CHAPTER

3

Building and Updating MDDBs

Analyzing Your Data

The key to creating an efficient MDDB is thorough analysis of both your data and its users. Armed with this analysis, you can determine

- the best order for the classification variables in the NWAY cube
- whether or not you should create subcubes
- which classification variables should be included in the subcubes.

While it is possible to create an MDDB that consists of the NWAY cube only, this approach would require that the NWAY cube fulfill all requests made by OLAP clients. While this approach saves on storage space, you can improve query performance if you create subcubes based on anticipated client requests. In order to create an MDDB that requires the least storage space while providing users with the best response time, consider the following issues:

- the warehouse data and how it will be used in the OLAP application

- the business problem and user expectations
- the need for an iterative approach.

By choosing and ordering classification variables for the NWAY cube and subcubes based on the needs of users, you will create an MDDB that meets the functional priorities dictated by your organization's business requirements.

Creating useful and efficient MDDBs is best accomplished as an iterative process that fits into the overall data warehouse and business intelligence strategies of your organization. You can analyze the usage patterns to determine whether your MDDB is defined correctly and make adjustments if necessary. For example, if a defined subcube is rarely or never accessed, it can be safely removed from the MDDB creation. Alternately, if there are subcubes that are not defined but are frequently requested, they can be added to the MDDB creation.

Using a Spiral Diagram to Order the Classification Variables

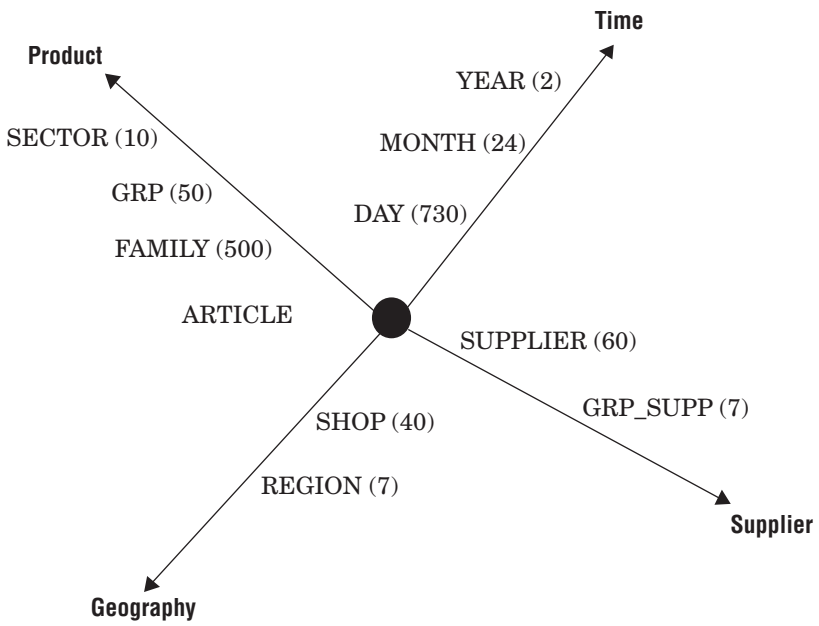
One way you can create an initial MDDB that will meet most users' requirements and performance expectations is to analyze your data by using a spiral diagram. By placing the classification variables on axes representing the hierarchies of the MDDB, you can develop an acceptable data model for the MDDB.

Imagine that you have a base table that contains retail sales information. The classification variables in this table can be grouped into four *dimensions*, or groups of variables with similar characteristics:

- product (SECTOR, GRP, FAMILY, ARTICLE)
- time (YEAR, MONTH, DAY)
- geography (REGION, SHOP)
- supplier (GRP_SUPP, SUPPLIER).

First, create an axis for each dimension. Then, place the classification variables on the appropriate axes (working from the outside to the center) in ascending order of cardinality (number of unique values), as shown in the following diagram. Each variable becomes a *dimensional level* for that dimension, with the outermost variable at the *top dimensional level*.

Figure 3.1 Axis Diagram

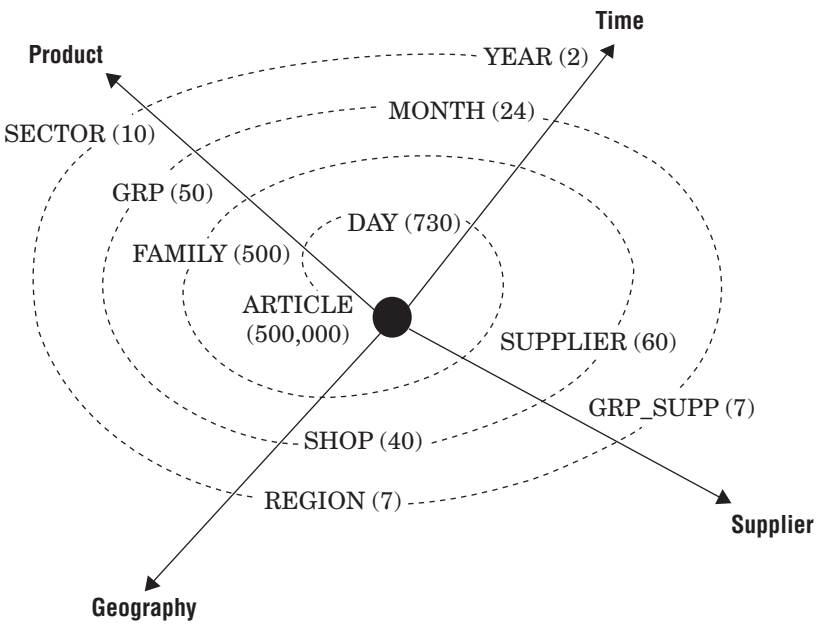


The placement of the axes in relation to each other can be significant. You might want to try several arrangements to find one that works best for you. The following are possible arrangements:

- arrange axes in descending order of likelihood of use
- arrange axes in descending order of cardinality at the top dimensional level
- arrange axes in descending order of number of levels.

Now you can draw a spiral on the diagram that will indicate a general ordering scheme for the variables. Starting on the outside with the classification variable with the lowest cardinality that is also the one most likely to be of interest to users, draw a line from this variable to an adjacent variable on the diagram. Continue in this manner, spiraling in toward the center. As in this example, you might have to deviate slightly from this pattern when multiple variables near the center have high cardinality. Here, the spiral is drawn so that FAMILY is after SUPPLIER instead of DAY.

Figure 3.2 Spiral Diagram



You can then produce an ordered list of classification variables based on the diagram. For this example, the order would be

YEAR
SECTOR
REGION
GRP_SUPP
MONTH
GRP
SHOP
SUPPLIER
FAMILY
DAY
ARTICLE

From this list, you can develop an intelligent choice of crossings. Start with the whole list and successively drop the last item in a "stair-step" fashion to form a new crossing, as follows:

```
YEAR SECTOR REGION GRP_SUPP MONTH GRP SHOP SUPPLIER FAMILY DAY ARTICLE
YEAR SECTOR REGION GRP_SUPP MONTH GRP SHOP SUPPLIER FAMILY DAY
YEAR SECTOR REGION GRP_SUPP MONTH GRP SHOP SUPPLIER FAMILY
YEAR SECTOR REGION GRP_SUPP MONTH GRP SHOP SUPPLIER
YEAR SECTOR REGION GRP_SUPP MONTH GRP SHOP
YEAR SECTOR REGION GRP_SUPP MONTH GRP
YEAR SECTOR REGION GRP_SUPP MONTH
YEAR SECTOR REGION GRP_SUPP
YEAR SECTOR REGION
YEAR SECTOR
```

```
YEAR
```

From this list of crossings, you can define the NWAY cube and as many subcubes as you want.

MDDB Memory Optimization

The primary factors affecting reporting performance using SAS/MDDB Server software are population of crossings and the storing of totals, but you can also optimize performance and tune the server to your particular environment by specifying how much memory the MDDB should be able to use during creation and reporting. This capability allows you to limit the resources that an individual MDDB request will take. You specify the memory limitations using the VMEMSIZE= and PKTSIZE= options in the MDDB procedure for creation of and reporting from MDDBs. See "Building an MDDB with the MDDB Procedure" on page 17 for details on MDDB procedure syntax.

The PKTSIZE= parameter determines the size of an internal heap that is used for storing analysis and statistic data nodes for the MDDB. The size of this internal heap also determines the maximum number of data nodes that can be written to disk at any one time. Ideally, this parameter should be set to a size that allows 1 to 10% of the total nodes in the NWAY cube to be stored in a single heap.

The VMEMSIZE= parameter determines the maximum amount of memory that is used for storing analysis and statistic data in memory; data that cannot be maintained in memory is written to temporary space in the WORK library. When you specify a non-zero value for the VMEMSIZE= parameter, analysis and statistical data is written to temporary disk space in a first in/first out fashion, as necessary to keep data memory usage at or below the specified value. Specifying a value for the VMEMSIZE= parameter could adversely affect performance for building and reading MDDBs, but in most cases, it will allow you to build and access larger data cubes than would be possible using the default value of 0.

Stored and Derived Statistics

When you create an MDDB, you have to decide which statistics should be stored with the MDDB. The statistics that can be stored with an MDDB are a minimum sufficient set that can be used to calculate other statistics when the MDDB is accessed. You can specify up to eight statistics for each analysis variable to be stored with the MDDB:

- N
- SUM
- SUMWGT
- UWSUM
- NMISS
- USS
- MIN
- MAX

Depending on which of the eight statistics are stored, up to 13 additional statistics can be calculated by SAS/MDDB Server software at run time, allowing for up to 21 available statistics. The following table indicates which statistics can be derived and which stored statistics are required:

Table 3.1 Stored and Derived Statistics

Statistic	Required Stored Statistics
NMISS	NMISS
SUMWGT	SUMWGT (available only if a WEIGHT is specified)
SUM	SUM (However, if a WEIGHT is specified, then SUM is the weighted sum.)
AVG	N, SUM (However, if a WEIGHT is specified, then SUMWGT and SUM are required instead.)
UWSUM	UWSUM
MIN	MIN
MAX	MAX
USS	USS
RANGE	MIN, MAX
PCTN	N
PCTSUM	SUM
CSS	N, SUM, USS (However, if a WEIGHT is specified, then SUMWGT, SUM, and USS are required instead.)
VAR, STD, STDERR, CV, T, PRT, LCLM, UCLM	N, SUM, USS (However, if a WEIGHT is specified, then SUMWGT is also required.)

See “VAR Statement” on page 19 for details on specifying a WEIGHT and using the SUMWGT statistic.

Building an MDDB

There are four ways that you can build an MDDB. You can use

- the MDDB procedure
- SAS/EIS software
- the SAS/MDDB Server classes
- SAS/Warehouse Administrator software.

This section provides instructions on how to build an MDDB by using each of these methods. You can choose one method over another based on the SAS software products that you use and with which you are most familiar.

Note: Regardless of the method that you choose, the information that you specify when creating an MDDB is similar to that specified when using the SUMMARY procedure. The NWAY cube correlates to the NWAY data produced by the SUMMARY procedure, and subcube data correlates to _TYPE_ records produced by the SUMMARY procedure. If you are already familiar with the SUMMARY procedure, keeping these similarities in mind could help you understand how to create MDDBs. △

Once an MDDB has been created, you can copy or transport it to any platform that supports the MDDB object type. For information on copying or transporting MDDBs, refer to "Transporting MDDBs Across Operating Environments" on page 31.

Building an MDDB with the MDDB Procedure

This section provides the syntax for the MDDB procedure and explains how to use the procedure to create an MDDB. An example PROC MDDB statement is also provided.

PROC MDDB <*option(s)*>;

CLASS *var1 var2 ...* / <*order-options*>;

HIERARCHY *class-var1 class-var2 ...* / <NAME=*name*|"*name*" DISPLAY=YES|NO|NODATA TOTALS=YES|NO>;

VAR *var1 var2 ...* / <*stat-options*>;

ADDHIER *class-var1 class-var2 ...* / <NAME=*name*|"*name*" DISPLAY=YES|NO|NODATA> <TOTALS=YES|NO>;

REMOVEHIER*class-var1 class-var2 ...* / <NAME=*name*|"*name*">;

PROC MDDB Statement

PROC MDDB <*option(s)*>;

You can use the following options in the PROC MDDB statement:

DATA= *dsname*
Use the DATA= option to specify the name of a SAS table that is to be used as the source for the MDDB. If you do not specify a table name, _LAST_ is used.

OUT= *libref.outmddb*
Use the OUT= option to specify the name of the MDDB that you are creating. The OUT= option is required.

IN= *libref.inmddb*
The IN= option is used during an incremental update of an MDDB; the name of the MDDB specified in the IN= option is the existing MDDB. The DATA= option is used to specify the name of the table that contains the incremental data that will be added to the data in the input MDDB and written out to the MDDB specified in the OUT= option.

LABEL= *description*
Use the LABEL= option to specify a description to be stored with the MDDB. The character description string can be up to 256 characters long. Enclose the description in quotes if it contains embedded blanks. This parameter is optional.

PW="*password*"
You can use the PW= option to specify a password that is to be associated with the MDDB. The password must be no more than eight characters and is not case-sensitive. Any passwords that are specified in the MDDB name will override the password specified as an option in the PROC MDDB statement. This parameter is optional.

You can specify read, write, and alter passwords using the same syntax as for data sets. For example, each of the following is valid:

```
out=libname.memname (pw     = "password"
                     read   = "read_password"
                     write  = "write_password"
```

```
                        alter  = "alter_password")
```

You can also specify the PWREQ= option to control whether a password requestor window appears when a required password is either missing or incorrect. By default, the password requestor window does not appear when creating an MDDB but does appear when reading an MDDB. To use the PWREQ= option, specify

```
pwreq = 'NO' | 'YES'
```

For example, you can specify the following code to ensure that a password requestor window appears when the required password is missing or incorrect:

```
libname.memname (pw    = "password"
                 pwreq = 'YES' )
```

Use the following as guidelines when you specify passwords:

- You can specify any combination of pw, read, write, and alter passwords.
- Spaces and quotes (single or double) are optional. The password is not case-sensitive.
- If you specify only **pw="password"**, that password is used for the read, write, and alter passwords.
- The read, write, and alter passwords override the pw password if the pw password is also specified.

VMEMSIZE=*msize*
The VMEMSIZE= option indicates the maximum amount of memory (in megabytes) used for keeping analysis and statistical data resident during the MDDB build or for each active cube at reporting time. The default value of 0 indicates no restriction. This parameter is optional.

PKTSIZE=*psize*
The PKTSIZE= option is used to specify the maximum amount (in kilobytes) of memory to be swapped at a time if a value has been specified for the VMEMSIZE= option. The block size should never go below the memory needed to hold the analysis and statistical data for a single node. The default value is 1024 KB. This parameter is optional.

TOTALS= YES|NO
A YES value for this option specifies that totals for the NWAY cube and all subcubes are stored with the MDDB, reducing reporting time but increasing the size of the MDDB and the time required to create it. The default value is NO. To store totals only for specific subcubes, use the TOTALS= option on the HIERARCHY statement and omit the totals option on the PROC MDDB statement.

CLASS Statement

CLASS *var1 var2 ...* / <*order-options*>;

Use the CLASS statement to specify variables from the base table that are to be used as the classification variables in the MDDB.

You can specify one or more CLASS statements. However, a given variable can only appear once in all CLASS statements. The class variable can be either numeric or character. If you do not specify a sort order, ASCENDING is used.

You can use the following options in the CLASS statement:

order-options
Use the *order-options* in the CLASS statement to specify the sort order for the classification variables. You can specify any of the following order options:

- ASCENDING
- DESCENDING
- ASCFORMATTED
- DESCFORMATTED
- DSORDER.

You can also specify a different sort order for each CLASS variable. To do this, use a separate CLASS statement for each variable to be sorted.

HIERARCHY Statement

HIERARCHY *class-var1 class-var2* ... / <NAME=*name* | *"name"* DISPLAY=YES | NO | NODATA> <TOTALS=YES | NO>;

You can define one or more subcubes to be stored in your MDDB by using a HIERARCHY statement. If you do not specify a hierarchy, only the NWAY cube hierarchy is stored in the MDDB. You can specify multiple CLASS variables; however, you can specify a CLASS variable only once in each HIERARCHY statement.

You can use the following options in the HIERARCHY statement:

NAME= *name* | *"name"*
Use the NAME= option to specify a name for your hierarchy. If the name contains a space or blank, it must be enclosed in quotes (see example, below).

If you do not specify a name for your hierarchy, the default name HIER *n* is used, where *n* is a number (beginning with 1).

DISPLAY= YES|NO|NODATA
This option will only have an effect at the time when someone chooses to register this MDDB in a SAS/EIS repository. At that time, a value of YES will be interpreted to mean that the specific hierarchy should be registered as a drill hierarchy. The default value of NO indicates that this hierarchy should not be specifically registered. The NODATA value means that only the SAS/EIS metadata is stored; no cell data is stored.

TOTALS= YES|NO
A YES value for this option specifies that totals for the hierarchy are stored with the MDDB, reducing reporting time but increasing the size of the MDDB. The default value is NO.

The following examples illustrate how to specify a subcube using the HIERARCHY statement:

```
hierarchy country region division /name=geo display=YES;

hierarchy country region division /name="geographic hierarchy";
```

Note: If you specify two or more identical hierarchies, SAS/MDDB Server stores only the first of the identical hierarchies and issues a warning that the duplicate hierarchies are not stored. △

VAR Statement

VAR *var1 var2* ... / <*stat-options*>;

The VAR statement enables you to specify variables from the base table to be used as the analysis variables in the MDDB.

You can specify one or more VAR statements. However, a given variable can only appear once in all VAR statements. The variables must be numeric. If you do not specify a statistic, SUM is used.

You can use the following options in the VAR statement:

stat-options

Use the *stat-options* in the VAR statement to specify the statistics to be stored for each analysis variable. Separate each statistic with a space. You can specify any of the following statistics options:

- MAX
- MIN
- N
- NMISS
- SUM
- SUMWGT
- USS
- UWSUM
- WEIGHT= numeric variable to use to weight the analysis variable

If you specify WEIGHT=, its value must be the name of a numeric variable in the data set. If you also specify SUMWGT, the weighted sum will be stored in the MDDB. If you specify only WEIGHT=, the weight will be used in calculating the SUM statistic, but the weighted sum will not be stored, and the other statistics that would be calculated based on the weighted sum will not be calculated (that is, they will have missing values).

If you specify SUMWGT but do not specify WEIGHT=, then the request to store SUMWGT will be ignored.

ADDHIER Statement

ADDHIER *class-var1 class-var2 ...* / <NAME=*name*|"*name*" DISPLAY=YES|NO|NODATA> <TOTALS=YES|NO>;

The ADDHIER statement enables you to update an MDDB by adding an hierarchy. The syntax and requirements for the ADDHIER statement are exactly the same as those for the HIERARCHY statement. The ADDHIER statement is valid only when updating an MDDB.You can have zero or more ADDHIER statements. See "Updating an MDDB Using the MDDB Procedure" on page 25 for details on the techniques for updating MDDBs.

REMOVEHIER Statement

REMOVEHIER *class-var1 class-var2 ...* / <NAME=*name*|"*name*">;

The REMOVEHIER statement enables you to update an MDDB by removing an hierarchy. The syntax and requirements for the REMOVEHIER statement are exactly the same as those for the HIERARCHY statement, except that there are no DISPLAY= or TOTALS= options on the REMOVEHIER statement. The REMOVEHIER statement is valid only when updating an MDDB. You can have zero or more REMOVEHIER statements.

If you specify the NAME option on the REMOVEHIER statement, only the hierarchy matching that name will be removed. Otherwise, all hierarchies containing only the specified classifiers will be removed.

See "Updating an MDDB Using the MDDB Procedure" on page 25 for details on the techniques for updating MDDBs.

Example 1: Building an MDDB Using the MDDB Procedure

This example shows you how to use the MDDB procedure to build an MDDB from the source table SASHELP.PRDSALE. The SASHELP.PRDSALE table contains the classification columns COUNTRY, REGION, DIVISION, PRODTYPE, PRODUCT, QUARTER, YEAR, and MONTH. The analysis variables are ACTUAL and PREDICT. Based on logical assumptions about how users would want to drill down through the data, you can write a PROC MDDB statement to create the correct MDDB with multiple subcubes that will meet anticipated user requests.

```
proc mddb data=sashelp.prdsale out=sasuser.mddb
      label='MDDB from SASHELP.PRDSALE';
   class product prodtype year quarter month country region division;
   hierarchy country region division /name="Geographic Hierarchy";
   hierarchy product year /name="Product-Time Hierarchy";
   hierarchy year quarter month;
   var predict /sum;
   var actual /n nmiss sum uss min max;
run;
```

The resulting MDDB is called SASUSER.MDDB. The NWAY cube contains each of the classification variables. One analysis variable, PREDICT, has the statistic SUM; the other analysis variable, ACTUAL, has the statistics N, NMISS, SUM, USS, MIN, and MAX. The HIERARCHY statements create subcubes that optimize drill-down performance. No matter where a user is in any of the drill hierarchies, there is a subcube with related aggregations.

Building an MDDB with SAS/EIS Software

This section provides instructions on how to use SAS/EIS software to build an MDDB. You supply information about the MDDB in a series of SAS/EIS windows. When you build an MDDB using SAS/EIS software, the MDDB will be registered automatically in the SAS/EIS metabase facility.

To build an MDDB with SAS/EIS software, you must first register the detail data in a SAS/EIS repository. The types of SAS/EIS reports that are produced from the MDDB will help you determine how to register the detail data. Your registration should contain columns defined with the CATEGORY and ANALYSIS attributes and can contain the HIERARCH table attribute. See the SAS/EIS software online Help for more details on the data requirements for specific reports.

Once you have determined the data requirements, register the detail data in a SAS/EIS repository. Then follow the steps below to build the MDDB.

1. Invoke SAS/EIS software, and double-click **Build EIS** in the EIS Main Menu.
2. In the Build EIS window, specify a path and an application database, if you have not previously done so. Select Add.
3. In the Add window, select **Data Access** from the Object Databases list box. Then select **Multidimensional database** from the Objects list box and select Build. The Multidimensional Database window appears, where you enter all the information needed to create an MDDB.
4. In the Multidimensional Database window, type a name and description in the **Name** and **Description** fields. Then select the right arrow beside the MDDB field to open the MDDB window.

5 In the MDDB window, specify information on where to save and register the MDDB that you are creating. You must register the MDDB in a repository. Use the down arrow beside the **Repository** field to specify a repository. You can also add password protection to the MDDB in this window. Select OK to return to the Multidimensional Database window.

6 Select the right arrow beside the **Table** field to open the Select Table window, where you specify the registered table to be used as input for the MDDB. Select the detail data that you registered in the repository. Select OK to return to the Multidimensional Database window.

7 Select the right arrow beside the **Dimensions** field to open the Column Selection window, where you select the dimension and analysis columns. Select OK to return to the Multidimensional Database window.

8 Select Create to build the MDDB. You will receive a message indicating that the MDDB has been successfully built. You can now specify your MDDB (instead of a table) in the objects that use MDDBs as input. If you do not select Create, the MDDB is not created or registered until the EIS application runs.

Note: Re-executing the MDDB application, by editing the MDDB and selecting Create or Test from the Build EIS window, or by using the RUNEIS APPL=*eis-app-name* command, will cause the MDDB to be re-created, overwriting any previous changes. △

Building an MDDB with the SAS/MDDB Server Classes

SAS/MDDB Server includes two classes that you can use to create MDDBs:

- MDDB Class
- MDDB_C Class.

Note: Two additional classes, MDDB_H and MDDB_M, are provided to help you work with MDDBs. The MDDB_H class reads an existing MDDB and returns header information. The MDDB_M class reads and returns data from the MDDB. For details on these classes, see the SAS/MDDB Server online Help. △

This section summarizes the functionality of the two classes that allow you to create MDDBs. For complete documentation of the classes, refer to the SAS/MDDB Server online Help.

MDDB Class

The MDDB class reads and summarizes a SAS table and stores the minimum sufficient set of summarized data in an MDDB library member. The methods specific to the MDDB class are

_handleError
handles errors that might occur during MDDB processing.

_summary
summarizes a table and creates the MDDB.

_updateMddb
updates an MDDB with the latest information specified in the table and in the original MDDB. You can also add and remove hierarchies.

MDDB_C Class

The MDDB_C class enables you to create an MDDB from any data source. You can use this class to create an MDDB that is

- not dependent on a particular data source (see "Creating an MDDB: Cell-by-Cell" in the SAS/MDDB Server online Help under "MDDB_C Class")
- dependent on a particular data source—that is, an MDDB built from a data set created by PROC SUMMARY (see "Creating an MDDB: Using a SUMMARY Data Set" in the SAS/MDDB Server online Help under "MDDB_C Class").

The methods specific to the MDDB_C class are

_addNode
: adds a node to the currently open cube.

_closeCube
: closes the current open cube.

_closeMddb
: closes the MDDB and stores it on disk.

_defineClass
: defines a classifier that is specified in _openMddb.

_fillFromSummaryDS
: fills an MDDB with data from a summary table.

_handleError
: handles errors that might occur during MDDB processing.

_isMddbComplete
: returns a value that indicates whether the minimum amount of data has been entered such that a _closeMddb method can be called.

_isMddbOpen
: returns a value that indicates whether an MDDB is open.

_openCube
: opens a cube specified in _openMddb and adds nodes to it.

_openMddb
: opens an MDDB and sets up basic header information.

Example 2: Building an MDDB Using the MDDB Class

The SCL code in this section shows how you could build the same MDDB as in "Building an MDDB with the MDDB Procedure" on page 17 using the MDDB class.

```
/*-- load the MDDB class to create the MDDB entry from data set--*/
/*-- using the CLASS instead of the PROC --*/
dcl object dataid=_new_ sasshelp.mddb.mddb();

init:

   /*-- create classification variables list --*/
   classlist=makelist();
   rc=setnitemc(classlist, 'ASCENDING', 'PRODUCT');
   rc=setnitemc(classlist, 'ASCENDING', 'PRODTYPE');
   rc=setnitemc(classlist, 'ASCENDING', 'YEAR');
   rc=setnitemc(classlist, 'ASCENDING', 'QUARTER');
```

```
   rc=setnitemc(classlist, 'ASCENDING', 'MONTH');
   rc=setnitemc(classlist, 'ASCENDING', 'COUNTRY');
   rc=setnitemc(classlist, 'ASCENDING', 'REGION');
   rc=setnitemc(classlist, 'ASCENDING', 'DIVISION');

   /*-- create hierarchies/subcubes--*/
   hlist=makelist();
   h2list=makelist();
   rc=insertc(h2list, 'COUNTRY', -1);
   rc=insertc(h2list, 'REGION', -1);
   rc=insertc(h2list, 'DIVISION', -1);
   rc=setniteml(hlist, h2list, 'GEOGRAPHIC HIERARCHY');

   /*-- create hierarchies/subcubes --*/
   h2list=makelist();
   rc=insertc(h2list, 'PRODUCT', -1);
   rc=insertc(h2list, 'YEAR', -1);
   rc=setniteml(hlist, h2list, 'PRODUCT TIME HIERARCHY');

   /*-- create hierarchies/subcubes --*/
   h2list=makelist();
   rc=insertc(h2list, 'YEAR', -1);
   rc=insertc(h2list, 'QUARTER', -1);
   rc=insertc(h2list, 'MONTH', -1);
   rc=insertl(hlist, h2list, -1);

   /*-- setup analysis and applicable stats --*/
   alist=makelist();
   a2list=makelist();
   rc=insertc(a2list, 'SUM',-1);
   a3list=makelist();
   rc=insertc(a3list, 'N',-1);
   rc=insertc(a3list, 'NMISS',-1);
   rc=insertc(a3list, 'SUM',-1);
   rc=insertc(a3list, 'NMISS',-1);
   rc=insertc(a3list, 'USS',-1);
   rc=insertc(a3list, 'MIN',-1);
   rc=insertc(a3list, 'MAX',-1);
   rc=setniteml(alist, a2list, 'PREDICT');
   rc=insertl(alist, a3list,  -1, 'ACTUAL');

   put 'Creating mddb: sasuser.mddb';

   dataid._summary('SASHELP.PRDSALE',    /*-- dataset name --*/
                   'SASUSER.MDDB',       /*-- mddb name    --*/
                   classlist,            /*-- list  of     ---*/
                                         /*-- classification vars--*/
                   hlist,                /*-- hierarchies list --*/
                   alist,
                   "MDDB from SASHELP.PRDSALE");
   rc=rc;
return;
```

Building an MDDB with SAS/Warehouse Administrator Software

SAS/Warehouse Administrator software provides a graphical user interface that enables you to specify the classification variables, analysis variables, and summary levels (hierarchies) for an MDDB. Additionally, you specify a location for storing the MDDB and other information specific to features of SAS/Warehouse Administrator software. For details about how to create an MDDB using SAS/Warehouse Administrator software, refer to the *SAS/Warehouse Administrator User's Guide*.

Updating an MDDB

It is not unusual to have gigabytes of source data that need to be summarized. Summarizing this volume of data can take hours or even days, depending on the type of CPU and other factors surrounding the data. Therefore, it is important to be able to update an MDDB with new data without having to completely rebuild the MDDB. SAS/MDDB Server software allows for incremental updates.

When you update an existing MDDB, remember that you cannot change the basic structure of the MDDB. You cannot add a classification variable, analysis variable, or statistic that was not in the original MDDB definition. If you want to perform any of these tasks, you will need to rebuild the entire MDDB from the original detail data. You can, however, performance-tune your MDDB by adding hierarchies (subcubes) that more closely match user data requests or delete hierarchies that are never requested.

This section discusses how to update an existing MDDB using the MDDB procedure, SAS/EIS software, and the MDDB class. Regardless of the method you use, an MDDB will be created containing the new data. During this process, the old MDDB must still exist, so make sure there is enough disk space before the update is attempted. When the process is complete, you must copy the new MDDB into a permanent library (if it is not already in one), delete the old MDDB, and then rename the new MDDB so that it has the same name as the old MDDB. If you do not give the new MDDB the name of the old MDDB and you are using the MDDB in SAS/EIS applications, the MDDB will not be recognized by your SAS/EIS metabase registrations.

Note: No explicit action is required to update an MDDB using SAS/Warehouse Administrator. When an MDDB is defined as part of the data flow of your data warehouse, it will be automatically updated by SAS/Warehouse Administrator software when the base table for the MDDB is updated. △

Updating an MDDB Using the MDDB Procedure

To update an existing MDDB using the MDDB procedure, you must specify the existing MDDB using the IN= option, specify the table that contains the update data using the DATA= option, and specify the name of the updated MDDB using the OUT= option. Note that the values of the IN= and OUT= options must be different. You can also use the LABEL= option to specify a label for the updated MDDB and the ADDHIER and REMOVEHIER options to add and remove hierarchies from an MDDB. When you update an existing MDDB, the CLASS, VAR, and HIERARCHY statements are ignored.

"Example 1: Building an MDDB Using the MDDB Procedure" on page 21 shows how to create an MDDB called SASUSER.MDDB. To update that MDDB, you will use a PROC MDDB statement to create a new MDDB that contains the update data. Data for the new MDDB will come from the existing MDDB and from an updated base table.

In the following statement, the DATA= option specifies the name of the updated base table, the IN= option specifies the existing MDDB, and the OUT= option specifies the name of the new MDDB. Note that the LABEL= option is also used.

```
proc mddb data=sasuser.sales in=sasuser.mddb
   out=sasuser.mddbnew label='Updated sales MDDB: 15 Nov 1998';
run;
```

After the procedure runs, you would use the DATASETS procedure to delete the old MDDB and rename the new MDDB so that it has the same name as the old MDDB:

```
proc datasets library=sasuser;
   delete mddb /mt=mddb;
run;
   change mddbnew=mddb;
run;
quit;
```

Updating an MDDB Using SAS/EIS Software

SAS/EIS software provides a graphical user interface that enables you to incrementally update an existing MDDB. The update data must exist in a SAS table that contains the variables in the MDDB. The format, informat, length, and type of the MDDB variables must also match those on the table. Any SAS table variables that are not in the MDDB are ignored during the update.

The Incremental MDDB Updates window in SAS/EIS software enables you to specify

- a SAS name that is unique for the application type in the current application database
- an optional description that appears in the application windows and in application selection lists
- the name of the MDDB to be updated
- the repository in which the MDDB registration is stored
- whether the MDDB will have a read-protect password
- the SAS table used to update the MDDB.

For detailed instructions on how to update an MDDB using SAS/EIS software, refer to the SAS/EIS software online Help.

Updating an MDDB Using the MDDB Class

The _updateMDDB method of the MDDB class updates an MDDB with the latest information specified in the table and in the original MDDB. For complete documentation of the _updateMDDB method, refer to the SAS/MDDB Server online Help.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/MDDB® Server Administrator's Guide, Version* 8 , Cary, NC: SAS Institute Inc., 1999.

SAS/MDDB® Server Administrator's Guide, Version 8

ISBN 1–58025–504–3

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999