QlikView

# AGGR

Explaining the AGGR function

## QlikView Technical Brief

March 2013

qlikview.com

QlikView

# Contents

QlikView

# Introduction

QlikTech is a leader in Business Discovery — user-driven Business Intelligence (BI). Our QlikView Business Discovery platform is what's next in business software. Today's business leaders want to enable users at all levels of the organization to leverage data to drive innovative decisions that push the business forward. QlikView puts those business users in control of exploring and exploiting their data without limits by delivering these capabilities:

**Insight Everywhere** — Business Discovery is a whole new way of doing things that puts the business user in control. Unlike traditional BI, where just a few people are involved in insight creation, Business Discovery enables everyone to create insight.

QlikView

# AGGR Defined

AGGR is a very powerful aggregation function that is sometimes overlooked in the user interface due to is not being properly understood or indeed a developer not being sure how it can be utilised. Often, a QlikView developer will revert to more complex scripting or pre-aggregating data to service an expression that is required in a chart, which can actually be solved by using AGGR.

You will see in the examples later in this document that AGGR can be used in both Expressions and Dimensions.

If we were to provide a short description of AGGR it would be -
***When it is used, the AGGR statement produces a virtual table, with one expression and grouped by one or more dimensions. The contents / result of this virtual table can then be used / aggregated by a further outer aggregation function(s).***

The QlikView help provides the following definition of AGGR (Advanced Aggregation):

## Advanced Aggregation

There is a special function for advanced aggregations:
aggr ([ distinct | nodistinct ] [{set_expression}]expression {, dimension})

Returns a set of values of expression calculated over *dimensions*. The result can be compared to the expression column of a 'local chart', evaluated in the context where the aggr function resides. Each dimension must be a single field. It cannot be an expression (calculated dimension).

If the expression argument is preceded by the nodistinct qualifier, each combination of dimension values may generate more than one return value, depending on underlying data structure. If the expression argument is preceded by the distinct qualifier or if no qualifier is used at all, each combination of dimension values will generate only one return value.

By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a *Set Analysis* expression.

By using this function in *Add calculated dimension...* it is possible to achieve nested chart aggregation in multiple levels. See also *Nested Aggregations and Related Issues*.

When used in chart expressions it is possible to achieve *Sum of Rows in Pivot Tables*.
Examples:
aggr( sum(Sales), Country )
aggr( nodistinct sum(Sales), Country )
aggr( sum(Sales), Country, Region )
count( aggr( sum(Sales), Country ))

QlikView

# AGGR Expressions

This first example will show you how to use a simple AGGR statement in a chart expression.

In this scorecard example, we want to be able to display our overall metrics at the country level, but also show two other more complex expressions.

- Largest average order value (by salesperson) for the specific dimension (in this case largest order value within each Country)
- Salesperson responsible for that largest order value

**Total Sales**

| Country | Total Sales | Largest Order | Avg Order Value | Largest Avg. Ord. Value | Salesperson Responsible |
|---|---|---|---|---|---|
| | **£3,950.00** | **£203.00** | **£109.72** | **£202.50** | **Barbara** |
| UK | £1,046.00 | £113.00 | £87.17 | £108.00 | Barbara |
| USA | £2,030.00 | £203.00 | £169.17 | £202.50 | Barbara |
| France | £874.00 | £99.00 | £72.83 | £97.50 | Bob |

We can easily calculate the Average Order Value for each country using a standard QlikView expression ( sum(Sales)/count(Order) ). However, we need to retrieve the LARGEST average order value for each country in the Dimension field. We therefore have to use AGGR to enable us to pre-calculate / or indeed use some nested aggregation.

So, firstly we will have to calculate the simple average order value….

**sum(Sales)/ count(Order)**

Next we need to try to find the Average order values for the individual salespeople within each country. To do this we will need to perform some nested aggregation / pre-calculation. Essentially we need to tell QlikView that we want to grab the Average order value for all of the salespeople within each country then display the largest of those.

To get the Average Order Value for the salespeople within each country we will have to put those dimensions within our AGGR statement….

**Aggr(sum(Sales)/ count(Order), Country, Salesperson)**

Using the expression above, if you imagine that QlikView is producing a table internally that gives you……

**Avg Order Value By Country**

| Country | Salesperson | Avg Order Value By Country |
|---|---|---|
| | | - |
| UK | Sue | 55 |
| UK | Fred | 84 |
| UK | Maureen | 85 |
| UK | Bill | 92 |
| UK | Bob | 99 |
| UK | Barbara | 108 |
| USA | Bob | 105.5 |
| USA | Bill | 152 |
| USA | Sue | 179 |
| USA | Maureen | 185 |
| USA | Fred | 191 |
| USA | Barbara | 202.5 |
| France | Sue | 29 |
| France | Barbara | 76 |
| France | Maureen | 77 |
| France | Bill | 78.5 |
| France | Fred | 79 |
| France | Bob | 97.5 |

Now QlikView has calculated the individual Average order values for each Salesperson in each Country, we now want to grab the largest of those. We do this by adding the MAX to the front of the expression.

**MAX(Aggr(sum(Sales)/ count(Order),Country, Salesperson))**

This expression will now let us remove the Salesperson dimension from our table and show the largest Avg Ord Val for all our salespeople at just the country level….

**Avg Order Value By Country**

| Country | Avg Order Value By Country |
|---|---|
| | **202.5** |
| UK | 108 |
| USA | 202.5 |
| France | 97.5 |

We have now added the first of our two complex expressions. The next requirement is to have the name of the salesperson responsible for these large Avg Ord Val displayed next to the values themselves.

To enable us to place the name of the salesperson, we will again need to utliise the AGGR function but also in conjunction with the FIRSTSORTEDVALUE() function.

**firstsortedvalue(***[{set_expression}][ distinct ] [ total [<fld {, fld}>]] expression [, sort_weight [, n]]***)**

returns the first value of *expression* sorted by corresponding *sort-weight* when *expression* is iterated over the chart dimension(s). *Sort-weight* should return a numeric value where the lowest value will render the corresponding value of *expression* to be sorted first. By preceding the *sort-value* expression with a minus sign, the function will return the last value instead. If more than one value of expression share the same lowest *sort-order*, the function will return null. By stating an n larger than 1, you will get the nth value in order.

**Examples:**

```
firstsortedvalue ( PurchasedArticle, OrderDate )
firstsortedvalue ( PurchasedArticle, -OrderDate, 2 )
firstsortedvalue ( A/B, X*Y/3 )
firstsortedvalue ( distinct PurchasedArticle, OrderDate )
firstsortedvalue ( total PurchasedArticle, OrderDate )
firstsortedvalue ( total <Grp> PurchasedArticle, OrderDate )
```

We will be using the same average order value calculation as before…

**aggr(sum(Sales)/count(Order),Country,Salesperson)**

Now, we need to tell QlikView that we want to grab the Salesperson for each of the largest Order Values. The FIRSTSORTEDVALUE function will let us do this.

The FIRSTSORTEDVALUE function tells QlikView to provide us with the **Salesperson**, for the specific Dimension or expressions specified in the **second portion of the function**. There is one small, very important, part of the expression. There is a minus symbol before the AGGR statement/expression. Within a FIRSTSORTEDVALUE function, you can specify the sort order of the array of data. In this case, **the minus symbol** tells QlikView to sort Descending (Largest to Smallest).

**FirstSortedValue(Salesperson,-aggr(sum(Sales)/count(Order),Country,Salesperson))**

QlikView

## AGGR & SUM(IF…)

We can take the use of AGGR in expressions a little further. We will stick to using the same Average Order Value Calculation for this first example.

Imagine I need to understand how many, if any, of our Salespeople have average order values of less than $100. I also want to show this in a text object.

Ok, so again, we need to first calculate the individual Average Order Values for each Salesperson in each Country. As per the previous section of this document you will use the AGGR statement with Country and Salesperson as the Dimensions to calculate over….

**aggr(sum(Sales)/Count(Order),Country,Salesperson)**

Now we have our familiar AGGR statement we can add a simple SUM and IF statement to enable us to count the number of Salespeople by Country, that have an average order value of less than $100. If you take a look at the table below, the IF statement will place a 1 next to every underperforming Salesperson (<100) and then adding a SUM to the expression will tell QlikView to sum the 1's up and enable us to display this number in a text object as you can see below.

**=sum(if(aggr(sum(Sales)/Count(Order),Country, Salesperson)<100,1,0))**

Example….

**Avg Orders <100 By Salesperson & Country**

11

**Avg Ord Val**

| Country | Salesperson | Avg Ord Val | if(aggr(sum(Sales)/Count (Order),Country, Salesperson)<100,1,0) |
|---|---|---|---|
| | | **110** | **0** |
| France | Sue | 29 | 1 |
| UK | Sue | 55 | 1 |
| France | Barbara | 76 | 1 |
| France | Maureen | 77 | 1 |
| France | Bill | 79 | 1 |
| France | Fred | 79 | 1 |
| UK | Fred | 84 | 1 |
| UK | Maureen | 85 | 1 |
| UK | Bill | 92 | 1 |
| France | Bob | 98 | 1 |
| UK | Bob | 99 | 1 |
| USA | Bob | 106 | 0 |
| UK | Barbara | 108 | 0 |
| USA | Bill | 152 | 0 |
| USA | Sue | 179 | 0 |
| USA | Maureen | 185 | 0 |
| USA | Fred | 191 | 0 |
| USA | Barbara | 203 | 0 |

QlikView

## AGGR & Rank()

AGGR can also be utilised with the rank function. Imagine you have a large amount of customers and you want to understand the top 3 sales rep's within a text object as you can see below.

**Top Selling Reps**

Barbara
Fred
Maureen

To achieve this we can use the Concat() and Rank() functions in conjunction with our AGGR statement. The completed expression is below.

**=concat(distinct IF(aggr(rank(sum(Sales)),Salesperson)<=3,Salesperson&CHR(13)))**

We will now show how the expression is constructed.

First we need to calculate our Sales:

**Sum(Sales)**

As a text object has no concept of dimensionality i.e. no dimensions to aggregate the expression over, we need to provide the expression with the dimension that we want to calculate at. As we have stated already, we want to show the top 3 Salespeople. Therefore, the AGGR statement must be used to tell QlikView we are calculating Sales at a salesperson level.

If you remember our original definition of AGGR, using the statement will create a virtual table, in this case it would look like the one below.

**aggr(sum(Sales),Salesperson)**

| Salesperson | sum(Sales) |
|---|---|
| | **3950** |
| Barbara | 773 |
| Fred | 708 |
| Maureen | 694 |
| Bill | 645 |
| Bob | 604 |
| Sue | 526 |

We then need to rank these Sales. The AGGR statement will get us our sales grouped by Salesperson, we can then add in a Rank() function to get the AGGR statement to return the Salespeople by Rank.

**aggr(Rank(sum(Sales)),Salesperson)**

| Salesperson | rank(sum(S… |
|---|---|
| | - |
| Barbara | 1 |
| Fred | 2 |
| Maureen | 3 |
| Bill | 4 |
| Bob | 5 |
| Sue | 6 |

The next step is to add in our IF statement to request that we only receive the top 3 people in the result. So, if the rank in our virtual AGGR table is <=3 then show the salesperson else null().

**IF(aggr(rank(sum(Sales)),Salesperson)<=3,Salesperson)**

Finally, we want to present the results of this together in a text object, not in a table/chart. We will need to therefore use the CONCAT() function to string the 3 Salespeople values together. There is also a clever trick to add a carriage return in to the CONCAT to place the values on their own rows in the Text Object. At the end of the statement you will see a CHR(13).

**=concat(distinct IF(aggr(rank(sum(Sales)),Salesperson)<=3,Salesperson&CHR(13)))**

Top Selling Reps

Barbara
Fred
Maureen

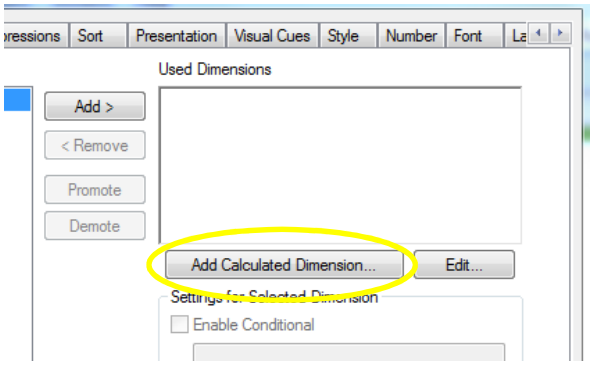
QlikView

# AGGR in a Dimension

The AGGR statement can also be utliised in a calculated dimension. In this example we want to create a dimension that is based on an aggregation.

This is a good example from the pharmaceutical industry. Sales reps visit physicians regularly to showcase and sell their new products. In this case, the company wanted to analyse the Sales by physicians, for those customers visited once, visited twice, etc.

To enable us to use AGGR in our dimension, we need to create a “Calculated Dimension” in our chart.




A dimension in any chart, needs to be a set/array/series of values to plot our expression over. And in this case, we want to plot the number of times Physician Visits have occurred. As we have mentioned earlier in this document, a QlikView developer would often perform some pre-aggregating of data in the script to achieve this.

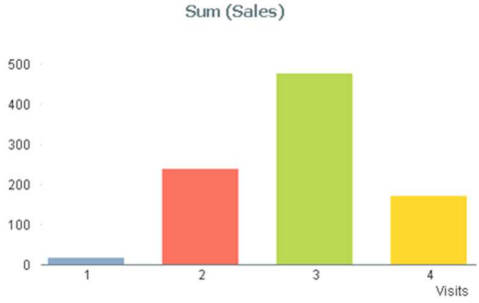
Using the AGGR statement in our dimension lets us aggregate (count in this case) the total Visits (VisitID) by each Physician (PhysicianID) and then plot the result of this on the axis of the bar chart. If we didn’t utilise AGGR the chart would return an error like below.

| =Count(Name) | sum(Value) |
|---|---|
| | **3590** |
| // Error in calculated dimension | 3590 |

Using the AGGR statement below, we can provide the bar chart with our series of values to plot our sales (Sum(Sales)) expression over.

**Aggr(Count(VisitID),PhysicianID)**

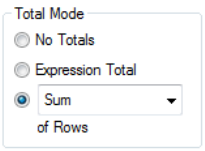
Our final chart…



QlikView

# Sum of Rows in a Pivot Table

When using QlikView straight tables, we are able to define the total mode for any of our expressions.




This total mode option is not available when working with Pivot Tables. However, it can be achieved using the aggr() function. It is especially valuable in a pivoted PivotTable (CrossTable) with more than one dimension.

In this example we have three displays/iterations of the same data. In the first straight table you can see a dimension and three expressions. The expressions are Plan, Actual, and Acc. Difference (the absolute difference between the Plan and Actual columns). In the total row, we want to have a sum of rows occurring.

Our initial Acc. Difference Expression look like this:

**fabs(sum(Planned-Actual))**

You can see in the image below. Using a Straight table, the Acc. Difference column presents 68. The correct sum of the rows.

Straight Table

| Region | Plan | Actual | Acc. Difference |
|---|---|---|---|
| 1 | 43 | 27 | 16 |
| 2 | 35 | 87 | 52 |
| | **78** | **114** | **68** |

However, if we wanted to display this data in a pivot table (as per the image below), the option to define the Total mode is not available to us. And the default display for the total row would be the “Expression Total”, in this case, showing 36.

Pivot Table

| Region | Plan | Actual | Acc. Difference |
|---|---|---|---|
| 1 | 43 | 27 | 16 |
| 2 | 35 | 87 | 52 |
| **Total** | **78** | **114** | **36** |

Using the aggr() function, it is possible to nest the inner basic aggregation with an outer aggregation, thus creating a nested aggregation that will be similar to the Sum of Rows option available in the StraightTable. In this specific example we have added the AGGR statement and provided the dimension of Region that is present in our table.

**aggr(fabs(sum(Planned-Actual)),Region)**

We then add the SUM() function to tell QlikView to sum the rows within.

```
sum(aggr(fabs(sum(Planned-Actual)),Region))
```

Pivot with "Sum of Rows"

| Region | Plan | Actual | Acc. Difference |
|---|---|---|---|
| 1 | 43 | 27 | 16 |
| 2 | 35 | 87 | 52 |
| **Total** | **78** | **114** | **68** |

## Conclusion

As you can see, AGGR is a very powerful function that can be utilised to create a more complex expressions and negate the need for “extra” scripting or pre-aggregating in the script. We have shown you a number of common scenarios where AGGR can help you solve a problem or indeed provide you with better analysis.