Paper 35-25

# *The SAS/MDDB® Shell Game: Options They Don't Show in SAS/HELP*

Blake Sanders, U.S. Census Bureau, Washington, D.C.
Mikhail Gruzdev, OAO, Greenbelt, MD

## Abstract

The SAS/MDDB product makes accessing data quicker and more efficient. Creating the files that work with SAS/MDDB, however, can be far from productive if you use an already mammoth data set. If the original data set is large enough, the creation of those files will take most of your available resources.

The solution is to combine the strategic segmenting of your original data set, the use of SAS® indexes and subsetting. By using these to create a SAS/MDDB file "on the fly", the quick and efficient access to data will remain the same while lowering the costs of production.

## Introduction

The Foreign Trade Division (FTD) of the U.S. Census Bureau compiles and publishes the U.S. trade statistics every month. From these statistics, the Data Dissemination Branch answers questions from the public and produces customized requests.

Much of the Data Dissemination Branch staff is clerical and cannot access the data without applications to produce "canned" reports. The flexibility gained by Data Dissemination due to the features of these SAS/MDDB applications has increased their productivity immeasurably.

The nature of trade data (large volume and numerous methods of classification) lead to the development of many such applications each of which was dedicated to a specific classification system. By producing a single SAS/MDDB file containing data for all classifications systems, a single application could be used. This would allow users unprecedented access to the data and flexibility in presenting it; however, the resources needed to create that file (given the original file's volume) are so enormous that it would not, in at least our case, be practical.

That does not mean it's impossible to create a single application with all of the benefits of SAS/MDDB.

## Basic Solution

The question became how to best circumvent the resource limitations of our situation. Creating the SAS data sets upon which the system could rely was done as a part of our monthly production. If we split those files along specific lines, much like telephone books for specific regions, it stood to reason that we could convert these smaller files. Taking that logic one step further, by subsetting these new files to include only what the user needed, we could definitely create a usable SAS/MDDB file. Creating the SAS/MDDB file "on the fly" was the way to go. This was confirmed when, upon further investigation, we realized that most of the requests would be fairly basic and could be done on a "standard issue" PC. With modest computing requirements, we designed the application to create the "on the fly" SAS/MDDB file on the user's hard drive.

The success of all of this grand planning, however, relied heavily on one thing: the ability of the Data Dissemination staff to know how to best perform a request. For example if you searched the phone book for all of the Smith families on Main Street, experience would tell you not to first search the entire book for everyone who lives on Main Street. Because the phone book is "indexed", it's much faster to look for the Smiths first. Likewise, we assumed that Data Dissemination would know best how to search for their data.

We included our own indexes in the data sets to cover those who might not consider this. By indexing the source files on all of the relevant category variables, the time to complete our subset would be as short as possible regardless of the person's experience. This improvement would not, however, be without cost. The extra space needed by the indexes was a small price compared to the space (disk and RAM) needed to produce the mammoth SAS/MDDB file that we originally considered.

With the theory mapped out, it was time to implement our plan of action:

- MAP OUT REQUESTS: By studying the way Data Dissemination's requests work, we were able to see the easiest way to segment the database and structure our design to take advantage of their work experience.
- SPLIT THE DATABASE: Smaller files take less time to search.
- CREATE AN INTERFACE: Provide a means for users to tell the application what they want.
- SUBSET DATA: Pull only what the user wants.
- LOCALLY CREATE SAS/MDDB FILE
- DISPLAY TO SCREEN

### Map out requests

By knowing how the user does their job, the task of designing software for them becomes easier. At first, our SAS/MDDB applications (one per classification system) mimicked the user's actions. Since SAS worked faster than the previous versions of their programs, Data Dissemination was more than happy.

Upon starting the "on the fly" SAS/MDDB database project (which later became known as "the flying MDDB"), we realized that many of the options built into the previous SAS/MDDB application were rarely used. Our research showed there was no need to build every answer into the database if only a small fraction would be used. Fortunately, the nature of SAS/MDDB files allowed us to include all of the data without precalculating all of the answers. Should the user request those answers, the SAS and the SAS/MDDB file could do the calculations as needed. Plus, with the size of the files we contemplated after segmenting and subsetting, any delay would be insignificant.

Speaking with Data Dissemination's employees, we found the common threads:

1.) "Static" Category variables: Two category variables appeared in every request
   a. Time period – the months or years for which the data described.
   b. Data type – imports or exports
2.) "Flexible" Category variables: These are the basis of all detailed trade data. One or more can appear in any request.
   a. Commodity – the product that is being imported or exported.
   b. Country – the country with which the U.S. is trading products
   c. District – the region of the U.S. where the goods entered or left the country. This variable is not available with all commodity classifications.
3.) Analysis variables:
   a. The availability of these varies depending on the request. Different types of value are always available. Quantity is only available under one classification system.

**Split the database**

Based on Data Dissemination's use, the data files were segmented according to time period, data type, and commodity classification. See Figure 1 for a representation.

Because of the variability in data use (with some categories being used more frequently than others) two versions were created for each of the segmented files: a "long" file and a "short" file. The "long" files contain all "flexible" category variables and all analysis variables. The "short" files contain the commodity and country "flexible" analysis variables as well as a small subset of analysis variables.

In the end, splitting the original data sets produced 240 files per full year of data. Each of these 240 files was small enough to be produced via base SAS while using a minimal amount of resources.

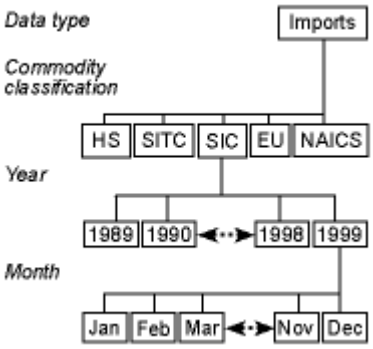


***Figure 1: Split the database according to data type, classification, year and month. Specifically, this shows the split for SIC imports for 1999.***

The naming convention for these files was simple. It utilized all of the factors upon which the data was segmented:

- Commodity classification:
  - HS (HS)
  - SITC (ST)
  - SIC (SC)
  - End-Use (EU)
  - NAICS (NA)
- Data type
  - Imports (I)
  - Exports (E)
- File type:
  - Long (L)
  - Short (S)
- Month
  - Jan (01)
  - Feb (02)
    …through…
  - Nov (11)
  - Dec (12)
- Year
  - 1989 (89)
  - 1990 (90)
    …through…
  - 1998 (98)
  - 1999 (99)

Using this system, the name for the "long" file of SIC imports for December 1999 would be: **SCIL1299.SD2**

**Create an interface**

The users needed an interface for two reasons:

- Identify the file(s) to use to create the user's subset
- Identify the criterion for the subset

SAS/AF® was chosen for this purpose.

The user selected a report they wanted to produce from a pull-down menu. The first six (6) reports are listed below.:

1. MDDB (HS10/country/district/moyr) IMPORTS
2. MDDB (HS10/country/moyr) IMPORTS
3. MDDB (HS10/country/district/moyr) EXPORTS
4. MDDB (HS10/country/moyr) EXPORTS
5. MDDB (HS6/country/district/moyr) IMPORTS
6. MDDB (HS6/country/moyr) IMPORTS

The list continued through all iterations of data type, file type and commodity classification. Based on the selection, variables were set to partially identify the data file(s). The following code is an example of how the menu would be built. How the file identifiers are set is talked about in the next section.

```
varlist=makelist();
varlist=insertc(varlist,
   ' 1.MDDB (HS10/country/district/moyr)
   IMPORTS', -1);
```

```
varlist=insertc(varlist,'76.MDDB
   (NAICS6/country//moyr) EXPORTS', -1);


substr:
   call notify('reptype',
      '_get_text_',reptype);
```

**Subset**

Depending on the report selected, variables will be set to identify the file(s) to process.

```
if substr(reptype,1,2) = ' 2' then do;
      sort='imp';
      dsnn='hsis';
      code='10';
      type='hs';
      link derive;
end;
```

```
 else if substr(reptype,1,2) = '76' then
do;
      sort='exp';
      dsnn='naie';
      code='6';
      type='naics';
      link derive;
end;

return;
```

Next, the user selected the months and years they wanted included in their subset. This identified the file(s) needed to complete the request.

Finally, the user would select the analysis variables for the request: commodity, country and/or district (depending on the type of report selected). After browsing the available categories (specific commodities or countries), the user selected their options and submitted the request.

The following code shows how the subset would take place.

```
/* Sample values */

/* Selected months are Jan 1998, */
/* Feb 1998 and May  1999        */

textdate='019802980599';

/* Selected codes are "0000000001" */
/* and "0000000002".               */

textcode='0000000001','0000000002';

/* Selected countriess are "1010"  */
/* and "4620".                     */

textctry='1010','4620';

/* Location of the original files   */
/* being subset.                    */

drive = 'd:'

derive:

/* ------------------------------- */
/* Loop for as many time periods as */
/* have been selected.              */
/* ------------------------------- */

do i = 4 to length(textdate) by 4;
textyr=substr(textdate,sum(i,-3),4);
rc=libname("Y"||substr(textyr,3,2)||"",
   ""||drive||"\inputs\Y"||
   substr(textyr,3,2)||"\"||
   sort||"\"||type||"",'','');

/* ------------------------------- */
/* If codes and countries have been */
/* selected..                       */
/* ------------------------------- */

if textcode ne '' and textctry ne '' then
do;
   rc=copy("Y"||substr(textyr,3,2)||
   "."||dsnn||textyr||
   "(where=("||type||code||"
   in("||textcode||
        ")and country in("||textctry||
        ")))","work.temp"||textyr||"");
end;

/* ------------------------------- */
/* If only codes have been selected */
/* …                                */
/* ------------------------------- */
```

```
else if textcode ne '' & textctry = ''
then do;
rc=copy("Y"||substr(textyr,3,2)||"."||dsn
n||textyr||
        "(where=("||type||code||"
in("||textcode||
        ")))","work.temp"||textyr||"");

end;

/* ------------------------------- */
/* If only countries have been     */
/* selected…                       */
/* ------------------------------- */

else if textcode = '' & textctry ne ''
then do;
rc=copy("Y"||substr(textyr,3,2)||"."||dsn
n||textyr||
        "(where=(country in("||textctry||
        ")))","work.temp"||textyr||"");

end;
call symput('textyr',textyr);
submit continue;

/* ------------------------------- */
/* Append the data subset in this  */
/* loop to the temp file…          */
/* ------------------------------- */

proc append base=work.temp
data=work.temp&textyr;
run;
endsubmit;
end;

/* ------------------------------- */
/* Check to see if there is data   */
/* for the items select by the     */
/* user.                           */
/* ------------------------------- */

das=open('work.temp','u');

/* ------------------------------- */
/* If not, apologize for the incon- */
/* venience and remove those items  */
/* from the menus so they don't    */
/* try that combination again.     */
/* Then, start the application     */
/* again.                          */
/* ------------------------------- */

if attrn(das,'ANY') = 0 then do;
_msg_ = 'NO DATA FOR THIS COMBINATION OF
PARAMETERS. TRY AGAIN';
call notify('obj1','_delete_all_');
call notify('obj3','_delete_all_');
call notify('obj6','_delete_all_');
call notify('obj12','_delete_all_');
call close(das);
stop;
return;
end;

/* ------------------------------- */
/* If there is data, close the     */
/* temp file and continue with     */
/* creation of the SAS/MDDB file.  */
/* ------------------------------- */

call close(das);
return;
```

**Create SAS/MDDB locally**

Once the data is subset, the SAS/MDDB file was created on the user's local directory:

```
libname mylib 'c:\';
proc mddb data=work.temp
out=mylib.mymddb;
        class var1;
        class var2;
        hierarchy var1 var2;
        var nvar1 / n sum;
        format var1 var2 $varsfrmt.;
run;
```

**Display to the screen**

To display the contents of the SAS/MDDB file, a SAS/EIS® multidimensional report was used.

SAS/EIS requires that all databases be registered in a metabase. All report objects must be registered in an application database. In each of these files resides the current location of the data set or report object on the computer or network. Figure 2 represents the contents of the metabase and where it looks for its contents.

If the items in the metabase are not at their expected location, the application will not function (see Figure 3). Our application takes advantage of this dependence. If the SAS/MDDB file is registered to a local directory, the application will try to access it there. Consequently, the metabase for this application is set to look for the SAS/MDDB file locally.

Finally, we achieved our objective. A user would run the program on their machine to subset the original data sets from the network. The application would create a SAS/MDDB file based on that SAS data set and save it on the user's machine. Next, based on the metafile, the application would access that SAS/MDDB file from the user's hard drive.

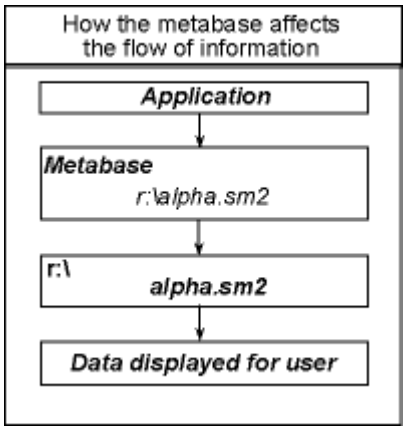


***Figure 2 - Metabase***

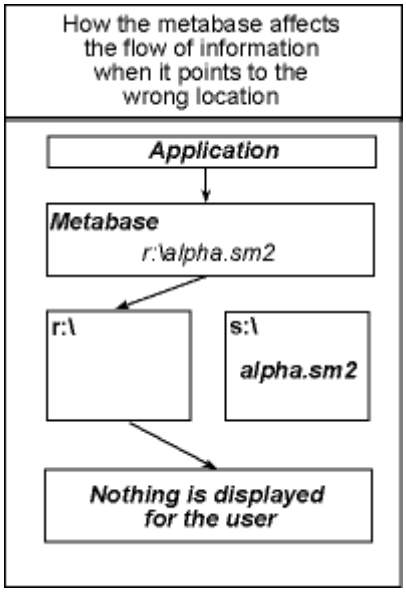


***Figure 3 - SAS/MDDB registered to one drive and residing on another.***

**Lessons learned along the way**

FTD uses several series of code numbers to identify items like commodities, countries and customs districts. This makes sorting and searching the data more simple and less time consuming. Processing 100,000 records of ten (10) character commodity codes is much easier than 100,000 records of 150 character descriptions.

When category variables can be identified by a unique code in addition to (or in place of) a description (as mentioned above), work with the code number. The resulting SAS/MDDB will be smaller because (in most cases) the unique identifier is smaller than its description. Plus, that description can appear in the appropriate place through the use of formats without unduly inflating the size of the file.

Applying those formats as late as possible in the process boosts performance more than when they are applied early. For instance, trying to apply the formats during the subsetting process only increases the size of the input file from which the SAS/MDDB is created. Applying them in the PROC SAS/MDDB statement leads to a faster creation time and smaller SAS/MDDB file.

The same kind of "flying MDDB" application is possible on the Internet via SAS/IntrNet™. It comes, however, with some modifications. Due to the nature of network computing, it's relatively easy to convince SAS that it's done more work that it actually has. Internet computing is different. We can't reply on the user's machine to do much beyond displaying the data. Therefore, all computations need to be done on the web server.

To make it work, each user needs to be uniquely identified and, based on that identification, registered in the metabase. For instance, someone with the user name "JOHN001" would need a SAS/MDDB file named "JOHN001.SM2" preregistered in the metabase before they could use the application. This way, the server would only display the data requested by that specific user.

**Conclusion**

Through a little slight of hand and some strategic planning, it is possible to take advantage of the power of SAS/MDDBs without having to burden your systems. By using the SAS system appropriately, the SAS/MDDB does not have to be created until it is absolutely needed. Applications can be organized so that, through understanding the pattern of user requests, requests are fast and efficient. Understanding these points gives users access to a powerful tool that they might not have had otherwise.

**Contact**

Blake Sanders
U.S. Census Bureau, Foreign Trade Division
FB 3, Rm 2158
Washington, D.C. 20233
- Phone: 301-457-2234
- Fax: 301-457-3932
- E-mail: bsanders@census.gov

Mikhail Gruzdev
U.S. Census Bureau, Foreign Trade Division
FB 3, Rm 2158
Washington, D.C. 20233
- Phone: 301-457-2234
- Fax: 301-457-3932
- E-mail: mikhail.gruzdev@ccmail.census.gov

**References**

SAS Institute Inc. (1990), *SAS Language*, Version 6, First Edition, Cary, NC: SAS Institute.

SAS Institute Inc. (1990), *SAS Procedures Guide*, Version 6, Third Edition, Cary, NC: SAS Institute.

SAS Institute Inc. (1994), SAS Screen Control Language, Version 6, Second Edition, Cary, NC: SAS Institute.

**Extras**

The following are available on the Internet:
1. This paper
2. All presentation materials
3. Extra code from the application

They are available at:
http://www.census.gov/foreign-trade/sugi25

---

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.