Paper 136-25

Convert Your Model T into a Ferrari – Unleash the MDDB

Ian Sutton, Pioneer Software Limited, Wellington, New Zealand

ABSTRACT

The SAS/MDDB® Server provides us with a powerful tool for enhancing the performance of SAS® System reporting, especially within EIS type environments. The trick is to use it with the appropriate types of data and to populate it in a sensible manner.

This paper discusses some of the fundamental concepts of what a Multi-Dimensional Database (MDDB) contains and how it works, starting from a very simple perspective and then introducing more detailed concepts that allow for efficient population of the MDDB. This paper also discusses when the SAS/MDDB Server is appropriate to use and also when it is not appropriate.

ARE FAST RESPONSE TIMES POSSIBLE WHEN YOU HAVE MASSIVE VOLUMES OF DATA?

Consider the possibility of being able to provide very fast reporting and analysis when your database consists of 10's of billions of records and is growing by more than 100 million records every day.

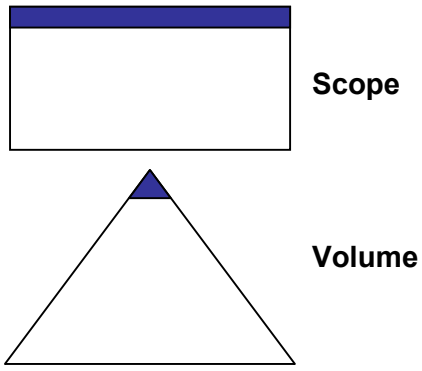
This is a very common problem for many companies with massive volumes of data like in telecommunications, but also for companies with significantly smaller volumes of data.

It is very realistic to provide fast response times to your users however this is dependent on exactly what nature of reporting your users require...

There are two extreme possible ways to report from your data:

- Broad Reporting – Getting the "Big Picture"
- Deep Reporting – Where users are interested in the detail

BROAD REPORTING - THE "BIG PICTURE"

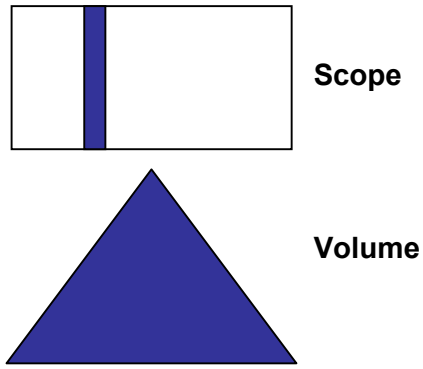


When using broad reporting users can see the entire scope of the data, however they are only looking at the surface level of this data. They can see the "big picture" but do not have access to the detail that was used to create this higher level view.

When using this type of reporting, it is usually possible to throw away 99% of the data volume, as it is the detail that makes up most of the data volume. The larger the original data size then the greater the difference between the storage requirements of the detail and the high level "big picture" view.

It is very easy to provide fast response times to this high level data because the summarized data volume involved is so small. The "deeper" users want to delve into the data then the more data volume is required and the slower the response times will be.

DEEP REPORTING - WHERE USERS ARE INTERESTED IN THE DETAIL



The other reporting extreme is when users are interested in the detail level of the data. This basically requires access to all of the original data and makes it very difficult to provide fast response times, and in some cases makes it impossible to provide fast response times when satisfying certain types of queries.

REPORTING PRIORITIES

We have found that the best approach to begin a project with is to start with the "big picture" view, for the following reasons…

EASE OF DEVELOPMENT

Very few resources are required to quickly create a high level view of your data. Not a lot of people are required and you do not need powerful machines therefore it can be done very quickly. This means that you can provide a functional and useable system very cost effectively and in a short period of time.

FOCUS ON IMPORTANT AREAS

By doing the "big picture" first, it is possible to identify the macro issues which need to be dealt with, which then provides you with the relative importance of various things, rather than getting wrapped up in a micro issue which may not be relevant in the greater scheme of things. This is true in both identifying where development effort should go, as well as from a business point of view, of what your company's business priorities should be.

BONUS - TESTING DATA INTEGRITY

An extra bonus of this approach, which we have often experienced, is that clients find that data integrity problems are very obvious when viewing the data at this level, especially when using graphical reporting and trends.

HOW TO EASILY GET THE "BIG PICTURE"

An approach used to provide this high level data, is to choose the dimensions which are of the most significant importance and to ignore the detail and high cardinality dimensions. The original data is then aggregated together by merging records of similar type up to the level of aggregation required.

For example, if a company has monthly product sales from the last 2 years, over 3 company divisions and 10 offices, and we group the products into 20 core product areas, then there is a maximum of 14,000 possible combinations of these values.

Months	(2 years = 24)
Company Divisions	(3)
Offices	(10)
Core Products	(20)

24 x 3 x 10 x 20 = 14,000 (maximum)

In reality not every combination will probably exist, so we should have less than 14,000 records but this would be the maximum number of records.

The original size of the source data is entirely unrelated to the size of this summary table which is produced. It doesn't matter if there were 15 thousand records or 15 billion records in the original source data. The only effect the size of the original data has, is how much computer time and resources are required to build the summary table prior to any reporting and analysis of it.

We can still achieve a reasonable level of flexibility with the reporting and analysis of this data. This summary table of 14,000 records gives us the ability to view any combination of…

MEASURES	BY	HIERARCHIES/DIMENSIONS
Number of Products Sold		**Time**
Cost of Products		Year
Sale Price (marked)		Quarter
Sale Price (sold for)		Month
		Company Division
Profit		Company Division
% Profit		**Geography**
Discount		Region
% Discounts		Office
Average Cost per Product		**Product**
etc.		Product Group
		Product

Any problem with performance? No !

There would probably be no problems with performance when using this data for reporting and analysis as it is so small. We would not need any special equipment or special SAS products in order to improve the reporting response times.

However, in reality…

Users may happily start with just a few very high level dimensions, however they very shortly want access to many more dimensions. This is because the more dimensions they have access to, the more flexibility they will have to analyse and report from this data.

A lot of data is not hierarchical which means that every single dimension that is added will add exponentially more complexity (and therefore volume) to the data. This is particularly the case with a lot of marketing data where analysts deal with a lot of attribute data, eg. Gender, Age, Income Level etc.

Another addition to the number of dimensions is the creation of entirely new dimensions. It is often possible to create new calculated dimensions to compromise the loss of the detail level of data. It is not possible to perform analysis on individual measure values as they have been aggregated together, however if we know in advance of certain factors which may want to be used in reporting we can create new dimensions to help provide this flexibility. Some types of new calculated dimensions are:

- Banding dimensions
- Scoring dimensions
- Segmentation dimensions

BANDING

Banding is when we identify particular pre-determined bands of a single measure. For example, with telecommunications data we could use the "length of call" measure to create bands "less than 1 minute", "1-5 minutes", …, "more than 1 hour", or with banking data we could use the "account balance" measure to create bands "overdraft", "$0-$100", "$100-$500", …, "More than $100,000".

SCORING

Scoring is the creation of a new dimension which can be the combination of any number of other values. For example, a finance company may have developed a way of rating the credit risk of clients based on a variety of stored factors, then categorizing these as a number between 1 and 10.

SEGMENTATION

Segmentation involves the creation of a dimension which categorizes the individual records based on some factors based on where they fit in the "big picture'. For example, to segment customers based on what revenue quartile they fit in. To do this you could have a dimension with four values, from the few big customers which make up the top 25% of your revenue, to the many smaller customers which make up the bottom 25% of your revenue.

The more hierarchies and dimensions which are added, the more flexibility the end users will have with their data analysis and reporting, however this also means more volume of reporting data and larger summary tables, which in turn means that the response times the users get will increase and their queries will be slower.

So what can we do about this? Introducing the SAS/MDDB Server…

WHAT IS THE SAS/MDDB SERVER?

The SAS Multi-Dimensional Database (SAS/MDDB Server) is a product designed to achieve a single purpose, and that is to increase performance. It provides no other advantages over what you can achieve with a summary table, however the improved performance is usually justification enough to introduce it in many situations.

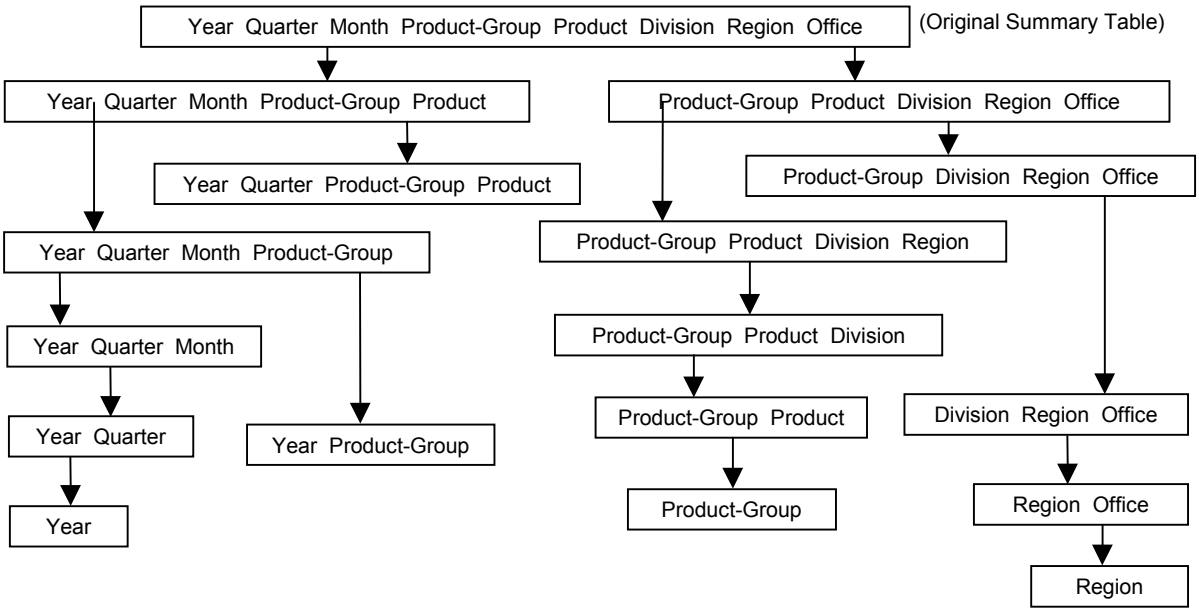
HOW IS THIS INCREASED PERFORMANCE ACHIEVED?

A Multi-Dimensional Database stores pre-aggregated summary tables in various different combinations and degrees of aggregation, so that most queries can be answered by using one of these higher level sub-tables rather than having to refer to the larger base summary table.

The performance is much faster because queries can typically be answered by referring to a much smaller table than would have otherwise been the case.

The SAS/MDDB Server also hides its internal complexities from the systems that request queries from the database.

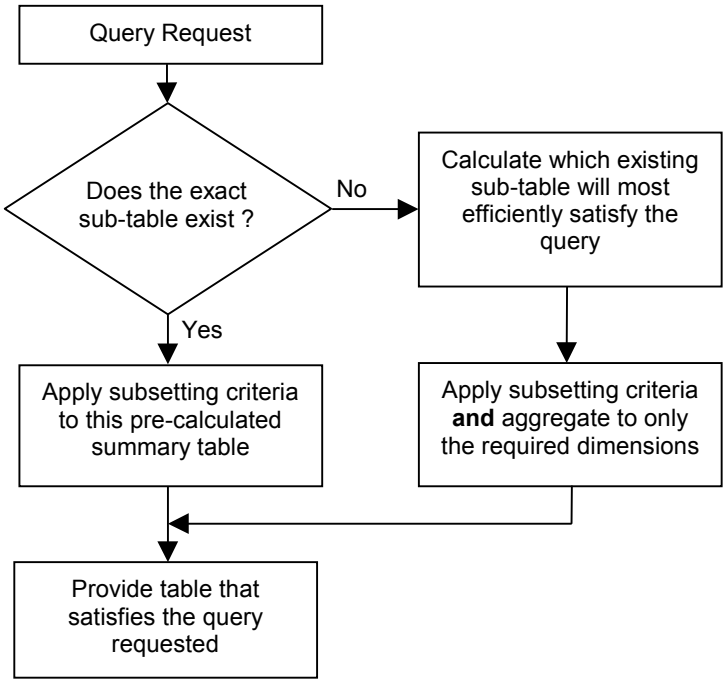
THE "INSIDE" OF AN MDDB



If we were to look into an MDDB we would see a structure something like the diagram above. This is made up of a single base summary table and many sub-tables all of which may or may not have their own sub-tables. The size of each sub-table becomes smaller and smaller with each iteration, where every single sub-table can be generated from the base summary table, and where every sub-table closer to the leaves of the branches can be generated from its parents.

In essence, the process of creating the base summary table is repeated again and again making smaller and smaller sub-tables by reducing the number of dimensions

HOW THE SAS/MDDB SERVER SATISFIES A USER QUERY



The diagram on the left shows the logic used by the SAS/MDDB Server to provide the results of a query. By using this logic it shields users from needing to understand the contents of the MDDB, or needing to know which sub-tables have been pre-aggregated.

NOTE: All reference to a particular sub-table of the MDDB is entirely done via the dimensions that it consists of. The dimensions which are used for subsetting and classification within a user query become the index which will identify the sub-table you require, or if no exact match exists then these dimensions will be used to work out which other sub-table will most efficiently be able to satisfy the query.

POPULATION STRATEGIES

Understanding how and why an MDDB improves performance is by no means "rocket science", however the process of selecting what you would like to populate the MDDB with can be a much more complex process.

WARNING : The number of possible combinations grow exponentially:

20 dimensions = more than 1 million possible combinations of dimensions (sub-tables)
(1,048,575 possible sub-tables)

50 dimensions = more than 1,000 trillion possible combinations of dimensions (sub-tables)
(1,125,899,906,842,623 possible sub-tables)

The more dimensions you have then the more complex this becomes, and this complexity increases at an exponential rate. If you have less than 10 dimensions then this is not very complex but if this is the case then you probably don't need an MDDB to improve performance as a simple summary table would probably be all that is required and no additional SAS System products would be required.

The trick to producing efficient MDDBs is to select the best sub-tables. If you have 20 dimensions then there are more than 1 million possible combinations of dimensions (sub-tables), but in most cases you can realistically improve performance to a very acceptable level with only one or two hundred of these million possible sub-tables.

If you have a large number of dimensions then the process of selecting which sub-tables to populate the MDDB with is not a trivial task, especially if you have a large number of hierarchies and non-hierarchical dimensions (like marketing attributes).

MANUAL SUB-TABLE SELECTION

There is one method that can be used to manually select the sub-tables for MDDB population. This method works with any number of dimensions, but be wary that this is simply a compromise, to increase the performance over what you would get without using the SAS/MDDB Server, but it will not provide optimal response times.

Step 1 Order the dimensions from the most commonly used and lowest cardinality to the least commonly used and the highest cardinality. This is the hardest part of the process, as sometimes you may have a conflict where a commonly used dimension is high in cardinality. Experience will teach you what order will be the most effective.

Step 2 Use this order of dimensions to create the first sub-table.

Step 3 Repeatedly add new sub-tables by taking the previous sub-table and dropping the last dimension, until there are no more dimensions left.

Step 4 Add in sub-tables for each single dimension.

EXAMPLE OF MANUAL METHOD:

The following is the order of dimensions. Notice that Office is last even though the cardinality of Product is higher. This is because Office is not used very often, and the Product dimension is used more often. (The value in the brackets in the dimension's cardinality.)

Year (2)
Division (3)
Product-Group (4)
Region (5)
Quarter (8)
Month (24)
Product (20)
Office (10)

The following is the list of dimension combinations (sub-tables) which you will get after Step 3:

Year Division Prod-Grp Region Qtr Month Product Office
Year Division Prod-Grp Region Qtr Month Product
Year Division Prod-Grp Region Qtr Month
Year Division Prod-Grp Region Qtr
Year Division Prod-Grp Region
Year Division Prod-Grp
Year Division
Year

Then the following sub-tables will be added with Step 4:

Division
Prod-Grp
Region
Qtr
Month
Product
Office

Finally the list of sub-tables can then be used to produce the MDDB by using the PROC MDDB procedure. The above example would result in the following code:

```
PROC MDDB data=source_dataset
          out=output_mddb;
  class YEAR DIVISION PROD_GRP REGION QTR MONTH
        PRODUCT OFFICE;
  var NUM_PROD / SUM;
  var COST     / SUM;
  var PRC_MRKD / SUM;
  var PRC_SOLD / SUM;
  hierarchy YEAR DIVISION PROD_GRP REGION QTR MONTH
            PRODUCT OFFICE;
  hierarchy YEAR DIVISION PROD_GRP REGION QTR MONTH
            PRODUCT;
  hierarchy YEAR DIVISION PROD_GRP REGION QTR MONTH;
  hierarchy YEAR DIVISION PROD_GRP REGION QTR;
  hierarchy YEAR DIVISION PROD_GRP REGION;
  hierarchy YEAR DIVISION PROD_GRP;
  hierarchy YEAR DIVISION;
  hierarchy YEAR;
  hierarchy DIVISION;
  hierarchy PROD_GRP;
  hierarchy QTR;
  hierarchy REGION;
  hierarchy MONTH;
  hierarchy PRODUCT;
  hierarchy CITY;
  run;
```

AUTOMATIC SUB-TABLE SELECTION

Due to the enormous number of possible combinations it is not typically practical to manually select which sub-tables you want and when done manually optimum response times will not be possible.

There are a variety of automatic sub-table selection methods which you can use to reduce the number of sub-tables produced. The most effective three methods we have found are described below.

USE OF HIERARCHIES

This is the method which SAS/EIS® uses when creating an MDDB and this is the best starting point to use for the list of sub-tables you want stored in the MDDB.

So using the hierarchies in our previous example of:

Time
- Year
- Quarter
- Month

Company Division
- Division

Geography
- Region
- Office

Product
- Product Group
- Product

We have to create a sub-table for every combination of hierarchy and every combination of the following dimensions within each hierarchy:

Time
- Year
- Year Quarter
- Year Quarter Month

Company Division
- Division

Geography
- Region
- Region Office

Product
- Prod-Grp
- Prod-Grp Product

As there are four hierarchies we can generate every combination of hierarchy by counting in binary with a four digit binary number, from 0000 through to 1111.

Then within each combination of hierarchy we must loop through all the dimension lists above adding a new sub-table for every combination of these.

This is the most complex part of the creation of the list of sub-tables. The remaining steps are considerably simpler.

DEPTH TRIMMING

This method and the next one are methods that we have developed ourselves which have proven to be vital in the process of selecting and reducing the number of sub-tables. This is especially true when you have more than about 12 dimensions, as the use of hierarchies alone does not reduce the number of sub-tables efficiently by itself.

Depth trimming is the most effective way of removing the large sub-tables which are not likely to be used. The "depth" referred to here is the number of dimensions that any single sub-table contains. When you have more than 15 or so dimensions then you end up with many sub-tables with a large number of dimensions, and hence are large in size. However, these sub-tables are very unlikely to be used to satisfy any queries.

If you were to trim to a depth of 9, then you would remove any sub-tables that contained more than 9 dimensions.

WARNING: This is a very powerful method of reducing the number of high-volume sub-tables but despite it's simplicity it has some serious implications if it is set too high or too low:

- If it is set too high then you will be wasting high volumes of space and your MDDB will take significantly longer to build than is necessary
- If it is set too low then your users will sometimes experience a sudden degradation of performance when they go beyond this threshold of the total number of dimensions they are using in their query for subsetting and classification.

We have found that for most organisations and types of data a sensible depth level to start with is about 8 or 9. To get an idea of what would be an appropriate starting point for you think about what the maximum number of dimensions that anyone is ever likely to use in any single query for either subsetting or classification.

The removal of these sub-tables from the list is simply a matter of looping through every sub-table in the list, counting how many dimensions would be in that sub-table and if this count is greater than the specified "depth" then we remove the sub-table from the list.

ENSURING DIMENSION EXISTENCE – "ALWAYS HAVE" DIMENSIONS

It is often the case that a single hierarchy or dimension will be used in almost all queries of the MDDB. This is often true of the time hierarchy/dimensions, as we rarely look at reports for time based data which don't contain a unit of time for classification or for subsetting. This means that we can throw away all sub-tables that do not contain those dimensions, and very significantly reducing the number of sub-tables.

The only bad side effect of this is that a query which does not involve this "special" dimension will have to aggregate the data to remove those dimensions. Though this is not usually at all significant and in most cases doesn't cause any noticeable loss of performance.

The removal of these sub-tables from the list is simply a process of looping through all sub-tables and removing any which do not contain the specified "Always Have" dimensions.

PULLING IT ALL TOGETHER – PRACTICAL APPLICATION OF THESE METHODS

We suggest the best method for implementing these methods is to create an application to help you with the selection of the sub-tables. This is the approach we have taken with our Futrix™ application and it works extremely well. This list of dimension combinations (sub-tables) can then be used as the input to the SAS/MDDB Server procedure (PROC MDDB) which will then populate your MDDB for you.

The more dimensions you have in a single MDDB then the more important it is to reduce the number of sub-table combinations. It is quite practical to create MDDBs with large numbers of dimensions (30, 40, 50…), but it becomes increasingly necessary to write your own application or use a product such as Futrix to select the sub-tables for you and to use many more methods of sub-table reduction, beyond simply the use of the hierarchy information.

If you want to write an application to produce the PROC MDDB code for you we suggest the following steps:

Step 1 Get the application to input the following information:
- source data set
- MDDB name to create
- all hierarchies and dimensions
- all measures and required statistics for each
- maximum depth (for step 3)
- any "Always Have" dimensions (for step 4)
- dimension order from "most commonly used/least cardinality" to "least commonly used/highest cardinality" (for step 5)

Step 2 Create a list of sub-tables by using the hierarchical dimension information

Step 3 Remove any of these sub-tables which have a depth greater than the specified amount

Step 4 Remove any of these sub-tables which do not contain the "Always Have" dimensions

Step 5 Add the sub-tables suggested by the manual method. This is using the dimensions in the order specified for a new sub-table and then for every additional sub-table removing the last dimension until none are left

Step 6 Add a sub-table which contains only a single dimension for all that don't already exist

Step 7 Output or run the PROC MDDB code which can now be created by using the list of sub-tables generated by the previous steps

MDDB LIMITING FACTORS

The limiting factors that you need to consider when building an MDDB are:

- the system memory available to run PROC MDDB
- how much time it will take to re-populate
- how much storage space it will require once built

The more sub-tables you have then the larger the storage required and the longer it takes. This is one reason that you must reduce the number of sub-tables as much as practically possible.

It is simply not practical (or even possible) to create an MDDB which contains thousands or tens of thousands of sub-tables. Remember that with only 20 dimensions there are more than a million possible sub-tables.

The factors which affect these limiting factors are:

- Number of dimensions
- Cardinality of each dimension
- Hierarchical nature of dimensions
- Sparsity between hierarchies
- Tuning of which sub-tables you store in the MDDB

WHEN NOT TO USE AN MDDB

It is not very appropriate to use an MDDB for reporting when users require the detail level dimensions. If they will often be using these dimensions then they may as well not use an MDDB. It is also not so appropriate when you have dimensions with a very high cardinality. The higher the cardinality then the less appropriate it is to use an MDDB. It may very well be the case that you use an MDDB which does not contain the high cardinality dimensions for some purposes and then use the summary level data set when you do need to access these dimensions.

An MDDB cannot be used for statistical analysis of individual measure values. This information has been lost in the reporting data as we have aggregated the measure values together. One way around this though can be to add new calculated dimensions, prior to populating the MDDB. These calculated dimensions can be approximations of the original measure values (banding). Though this is a compromise and often more flexibility is required or a finer granularity on the bands is needed (which then results in higher cardinality).

When using an MDDB for reporting and analysis it is also not possible to specify any subsetting criteria on the measure values. For example, you can't subset a telecommunications report to only look at telephone calls that lasted for more than an hour (because the "duration of call" is a measure and the detail of which is lost when aggregated to summary tables).

ADVANCED POPULATION OF AN MDDB – SELF-OPTIMISATION

It is possible to very efficiently populate the MDDB if the following information has been tracked and logged for all the queries that the users have made:

- which MDDB is being reported from
- all the dimensions used for each query
- the response time for each query

By using this tracking information it is possible to get the MDDB to optimise itself over time based on exactly how it is being accessed. The MDDB becomes more and more finely tuned over time so that sub-tables which are never used are not stored and other sub-tables are added if they are being used a lot and the user has received a slow response time.

The result of this process is that over time the response times the users' experience become faster and faster and the storage space required for the MDDB becomes smaller and smaller.

COMBINED APPROACH

It is often the case that you should use two separate types of data storage for two separate types of analysis.

MOLAP (Multi-dimensional On-Line Analytical Processing)
vs
ROLAP (Relational On-Line Analytical Processing)

Everything discussed in this paper so far has been MOLAP, which is when you are dealing with aggregate dimensional based reporting. This can answer many of the user queries, however it is not possible to satisfy all queries as some detail has been lost during aggregation. ROLAP is the other extreme where relational tables are used to satisfy queries and the processing is performed on the original tables but only joining the minimum number of tables. This can then satisfy every user query however much larger volumes of data need to be processed to satisfy some types of queries, and therefore performance can be much slower and the response times are often unpredictable unless the user has a strong knowledge of the database structure.

The SAS Institute has now introduced a solution which goes some way towards giving us the advantages of both MOLAP and ROLAP…

HOLAP (Hybrid On-Line Analytical Processing)

The SAS Institute in Europe has developed what in known as HOLAP, which goes some way to creating a single database engine which gives the best of both worlds. It does not cope with all relational database structures but it can handle some simple relational structures. Its key advantage is that it allows you to distribute a single database engine over multiple different platforms and mix multiple different storage levels from MDDBs to detail level tables into the single engine.

CONCLUSIONS

The SAS/MDDB Server fills a very important hole in the SAS suite of products. It certainly does fulfil its promise to improve performance of dimensional based reporting.

To maximise use of the MDDB, it is often necessary to add in your own new calculated dimensions which increases the power and flexibility of the analysis and reporting possible when using the data from an MDDB.

If you are trying to populate MDDBs when you have a reasonable number of dimensions then you need to have a good appreciation

of how to efficiently select which sub-tables will be stored. The more dimensions you have in a single MDDB then the more you need tools to help you reduce the number of sub-tables to only those that are most likely to be used.

CONTACT INFORMATION

If you have any questions about any aspect of this paper or its contents then please feel free to contact me directly:

Author:	Ian Sutton
Company:	Pioneer Software Limited
Web:	www.futrix.com
Email:	ian@futrix.com