Paper 232-2007

Beyond the Basics: Advanced OLAP Techniques

Ben Zenick, Zencos Consulting LLC, Durham, NC
Brian Miles, Zencos Consulting LLC, Durham, NC

ABSTRACT

This paper will address advanced OLAP capabilities that historically could not be taken advantage via the SAS platform. The paper will include real-world examples demonstrating how the SAS v9 platform has allowed great accessibility to the world of OLAP. Have you ever attempted to apply if/else logic directly into an OLAP cube that will allow the engine to make certain decisions? Have you had difficulty applying Advanced Time Period Manipulation in an OLAP structure? Can you apply different calculations to a measure based on a certain level in a hierarchy? How can you deliver Non-Additive calculations in OLAP? How can traffic light images be represented on an OLAP Report? This paper will address these questions as well as the following information custom options for cube building, using external aggregation tables, advanced MDX topics, and Cube display options.

INTRODUCTION

With version 9 of SAS came with a completely new OLAP framework that would give developers the ability to customize OLAP report data in ways never before possible from within SAS. Multidimensional Expression Language is the foundation of v9 SAS cubes and is responsible for the increased flexibility and functionality. There are more ways to access, manipulate, and display cubes than previously possible. This paper will focus on these new concepts to SAS OLAP and will guide you in the utilization of these new tools.

ALTERNATIVES FOR CUSTOM REPORTING USING SAS CUBES

PROC SQL

Proc Sql can use pass thru to OLAP to convert results of a multidimensional query into a SAS dataset. This allows applications to leverage the increased performance of a cube and still perform advanced analytic functions that are available via the SAS language.

In the following example a basic MDX query is run against a cube to extract data and then a Proc Univariate is run against the data to get detailed information. The Proc Univariate is then used to determine what the overall quartiles are for the products.

```
proc sql;
connect to olap (host= "&server" port=5451 protocol=bridge user="&user"
pass="&pass" repository= Foundation olap-schema="SASMain - OLAP Schema");
/* pass thru to OLAP line */
create table work.MDXOut as select * from connection to olap
(
          /* My MDX Code goes here */
          SELECT
          {[Measures].[Actual],[Measures].[Predict]} ON COLUMNS,
          {[Products].[Product].Members} ON ROWS
          FROM SUGI
          WHERE ([Time].[All Time].[1994])
          /* End of MDX Code */
);
disconnect from olap;
/* Disconnect from the OLAP Server */
quit;

/* must sort first */
proc sort data=MDXOut;
   by product;
run;
```

```
Proc Univariate data=work.mdxout Noprint ;
    By product;
    Var Actual;
    Output out=work.uniOut

    Q1=Q1
    Median=Q2
    Q3=Q3
    MAX=Q4;
Run;
```

STORED PROCESS (USING PROC SQL)

The output from the previous query can be leveraged by the SAS Stored Process infrastructure to stream graphics and multiple ODS formatted report outputs from the cube. The users are allowed to pass in the start month and end month; once the query runs it returns a table and graph.

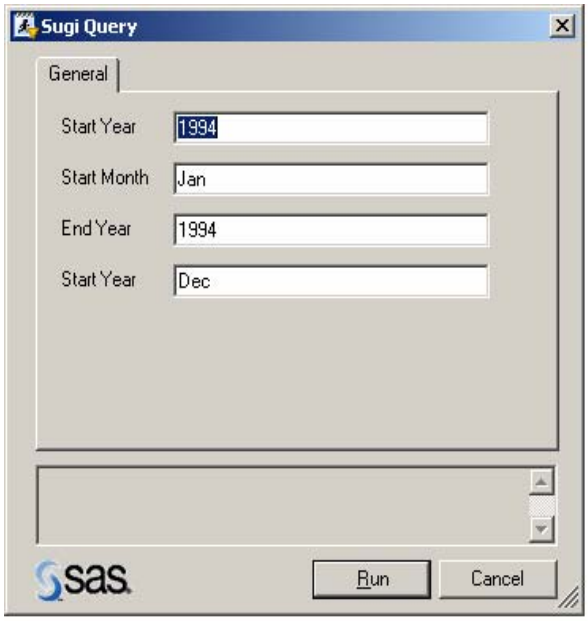
The code for the stored process will be similar to the example above (without the Sort and Univariate procedures); except a stored process will be used to pass a start date and end date to the code. The parameters that are going to get passed into the stored process are *endmonth*, *endyr*, *startmonth*, and *startyr*. The end user can select the time period that they would like to query by and the results will get streamed back to them.

This method is very similar to passing subsets into reports via a URL. Notice that the month subset syntax is not placed directly on the WHERE clause due to an MDX limitation that will only allow one member from each dimension to be placed on the WHERE clause. To circumvent the issue a set with the time subsets must be created and stored in one aggregated member. That member then gets placed on the where clause, which in turn applies the subset.

```
*ProcessBody;

%global  endmonth endyr startmonth startyr;

%stpbegin;
/* End EG generated code (do not edit this line) */

proc sql;
connect to olap (host= "&server" port=5451 protocol=bridge user="&user"
pass="&pass" repository= Foundation olap-schema="SASMain - OLAP Schema");

/* pass thru to OLAP line */
create table work.MDXOut as select * from connection to olap
(
with set [monthSet] as
{[Time].[YrMonth].[All YrMonth].[&startYr].[&startMonth] : [Time].[YrMonth].[All
YrMonth].[&endYr].[&endMonth]}
member [Time].[Months].[All Months].[sub] as 'Aggregate([monthSet])'

          /* My MDX Code goes here */
          SELECT
          {[Measures].[Actual],[Measures].[Predict]} ON COLUMNS,
          {[Products].[Product].Members} ON ROWS
          FROM SUGI
          WHERE ([Time].[Months].[All Months].[sub])
          /* End of MDX Code */
);

disconnect from olap;
/* Disconnect from the OLAP Server */
quit;
```

```
proc print data=work.mdxout;
run;

/* Begin EG generated code (do not edit this line) */
%stpend;
/* End EG generated code (do not edit this line) */
proc print data=work.mdxout;
run;

/* Begin EG generated code (do not edit this line) */
%stpend;
/* End EG generated code (do not edit this line) */
```

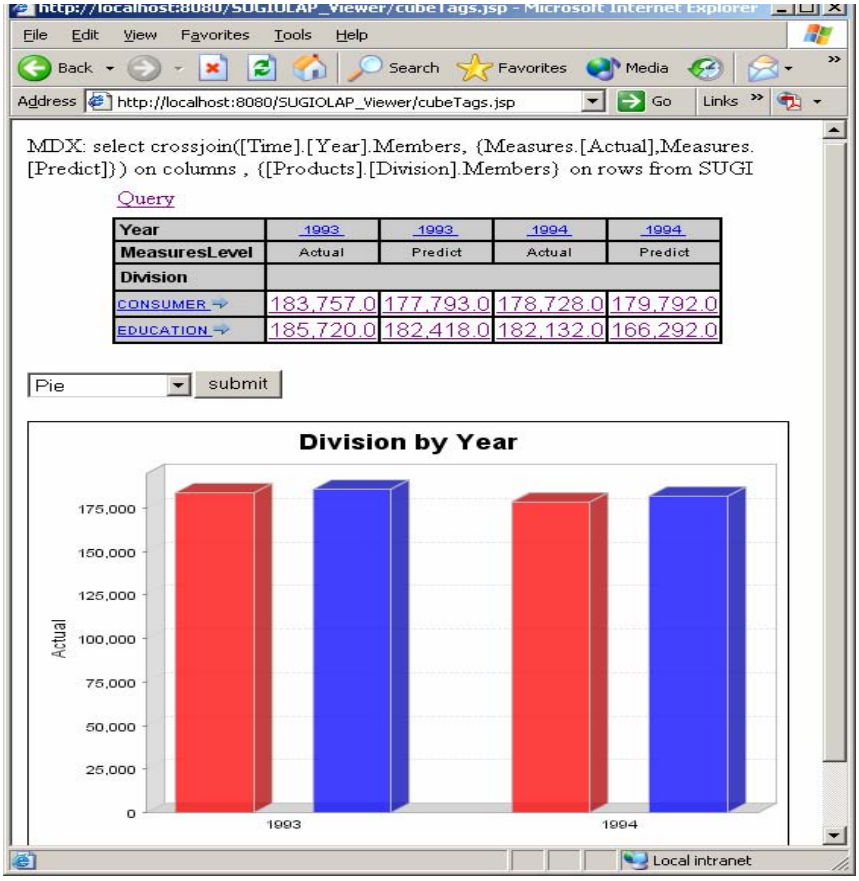
Sample output from the Stored Process is shown below.



Obs	Division	ProdType	Product	Actual	Predict
1	CONSUMER	FURNITURE	BED	37063	37838
2	CONSUMER	FURNITURE	SOFA	35574	34811
3	CONSUMER	OFFICE	CHAIR	35703	32627
4	CONSUMER	OFFICE	DESK	36926	40592
5	CONSUMER	OFFICE	TABLE	33462	33924
6	EDUCATION	FURNITURE	BED	35511	29106
7	EDUCATION	FURNITURE	SOFA	33582	29162
8	EDUCATION	OFFICE	CHAIR	39557	33985
9	EDUCATION	OFFICE	DESK	36139	36776

JAVA OLAP API

In V9, AppDevStudio provides classes that allow developers to build custom OLAP viewers. An example of these classes is com.sas.storage.olap.OLAPDataSetInterface. The OLAPDataSetInterface executes an MDX query and then the interface can be used to display reports. It does take a considerable effort and knowledge of the Java Language to develop drilling, expanding, click thru to detail data, graphs, etc. Below is a sample of one of the custom OLAP viewers that was created with this interface. As the user drills through the hierarchies the MDX is generated dynamically and is submitted to the OLAPDataSetInterface. The functionality of this class essentially allows developers to create OLAP reports in any format, style, or display.



CUSTOM OPTIONS WHEN BUILDING CUBES

Calculations that are processed at runtime can be added to the Proc OLAP code and they can be referenced by reporting tools just like the rest of the measures in the cube. Defined Members and Sets are ways to add custom runtime logic to a cube.

DEFINED MEMBERS

These were commonly referred to as computed columns in version 8 of SAS. The statements can be defined within the Proc OLAP code with options like formats available. All of the SAS and Custom OLAP client tools will treat defined members in the same fashion as all of the other measures that were defined by “Measure” statements.

Example of a basic defined member:

```
DEFINE MEMBER "[SUGI].[Measures].[Difference]" AS
'([Measures].[Actual] - [Measures].[Predict]), format_string="10.1"';
```

Defined Members can also use other defined members in their calculations. This example uses the Difference measure that was defined in the prior statement.

```
DEFINE MEMBER "[SUGI].[Measures].[Difference*2]" AS
'([Measures].[Difference]*2), format_string="10.1"';
```

These members can also be removed without rebuilding the cube by issuing an “Undefine” statement.

Below is an example of the MDX code that would be used to access the new measures.

```
SELECT
    {[Measures].[Difference],[Measures].[Difference*2]} ON COLUMNS,
    {[Products].[Product].Members} ON ROWS
FROM SUGI
```

SETS

A set is used to define a slice of the cube for calculated members to use in their calculations. A set consists of measures from existing dimensions or hierarchies in the cube. A set can also contain another set and use functions such as union and intersect to piece sets together.

Example:

```
DEFINE SET '[SUGI].[EducationCanada]' as
'Crossjoin([Products].[All Products].[EDUCATION],
[Geo].[All Geo].[Canada])';
DEFINE SET '[SUGI].[ConsumerGermany]' as
'Crossjoin([Products].[All Products].[CONSUMER],
[Geo].[All Geo].[Germany])';

DEFINE MEMBER "[SUGI].[Measures].[Percent_CanEdu]" AS
'Aggregate([EducationCanada],[Measures].[Actual]), format_string="Percent10.1"';

DEFINE MEMBER "[SUGI].[Measures].[Percent_GerCons]" AS
'Aggregate([ConsumerGermany],[Measures].[Actual]), format_string="Percent10.1"';
```

IIF / ELSE

Conditional logic can be used to define calculations based off of the current level, the current classification, or the current value that is being displayed. An iif function has three arguments:

1. the condition
2. what to do if the condition is true
3. what to do if the condition is false

If there is more than one condition, multiple iif functions are used and nested within one another. Here is an example of conditional logic with three conditions defined by comparing measures to determine how to set a traffic light value:

```
DEFINE MEMBER "[SUGI].[Measures].[TLight]" AS
'iif([Measures].[Difference] > ([Measures].[Actual] / 10),
    1,
    iif([Measures].[Difference] > ([Measures].[Actual] - ([Measures].[Actual] +
    [Measures].[Actual] / 10)),
    2,
    3
      )
 ), format_string="1."';
```

The logic sets TLight equal to 1 if Actual is greater than 10% of actual. If Actual is less than 10% but greater than -10% the measure is set to 2. Otherwise, the value is set to 3.

BUILDING ROLAP CUBES WITH EXTERNAL AGGREGATION TABLES

When a request is made to a cube, the results can be returned from a variety of source types including but not limited to ROLAP and MOLAP. MOLAP is defined as storing all of the data for a cube in a single physical OLAP structure. The acronym stands for Multidimensional On Line Analytical Processing. ROLAP on the other hand allows developers to have options for storing aggregation tables. Aggregation tables are built to define data in the cube that is available to satisfy query requests without having to summarize the data on the fly. They are pre-defined and pre-summarized structures. ROLAP provides the capability of storing these structures either as part of the physical cube, or in another type of data structure (i.e. SAS Tables, SPDS, or any RDBMS). As a result of this, the acronym stands for Relational On Line Analytical Processing for the historically common way of storing data in a RDBMS to retrieve from a Multi-Dimensional Report. There are a variety of different reasons to store aggregations in an external structure. Some of the reasons are;

1. That the structure provides the ability to perform different types of load techniques beyond the current MOLAP supported refresh;
2. Allows very large data structures to be summarized, while not running into some of they typical system resource constraints one may encounter when building MOLAP cubes; and
3. Allows the aggregated structures to be easily consumed by other client applications beyond OLAP clients.

The data structures / aggregations statements must still be defined in the Proc OLAP code, but by using the options defined in the sample Aggregation statement below, the Proc OLAP will not attempt to build the physical structure in the cube, instead it will simply add the metadata definition of the table that defines the Aggregation.

```
AGGREGATION Year Quarter Month Country Region /
            TABLE = libref.tablename DATAPATH=('/path-to-temp-data') INDEX
             INDEXPATH=('/path-to-temp-indexes') PARTSIZE=256';
```

In the aggregation above, an external summary table was created that crossed the Time Hierarchy by the Geo Hierarchy called libref.tablename. Subsequent aggregations tables can be built from each other by using a previously defined aggregation table as input, as long as the input table contains all of the necessary levels. A common approach for building ROLAP models is to store large NWAYS, that shouldn't be hit very often, but must be in every OLAP structure, in an aggregation table, so there is little impact on the physical cube from both a build and access perspective.

CUSTOM OPTIONS WHEN DISPLAYING CUBES

For most custom applications, MDX will need to be dynamically generated to apply subsets, calculated members, traffic lighting, trending, allowing users to define their own hierarchies from the front end, etc. This requires dynamic members and sets as well as the base MDX code to be generated on the fly.

DYNAMIC MEMBERS AND SETS

Members and Sets can be dynamically defined at query time just like they are defined in Proc OLAP, but they require slightly different syntax. Instead of using the "Define" statement, you must use a 'WITH' statement to create a query time member or set. There are ways to define sets and members that will persist for a user session, but it is preferable to use the query time 'WITH' statement since the same variables generally change across user requests. Unlike the Define statements in Proc OLAP, the 'WITH' keyword only appears once.

Here is an example of defined sets and members at query time. There is a set that is used to subset the report for data from Jan 1994 to Dec 1994. Also, a measure named GermanySum was created that sums up all of the actual values for Germany – even though you may be looking at another slice of data.

```
proc sql;
connect to olap (host= "&server" port=5451
protocol=bridge user="&user" pass="&pass"
repository= Foundation olap-schema="SASMain - OLAP Schema");

/* pass thru to OLAP line */
create table work.MDXOut as select * from connection to olap
(
```

```
/* My MDX Code goes here */
with
set [monthSet] as {[Time].[YrMonth].[All YrMonth].[1994].[Jan]              :
[Time].[YrMonth].[All YrMonth].[1994].[Feb]}

member [Time].[Months].[All Months].[sub] as 'Aggregate([monthSet])'
member [Measures].[GermanySum] as
      'Aggregate({[Geo].[All Geo].[Germany]},[measures].[Actual])'

           SELECT
           {[Measures].[Actual],[Measures].[GermanySum]} ON COLUMNS,
           {[Products].[Product].Members} ON ROWS
           FROM SUGI
            Where ([Time].[Months].[All Months].[sub])
/* End of MDX Code */
);

disconnect from olap;
/* Disconnect from the OLAP Server */
quit;
```

EXAMPLE: WHAT IF YOU NEED TO PASS A SUBSET FROM A URL?

One good example of how to do this would be to use a SAS Stored Process. From the Stored process example above, a web application can call the Stored Process application with the appropriate parameters and apply the subset. Also, if you use the App Dev Studio Java OLAP API to build a custom viewer you can alter the MDX that you pass into the OLAPDataSetInterface based off input parameters that come in.

EXAMPLE: HOW DO YOU GET TRAFFIC LIGHTING IMAGES INTO MY OLAP REPORT?

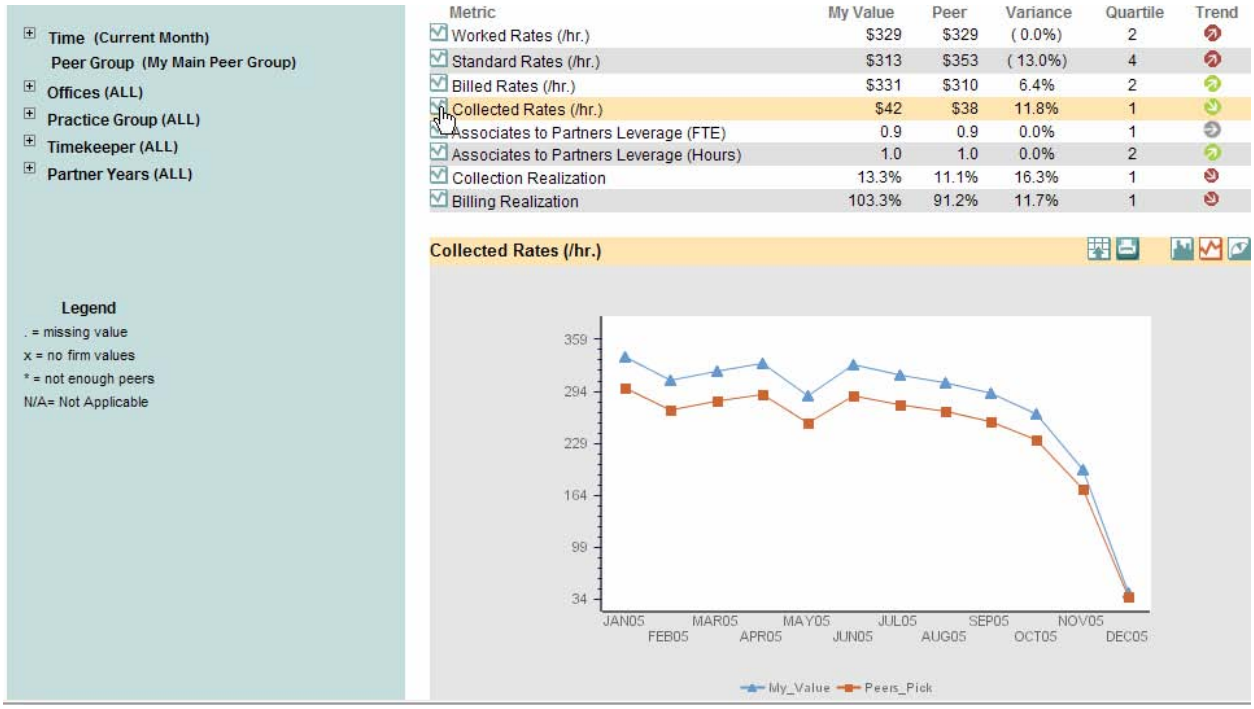
There are multiple ways to accomplish traffic lighting in OLAP. The most efficient ways usually involve creating custom formats and applying them to a defined member or applying them after the data has been extracted. In all cases, a defined member added to the cube to flag the report records based off of calculations.

```
DEFINE MEMBER "[SUGI].[Measures].[TLight]" AS
'iif([Measures].[Difference] > ([Measures].[Actual] / 10),
     1,
      iif([Measures].[Difference] > ([Measures].[Actual] - ([Measures].[Actual] +
[Measures].[Actual] / 10)),
           2,
           3
      )
 )

    , format_string="1."';
```

After there is a condition and the OLAP report records are flagged you need to find a way to turn the flag into an image. Usually a custom format must be created to map a flag to an html image. The format can then be applied in 3 ways:

1. The format_string option of the defined member can use the custom format
2. A data step after the MDX statement can apply the format to the results if it is a stored process
3. If it is a Java custom web application that isn't using either of the previous approaches, the format will need to be applied from the application.

The example below shows a Custom OLAP Viewer that has images applied as Trends to the viewer based on a defined member similar to the one above while also utilizing Advanced Time Manipulation described later in the paper.



ADVANCED EXAMPLES

AVERAGE REVENUE PER PRODUCT

The complication with performing this calculation in version 8 stemmed from needing a distinct count of products at any given level to divide by. With the use of the MDX count function you can add a measure to perform this when building the cube or at query time. The same logic would be used for Average Revenue per Customer calculations in OLAP.

```
DEFINE MEMBER "[SUGI].[MEASURES].[Avg_Rev_Per_Product]" AS
'([Measures].[Actual] /
COUNT([Products].[ProdID].[All ProdID].Children))';
```

PERFORMING CALCULATIONS BASED OFF CURRENT LEVEL (NON ADDITIVE MEASURES)

Non additive metrics usually mean that a particular calculation should behave differently based off of the slice of data that is being displayed. In version 8 OLAP this was very tough to accomplish and it usually required front end application changes and metabase extensions. With MDX it only takes about 4 lines of code.

Below are two ways of defining the same calculation:

```
Define MEMBER "[SUGI].[Measures].[New_Actual]" AS
    'iif(NOT([Time].currentmember.level is null) AND  [Time].currentmember.level is
[Time].[Month],
            Avg([Geo].[All Geo].[Germany],[Measures].[Actual]),
            [Measures].[Actual])';

Define MEMBER "[SUGI].[Measures].[New_Actual2]" AS
    'iif(NOT([Time].currentmember.level is null) AND
[Time].currentmember.level.Name = "Month",
      Avg([Geo].[All Geo].[Germany],[Measures].[Actual]),
       [Measures].[Actual])';
```

ADVANCED TIME PERIOD MANIPULATION - ROLLING 12 MONTHS

Creating rolling 12 month calculations from cubes has always presented challenges since multiple month values roll into the calculation for one month. The value for a month must be overwritten with the aggregation of the prior 12 months at a given level of a report. Here is a piece of code that generates a rolling 12 calculation in MDX. Notice that it will check to see if there are values for the prior 12 months and if not the calculation defaults to 0. If there are at least 12 prior members to the currentMember (for month), an aggregation of actual is taken over the last 12 months.

```
DEFINE MEMBER "[SUGI].[Measures].[Rolling_Actual]" AS
'iif([Time].[Months].CurrentMember.lag(12) is NULL ,
 0,
Aggregate([Time].[Months].CurrentMember.lag(1):[Time].[Months].CurrentMember.lag(12
) ,[Measures].[Actual]))';
```

ADVANCED TIME PERIOD MANIPULATION - YTD

```
proc sql;
connect to olap (host= "&server" port=5451
protocol=bridge user="&user" pass="&pass"
repository= Foundation olap-schema="SASMain - OLAP Schema");

/* pass thru to OLAP line */
create table work.MDXOut as select * from connection to olap
(
with MEMBER [Time].[All Time].[YTD] AS
'Sum([TIME].[ALL TIME].[1994])'
MEMBER [Time].[All Time].[Difference] AS
'([TIME].[ALL TIME].[1994] - [TIME].[ALL TIME].[1993])'
    /* My MDX Code goes here */
    SELECT
    {[Measures].[Actual],[Measures].[Predict]} ON COLUMNS,
    {[Time].[Time].[Year].Members, [Time].[All Time].[Difference],
    [Time].[All Time].[YTD]} ON ROWS
    FROM SUGI
    /* End of MDX Code */
);

disconnect from olap;
quit;
```

CONCLUSION

Many advances have been made in SAS' OLAP technology when assessing the capabilities between SAS v8 and SAS v9. These changes are evident in the flexibility that is available to developers; allowing them to generate complex logic and queries with Proc OLAP and Proc SQL using MDX. The ability to create complex calculations within a cube; or on the fly from any application truly opens up OLAP to be used for more than just typical Multidimensional Reporting. This same flexibility and adherence to industry standards allows developers to build reporting applications that have a much more tailored and customizable look and feel for their clients. These OLAP applications can now truly integrate the Statistical and Analytical Power of SAS into one highly optimized data structure.

CONTACT INFORMATION

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

Ben Zenick
Zencos Consulting LLC
2530 Meridian Parkway Suite 300
Durham, NC 27713
Work Phone: 919.806.4412
Fax: 800.858.9315
E-mail: ben.zenick@zencos.com
Web: www.zencos.com

Brian Miles
Zencos Consulting LLC
2530 Meridian Parkway Suite 300
Durham, NC 27713
Work Phone: 919.806.4412
Fax: 800.858.9315
E-mail: brian.miles@zencos.com
Web: www.zencos.com