Paper 37-26

OLAP Best Practices

What You Need to Consider When Building and Deploying an OLAP Application

Greg Henderson, SAS Institute, Cary, NC

ABSTRACT

SAS/OLAP Server® Software is a powerful, flexible and scalable OLAP solution. It's hybrid OLAP (HOLAP) architecture provides the flexibility to store data in whatever format is most appropriate based on the characteristics of the data and how the data will be used. This also means, however, that choices must be made as to what is the most appropriate data model for a successful OLAP application. This paper will identify and discuss the issues that need to be taken into account when designing and deploying an OLAP application.

INTRODUCTION

As OLAP applications have evolved and matured from simple sales analytics to other analytic areas throughout the organization, OLAP tools have also had to mature to meet the ever-increasing demands for aggregate analysis. Nowhere is this more evident than in the recently conceived areas of e-intelligence and web analytics. The nature of data generated by the web has put a tremendous strain on many systems, OLAP included. When you consider the vast amounts of data generated, combined with the inherently high cardinality of that data, many traditional OLAP technologies just cannot scale up to the challenge.

To answer these increased demands, SAS Institute made a strategic decision late in the version 6 development cycle to extend its traditional multidimensional data base engine, SAS/MDDB Server® software, to support a new Hybrid OLAP (HOLAP) architecture that would provide the necessary scalability and flexibility to answer the demands of a web driven world. In version 8, the HOLAP architecture has been completely integrated into the SAS System, and now carries a new name, SAS/OLAP Server® software.

As OLAP technology evolves and matures, so too must our thought processes, data modeling strategies and presumptions. Although there is not enough space in this document to cover every minute detail of the process of building successful OLAP data models, the intent is to get the reader to think in new ways. For, with the new HOLAP infrastructure, we are no longer limited by technology, but only by our imagination and creativity.

WHAT IS OLAP?

No OLAP document would be complete without first defining what is meant by OLAP. In a formal sense, OLAP (OnLine Analytical Processing) is defined as fast access to large amounts of summarized data. Implied in this definition is the concept of dimensionality. For without dimensions, there would be nothing to summarize the data by. Thus, a more generalized definition might be the ability for users to quickly interrogate large amounts of data, at varying levels of detail, across a variety of combinations business dimensions.

OLAP is full of a myriad of terms and acronyms, which are often ill defined. Thus, before digging too deeply into the bowels of OLAP, it is prudent to provide a few basic definitions of some of the OLAP terms that will be used in this paper:

Dimension – A business perspective that is useful for analyzing data by or across. Often times referred to as a hierarchy. Examples are Time, Product or Geography.

Level – Dimensions are often made up of various levels of detail. For example, the Time dimension may consist of Year, Quarter and Month levels. Some dimensions will only have one level, in which case the level is implied. When referring to the physical representation of data, a dimension level is sometimes referred to as simply a dimension, or for those familiar with SAS terminology, a class variable.

Member – A given value of a dimension level. For example, members of the Year level of the Time dimension could be "1998", "1999", and "2000". The number of unique members at any dimension level is referred to as the **cardinality** of that dimension level.

Measure – The ultimate business measure that is being aggregated. For example, Sales or Profits. Measures also have statistics associated with them such as Sum, Count, Average, etc. Those familiar with SAS terminology may refer to these as analysis variables.

OLAP DATA STORES

What really differentiates OLAP from other query and reporting systems is the idea of fast access (or high performance), combined with large amounts of data. Traditional query and reporting systems just are not designed for this level of online performance. Although the definition for OLAP does not explicitly specify any type of storage scheme for the underlying data, it is presumed that in order to accomplish the desired performance, some specialized data stores are required.

These specialized data stores contain data that is pre-summarized, or aggregated, across those combinations of dimensions that users want to see. Thus, an OLAP database can be visualized as a series of subtables, where each subtable represents a specific combination of dimension levels. Within each subtable, the rows represent each unique combination of the members of the dimension levels for that subtable. The columns represent the aggregated values of the measures for each unique combination.

For example, let's consider a simple situation where there are two single level dimensions, Year and Country, and one measure, Sales. If these are aggregated into a multidimensional database, the result is potentially 3 subtables – one for the summaries of each Year, one for the summaries of each Country, and one for the cross tabulation of Country by Year. If there are 2 years of data, and 3 countries, the Year subtable would contain 2 records, the Country subtable would contain 3 records, and the Country * Year subtable would contain 2*3 = 6 records. Table 1 illustrates what the resulting subtables would look like.

Year	Sales
1999	$850,000
2000	$1,060,000

Country	Sales
US	$1,300,000
Canada	$500,000
Mexico	$110,000

Year	Country	Sales
1999	US	$600,000
1999	Canada	$200,000
1999	Mexico	$50,000
2000	US	$700,000
2000	Canada	$300,000
2000	Mexico	$60,000

Table 1: Simple Aggregate Subtables

Note the word "potentially" in the previous paragraph. Again, the definition of OLAP does not mandate that all possible combinations are stored as persistent aggregations (ie. physical subtables). As long as the Year * Country subtable exists, Year and Country aggregates could be independently derived from the Year * Country subtable at run time. In this case, that would involve rolling up only 2 or 3 values respectively, so there would not be any significant performance degradation.

In a more generalized sense, the statement can be made that the only subtable that is required in an OLAP database is the one that crosses all of the dimensions levels, which we call the N-Way subtable. Most real world databases, however, are not as simple as the one described above, and to get acceptable performance, some subtables need to be pre-aggregated and persisted in the OLAP database. A good portion of this paper will be spent discussing how to determine which subtables to persist in order to obtain a good balance between performance and storage requirements.

STORAGE ARCHITECTURES

Once it is determined which aggregations to persist in the OLAP database, decisions must me made as to how to physically store that data. Traditionally, OLAP systems have operated on top of one of two underlying data architectures, multidimensional OLAP (MOLAP) and relational OLAP (ROLAP). Both of these architectures provide the capability of presummarizing data across the various dimensions that users want to see. The differences have to do with performance, scalability, simplicity, and resource utilization.

The earliest OLAP engines were primarily multidimensional (MOLAP) in nature. This required a specialized data store that would hold the aggregated data in a format whereby it could be easily retrieved by multidimensional queries. Most MOLAP's use a single operating system file to store the entire database, and have indexing implied and built into the structure.

In the early 1990's, however, the concept of a dimensional data model that could be represented in a relational database (RDBMS) was presented by Ralph Kimball. The basis of this concept lies in the star schema, and its derivative, the snowflake schema. As this concept gained in popularity, some vendors began to structure OLAP architectures on top of the star schema model, and thus was born the ROLAP architecture.

ROLAP OR MOLAP, WHICH IS BETTER?

As these two technologies evolved, a large debate began as to which was the superior architecture for OLAP applications. There were passionate opinions on both sides, but in reality each had it's own benefits and drawbacks. Which was best ultimately depended upon the underlying reporting requirements and the nature of the data.

MOLAP engines were preferred by many due to their simplicity. Because all of the aggregation rules, relationships and indexing were inherent in the data structure, they were very easy to build and maintain. In addition, they were often much smaller and much faster than comparable ROLAP architectures. There was, however, one key drawback. The MOLAP model was not highly scalable. As the amount of aggregated data increased, the simplicity of the single file architecture thus became its nemesis.

Especially problematic for MOLAP's was high cardinality data. Relational databases do a much better job of storing and retrieving small subsets of large data. Thus, the star schema and ROLAP architecture was often more suited for high cardinality data. In fact, many OLAP applications were forced into being implemented as ROLAP's simply due to the cardinality of one key dimension, often times a customer or product dimension.

In addition to scalability, many IT shops preferred ROLAP's because they used technology that most IT professionals were already familiar with, the RDBMS. In addition, if the data were stored in an RDBMS, it would be open for use in non-OLAP applications as well.

The following table highlights the key benefits and drawbacks to each approach:

Architecture	Benefits	Drawbacks
MOLAP	Fast	Not Scalable
	Small	Unknown Technology
	Easy to Maintain	
ROLAP	Very Scalable	Difficult to Maintain
	More Familiar Technology	RDBMS Overhead
	Flexible	

Table 2: OLAP Architecture Comparison

HYBRID OLAP (HOLAP), THE POWER TO CHOOSE

A true hybrid OLAP approach lets the application designer choose a combination of ROLAP and MOLAP architectures based on the reporting requirements of the user, the system resources available, and the nature of the data. For example customer and product dimensions are sometimes problematic in a MOLAP architecture due to their high cardinality. In a HOLAP architecture, these dimensions can be stored in a more scalable ROLAP schema, while all other dimensions are stored in a more manageable MOLAP architecture.

In addition to having the flexibility to store the data in different underlying architectures, the SAS HOLAP architecture allows each piece of the model to be stored on a separate computing platform, thereby further increasing the scalability across host systems.

Because of this increased flexibility, the application designer must make choices about not only what subtables to store in an OLAP database, but also how to physically store them. The next few sections will discuss some guidelines on how to make these choices. Since each set of choices has tradeoffs, it is imperative that the application designer understand the nature of the data that is being put into the model, as well as how the end users will be using the model. A data and business requirements assessment, thus, becomes one of the most critical steps in the application design process.

OPTIMIZING THE MODEL: DETERMINING PERSISTENT AGGREGATES

The introduction briefly touched on the idea of persistent subtables in an OLAP data model. On one extreme, the model could be optimized for minimum disk space, in which case we would only store one persistent subtable, the N-Way. This would have the benefit of a small data store, most likely at the expense of degraded performance since most rollups and cross tabulations would have to occur at run-time of reports.

On the other extreme, every possible combination of dimension levels could be persisted as a series of subtables. In theory, this would maximize performance; however, in practicality this is not true due to the huge size and complexity of indexing required to implement such a scheme. Although this technique might apply to the simple data model laid out in the introduction, real world models will typically have many more dimensions and dimension levels than this. The number of possible subtables for any OLAP model can be defined as

$$2^n-1$$

where n is the total number of dimension levels that exist in the model. Thus, for a model with 20 dimension levels, there would be over a million possible subtables!

To further illustrate how the above formula is derived, consider a binary example whereby each dimension level represents a digit within a binary number. The number of digits is equal to the total number of dimension levels. To represent each possible combination of dimension levels, it's binary digit can either be "1" indicating it is present, or "0" indicating it is not present. This would yield 2^n possible combinations. Since there would be no use in a subtable that didn't include any dimension levels, 1 is subtracted for the case where all digits would be "0". For those familiar with PROC SUMMARY, this binary representation is how the _TYPE_ variable is derived.

So, now that it is understood that persisting every subtable nor only persisting the N-Way is optimal for most OLAP models, let's consider some other techniques for determining which subtables to persist in a well-optimized OLAP database. Which technique is best primarily depends upon how users are going to be reporting against the data. The goal, however, is to persist those subtables that are going to be accessed most often, and minimize the number of times that the lowest level subtables (ie. N-Way) are accessed.

STAIRSTEP METHOD

The stairstep method involves persisting those crossings that exist on a set of predefined reports. Typically, this involves crossing all of the levels of dimensions that will exist on a single cross tabular drilldown report, and then repeating this process for each subsequent predefined report.

For example, let's consider a report where the Product dimension (consisting of Category, Item and SKU) is crossed with the Time dimension (consisting of Year, Qtr and Month). In order to make sure that there is an exact match subtable for each possible combination of levels of these dimensions, first start with the Cartesian product of all levels of both dimensions. Then, "stairstep" down each dimension by dropping off levels in sequence. The result would look like the following:

```
Category Item SKU Year Qtr Month
Category Item SKU Year Qtr
Category Item SKU Year
Category Item     Year Qtr Month
Category Item     Year Qtr
Category Item     Year
Category          Year Qtr Month
Category          Year Qtr
Category          Year
```

If the report contains more than two dimensions, the process is simply extended to include the additional dimension. Also, if the reports need the capability to be subset on certain dimension levels not contained in the axes dimensions, those dimension levels would need to be added to each crossing. This is a very important point in the stairstep model, because if the subsetting dimension level is not represented in any subtables, the N-Way will be used to satisfy all requests that include the subset.

Note that this technique is optimized only for drill downs operations (ie. subsetting the data by the selected parent member when moving to the next level). It is not fully optimized for down operations, which is when the report jumps to the next level without subsetting on any member of the parent. To fully optimize for this type of reporting, it would be necessary to include every possible combination of levels for the dimensions used in the report.

As you can see, the stairstep method is very effective if the types of reports can be predetermined. However, one of the key benefits of OLAP is that it allows the user to "slice and dice" the data any way they want. The next technique will optimize a model for more ad-hoc reporting.

SPIRAL METHOD

Although the Stairstep technique is very effective for determining which subtables to persist in a controlled reporting environment, the real power of OLAP databases is the ability for the user to create ad hoc reports, or slice and dice through the data. A very common use of OLAP tools is for a user to identify an anomaly in some business measure, and then slice this anomaly across various combinations of dimensions to try and determine why the anomaly exists.

To optimize for these types of reports, the modeling effort must be approached a little differently. Since ad hoc queries often combine a limited number of levels from many different dimensions (as opposed to many levels of a limited number of dimensions in the Stairstep example), it is beneficial to persist subtables that contain levels from many of the dimensions in the model. To keep the size of the overall database manageable, it is also desired to persist high cardinality data in as few subtables as possible.

The Spiral technique provides a model for this ad hoc environment. To get subtables that contain levels from as many dimensions as possible, and also limit the number of high

cardinality dimension levels in those subtables, the dimension levels need to be ordered based on their dimension and cardinality.

Figure 1 illustrates how this can be accomplished. Notice that each dimension has been placed on a set of vectors originating from a common center point. Next, the levels for each dimension are placed on the appropriate vector, with the highest level of summary furthest from the center, and the lowest level of summary closest to the center. It is also helpful to list the cardinality of the level next to it on the vector. Typically, cardinality will increase when moving from high summary to low summary levels.

Once this is done, the levels can be ordered by starting at the highest level of the dimension that will be most commonly used in the application, and then draw a line to the highest summary level of the next most common dimension. Continue this until the highest level of all dimensions is connected. Then, move to the next level and repeat the process. Notice that the diagram begins to look like a spiral leading to the central intersection of the dimension vectors. As the line approaches the center, some dimensions will run out of levels before others. Once this happens, use the cardinality of the levels to determine where to go next, going from lowest cardinality to highest.

When complete, the diagram should look like Figure1.

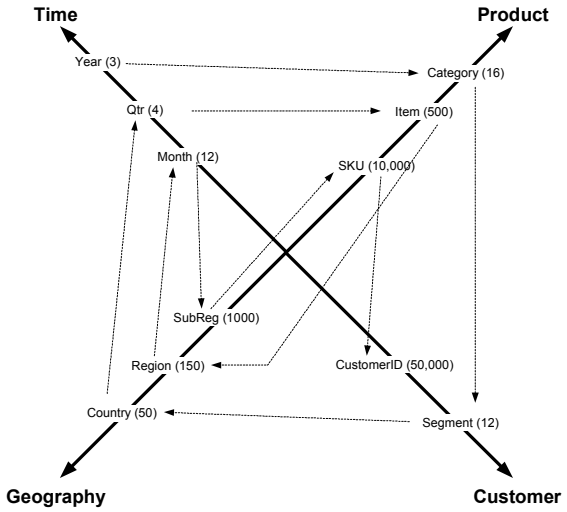

FIGURE 1 – SPIRAL OPTIMIZATION TECHNIQUE

By following the lines, a prioritized list of dimension levels can be determined. For our example, it would look like:

```
Year Category Segment Country Qtr Item Region Month SubReg SKU CustomerID
```

This simply produces an N-Way, but if we then stairstep this subtable down using the priority order, we should obtain a reasonably optimized set of subtables. The resultant model would look like:

```
Year Category Segment Country Qtr Item Region Month SubReg SKU CustomerID
Year Category Segment Country Qtr Item Region Month SubReg SKU
Year Category Segment Country Qtr Item Region Month SubReg
Year Category Segment Country Qtr Item Region Month
Year Category Segment Country Qtr Item Region
Year Category Segment Country Qtr Item
Year Category Segment Country Qtr
Year Category Segment Country
Year Category Segment
Year Category
Year
```

Since the highest cardinality dimension levels only occur in a very few subtables, the size of the resultant database is controlled. Also, every subtable contains levels from as many dimensions as possible. Although many queries will not be answered by an exact subtable with this model, most queries should be able to be satisfied without going all the way back to the N-Way.

OTHER TECHNIQUES

The Stairstep and Spiral techniques are just a sampling of strategies that can be used to optimize the persistent subtables that are stored in an OLAP model. These techniques can be used on their own, combined with each other, or combined with other techniques that will allow the OLAP model to ultimately serve the performance requirements of the user.

These techniques are used as a starting point when designing the model. Once the model is deployed, it will need to be monitored, and then adjusted based upon how users are actually exploiting it. How to monitor the model is discussed in more detail in the section on Deploying the Model.

OPTIMIZING THE MODEL: PARTITIONING STRATEGIES

Once it has been determined which subtables are to be persisted in the OLAP database, the next step is to decide how that data will be physically stored. The SAS HOLAP architecture provides an extremely flexible and scalable way to physically implement the model.

Due to the simplicity and performance benefits of MOLAP on data that has small to medium cardinality, a single MOLAP structure, or MDDB, should be used as a starting point for any model. Only when performance issues, data size issues or other special circumstances arise should one consider partitioning the model into separate physical structures.

There are two ways that an OLAP model can be partitioned. Stacking refers to storing different subtables in separate physical structures. Racking refers to taking a single subtable, or set of subtables, and partitioning them based on the value of one or more dimension levels. Each of these techniques, and when they might be appropriate, is discussed in the following sections.

STACKING

One way to partition an OLAP database is by stacking subtables. In stacking, each persistent subtable can be stored in its own physical structure. The structure can be any data format that SAS can read, but typically it is SAS MDDB's, summary datasets, or RDBMS tables.

Stacking is typically used to solve the following issues:

- High cardinality data that does not scale well in an MDDB.
- Large subtables that do not fit in memory.
- Special aggregation rules.
- Summary data already exists in a non-MDDB format.

High Cardinality

As mentioned in the introduction, high cardinality dimension levels often do not scale well in MOLAP structures. To get around this, stacking can be used to store those persistent subtables that contain high cardinality dimension level is a more appropriate structure. What defines "high" cardinality is a relative term, and will depend on the hardware platform, the total number

of dimensions and levels in the model, and the number of persistent subtables, among other things. However, dimension levels that are approaching 1000 members should be monitored for performance issues.

Using the example illustrated in the Stairstep technique, let's assume that the dimension levels have the following cardinality: Year(3), Quarter(4), Month(12), Category(16), Item(500), SKU(10,000). Obviously SKU is the dimension level that will be of the greatest concern. To stack this dimension level, we would take every persistent subtable that contains this level (including the N-Way) and break those subtables out of the MDDB. In this case we would remove the following subtables from the MDDB and stack them on top of it:

```
Category Item SKU Year Qtr Month
Category Item SKU Year Qtr
Category Item SKU Year
```

These subtables could then be stored in a RDBMS or SAS dataset, which are typically more suited to retrieving small subsets of values from a large dimension level. When stacking subtables for high cardinality dimension levels, it is imperative to index the tables on the dimension level variable that caused this subtable to be stored separately – in this case, SKU.

Memory Issues

Although medium to high cardinality dimensions do not always create enough of a performance problem to warrant stacking them, addressable system memory often will become an issue. SAS requires each subtable of an MDDB to be able to fit into memory to be exploited. On 32 bit systems, this limit is 2GB; on 64 bit systems, the limit is typically controlled by the amount of RAM allocated to the SAS session. By partitioning subtables that exceed these limitations into other structures, the limitations can be extended. Typically high cardinality dimension levels will not be accessed without first being subset to a reasonable number that can be presented on a report. By letting the RDBMS subset this data before it is summarized, the amount of data passing through the OLAP engine is reduced.

Special Aggregation Rules

In the introduction, one of the benefits mentioned for MOLAP architectures was the fact that the aggregation rules and logic were built into the OLAP engine. Most data can be summarized across any dimension using simple aggregation rules that are common to all measures across all dimensions. There are however, rare instances where certain business measures do not conform to these rules. For example, account balances are not additive across the levels of the time dimension, since (unfortunately!) the balance for the year is not equal to the sum of the balances of the individual days, months or quarters. These are often called "non-additive" measures, and their special aggregation rules can normally be handled more flexibly in a two dimensional structure.

Stacking allows us to break out all subtables that contain the non-additive dimension, and manually aggregate those subtables using the appropriate business rules in SAS Data Step or SQL code. When these types of dimensions exist, it is imperative to persist **all** possible subtables that can include any levels of the dimensions that are not additive. Otherwise, the OLAP engine may attempt to perform some aggregations at run time using the predefined aggregation rules of the OLAP engine.

Obviously, this can potentially lead to a very large number of persistent subtables, so a star schema model is often an appropriate storage mechanism.

Existing Summarized Data

Another application for stacking is to include existing summarized data without replicating it in the OLAP database. This data might be in an existing data warehouse or some other source. Instead of replicating the already summarized data, it can simply be combined with other summarized data that is not in the warehouse, and "stacked" into the model. SAS/Access® software makes it very easy to include existing RDBMS based star and snowflake schemas into an OLAP data model.

As you can see, stacking can provide the OLAP model with much needed scalability and flexibility, adding complexity to the model only when needed. One further benefit of stacking is that each stacked data source can be partitioned onto separate server platforms to further increase the scalability. In addition, SAS/Access® software will pass much of the subsetting work through to the RDBMS, which limits the amount of data that moves across the network.

RACKING

Whereas stacking involves partitioning multiple subtables into multiple physical structures, racking involves partitioning a single subtable into multiple physical structures. The Time dimension is often used in racking models, whereby the separate structures are built for each time period. For example, a separate MDDB could be built for each month. When using racking, the OLAP engine only accesses the physical structures needed to satisfy any given query. Thus if the model contains monthly MDDB's and a report needs just one month, only one MDDB would be accessed. If a report requested multiple months, then multiple MDDB's would be accessed and joined together.

One real benefit of this approach is at build time. If data is refreshed on a monthly basis, then instead of rebuilding the entire OLAP database, only the month MDDB affected by the new data will be rebuilt.

Racking can also be used on other dimensions when reports are typically subset on that dimension. An example might be a marketing analysis application where each marketing manager has a set of products they are responsible for. Each manager could thus have their own MDDB containing only the data for their products. The advantage to this approach is that by subsetting the high cardinality (Product) dimension, the subsets should be small enough for MOLAP implementation. If a consolidated view is needed, the HOLAP racking capabilities can join the disparate MDDB's together in a report when needed.

In summary, racking and stacking provide the OLAP model with scalability and flexibility. It allows the majority of the model to be implemented in a single, manageable MDDB, while only breaking out the pieces necessary to address scalability or flexibility issues. In addition, stacking and racking can be combined in the same model, as needed, to accomplish some of the benefits of both techniques. This is referred to as "stracking."

SKELETON MDDB

In order to hide the complexities of the underlying model from end users, the SAS HOLAP implementation uses the concept of a skeleton (or proxy) MDDB. This skeleton contains all of the attribute information of the entire OLAP model, but no data. Instead, it stores metadata about the distributed data model that has been defined. The key benefit of this is that changes made

to the underlying physical structure of the model can be transparent to the end user. Thus, it is prudent to consider using the HOLAP skeleton, even if the initial data model is not partitioned. This will allow a more seamless conversion to a partitioned model later if one is needed.

DEPLOYING THE MODEL

Once the model has been designed, and the physical layer has been determined, the final step is to actually build it and deploy it to users. The focus of this document is really on the modeling issues, however for completeness, I'll address some deployment topics at a high level.

SECURITY

As with any data, sensitivity is often an issue with OLAP databases. The SAS OLAP Server provides a role based access control subsystem that facilitates the following types of security:

Cube Level – Which cubes
Application Level – Which reports
Hierarchy Level – Which levels
Column Level – Which measures
Row Level – Which members

Security issues should be addressed during the planning and design phase, and implemented through the access control subsystem accordingly.

In addition to security, the access control subsystem can be used as a filtering tool to facilitate easier report development. For example, if product managers each need a report for only their products, the row level security can be set for each product manager so that they can only view data for their products. Thus, the report developer can build a single report for all product managers, and the access control subsystem will control what data is surfaced to each user.

More information on the Access Control subsystem can be found in the SAS/OLAP Server Administrator's Guide.

BUILDING THE MODEL

There are several ways to build the model once it is designed. In addition to PROC MDDB, both SAS/EIS® Software and SAS Enterprise Guide® Software provide GUI interfaces for building MDDB's. However, SAS/Warehouse Administrator® Software is the preferred tool. It encompasses the entire ETL process, not just the creation of the OLAP model, and provides facilities for scheduling refresh jobs and documenting the process via metadata. In addition, it has built-in support for many of the optimization techniques discussed in this paper. Further information on building OLAP models using SAS/Warehouse Administrator can be found in the SAS/Warehouse Administrator Reference Guide.

If using PROC MDDB, remember to order the HIER statements for building subtables from those with the greatest number of dimension levels to those with the fewest. This will optimize the build time performance of the MDDB because each subsequent table can be built from a previous summary table instead of always going back to the N-Way or detail data.

Also, if the model has been partitioned using racking or stacking, it is possible to build the various physical structures in parallel. This is especially relevant in a multi platform environment, and can significantly reduce the total amount of time required to rebuild or refresh the database.

END USER REPORTING

Once the database is built, a reporting environment needs to be established to facilitate exploitation of the database. SAS provides a variety of OLAP client interfaces that are appropriate to different types of end users. SAS/EIS® software provides a full client interface that includes a variety of tabular and graphical objects. In addition, having the full power of SAS on the client provides additional analytic capabilities that are outside the scope of most OLAP applications.

The web can also be a very effective delivery mechanism for OLAP reports. SAS offers OLAP client interfaces based on several platform independent technologies such as HTML, Java Applets and Java Server Pages. The web based OLAP viewers are also highly integrated into the SAS Business Information Portal, allowing user to subscribe to content contained in OLAP databases.

Finally, SAS supports the OLE DB for OLAP standard as both a provider and a consumer. Thus, SAS OLAP databases can be surfaced via OLE DB for OLAP compliant interfaces such Microsoft Excel™ or SAS Enterprise Guide® software.

ONGOING OPTIMIZATION

Once the model has been designed and deployed to end users it is imperative to revisit the design on a regular basis to ensure that the assumptions made during the design phase are still valid, and that nothing substantial was overlooked.

The HOLAP data provider generates a log file that contains information about all requests for data. The types of information contained in the log are the time it took to satisfy the query, the name of the data source(s) and subtable(s) used, the number of items returned, etc. By default, this file is stored as a SAS dataset in the WORK library for each SAS user session. It is, however, often advantageous to store the log files in a more permanent location. This can be accomplished by setting an attribute on the data provider class.

Because the log file displays subtable names, it is a good idea to provide a descriptive name for each subtable that is persisted in the model. Otherwise the log will display the subtable names as a number that was system generated when the subtable was created. A good naming convention is to use the names of the dimension levels that are included in that subtable.

CONCLUSION

The HOLAP capabilities contained in SAS/OLAP Server® software provide a robust level of scalability and flexibility. However, to fully leverage the power of HOLAP, a data model must be carefully planned and executed that takes into account both user reporting requirements and the characteristics of the data to be reported. By carefully understanding both the requirements and the data, highly scalable OLAP applications can be delivered to help end users solve sophisticated and complex business problems by leveraging vast amounts of data, which are aggregated and presented in an easily consumable format.

REFERENCES

Weinberger, Ann and Mattias Ender. *The Power of Hybrid OLAP in a Multidimensional World*. SUGI 25, Paper 133-25.

Wright, Ken. *New Features in Warehouse Administrator*, SUGI 25, Paper 114-25.

Moorman, Mark. *The Art of Designing HOLAP Databases*. SUGI 24, Paper 139.

SAS/EIS® Technical Report: HOLAP Extensions, Release 6.12. SAS Publication 56564.

SAS/OLAP Server Administrators Guide, Release 8.1. SAS Publication 57924. SAS Institute Inc, Cary, NC

SAS/Warehouse Administrator Users Guide, Version 2.0, First Edition. SAS Publication 56799. SAS Institute Inc, Cary, NC

Kimball, Ralph. 1996. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. (John Wiley and Sons, 1996).

ACKNOWLEDGMENTS

I'd like to thank Rob Stephens for providing the opportunity to collect many of my random ideas into this document. Also, Mark Moorman, Stu Levine and Ben Zenick for invaluable content and review assistance. Finally, thanks to the entire OLAP development team at SAS Institute for putting together the most flexible and robust OLAP technology possible.

CONTACT INFORMATION

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

Greg Henderson
SAS Insitute Inc.
SAS Campus Dr
Cary, NC 27513
Ph: (919) 531-2152
Fax: (919) 677-4444
Email: greg.Henderson@sas.com
Web: http://www.sas.com