Paper 433-2011

Measures, Members, and Sets, Oh My!

Advanced OLAP Techniques

ABSTRACT

Going beyond basic OLAP measures, members and sets gives the analytical user more flexibility to answer complex business questions efficiently and effectively. Many times business requirements overtake the basic aggregate measures of an OLAP cube and require customized measures that may or may not be non-additive. For example, Percent of Totals, Weighted Averages, and Ratios across time cannot be determined through the standard aggregation definitions of OLAP. The creation of custom OLAP member sets may be necessary to present these custom measures. These custom sets can present specific members within multiple levels of a hierarchy or across many dimensions. MDX allows for the creation of many highly customized measures, members, and sets but has its limits. There are ways to move beyond these limits and create more dynamic and complex functions using SAS Macros® in conjunction with custom MDX. Utilizing SAS Macros created by pre-processing base SAS code will open a new world of dynamic functionality that is accessible within SAS OLAP cubes. This paper will discuss how to create custom measures, members and sets using the standard OLAP approach and ways to combine base SAS and macros variables to define more powerful members and sets.

INTRODUCTION

This paper uses the PRICEDATA dataset from the SASHELP library. For context, the current month is October 2002. The data revolves around monthly sales for a shoe store with outlets across the United States.

As a general tip, SAS Enterprise Guide ® can be used to rapid prototype measures, members, and sets. Once the correct MDX is written, the code can be defined in OLAP Cube Studio ® or in the base PROC OLAP SAS code that creates the cube. This paper focuses on writing PROC OLAP code directly, rather than using OLAP Cube Studio. This is a best practice for development purposes which decreases development cycles and improve troubleshooting ability.

MEASURES

The most common aggregations are sum, count, min, max, and average. These can be used as building blocks for much more complex calculations. The general idea is to break the calculation down into steps and compute through multiple measures that relate to one another. Remember to set the solve order appropriately.

PERCENT OF TOTAL

Percent of total measures are those that rely on the parent total to compute a percent of the parent total. This can be useful when showing the distribution of a measure across a dimension. A stacked bar graph shows this nicely.

```
DEFINE Member "[SGF].[Measures].[Percent of Total Sales]" AS
'([Product].[Product].CurrentMember, [Measures].[SalesSum])/
 iif(([Product].[Product].CurrentMember.Parent, [Measures].[SalesSum]) = 0
OR ([Product].[Product].CurrentMember.Parent, [Measures].[SalesSum]) = NULL,
         ([Product].[Product].CurrentMember, [Measures].[SalesSum]),
        ([Product].[Product].CurrentMember.Parent, [Measures].[SalesSum])
             ),FORMAT_STRING="PERCENT10.2" ' ;
```

ANNUALIZED TOTALS

Annualization of a measure can be approximated with simple calculations; for example take a monthly measure and multiply by 12. This simple calculation does not take into consideration leap years or the number days in a month.

[June Sales] * [Months in Year]
$1,546,093 * 12 = $18,553,116

[June Sales] / [Days in Month] * [Days in Year]
($1,546,092.74 / 30) * 366 = $18,862,331.43

Measures			Sales Sum	Annualized Sales
Year	Quarter	Month		
1998			$16,730,227	$16,730,227
1999			$16,533,201	$16,533,201
2000			$18,755,067	$18,755,067
	1		$4,709,537	$18,941,653
	2		$4,649,952	$18,702,003
		April	$1,550,395	$18,914,818
2000	2	May	$1,553,464	$18,340,897
		June	$1,546,093	$18,862,331
	3		$4,684,854	$18,637,572
	4		$4,710,725	$18,740,492
2001			$13,137,031	$13,137,031
2002			$12,935,040	$12,935,040

Figure 1: Showing annualized sales across YQM hierarchy.

Month Days, Quarter Days and Year Days measures are base measures that are the number of days in the Month, Quarter of Year, and are used to reduce the Sales Sum to the lowest grain before the annualization can occur.

Define the Monthly Annualization measure:

```
DEFINE Member "[SGF].[Measures].[Monthly Annualization]" AS
    '([Measures].[salesSUM] / [Measures].[monthdays]) * [Measures].[yeardays],
             format="dollar20.0"';
```

Define the Quarter Annualization measure:

```
DEFINE Member "[SGF].[Measures].[Quarter Annualization]" AS
    '([Measures].[salesSUM] / [Measures].[quarterdays]) * [Measures].[yeardays],
                    format="dollar20.0"';
```

Lastly define a measure which checks the current level for QUARTER or MONTH types and display one of the two measures above. This is a good example of how measures can relate to one another as long as they are defined in the correct order or the solve order variable is set.

```
DEFINE Member "[SGF].[Measures].[Annualized Sales]" AS
    'iif([Time].[YQM].CurrentMember.Level.Name="Month",
       [Measures].[Monthly Annualization] ,
       iif([Time].[YQM].CurrentMember.Level.Name="Quarter",
         [Measures].[Quarter Annualization],
         [Measures].[SalesSum]
             )), format="dollar20.0" ' ;
```

Point in Time Versus Additive Measures

Sometimes measures in the source data might be summarized to a certain level of time or are simply “as of right now” or the sum of the entire population. In our sample dataset we have a column called ‘employee’ which is the number of employees at that month in time.

	Measures	Employee Count
Year	*Month*	
1998		102
1999		102
2000		50
2000	January	102
	February	102
	March	102
	April	102
	May	102
	June	60
	July	102
	August	102
	September	102
	October	102
	November	102
	December	50
2001		102
2002		135

Figure 2: Point in Time measures do not aggregate at parent members.

We want to show the 'Employee Count' by month but do not want to aggregate it above month when viewing the year. The idea is to use conditional logic and the isleaf() function to check if the current member is the lowest possible level of time and only aggregate for the lowest grain of the source data, the MONTH in our case.

```
DEFINE Member "[SGF].[Measures].[Employee Count]" AS
 'iif(isleaf([Time].[YM].CurrentMember),
    [Measures].[employeeMax],
    ([Time].[YM].LastChild, [Measures].[employeeMax])
  ),  format="comma8."';
```

The MDX logic follows these steps:

1. If the current member is a leaf member, it has no children and it is the last level in the hierarchy.
2. If this is the case, show the standard measure which has the desired measure at the lowest grain.
3. If the current member is not a leaf member, then use the LastChild function to determine the last child of the year (month) and return the measure for that level.

MEMBERS

Calculated members are custom aggregations of members within a hierarchy. They can be shown simultaneously with the same parent hierarchy and remain displayed even if the parent hierarchy is expanded or drilled.

SHOWING A ROLLING MONTH AGGREGATION USING A CALCULATED MEMBER

One common task is to show aggregations of time simultaneously. For example, business users may want to see 12, 24, and 36 month aggregations at the same time.

Columns: YM > All YM | Sales Sum

Rows: Product > All Product

Filters:

Year	1998	1999	2000	2001	2002	12 Months	24 Months	36 Months
Measures	Sales Sum	Sales Sum	Sales Sum	Sales Sum	Sales Sum	Sales Sum	Sales Sum	Sales Sum
Product Line								
Line1	$2,487,922	$2,453,661	$2,772,305	$1,947,573	$1,907,493	$2,238,068	$4,319,868	$7,047,880
Line2	$2,409,096	$2,364,162	$2,691,951	$1,892,619	$1,861,708	$2,172,358	$4,209,655	$6,842,871
Line3	$4,353,579	$4,340,822	$4,902,776	$3,406,555	$3,381,568	$3,968,308	$7,589,958	$12,427,956
Line4	$5,452,763	$5,388,396	$6,125,792	$4,298,942	$4,223,540	$4,935,858	$9,553,402	$15,546,754
Line5	$2,026,867	$1,986,161	$2,262,243	$1,591,341	$1,560,732	$1,825,992	$3,533,470	$5,751,350

Figure 3: Showing multiple aggregations of time using calculated members in Enterprise Guide.

From the example above, the user can see the 12, 24, and 36 month rolling totals while drilling into the standard time hierarchy on the left.

To create the '12 Month' calculated member, here is the OLAP statement.

```
DEFINE MEMBER "[SGF].[Time].[YM].[All YM].[12 Months]" AS
     'Aggregate(
        {
          [Time].[YM].[All YM].[2001].[November],
          [Time].[YM].[All YM].[2001].[December],
          [Time].[YM].[All YM].[2002].[January],
          [Time].[YM].[All YM].[2002].[February],
          [Time].[YM].[All YM].[2002].[March],
          [Time].[YM].[All YM].[2002].[April],
          [Time].[YM].[All YM].[2002].[May],
          [Time].[YM].[All YM].[2002].[June],
          [Time].[YM].[All YM].[2002].[July],
          [Time].[YM].[All YM].[2002].[August],
          [Time].[YM].[All YM].[2002].[September],
          [Time].[YM].[All YM].[2002].[October]
        })';
```

As you can see this is a tedious task that can become overwhelming for larger number of months. Later in this paper we will discuss how to make this more dynamic and efficient using a 24 and 36 month example.

Suppose you wanted to create a custom member that displayed the aggregate of years prior to a certain year. This can be done using the same DEFINE MEMBER statement as shown above and would look something like this:

Year	Pre 2000
Measures	Sales Sum
Sales Region	
Region1	$4,941,583
Region2	$13,467,659
Region3	$14,854,186

Figure 4: Creating a calculated member called 'Pre 2005'.

As you can see again, there would be a lot of manual effort around maintaining the static code as time moves forward. Later this paper will again discuss how to write dynamic members.

SETS

Custom member sets are sometimes needed to present measures more effectively. The method used to create a custom member set depends on the output needed. Dashboards lend themselves very nicely to member sets because it simplifies the number of objects needed to present as an indicator. For example, a company may have multiple regions across the country but has a specific region it focuses on more often than others. A member set can be defined which isolates this region.

Measures	Sales Sum
Sales Region	
Region2	$31,604,836

Figure 5: Isolating 'Region2' using a member set.

Member sets are more useful when isolating custom groupings of time. For example, a dashboard indicator may need to always show the last 7 days of a metric or any relative grouping of time. Since time is continuous and changes as more data is added to the cube, it is not effective to define a specific set of time using the standard approach. Later this paper will show how to use base SAS code to dynamically write the MDX needed for a desired grouping of time.

Member sets can be defined in the base PROC OLAP code to build a cube or within OLAP Cube Studio. Information Map Studio ® currently does not provide a simple way to add custom member sets. To add, simply create a new data item and reference the member set using MDX syntax.

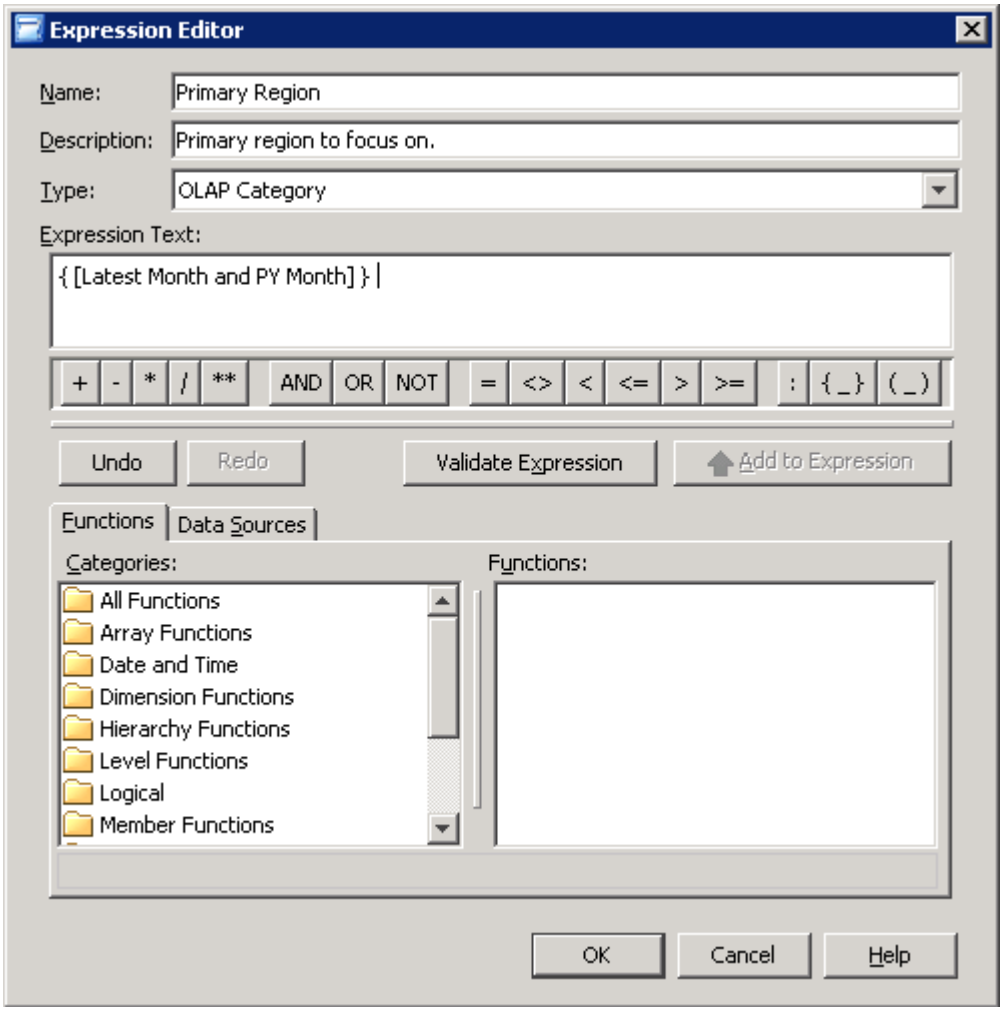

Figure 6: Defining a member set in Information Map Studio.

MEASURES, MEMBERS, AND SETS, OH MY!

A key concept of designing cubes is that elements within the cube can be related. Another key concept to remember is that the code used to build cubes can have pre-processing code to determine virtually any custom measure, member, or set. Pre-processing code can be used to dynamically write the MDX needed to produce complex relative time sets or calculated members. The past sections of this paper have shown how to create a few unique measures, members, and sets. Now we will combine all of these and show the true power of OLAP.

ENHANCING THE ROLLING MONTH CALCULATED MEMBER

Let's go back to the example of showing 12, 24, and 36 month aggregations of time simultaneously. Originally we manually specified the past 12 months. This can be very cumbersome and irrelevant as time move forward. To overcome this we can use the power of base SAS to dynamically write the MDX needed for the calculated member.

Here is code that is run as a pre-processing step before the OLAP cube is generated.

```
%let maxdate = '05OCT2002'd;
/* Generates 24 months inclusive of current month */
DATA rolling24months;
  length mdx $50;
  format date mmddyy10.;
  do x = 0 to 23;
     date = intnx('month', &maxdate , -1 * x);
        mdx = "[Time].[YM].[All YM].[" || strip(put(year(date),4.)) || "].[" ||
                  strip(put(date,monname9.)) || "]";
     output;
  end;
  drop x date;
run;
proc sql noprint;
       select mdx into :rolling24months_mdx separated by ', ' from rolling24months;
quit;
```

For our example, the MAXDATE macro variable is manually set to the latest available date in the source dataset. A more dynamic date, such as the today() function, or querying the data at dynamically should be used in a live environment.

To define the actual calculated member, this is the segment of code that is necessary.

```
DEFINE MEMBER "[SGF].[Time].[YM].[All YM].[24 Months]" AS
     "Aggregate( {&rolling24months_mdx} )";
```

This rolling 24 month calculation will be dynamically generated every time the cube is generated.

MAKING THE MEMBER SET MORE DYNAMIC

Analysts may want to compare a current month metric to the same month a year ago. Here is the pre-processing code necessary to accomplish this.

```
DATA monthPmonth;
  length year $4 mdx $100;
  format date date9.;
  date = intnx('month', &maxdate, -12);
  mdx = "[Time].[YM].[All YM].[" || strip(put(year(date),4.)) || "].[" ||
strip(put(date, monname9.)) || "]";
  output;
  date = intnx('month', &maxdate, 0);
  mdx = "[Time].[YM].[All YM].[" || strip(put(year(date),4.)) || "].[" ||
```

```
strip(put(date, monname9.)) || "]";
  output;
run;
proc sql noprint;
       select mdx into :monthPmonth_mdx separated by ', ' from monthPmonth;
quit;
```

Here is the DEFINE step needed to create the member set.

```
DEFINE SET '[SGF].[Latest Month and PY Month]' as "{&monthPmonth_mdx}";
```

TIME LEVEL ORDINAL MEASURE

Certain measures may need to know what hierarchy is being shown in order to dynamically adjust how it performs its math. As described earlier, knowing which Time hierarchy is crucial for displaying point in time measures such as the Employee Count accurately. The "Percent of Total Sales" measure could also be even more dynamic if there are multiple hierarchies in a single dimension. For example, if the Product dimension had more than one hierarchy, a new measure could be defined which checks for these different hierarchies so the correct calculation is used. This measure is not intended to be shown to the end users but rather used by the true 'Employee Count' measure. The key to this type of measure is using the .ordinal function.

```
DEFINE Member "[SGF].[Measures].[TimeLevel]" AS
 'iif([Time].[YQM].currentmember.level.Ordinal > 0, "YQM",
   iif([Time].[YM].currentmember.level.Ordinal > 0, "YM",
     "UNKNOWN" ))';
```

Once the 'TimeLevel' measure is defined, we can use it to enhance the 'Employee Count' measure and the 'Percent of Total Sales' measure. The 'TimeLevel' measure is used to determine what hierarchy is being shown so the MDX code can be written to reference more than one hierarchy. More hierarchies could technically be added but the MDX code becomes more complicated.

Here is the 'Employee Count' measure enhanced so both the YM and YQM time hierarchies can be used. Originally this measure only worked accurately for the YM hierarchy. The tricky part is determining the hierarchy shown so the correct number of nested LastChild functions are used. The level name has to also be referenced. For example, if the YQM hierarchy is shown we will want to take the last child of the last child for the YEAR level total and the last child of the QUARTER level for the Quarter total.

```
DEFINE Member "[SGF].[Measures].[Employee Count]" AS
 'iif(isleaf([Time].[YM].CurrentMember) OR isleaf([Time].[YQM].CurrentMember),
[Measures].[employeeMax], iif([Time].[YQM].CurrentMember.Level.Name = "Quarter",
iif([Measures].[TimeLevel] = "YQM", ([Time].[YQM].LastChild,
[Measures].[employeeMax]), null),
iif([Time].[YQM].CurrentMember.Level.Name = "Year" OR
[Time].[YM].CurrentMember.Level.Name = "Year",
iif([Measures].[TimeLevel] = "YM", ([Time].[YM].LastChild, [Measures].[employeeMax]),
iif([Measures].[TimeLevel] = "YQM", ([Time].[YQM].LastChild.LastChild,
[Measures].[employeeMax]), null)), null))
),  format="comma8."';
```

The output would look like this:

			Measures	Employee Count
Year	Quarter	Month		
1998				102
1999				102
2000				50
2000	1			102
	2			60
	2	April		102
		May		102
		June		60
	3			102
	4			50
	4	October		102
		November		102
		December		50
2001				102
2002				135

Figure 7: Enhancing the 'Employee Count' so it can be shown on the YQM hierarchy.

MEMBER SET INCLUDING A PREVIOUSLY DEFINED CUSTOM MEMBER

Suppose a standard view of time was to group older years into a single member and still show years following.

Measures	Sales Sum
Year	
Pre 2000	$33,263,428
2000	$18,755,067
2001	$13,137,031
2002	$12,935,040

Figure 8: Showing a member set that includes a custom member.

This can be done by referencing the custom member in a predetermined member set.

```
DEFINE SET '[SGF].[Special Year View]' as "
   {
      [Time].[YM].[All YM].[Pre 2000],
      [Time].[YM].[All YM].[2000],
      [Time].[YM].[All YM].[2001],
      [Time].[YM].[All YM].[2002]
   }";
```

Again, we have static references in this code. You can learn from prior examples how to use base SAS code and macro variables to produce this code dynamically so that years are automatically written as time progresses.

CONCLUSION

Many business problems can be solved much more efficiently with the use of OLAP. Using customized measures, members, and sets to meet business requirements extends OLAP even further by providing a more complex and

dynamic framework. Using OLAP simplifies the source data model by computing necessary variables on the fly. OLAP can reduce the data complexity and streamline the development infrastructure which promotes faster results in the longer run.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Stephen Overton
Email: stephen.overton@gmail.com
Phone: (919) 341-9667

Bryan Stines
Email: btstines@gmail.com