount() aggregation function in the following search expression:

```
'=NullCount(Field)>0'
```

If your users make such a selection often, it may be a good idea to save the advanced search in a bookmark. The bookmark will remember the advanced search and perform this every time that it is applied. This simplifies a lot for the user.

Create a field in the data model that allows selecting excluded values

A third possibility is to prepare logic for selecting NULLs and missing values in the script, i.e. by creating fields that hold the information necessary.

For the missing phone numbers (true NULLs), you could add a Boolean field that indicates whether the record has a phone number of not:

```
People:
LOAD   PersonID,
       If( Len(Trim(Phone))>0, 'Yes', 'No') as [Has phone],
       ... FROM People;
```

For the customers that have not placed any orders (missing values type one), you could add a Boolean field that indicates whether the customers has placed any orders or not (whether the key is represented in another table):

```
Orders:
LOAD   CustomerID,
       ... FROM Orders;

Customers:
LOAD   CustomerID,
       If( Exists(CustomerID), Dual('Yes', True()), Dual('No', False()) ) as [Has orders],
       ... FROM Customers;
```

The Exists() function compares the value of CustomerID in the Customers table with all previously loaded values of this field. If the value has been loaded before, the Exists() function returns TRUE; if not, it returns FALSE. Note that in order for this to work, the Orders table must be loaded *first*.

The Dual() functions are not necessary, but by using them the created field can be used directly as a Boolean flag in conditions.

This approach has the advantage that it is obvious for the user how to find the missing values and at the same time all other search possibilities are maintained.

NULLs and missing values in Set analysis

Most of the above is applicable also in set analysis.

In set analysis, searches are made using double quotes. Also advanced searches can be made, and a search for NULLs can be made using the logic described above. Hence:

```
Concat({$<PersonID={"=NullCount(phone)>0"}>} distinct PersonID)
```

evaluated on the data in the example above in the paragraph “Using advanced search” will evaluate to ‘Z’, exactly as in the example in this section, i.e. the people that do not have a phone number.

Empty element sets, either explicitly e.g. <Product = {}> or implicitly by a search with no hits <Product = {"Perpetuum Mobile"}>, mean *no product*, i.e. they will result in a set of records that are not associated with any product.

Note that the set modifier <Product = > is not the same as <Product = {} >. The former merely removes the existing selection in the field, whereas the latter returns an empty set.

Set operators

In set analysis, set operators can be used to find the complement to a selection, i.e. the excluded records. For example, the set expression

{$<OrderID={"*"}>}

will pick out *all* possible OrderIDs – but not the NULLs – and consequently

{1-$<OrderID={"*"}>}

will return the complement: it will pick out all customers that have an empty OrderID set, i.e. that have not placed any orders.

Implicit field value definitions

Finally, in set analysis it is also possible to use the implicit field value definitions P() and E(), which returns the set of possible values and excluded values, respectively. The E() set function is especially useful. Example:

{$<Customer = E({1<Product={"*"}>})>}

This expression will pick out the customers that are excluded when all products are selected, i.e. the customers that have not bought anything.

Further:

{$<Customer = E({1<Product={'Shoe'}>})>}

This specific expression will pick out the customers that are excluded when the product “Shoe” is selected, i.e. the customers that have not bought any shoes.

HIC