The SAS Multidimensional Database Procedure

Timothy J Harrington, Trilogy Consulting Corporation, Kalamazoo, MI**, USA**

ABSTRACT

This paper introduces a SAS® version 6.12 development from the SAS Institute, the Multi-Dimensional Database (MDDB) and the SAS/Multidimensional Database Procedure ®, PROC MDDB. A MDDB is a data storage object which stores the presummarized statistics calculated from a very large volume of transaction data for fast and convenient access. MDDBs are primarily designed for use with the SAS/Executive Information System® (SAS/EIS). PROC MDDB provides a convenient way of creating MDDBs out of SAS data sets. This paper also compares the MDDB with the conventional SAS data set.

INTRODUCTION

The SAS MDDB is a data structure, similar to a SAS data set, but with built in presummarization and accessability. The type of data the MDDB is intended to handle is large volumes of historic transaction data, such as a retail sales ledger. The primary features of the MDDB are:

- Calculation and storage of presummarized statistics of large quantities of related data
- Fast access through range queries
- Easy data navigation
- Built in data integrity

The MDDB has the ability to store summaries of very large (potentially millions of observations) volumes of related data. This data can be updated with new observations regularly or on an occasional ad-hoc basis. Thus, the MDDB is ideally suited to be the heart of a data warehouse, or any centralized data repository.

CONCEPTS

A data warehouse must be designed to reflect the information queries that will be asked of it, this is critical to building a successful database. A generic definition of 'Successful' is the timely retrieval of required information with accurate, up to date, and applicable facts presented in an appropriate format, and accessible in a user friendly and navigatable way.

A data warehouse, or a part of a data warehouse can be thought of as a *multi-dimensional* table. Information in a MDDB is retrieved at the intersection of *ranges* of keys. Information pertaining to the keys is stored in separate key tables. For example, if one of the ranges was a series of generic product codes, only the product code is stored in the dimension table. Information about each product, such as its description, size, color, and regular retail price are stored with the key in a separate product information table. The user front end of the system uses the product information table to display the user understandable information about the product on the screens and reports. A retail business may have a Sales MDDB containing information about numbers of products sold, and the price paid by the customer at each sale. The sales revenue for a given product, date, and store is queried with a call to the database:

```
GET ( PRODUCT = <product code>, DATE = <mmddyy>,
    STORE = <location code>, REQUIRED INFORMATION =
    number of units sold, total sales revenue)
```

This call would retrieve the number of units sold and the sum of all the prices paid for each of the product units sold, this being the total sales revenue for the given product, on the given day, at the given store. Not only can one product, date, and store be specified, but *ranges* of these can be input too. 'Product' can refer to a whole brand name, a certain category of goods, or all the items on a sales promotion. 'Date' can refer to not just one particular day, but any range of days, a month, a week, a certain day of each week, or days which are holidays. Similarly 'Store' can refer to all the stores in a particular region or state, or all of a certain type or size of store. Whatever the *range queries,* the corresponding total, or *additive statistics* are retrieved for those ranges. A *range query* is the equivalent of a WHERE clause in a SAS DATA step or in SQL.

An important point to note is that each dimension has a *grain* or limit of resolution. In the example above the product dimension grain is the product, or sale unit, itself. The date dimension grain is one day, and the store dimension is each individual store. Each *cell* of the MDDB contains the statistics pertaining to the dimensions that intercept that cell. A database holding one year's worth of sales data for 1000 different products at 50 stores would contain 365*1000*50 = 18,250,000 cells (assuming a store is open every single day of a non-leap year). In turn, each cell then contains the summary statistics for that day, that product, and that store. All of the sales transactions for one product during one day at one store are summarized in the corresponding cell when the MDDB is built or updated. During the initial design phase of the data warehouse a suitable grain must be chosen for each of the dimensions. If too low a resolution is used valuable detail is lost, for example if the date

dimension has a grain of one week days cannot be retrieved or compared. Comparing sales statistics for Monday's with the corresponding statistics for Tuesdays is no longer possible. Conversely, if too high a resolution is used *sparsity failure* will occur, that is the MDDB will contain a vast number of cells, many of which would be empty (points in the table with zero or missing values). If, for example, the date resolution was one hour instead of one day there would be some hours when no units of the product sold at all (zero values) and other hours when sales were not possible (missing values), such as when the store was closed.

Each cell in the MDDB contains the statistics pertaining to the dimension values which point to it. Such statistics must be *additive* across the dimensions so that any sum or breakdown of each range results in a meaningful compound statistic. For example when a date range from Monday to Sunday inclusive is taken the total sales for that week are the sum of each of the totals for each of the seven days. Examples of such additive statistics are the total number of observations (N), the sum of values for the observations, the number of missing values, the maximum, and the minimum. Examples of statistics which are not additive are the median, mode, and mean, however non- additive statistics are easily calculated using the aggregate range statistics. For example the mean sales revenue for January is calculated by dividing the sum of the revenues for the 31 days by the sum of the number of sales (N) for the 31 days.

To improve performance further the MDDB may be implemented with *aggregate cells* of precalculated range statistics. An example is each month's sales statistics are stored as the accumulated statistics for all the days in each month in each of twelve cells for each year. If a range query with one or more complete months of data are specified the complete months' data are retrieved directly from the applicable aggregate cells instead of being added from all the days of each month. This aggregation enhances performance when the aggregated ranges are frequently all or part of the queries being asked of the database.

Finally there is the navigation factor, or *drilling* ability. *Drilling down* is accessing a subset of the last data retrieval. For example, if a particular type of product is currently being viewed, the ability to see any one or set of brands of that type of product should be only a mouse click away. The brand data should then allow drilling down to the individual product data with another single mouse click.
Drilling down is taking one or more sub-ranges of the current range query and is easily implemented in a MDDB. Drilling up is the exact opposite of drilling down, moving up to a superset of the range. Aggregate cells make drilling up more efficient, particularly when corresponding ratios and percentages need to be maintained (For example what percentage of soft drink sales was diet pop yesterday, last week, last month?). Drilling across is linking to data with one or more attributes in common, such as moving from sales of one brand of jewelry to another brand of jewelry, or from sales of necklaces to sales of earrings. Internal indexing of the dimensions allows such range changing to be performed very quickly.

The major advantages of the MDDB over other data structures are:

- There is no redundancy of information. This together with aggregate totals and data compression results in great space savings
- Aggregate statistics, ordering of data, and comparisons are very quick and easy to perform
- Range queries conform to SQL standards and hence are easily performed by existing data query tools.
- There is built in integrity regardless of what combination of any range queries are used.

The only significant disadvantages of the MDDB are the data must be historical and non-volatile, that is it cannot frequently be changed in an ad-hoc fashion, and some sparsity (unused cells) is unavoidable. Changes in the data can, however, be periodically rolled back into the database at the time updates are made. For example sales returns refunds (if not handled separately from the sales ledger) can be subtracted from the applicable sales ledger cells at the time the MDDB is updated.

THE SAS MDDB

A SAS implementation of a MDDB is created or modified using either

- SAS/Data Warehouse Administrator ®
- SAS/EIS
- PROC MDDB

Whichever of these three methods are used the resulting MDDB is the same. PROC MDDB is a quick way to build a few temporary MDDBs to use with SAS/EIS to produce a series of presentation charts. If the MDDB is created using SAS/EIS or the SAS Warehouse Administrator it is automatically registered in SAS/EIS, but if PROC MDDB is used and the MDDB is to be used by SAS/EIS it must be registered in SAS/EIS by using the ADD option in the SAS/EIS Metabase Window.

PROC MDDB is used to build or update a MDDB from a SAS Data set. PROC MDDB is functionally the same as PROC SUMMARY in terms of the calculation and storage of statistics, the difference is an object of the summarized structure is created, and only statistics which are additive across ranges can be specified.

THE MDDB COMPARED WITH THE DATA SET

Data Consistency and Integrity Using PROC MDDB greatly simplifies the process of the database build, and does not involve any mathematical calculation on the part of the programmer. There is therefore much less opportunity for error. The structuring and calculation processes of PROC MDDB are highly integrated and standardized and so there is not a problem of rounding error differences. Programmed calculations done by hand can result in two means for the same data being slightly different because of rounding errors. A MDDB is much simpler to make regular updates to with new data using PROC MDDB instead of using a DATA step or PROC SQL to update a data set.

Less Code An MDDB procedure uses only a few lines of easy to read code and not many comments are needed to explain what is being done. Usually a DATA step or a PROC SQL requires more complex code to achieve the same goal. Using a MDDB is therefore relatively easy to learn and easy to teach.

Space Savings Precalculation and elimination of redundancy as well as data compression and internal indexing reduce storage media consumption, memory page faults, and make good use of buffer caching.

Speed The above advantages result in a much faster build time for the MDDB than for a Data set containing the same data with the same precalculations. Run time reading of the MDDB is generally between 5 and 20 times faster. For example a SAS/EIS presentation written using data sets takes between 15 seconds and two and a half minutes to load a new screen of data. That same SAS/EIS presentation using the same data, but stored in MDDBS has corresponding data load times of between 3 and 25 seconds.

Portability A SAS MDDB is generally creatable and portable across common platforms, but further investigation and documentation of MDDB portability issues is needed.

Labels and Formats Both labels and formats can be used on the dimensions and variables within a MDDB. The MDDB itself can have a label.

Security A MDDB may be password protected just like a SAS data set.

Sorting A MDDB may be ordered by its dimension values in ascending or descending sequence using actual or formatted key values when the MDDB is built or updated. PROC MDDB may be used to change the sort order. (PROC SORT cannot be used on a MDDB). Each dimension variable has its sort order defined independently. The sort order may be specified as being the same as that defined for the input data set.

Hierarchies A MDDB may have one or more sub MDDBs defined within it. These sub-databases are named hierarchies, which are handled in exactly the same manner as a complete MDDB. In fact the complete MDDB is the 'NWAY' (all dimensions) hierarchy.

Limits Due to the very integrated and compact nature of a MDDB large numbers of dimensions and hierarchies may be used with very large volumes (> 10Mb) of data. Table 1 shows the comparison of PROC MDDB being used to build a MDDB and PROC SUMMARY being used to build a data set with the same structure and statistics from the same input data set. In each case the statistics calculated are *n, sum, max*, and *min*. A 'dimension' corresponds to a CLASS or BY variable in a SAS procedure. A 'variable' corresponds to a VAR variable.

Table 1 PROC MDDB v PROC SUMMARY Benchmark Test Results

Original Data set				PROC MDDB Creation		PROC SUMMARY Creation	
Size (Mb)	Number of Observations	Number of Dimensions	Number of Variables	Time (secs)	Size (Mb)	Time (secs)	Size (Mb)
44.2	1,000,000	2	1	31	16.6	42	33.0
8.6	10,000	10	10	11	4.0	366	6.1
8.6	10,000	50	50	97	18.5	<Size limit Exceeded>	

Accessibility An MDDB is readable using the SAS Display Manager and is managed by most (but not all) existing SAS utilities such as PROC DATA SETS. A MDDB cannot be read by PROC SQL or be used like a Data set. Neither can a MDDB be a 'view' of changing data. Unless the SASSFIO engine is used (see below), a MDDB cannot be input to a DATA step with a SET, MERGE, or MODIFY statement because a MDDB does not contain ordered observations like a data set. However, an MDDB is easily built or updated from a data set. The SAS MDDB is primarily designed for use with SAS/EIS for the display of data in objects such as charts and graphs, ideal for live presentations. SAS/EIS regards a MDDB (or a hierarchy

within a MDDB) as a data object in the same manner as a data set. However, there are two other ways data can be extracted from a MDDB:

- Using Screen Control Language (SCL), which is ideal for SAS/AF® Frames applications.

- Using the SASSFIO engine, specifying

"LIBNAME <LIBREF> SASSFIO <PATH> <HIERARCHY>

where LIBREF is the name used to reference the MDDB location, PATH is the directory path (host operating system dependent) to the MDDB file or one of its hierarchies (see below), and HIERARCHY is the name of the hierarchy or the whole MDDB.

Documentation The SAS MDDB is fully documented in the SAS Help facility.

DATA PREPARATION

The structure of each MDDB and its role in any larger data management system such as a data warehouse must be well defined. Also the input data must be contained in a known structure in a data set which will be reused with different data each time the MDDB is updated. Particular issues are:

1) Which variables and statistics to use.

2) What happens to any error in the input data as observations are collated together and the statistics are calculated. Small errors in one of the variables may, on average, cancel out when a large number of observations are accumulated. However, a small rounding or truncation error may be amplified when many thousands of observations with that error are accumulated. For example if 10000 observations have a variable X, which should be an integer but is instead 0.01 too small (a 10 is stored as 9.99) the resulting sum would be 9,990 instead of 10,000, short by a factor of 10, this could mean a wrong figure is taken as a maximum, or a total is reported as 9,995 instead of the correct total of 10,000.

3) The use of formats and labels. These should be defined for the MDDB, even though formats and labels from an input data set are taken as defaults. The input data set attributes may be changed on a future update.

4) Data volatility. The data must be historic in nature and should not be changed once the MDDB has been built. Any erroneous data should be corrected or filtered out before the MDDB update stage. Though corrections to previous input data errors may be applied at the next update.

THE SAS MDDB PROCEDURE

The general syntax of the MDDB procedure is:

```
PROC MDDB DATA = <data set>
            IN = <existing MDDB>
           OUT = <new MDDB>
         LABEL = "<Label text>"
      PASSWORD = "<Password>"
;
   CLASS <var1> <var2> .../ <sort options>
;
   HIERARCHY <var1> <var2>...
/NAME="<name>"        DISPLAY= Yes/No
;
   VAR <numvar1> <numvar2>.../<stat-
options> ;
RUN;
```

The DATA *parameter Data Set* is the SAS dataset containing the source data to be used in creating or updating the MDDB. Data set options such as WHERE, DROP, and KEEP can be used in parenthesis after the data set name. Two or more input data sets with corresponding variables and variable types may be specified. The input data set(s) are unchanged by PROC MDDB.

The IN parameter *Existing MDDB* specifies an existing MDDB to used in the creation of the new MDDB. This IN MDDB must be of the same structure (e.g. An earlier version of the database) as the new MDDB being created. If an IN MDDB is specified there must be no other MDDB definition statements such as CLASS, HIERARCHY, or VAR (see below) since the new MDDB will inherit all the characteristics, including the labels and formats, of the existing MDDB. Two or more such input MDDBs may be specified. The input MDDBs are unchanged by PROC MDDB.

Either a DATA = data set or an IN = MDDB must be specified. If both of these parameters are present in the same PROC MDDB the new MDDB is updated with data from both the data set and the input MDDB.

The LABEL is optional and specifies a label, of up to 40 characters in length, for the MDDB being created.

The PASSWORD is optional.

The CLASS statement defines one or more class (dimension) variables. This statement has the same meaning as the CLASS statement in PROC SUMMARY and PROC TABULATE. In the resulting MDDB the totals and overall statistics are shown for the analysis variables within each class. The hierarchy of the class variables is the order in which they are specified in the CLASS statement(s). There must be at least one class variable and there may be two or more CLASS statements, this is

required if different class variables are to have different sort options. Class variables may be alphabetic or numeric. The sort options define the sort order within the class, the default being alphanumeric or the sort order of the IN = MDDB if there is one. One of the following sort options can be used with each CLASS statement:

ascending	Ascending alphanumeric
descending	Descending alphanumeric
ascformatted	Ascending by format value
desformatted	Descending by format value
dsorder	The same order as in the input dataset

The HIERARCHY statement defines a separate class subtable. More than one such subtable may be defined by using multiple HIERARCHY statements. HIERARCHY is the same as CLASS except the sub table is stored as a separate table in the same MDDB. The HIERARCHY variables must also be present in a CLASS statement, but any CLASS variable cannot be used more than once in any given HIERARCHY statement. The HIERARCHY options NAME and DISPLAY enable the subtable to have a user defined name. If NAME is not specified the default values of HIER1, HIER2, and so on are used for each subtable in the order they are assigned. If the DISPLAY = YES option is used the hierarchy can be viewed with the SAS BROWSE window (Refer to Browsing a MDDB below).

The VAR statement defines one or more analysis variables for statistics to be precalculated. All variables must be numeric, multiple VAR statements can be used, but each variable cannot appear more than once in all the VAR statements. More than one VAR statement is needed when different statistics options are used. The available statistics options are N, SUM, MAX, MIN, NMISS, USS, SUMWGT, UWSUM, and WEIGHT = <numeric variable in the input dataset>. (SUMWEIGHT is only calculated if WEIGHT is specified). If no statistics options are specified the default of SUM is used. Missing values are produced when there are no observations in the input dataset corresponding to the row and column (cell) in the MDDB. With the exception of the NMISS statistic missing values are ignored, but if all of the values of the analysis variable are missing for the class the resulting values of the statistics are missing, except for NMISS which shows the total number of the missing values. The aggregated cells holding compound statistics for each hierarchy are stored at rows and columns labeled _TOTAL_.

EXAMPLE

```
PROC MDDB DATA=LEDGER01 OUT=JANSALES
   LABEL='January Sales' ;
   CLASS STORE BRAND PRODUCT DATE ;
   HIERARCHY STORE DATE / NAME =
      STORES DISPLAY = YES
   VAR SALES REVENUE / N SUM MAX MIN ;
RUN ;
```

The data set named LEDGER01 contains sales figures for the month of January in the variables STORE, BRAND, PRODUCT, DATE, SALES, and REVENUE. SALES is the total number of units sold for each product in each brand at each store each day in the month of January. This PROC MDDB takes this data set and creates a MDDB labeled 'January Sales'. The overall hierarchy of the whole table (NWAY) uses the variables STORE, BRAND, PRODUCT, and DATE as the dimensions. The HIERARCHY statement produces a sub-table of SALES and REVENUE statistics involving just the STORE and the DATE only. This sub-table is named 'STORES' and the DISPLAY = YES option enables this sub-table to be displayed when the MDDB is browsed. The analysis variables are SALES (the number of units sold), and REVENUE, the price paid by each customer for each purchase. The statistics being generated for both the main database and any sub-tables are N (count), SUM, MAX, and MIN. The class variables are ordered in ascending sequence, for example STORE is ordered by store name, but sort options could be used accordingly.

If at the close of February's business February's sales are contained in the Data set LEDGER02 an MDDB of February's sales, FEBSALES, is produced in the same way, however if the management also require a Year-to-Date MDDB it can be created using PROC MDDB in either of the three ways shown in Table 2.

Table 2 Methods of combining data sets and MDDBs to form a compound MDDB.

```
    PROC MDDB DATA=LEDGER01 LEDGER02
       OUT=YTDSALES
       LABEL='Year to Date Sales' ;
       CLASS STORE BRAND PRODUCT DATE ;
       HIERARCHY STORE DATE / NAME =
          STORES DISPLAY = YES
       VAR SALES REVENUE/N SUM MAX MIN;
    RUN ;

or: PROC MDDB DATA=LEDGER02 IN=JANSALES
       OUT=YTDSALES
       LABEL='Year to Date Sales' ;
    RUN ;

or: PROC MDDB IN=JANSALES FEBSALES
       OUT=YTDSALES
       LABEL='Year to Date Sales' ;
    RUN ;
```

In the first case the MDDB is made from two similar input data sets, and the structure of the MDDB must be fully defined. In the second case the MDDB is made by updating the MDDB JANSALES with the data in the

LEDGER02 data set and storing the resulting MDDB in YTDSALES. This resulting MDDB inherits its structure definition from JANSALES, except for the label which is overwritten by the new definition. (CLASS, HIERARCHY, and VAR statements cannot be specified when there is an input MDDB). In the third case the JANSALES and FEBSALES MDDBs are combined into the resultant MDDB YTDSALES. An MDDB for the entire year's sales could be built in this way using all twelve months of data. In all cases the input DATA data sets and the IN MDDBs are left unchanged.

READING DATA FROM A SAS MDDB

An MDDB, once registered in SAS/EIS, is ready for use with SAS/EIS. A second way to read a MDDB is to create a data set from it using SCL, however the easiest way to read a MDDB from base SAS is to use the SASSFIO engine. If the MDDB JANSALES in the above example was in the UNIX directory /COMPANY/RETAIL the following statements effectively define the HIERARCHYs within JANSALES as data sets:

```
LIBNAME SALEINFO  SASSFIO
'/COMPANY/RETAIL/JANSALES.SSM01'  ;
```

NB. SSM01 is the file type/extension and is host operating system dependant.

The whole MDDB is the NWAY HIERARCHY, so it is referenced as such, e.g.

```
PROC PRINT DATA=SALEINFO.NWAY ;
RUN;
```

To access a sub-table as a data set it is referred to using its HIERARCHY name with the library reference , e.g.

```
PROC PRINT DATA=SALEINFO.BYSTORE;
RUN;
```

BROWSING A MDDB

A MDDB is shown in the SAS DIR window with the MDDB name as the SAS file and 'MDDB' as the filetype. Entering 'L' next to the MDDB displays its contents in the SAS BROWSE window. The rows and columns are displayed in the order defined in the order options of each CLASS statement. Formatted values and labels are shown where they are defined. By default the smallest HIERARCHY table is displayed. Clicking the right mouse button displays each HIERARCHY sub-table in turn, including the whole NWAY table. Totals and sub-totals are displayed for each class or hierarchy level including the grand totals for the whole NWAY table. Such total columns and rows are labeled as '_TOTAL_'. The statistical columns are labeled as _ANLSYS_ with the appropriate statistic notation such as N, MAX, MIN accordingly. In version 6.12 an MDDB cannot be read or changed by using FSEDIT.

VIEWING THE STRUCTURE OF A SAS MDDB

When in the SAS DIR window entering a 'S' next to the MDDB name displays the internal structure of the MDDB and it's variables. PROC CONTENTS cannot be used to show the structure of a MDDB.

RENAMING OR DELETING A SAS MDDB

To rename or delete an MDDB the PROC DATASETS procedure can be used. The following example renames the MDDB OLDMDDB to NEWMDDB and deletes the MDDB TMPMDDB.

```
PROC DATASETS   NOLIST   LIB = WORK ;
    CHANGE   OLDMDDB  =  NEWMDDB ;
    DELETE   TEMPMDDB ;
QUIT ;
```

Alternatively, the Rename and Delete options can be used in the SAS/EIS Metabase.

CONCLUSIONS

The MDDB is an excellent data structure for handling very large amounts (greater than 1Mb) of data. It is particularly good for SAS/EIS, since the response time when running SAS/EIS is markedly improved over using data sets. At the same time the convenience of SAS/EIS for preparing reports and charts is maintained. However, the speed and response times for SAS/EIS when using the MDDB need to be improved still further. Many organizations have been reluctant to use SAS/EIS due to its slowness when reading from data sets. PROC MDDB provides a convenient way to create an MDDB from other data sources and other SAS applications, and being able to browse the MDDB means developers can validate the information it contains. However, investigation needs to be done into creating an MDDB from a remote data source, and into portability issues, but since MDDBS are quick to create they can be created as temporary structures at application run time, unless very large numbers of variables (more than about 10 class and analysis variables) are to be used. The SAS MDDB is still to some extent in an experimental phase, in future releases it is likely to be developed and documented further, particularly regarding cross-platform compatibility. In future releases of SAS the MDDB is likely to become more accessible with many SAS tools and packages such as SAS/ACCESS® and SAS/CFO Vision®.

ACKNOWLEDGMENTS

The following provided information, advice, and assistance in preparing this paper:

Thomas Beauregrad, Trilogy Consulting Corporation
Andrew Norton, Trilogy Consulting Corporation
SAS On-Line Support Services
'The Data Warehouse Toolkit' book by Ralph Kimball PhD,John Wiley & Sons Inc. New York, 1996.

Timothy J. Harrington, Trilogy Consulting Corporation, 5148 Lovers Lane, Kalamazoo, MI 49002 (616) 344 1996.
TJHARRIN@TRILOGYUSA.COM