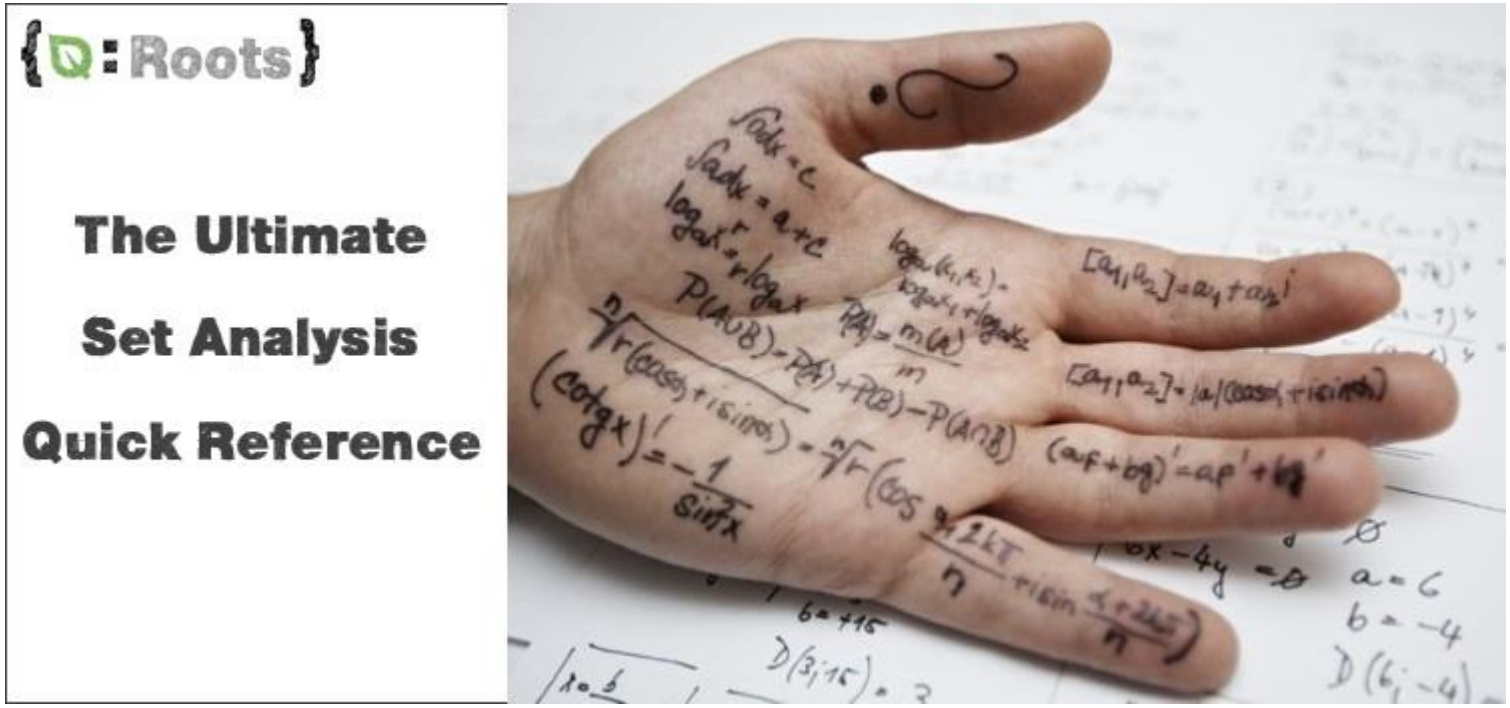
LivingQlik Roots: The Ultimate QlikView Set Analysis Reference

April 25, 2017
Aaron Couron



QlikView Set Analysis – A Scary Subject

Set Analysis is a scary subject. I would surely claim it is the most difficult part of developing your QlikView front-end.

I have waited to throw my hat in the ring regarding set analysis. This is mostly because I didn't feel I had much to add, but also because Set Analysis in QlikView is a formidable subject.

But the time has come for LivingQlik to cover Set Analysis. Firstly, no self-respecting blog would be complete without tackling the subject. Secondly, I feel that there is room for a compete reference presented in a clear voice. Can LivingQlik be that voice? Maybe?

This is the ultimate cheat sheet for Set Analysis in QlikView. It includes the syntax definitions and options, many examples and tons of tips and tricks to make the most out of this complex topic.

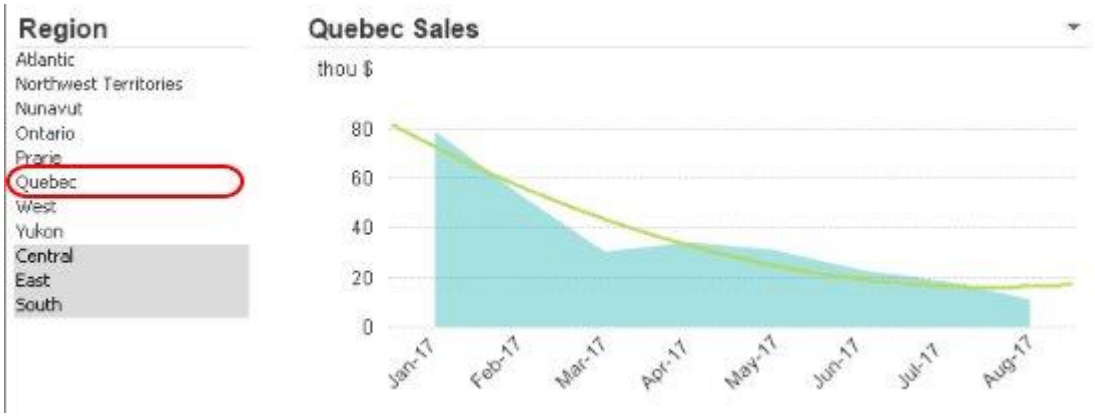
One thing I tell students is to not beat themselves up if they don't absorb everything there is to know about Set Analysis in one sitting. It is a difficult subject for all skill levels, from newbie to experienced. I know I still run into situations that make me scratch my head and I have been doing this for a while.

What is Set Analysis?

The purpose of Set Analysis is simple.

Set Analysis enables a Qlik aggregation expression to be calculated with a set of selections that differ from the user's selected values.

When we use an expression with set analysis inside a chart, the chart will be calculated based on the data set within the expression rather than the green selected values and white associated values that would normally filter the chart.



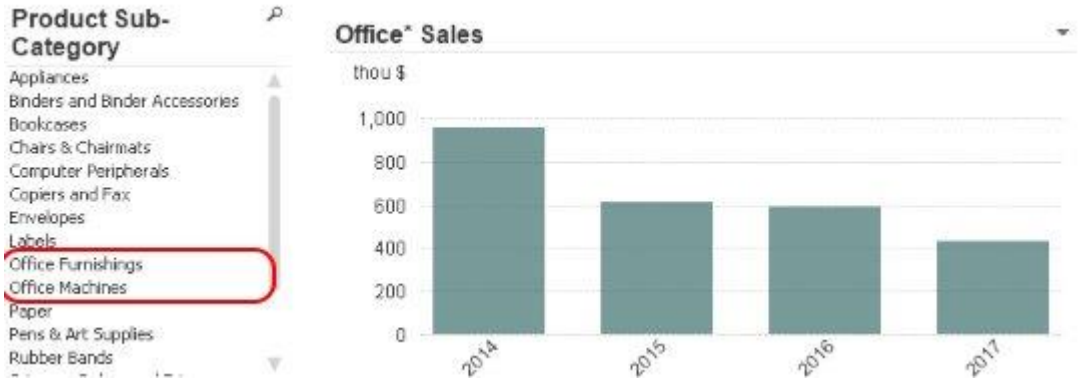
This line chart shows Quebec sales even when Quebec is not selected

Sales YTD vs. LYTD

Sales 2017 YTD	Sales 2016 YTD	Var %
2,378,281	2,041,607	16.5%

Comparing points-in-time would be difficult without set analysis.

Here we look at a month vs the same month last year.



Looking at Sub-Categories that begin with the word “Office”

How Does Set Analysis Work?

Admittedly, the syntax for QlikView Set Analysis is not easy. Here, I will lay it out as plainly as possible. Where there are several options, I have attempted to highlight the more common use-cases.

Set Analysis is always enclosed in curly braces and must exist within an aggregation like sum(), avg(), max(), min(), or only(), for example. In fact, Set Analysis is the *only* place where curly brackets are utilized in Qlik syntax.

Sum up the sales for all data excluding the current selections limited to the Quebec Region.

The syntax for Set Analysis is broken out kind of like grammar rules break out a sentence. Some say there are 3 parts but I detail 4 (Element List is really part of the modifier but this is where much of the "action" takes place):

- **Identifier** – the selection state established for your set
- **Operator** – Method for combining two or more identifier-modifier sets together
- **Modifier** – Assigns specific field values to your set expression
- **Element List** – Within the modifier, the specific selections assigned to your modifier field

Identifier – the selection state established for your set

An identifier is an optional prefix that determines the starting selection state of our set. Identifiers get applied to the modifiers they precede.

The default identifier is the dollar-sign {$} which tells the expression to respect and apply the user's current selections *in the default state* before evaluating the set. Because developers are usually using the default state of the data model, developers often leave the dollar sign out since we are already in the default state. If your object or sheet is responding to another state, and you actually want the expression to reference the default state, then the dollar-sign would be needed. Omitting the identifier just means we are responding to the state of the object or sheet following normal inheritance rules.

$
Respects user selections in the default state of the object
1
All the data (ignores all user selections)
BookmarkID
The selections that would be invoked by a bookmark
Alternate State ID
The selections from an alternate document state
$1
Previous Selection (like hitting back) (ignores all user selections)
$2
2nd previous selection
$_1
Forward Selection
$_2
2nd forward Selection
0
Empty Set

```
avg({1}m_sales)
```

Average of sales for all data in the application (ignore all user selections).

is

```
max({$}m_sales)
```

the same as

```
max(m_sales)
```

?

When working in the default state we often omit the $ as an operator as it is redundant. But be careful about omitting the identifier when working with other states.

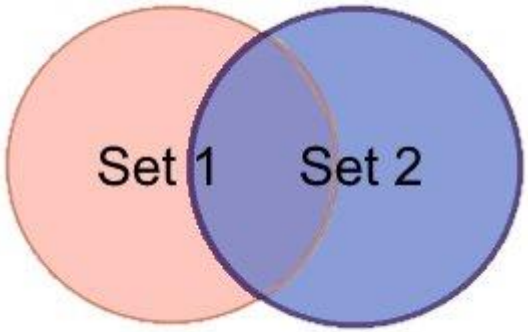
Operator – Method for combining two or more identifier-modifier sets together

Operators are used to join two identifier-modifier sets. Operators can also be used inside a modifier element (more on this in the Element List section). There are only 4 variations.

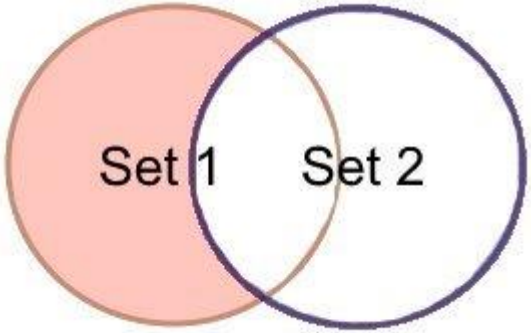
Operator Definition

\+

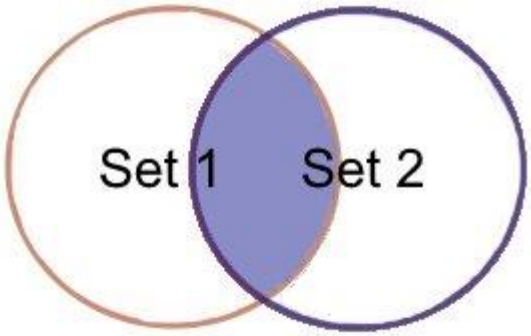
unions two sets together



–
exclusion of second set from the first

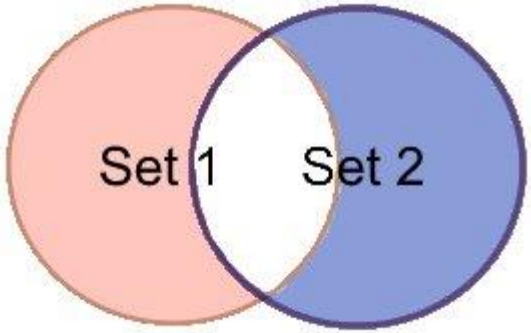


*
intersection of two sets



/

symmetric difference (values that belong to either but not both sets)



Identifiers and operators can be used to create some very interesting sets that might (or might not) have value to you. I find that I use the intersection operator more often than any of the others, but in all reality, the sets I require generally do not require an operator.

```
sum({<Region={'Ontario'},[Product Category]={'Furniture'}>+
    <Region={'Nunavut'},[Product Category]={'Technology'}>}m_profit)
```

Sum of profit for Region Ontario and Product Category Furniture but also including profit for Region Nunavut and Product Category Technology.

```
sum({1-$}m_profit)
```

Sum of profit for all the data in the app excluding the current user selections.

```
sum({<Year={$(=max({1}year(Date_Field)-1))}>*
    <Year=,Date_Field={"<=$(=addmonths(max({1}Date_Field),-12))"}>}m_profit)
```

Sum of profit for last year but only for the period less than today one year ago. This is profit LYTD.

```
sum({<Manager={'Chris','Erin'}>/<Year={2017}>}m_profit)
```

Sum of profit for Chris and Erin or for Year 2017, but not for Chris and Erin in 2017.

Modifiers – Assigns specific field values to your set expression

A modifier gives you a way to assign specific values to fields when determining your set. The modifier is always enclosed by angle brackets: <>. Here you will list field names, each followed by an equal sign and then an element list. The modifier is where things start to get interesting.

```
sum({<Region={'Quebec'}>}m_sales)
```

Sum of sales for Quebec Region.

Within a modifier you can:

…assign values to multiple fields separated by commas:

```
sum({<[Ship Mode]={'Express Air'},Region={'Prarie'}>}m_sales)
```

Sum of sales for Ship Mode Express Air in Region Prarie.

…assign a field to the same field in another state

```
sum({<Manager=Group1::Manager>}m_sales)
```

Sum of sales transferring the Manager selections in Group1 to the default state.

Element Lists – The specific selections assigned to your modifier field

The portion of the modifier after the equal sign is called the element list. Values listed or evaluated here become specific selections for your set evaluation. The element list is surrounded by curly braces and should not be confused as an inside or nested set. We can do many awesome things with the element set and it is often where I spend the most time when designing Set Analysis expressions.

Within an element list you can:

…assign more than one value to a field separated by commas

```
sum({<[Ship Mode]={'Express Air','Regular Air'}>}m_sales)
```

Sum of sales for Ship Modes Express Air and Regular Air.

…utilize wildcard searches

```
sum({<[Product Name]={'Avery*'}>}m_sales)
```

Sum of sales where product name begins with "Avery".

For element values, it is best to use single quotes to identify a string although if the text does not contain spaces or other special characters, you can get away with not using them. When expecting numeric values, no quotes are necessary. Be advised that date fields can be tricky and will conditionally require quotes based on the evaluation and format of the field.

```
sum({<Year={2017}>}m_sales)
```

A field used inside a modifier overwrites the user's selection for that field. Therefore we can do things like "ignore the user selections for this field". But there is some nuance in how you should approach this. For example, the two following examples will treat the Manager field slightly differently and *could* result in different calculations:

```
sum({<SalesPerson={'*'}>}m_sales)
```

Sum of sales respecting all user selections and selecting all possible (non-null) values for the field SalesPerson.

```
sum({<SalesPerson=>}m_sales)
```

Sum of sales respecting all user selections ignoring user selections in the SalesPerson field.

Searching for the Element List Values

One way of developing your element list is to search for it. Enclose your advanced QlikView search syntax inside double quotes within your element list. This is especially helpful to limit your set to a date range for example.

```
sum({<Date_Field={">$(=date(max(Date_Field)-30))"}>}m_sales)
```

Sum of sales for the last 30 days from the maximum current date selection.

Here we have closed the range.

```
sum({<Date_Field={">=$(=date(today()-31))
        <=$(=date(today()-1))"}>}m_sales)
```

Sum of sales for the last 30 days measured from yesterday backwards

You can define a calculation to search for values.

```
avg({<Region={"=rank(avg(m_sales))=1"}>}m_sales)
```

Avg of sales for the dimension value that ranks number 1

Or even assign a field to another field.

```
sum({<Date_Field={"=Date_Field=[Ship Date]"}>}m_sales)
```

Sum of sales for Date_Field where the Date_Field is the same as the Ship Date (same day shipping)

P() and E() – Special Elements

The P() and E() special elements return the element set that is possible or excluded (respectively) from a field or expression. I don't have requirements that call for the possible or excluded functions very often.

```
sum({<Manager=p({1<[Product Sub-Category]={'Labels'}>}Manager)>}m_sales)
```

Sum of sales respecting current selections for the Managers that have ever sold Product Sub-Category Labels.

And an example with e():

```
sum({<[Product Name]=e({<Year={$(=(max({1}Year)-1))}>}[Product Name])>}m_sales)
```

Sum of sales respecting current selections for the Product Names that were not associated (not sold) last year.

Passing States into the Modifier

Another interesting technique involves alternate states you might have in your document. We have already seen the possibility of using these states as operators. But we can also use them in the modifier to pass the selections in a field from one state to another.

```
sum({<Region=Group1::Region>}m_sales)
```

Sum sales in the current inhereted state passing in the selected values from the Group1 state.

Dynamic Elements – Dollar-Sign Expansion

Dollar-sign expansion is a frequently used feature of Set Analysis. Think of the example where you want to display sales for this year-to-date. I could just type whatever the current year is into the modifier element list, but then I would have to come back to this expression on January 1st and change the year.

Thankfully, you will not need to hard-code a year and then change to the next year every January. Using the concept of Dollar-Sign Expansion, you can dynamically assign the value to the modifier element.

Syntax inside the dollar-sign expansion will be evaluated once for the entire chart before the rest of the expression is calculated. This is usually great. But because of this, you cannot leverage set analysis to do row-by-row evaluations, expecting the set to change for each dimensional value. For those occasions, use an if statement.

Here are some examples of some point-in-time sets using dollar-sign-expansion. These are examples of how I do it, but there are many ways to "skin the cat" and different approaches have different advantages.

```
sum({<Year={$(=max({1}Year))}>}m_sales)
```

Sum of sales for the maximum year in the data.

```
(sum({<Year=,MonthYearID={$(=max(MonthYearID))}>}m_sales)/
sum({<Year=,MonthYearID={$(=max(MonthYearID)-12)}>}m_sales))-1
```

Sum of sales for the maximum month in the data vs. the same month in the prior year.

```
(sum({<Year={$(=max({1}year(Date_Field)))}>}m_sales)/
    sum({<Year={$(=max({1}year(Date_Field)-1))}>*
    <Year=,Date_Field={"<=$(=addmonths(max({1}Date_Field),-12))"}>}
m_sales))-1
```

Sum of sales for maximum year-to-date vs. the prior year-to-date.

Operators within Modifiers

The use of operators is not limited to the beginning of the set. You can also utilize modifiers within a modifier either as an assignment operator or inside an element list (You can ignore the syntax checker here)

```
sum({$<[Product Sub-Category]=[Product Sub-Category]+{'Envelopes','Labels'}>}m_profit)
```

Sum of profit for the currently selected Product Sub-Categories adding the selections Envelopes and Labels to this.

You can also assign the operator before the element list. This has the exact same result as the above example.

```
sum({$<[Product Sub-Category]+={'Envelopes','Labels'}>}m_profit)
```

Sum of profit for the currently selected Product Sub-Categories adding the selections Envelopes and Labels to this list.

A word of caution here. If you need to use multiple operators within the modifier, put them all in the element list rather than utilizing both the element list and the assignment operator before the list. Using the second example below will not exclude Paper.

Use this syntax:

```
sum({$<[Product Sub-Category]=[Product Sub-Category]+{'Envelopes','Labels'}-{'Paper'}>}m_profit)
```

rather than this:

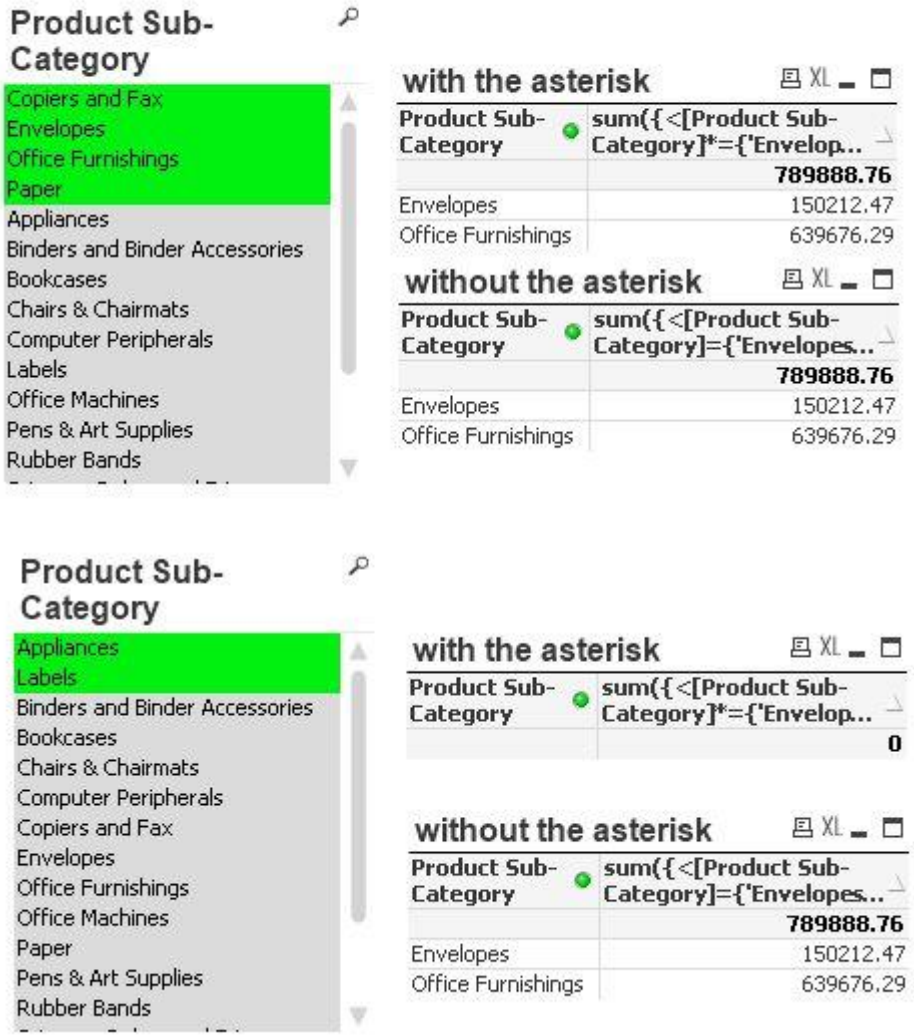
```
sum({$<[Product Sub-Category]+={'Envelopes','Labels'}-{'Paper'}>}m_profit)
```

Another great example is the use of asterisk and equals. This has the effect of considering the user's selection from the modifier field before calculating the set. One could argue this methodx of developing sets is actually *more intuitive* to the user than the "normal" approach of completely overwriting the user's selection.

```
sum({<[Product Sub-Category]*={'Envelopes','Office Furnishings'}>}m_sales)
```

Sum of sales for Product Sub-Categories Envelopes and Office Furnishings, respecting the users selections within that field.

Here we have the expression written with and without the asterisk operator so you can see the difference in behavior.



General Tips

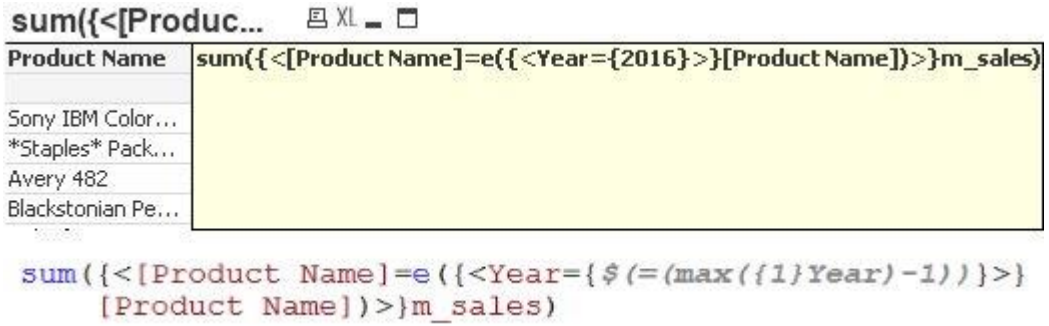
You can use sets within other sets

```
sum({<Year={$(=max({1}Year))}>}m_sales)
```

When using nested aggregations with the aggr() function, it might be necessary to use the same set in more than one place in your expression

```
avg({<Region={'Ontario'}>}aggr(
    sum({<Region={'Ontario'}>}m_profit),Year))
```

For QlikView, debugging your set can be done by putting your expression into a straight table and leaving the expression label blank. The dollar-sign expansion portions will evaluate in the header row. As a rule, I leave my set analysis labels blank until I have confirmed that the dollar-sign expansion is working and the expression is accurate overall.




```
sum({<[Product Name]=e({<Year={$(=(max({1}Year)-1))}>}
    [Product Name])>}m_sales)
```

Take care when using set analysis to *label your expression appropriately*. By definition, you are affecting the normal selection state of the calculation. Be sure you clearly tell the user how that expression is actually calculating. Incorporating the modifier value might be a good way to do that.x


```
='Sales for products not sold in ' & (max({1}Year)-1)
```

You can use flags to identify month-to-date and other points in time to use in sets. But keep in mind these will not be dynamic (the date range evaluated will not change with date selections).

```
sum({<CurYTDFlag={1}>}m_sales)
```