Summarizing Impossibly Large SAS® Data Sets for the Data Warehouse Server Using Horizontal Summarization

Michael A. Raithel, U.S. Customs Service

Abstract

Data warehouse applications thrive on pre-summarized data. When a warehouse's data originates on a mainframe computer, it makes sense to take advantage of the mainframe's power to summarize the data before porting it to the warehouse server. The SAS System is the perfect tool to use to summarize data bound for a data warehouse. However, summarizing very large SAS data sets with many CLASS variables can be problematic even on a mainframe computer.

This paper introduces the concept of Horizontal Summarization. Horizontal Summarization is a methodology that can be employed to summarize very large SAS data sets by every combination of a specific set of CLASS variables. Horizontal Summarization is designed to work with the mainframe operating system and avoid common out-of-memory and work-space-shortage abends. Horizontal Summarization was specifically developed and employed as the front-end summarization tool for a data warehouse application.

Introduction

When you need to summarize a SAS data set to create a fact table for a data warehouse, the SUMMARY procedure is the ideal tool. However, the method PROC SUMMARY uses to summarize data via CLASS variables often makes it difficult to successfully summarize a very large SAS data set with many CLASS variables. If you have ever attempted such a summarization, then you have probably experienced problems with long-running batch jobs, memory abends, and/or work space shortages.

The root of these problems lies in the way that the SUMMARY procedure processes data via a CLASS statement. The CLASS statement instructs the SAS System to form subgroups based on every distinct combination and every discrete value of each CLASS variable. PROC SUMMARY keeps these combinations in a matrix in memory, sorting the input observations as they are summarized. Because of this approach, the CLASS statement is designed to handle a small number of discrete values or distinct levels. Summaries having CLASS variables with a large range of values, or that have many CLASS variables, can experience processing delays and computer resource problems.

The summarization that PROC SUMMARY traditionally executes can be thought of as "vertical summarization". It is "vertical" because the SUMMARY procedure builds a vertical matrix in memory of the different levels/values of the class variables. For example, consider the following code:

```
proc summary data=cdsales;
     class state city cdid;
     var  salesamt;
     output out=sumsales sum=;
run;
```

The above code would Create the following basic vertical matrix during execution of the SUMMARY procedure:

type	**Description**
0	total salesamt of all observations
1	total salesamt for each discrete cdid
2	total salesamt for each discrete city
3	total salesamt for each discrete cdid/city combination
4	total salesamt for each discrete state
5	total salesamt for each discrete cdid/state combination
6	total salesamt for each discrete city/state combination
7	total salesamt for each discrete cdid/city/state combination

In the example, **_type_** 1 through 7 would each be composed of vertical sub-levels. Each sub-level would be constructed from the discrete values of its CLASS variables. For example, if there were 50 discrete values of **STATE**, **_type_** 4 would have 50 vertical sub-levels. Each sub-level would hold the running total of **SALESAMT** for the particular value of **STATE**. It is easy to imagine how the matrix could grow, vertically, with 50 discrete values of **STATE**, and hundreds of discrete values of **CITY** and **CDID**. That is the problem with traditional, "vertical" summarization.

Horizontal Summarization Overview

Horizontal Summarization minimizes the "verticalness" of a summarization by performing each **_type_** of summary separately. The separate summaries may be done sequentially, or in parallel batch jobs. The resulting output SAS summary data sets can then be combined into a single data set, containing all summary **_type_**'s, or they can remain as separate data sets.

In the example above, Horizontal Summarization would employ eight different summaries to sum the **CDSALES** data set. Each summary would use the **NWAY** option and the **BY** statement. Each summary would use a single variable combination that corresponded to one of the **_type_**'s, above. For example, the second summarization, which corresponds to **_type_** = 1, would used only **CDID** as its BY variable. The third, corresponding to **_type_** = 2, would use **CITY** as its sole BY variable; the fourth would use **CDID*CITY**, and so on. Together, the eight output SAS data sets would contain the fully summarized CDSALES SAS data set.

Horizontal Summarization of a SAS data set is realized by implementing the following algorithm. The first three steps get the data into the proper shape for the multiple summarizations. The last step actually does the bulk of the Horizontal Summarization work.

1. Create a new variable, called STRING, in the SAS data set that is to be summarized. STRING is set equal to the concatenation of the variables you intend to use in your summarization. In the example above, we would have:

   ```
   string = state || city || cdid;
   ```

2. Sort the original SAS data set by variable STRING.

3. Summarize the original SAS data set by STRING, using the NWAY SUMMARY option and the BY statement. This yields an output data set that is summarized at the lowest level of granularity for the key variable, STRING. Since STRING is the concatenation of all ("CLASS") variables, the resulting data set is now summarized at the highest value of **_type_**. In the example, STRING is the concatenation of STATE, CITY, and CDID, so the output data set is now summarized at the equivalent of **_type_** = 7.

4. Iteratively modify, sort and summarize the SAS data set created in Step #3 to get each of the other summary **_type_**'s. This is done by systematically doing the following:

   A. Blank out the portion of STRING that corresponds to the variables not being used as “CLASS” variables in a particular **_type_**. In the example, when the **_type_** = 5 summary is about to be done, the middle bytes of STRING, corresponding to CITY are set equal to blanks. Thus, only the values of CDID and STATE are taken into account in the SUMMARY procedure.

   B. Sort the data set by (the newly modified values of) STRING.

   C. Execute PROC SUMMARY with the NWAY option and a "BY STRING" statement.

   D. In the output data set, decompose STRING back into its original component variables via the SUBSTR function. In the example, this would be coded:

      ```
      state = substr(string,1,2);
      city  = substr(string,3,20);
      cdid  = substr(string,23,8);
      ```

Broken into its fundamental steps, Horizontal Summarization is not overly complicated. By executing summarizations serially, and limiting the number of "CLASS" variables to a single BY variable (STRING) in each summarization, no overly large CLASS variable matrix is created in computer memory. The only real complication comes in having a methodology to systematically control the execution of Step #4.

Controlling Horizontal Summarization

Horizontal Summarization makes use of a binary counter and bit testing to determine which SUMMARY **_type_** should be executed. The binary counter indicates which sub-field(s) in STRING should have their values used in the summary, and which should be set to blanks. Only the sub-fields that contribute to a particular **_type_** of summary are left non-blank. To better understand this concept, consider the table on the next page.

STATE	CITY	CDID	_TYPE_	Binary Counter	Portion of STRING Represented
0	0	0	0	'000'b	_____ ____ ____
0	0	1	1	'001'b	_____ ____ cdid
0	1	0	2	'010'b	_____ city ____
0	1	1	3	'011'b	_____ city cdid
1	0	0	4	'100'b	state ____ ____
1	0	1	5	'101'b	state ____ cdid
1	1	0	6	'110'b	state city ____
1	1	1	7	'111'b	state city cdid

In the table above, the variable CDID only contributes to the summarization when the **_type_** values are 1, 3, 5, and 7, and the binary counter values are '001'b, '011'b, '101'b, and '111'b. When the **_type_** values are 0, 2, 4, 6, and the binary counter values are '000'b, '010'b, '100'b, and '110'b, CDID does not contribute to the summarization. So, the algorithm for Horizontal Summarization blanks out the portion of STRING occupied by CDID when the binary counter is equal to '000'b, '010'b, '100'b, and '110'b. Of course, the same is done for the non contributing binary counter values of STATE and CITY.

The binary counter is a variable of format **binaryX.**, where X = the number of individual variables that were concatenated to form STRING. In the example, the binary counter would be declared: **format counter binary3.;** Horizontal Summarization uses the binary counter to represent the 2**X possible combinations of STRING summarization patterns. In the example, since X=3, there are 2**3, or 8 possible patterns for separate summarization of STRING. If there had been 7 CLASS variables, then **format counter binary7.** would be used; and there would be 2**7, or 256 possible summarization patterns.

Notice that though there are 8 patterns for summarization in the example, the binary counter values range from 0 ('000'b) to 7 ('111'b). That means that the DO LOOP controlling Horizontal Summarization has to range the binary counter from 0 to 7, and not from 1 to 8. Of course, 0 to 7 is still eight overall iterations. Put in more generalized terms, the DO LOOP controlling Horizontal Summarization must range the binary counter from zero to (2**X) - 1.

The binary counter is used in the logic outlined in Step #4.A, in the preceding section. The counter is incremented with every iteration of steps #4.A through 4.D. This allows Horizontal Summarization to focus on each subsequent pattern of STRING; blanking out the fields corresponding to zero in the binary counter. To appreciate this concept, refer to the Macro %BIGLOOP in the code example, below.

Horizontal Summarization Example

The final pages of this paper contain a full example of the code necessary to perform a Horizontal Summarization. In the example, the goal is to supply a data warehouse application with summarized data of CD sales. The data is already stored in a large SAS data set. There are six variables that must be summarized into every possible combination of values for variables: YEAR, MONTH, COUNTRY, STATE, CITY, and CDID.

Since there are six variables that are to be used in the summarization, the binary counter is set to **binary6.** There are 2**6, or 64 possible summarization patterns for the concatenated field STRING. Thus, the binary counter will range from 0 ('000000'b) to 63 ('111111'b).

The BIGLOOP macro produces sixty-four output data sets, each with an output name of SUMxxx, where xxx = 000 through 063. Each of the output data sets contains the individual variables that were used in the summarization, though some contain blanks in the data sets where they did not contribute to the summarization. Additionally, the binary counter is included in each data set so that one can tell which **_type_** of summarization the particular observation participated in. This is VERY important if all of the observations in all of the data sets are later combined into a single data set.

Though there is not room enough to illustrate it in this paper, the processing done in the BIGLOOP macro could easily be optimized. This could be done by first splitting out and running the part of the program that creates the NEW.BASEDATA SAS data set. Then, multiple batch jobs, that each processed a portion of the sixty three summarizations

could be submitted concurrently. For instance, if two "BIGLOOP" batch jobs were submitted, the first would contain the following do loop: **%do bicount = 000 %to 032 %by 1;** and the second job would contain the following do loop: **%do bicount = 033 %to 063 %by 1;** Of course, they would both read the same SAS data set (NEW.BASEDATA) as input, but they would write to separate output SAS data libraries.

Summary

Horizontal Summarization was developed to feed a data warehouse application after a conventional PROC SUMMARY failed. The original objective was to summarize the values of 19 variables, in 980,000 observations, using 8 CLASS variables. The first attempt failed after over two days of thrashing on a large IBM mainframe. Horizontal Summarization completed the task during the course of a normal work day. Horizontal Summarization is a powerful methodology that can help you to feed your data warehouses with summaries of impossibly large SAS data sets. Give it a try!

Trademarks

References

Raithel, Michael A.
Tuning SAS Applications in the MVS Environment
Cary, NC:SAS Institute Inc., 1995
303pp.

SAS Institute Inc.
SAS Procedures Guide, Version 6, Third Edition
Cary, NC:SAS Institute Inc., 1990
705pp.

Contact Information

Michael A. Raithel
PO Box 406
Garrett Park, Md. 20896
E-mail: maraithel@mcimail.com

Horizontal Summarization Example SAS Code

```
*****************************************;
* this data step concatenates the key variables to one *;
* long variable ('string'). then it outputs to a sas   *;
* data set.                                             *;
*****************************************;
data new.rawdata(drop=year month country state city
                     cdid);
set base.rawdata;

length string $ 30;

string = year || month || country || state || city || cdid ;
run;

*****************************************;
* sort by 'string' for following summarization.         *;
*****************************************;
proc sort data=new.rawdata;
     by string;
run;

*****************************************;
* summarize the data by 'string' to get the lowest      *;
* level of granularity. this is the 'mother of all      *;
* summarizations'-- every subsequent summarization*;
* will use the output dataset                           *;
*****************************************;
proc summary nway data=new.rawdata;
     by string;
     var totsales basprice numsold notsold numretur
         damaged;
output out=new.basedata(drop=_type_ _freq_) sum=;
run;

*****************************************;
* all of the data preparation work is done. the data set*;
* new.basedata will be used as input to the horizontal*;
* summarization iterations that follow                  *;
*****************************************;
```

```
 /*****************************************/
 /* the big loop that executes the data & proc steps    */
/*****************************************/
%macro bigloop;
%do bicount =  000 %to 63 %by 1;
*******************************************;
* this data step examines the binary counter and sets   *;
* components of  'string' to blanks when the            *;
* corresponding bit of the binary counter is zero.  the  *;
* variable 'lenvars'  is used to point to the beginning   *;
* of each variable that was concatenated to form         *;
* 'string'.                                              *;
*******************************************;
data temp;
set  new.basedata;

format x binary6.;

x  = &bicount;

lenvars = 1;

if x = '0.....'b then
   substr(string,lenvars,4) = '    ';  /*year*/

   lenvars = lenvars + 4;

if x = '.0....'b then
   substr(string,lenvars,2) = '   ';   /*month*/

   lenvars = lenvars + 2;

if x = '..0...'b then
   substr(string,lenvars,2) = '    ';  /*country*/

   lenvars = lenvars + 2;

if x = '...0..'b then
   substr(string,lenvars,2) = '   ';   /*state*/

   lenvars = lenvars + 2;

if x = '....0.'b then
   substr(string,lenvars,10) = '   ';   /*city*/

   lenvars = lenvars + 10;

if x = '.....0'b then
   substr(string,lenvars,10) = '   ';   /*cdid*/

run;

*******************************************;
* sort and summarize the file by the large 'string'       *;
* variable to get an nway summarization.                  *;
*******************************************;
proc sort data=temp;
        by string;
run;

proc summary nway data=temp;
       by  string;
       id  x;
       var totsales basprice numsold notsold numretur
            damaged;
output out=sum&bicount sum=;
run;

*******************************************;
* clean up the work area before proceeding.               *;
*******************************************;
proc datasets library=work;
       delete temp;
run;

*******************************************;
* de-compose the 'string' variable into its component *;
*  parts   and write out to a permanent data set.         *;
*******************************************;
data new.sum&bicount(drop=string);
set  sum&bicount;
     by    string;

length  year $ 4  month $ 2  country  $ 2   state $ 2
           city $ 10 cdid  $ 10;

year      = substr(string,1,4);
month     = substr(string,5,2);
country   = substr(string,7,2);
state     = substr(string,9,2);
city      = substr(string,11,10);
cdid      = substr(string,21,10);

run;

*******************************************;
* clean up the work area before proceeding.               *;
*******************************************;
proc datasets library=work;
       delete sum&bicount;
run;

%end;                /** end of the big do loop **/

%mend bigloop;    /**  end of the bigloop macro **/

%bigloop;            /**  invoke the bigloop macro **/
```