Paper BB-02-2015

Greenspace: A Macro to Improve a SAS® Data Set Footprint

Brian Varney, Experis Business Intelligence and Analytics Practice

ABSTRACT

SAS® programs can be very I/O intensive. SAS data sets with inappropriate variable attributes can degrade the performance of SAS programs. Using SAS compression offers some relief but does not eliminate the issue of inappropriately defined SAS variables. This paper intends to examine the problems inappropriate SAS variable attributes can cause as well as a macro to tackle the problem of minimizing the footprint of a SAS Data Set.

INTRODUCTION

This paper intends to examine the problem of working with SAS Data Sets that have inappropriate variable attributes. This paper will also examine a SAS macro to remedy the situation by altering character variable attributes to only use as much storage space as necessary based on the variable values in the data. This paper does not address shortening the length of numeric variables as it could lead to problems in numeric precision.

THE PROBLEM

The focus on this paper is about character variables that are assigned storage lengths that are larger than needed. For example, there are times that data ends up in a database that have most of the character variables defined as a variable length character with a long storage length. i.e. gender stored as a varchar 500. Even though the database can handle this efficiently, when the file is being processed in SAS, it is treated as a fixed length character variable. As far as storage is concerned, one can turn on the compression option but the savings only occur during reading and/or writing the SAS data set to disk. During data processing, the variables are expanded and treated with their full storage length.

SAMPLE DATA SAS DATA SET WITH POORLY DEFINED VARIABLE ATTRIBUTES

The data set used for this paper is not very large compared to what is encountered in real projects. This data set has 500,000 records and 10 variables.

```
* option to show diagnostics.;
options fullstimer;

* turn on data set compression;
options compress=yes;

* set number of records for sample data set;
%let nobs=500000;

** Data set with unnecessarily long character fields.;
data with_long_vars;
   length gender race
          city county state country
          firstname lastname nickname
          favoritecolor $500;
   do id=1 to &nobs.;
      output;
   end;
run;
NOTE: The data set WORK.WITH_LONG_VARS has 500000 observations and 11 variables.
NOTE: Compressing data set WORK.WITH_LONG_VARS decreased size by 99.44 percent.
      Compressed is 70 pages; un-compressed would require 12500 pages.
```

```
NOTE: DATA statement used (Total process time):
      real time           2.22 seconds
      user cpu time       1.77 seconds
      system cpu time     0.10 seconds
      memory              2113.53k
      OS Memory           9344.00k
      Timestamp           08/05/2015 08:48:50 AM
      Step Count                        1  Switch Count  0
```

SAMPLE DATA SAS DATA SET WITH MORE APPROPRIATELY DEFINED VARIABLE ATTRIBUTES

```
data with_shorter_vars;
   length gender $7
          race $40
          city $100
          county $100
          state  $100
          country $100
          firstname $100
          lastname $100
          nickname $100
          favoritecolor $40;
   do id=1 to &nobs.;
      output;
   end;
run;
NOTE: Compressing data set WORK.WITH_SHORTER_VARS decreased size by 97.51 percent.
      Compressed is 154 pages; un-compressed would require 6173 pages.
NOTE: DATA statement used (Total process time):
      real time           0.41 seconds
      user cpu time       0.37 seconds
      system cpu time     0.03 seconds
      memory              580.68k
      OS Memory           8932.00k
      Timestamp           08/05/2015 08:49:12 AM
      Step Count                        4  Switch Count  0
```

Notice in the above two examples that the data set with the more appropriately defined attributes is much more efficient. Even though compression is turned on, it is much less efficient to have character variables with storage lengths that are longer than necessary. The data set above is only 500,000 records and 10 variables. In practice data with millions of rows and hundreds of variables is very common.

Metric	Long Variables	Short Variables
File Size	13.9MB	9.7MB
Real Time	2.22 seconds	0.41 seconds

THE SOLUTION

There are a few solutions to this problem of poorly defined SAS data set variable attributes.

1) Get the attributes changed in the source. This is the best case in that it is easiest for you and anyone else that needs to work with the data set. However, the entity providing you the data may not be willing, able, or have the time to make the changes.
2) Build a SAS view between the data source and your SAS data set environment.
3) Use the greenspace macro to minimize the footprint of your SAS data set(s)

BUILD A SAS VIEW AGAINST THE DATA SOURCE

One method to reduce the impact on processing is to build a SAS View in between the data source and your target data library. A view is a set of instructions on how to obtain the data. Every time you use the view, the data is pulled from the source.

```
proc sql;
   create view wlv_view as
   select id,
          gender length=7,
          race length=40,
          city length=100,
          county length=100,
          state  length=2,
          country length=100,
          firstname length=100,
          lastname length=100,
          nickname length=100,
          favoritecolor length=40
   from with_long_vars;
quit;
```

THE GREENSPACE MACRO

In the interest of minimizing the footprint for a data set for your project data library, it would be best to have an automated process (macro) to calculate the minimal length necessary for storing the data in each of your character variables. The following is a high level overview of the steps that the greenspace macro carries out.

1) Reads the metadata for the input SAS data set and stores the variable names, types, lengths, formats, and labels into sequential macro variables.
2) Uses PROC SQL to read through the input SAS data set and store the maximum character value for each character variable. If the variable is numeric, it just uses the original length.
3) A SAS data step is used to write out the data set to the output data set designated in the macro parameter. The lengths from step 2 are used in a length statement prior to the set statement to force the new more efficient storage length. The format is also adjusted to match the new storage length for the character variables.
4) The informats are wiped out since they no longer serve a purpose.
5) PROC CONTENTS are run on the original input and new output SAS data sets for review.

The full documented code follows.

```
/*-------------------------------------------------------

  Program: greenspace.sas
  Programmer: Brian Varney
  Date:

  Purpose:
     Shorten the storage length of character
     variables to the longest value in that variable
     within the input data set.

  Parameters:
     DSIN:  The input data set name. this can be a one
            or two level name.

     DSOUT: The output data set name. This can be a one
            or two level name.

  Cautions:
```

```
     I would not recommend making the output data set
       the same as the input data set. If something did not
       go as planned, you could lose the input data set.
     If a character variable has a user defined
       format applied, it will be over written by a more
       generic format.
     If there are indexes on the data set they will
       be removed when the data set is re-written.
-------------------------------------------------------------*/
%macro greenspace(dsin=,dsout=);

************************************************************;
** Keep as many macro variables local to the macro as     **;
** possible.                                              **;
************************************************************;
%local libn memn numvars i;

************************************************************;
** Split the input data set name into the two levels if   **;
** appropriate.                                           **;
************************************************************;
%if %index(&dsin.,.)=0 %then
%do;
   %let libn=WORK;
   %let memn=%upcase(&dsin.);
%end;
%else %do;
   %let libn=%upcase(%scan(&dsin.,1,.));
   %let memn=%upcase(%scan(&dsin.,2,.));
%end;
%put &libn.  &memn.;

************************************************************;
** Grab all of the variable information and put into      **;
** macro variables.                                       **;
************************************************************;
proc sql noprint;
   select name, type, length, format, label
            into :name1-:name99999,
                 :type1-:type99999,
                 :length1-:length99999,
                 :format1-:format9999,
                       :label1-:label9999
   from dictionary.columns
   where libname="%upcase(&libn.)" and
             memname="%upcase(&memn.)"
; quit;
%let numvars=&sqlobs.;

************************************************************;
** Calculate the maximum character value length for each  **;
** variable and grab the current storage length for each  **;
** numeric variable.                                      **;
************************************************************;
proc sql;
   create table &memn._maxlengths as
   select
```

```
%do i=1 %to &numvars.;
%if &&type&i.=char %then
%do;
      max(length(&&name&i.)) as &&name&i.
%end;
%else %do;
      &&length&i. as &&name&i.
%end;
%if &i ne &numvars. %then
%do;
      ,
%end;
%end;
   from &dsin.
;quit;

************************************************************;
** Transpose the data so we can load the desired          **;
** variable storage lengths into macro variables.         **;
************************************************************;
proc transpose data=&memn._maxlengths
               out=&memn._maxlengthsv(rename=(col1=maxlength))
               name=varname;
run;

************************************************************;
** Load the desired variable lengths into macro           **;
** variables.                                             **;
************************************************************;
proc sql noprint;
   select maxlength into :maxlength1-:maxlength99999
   from &memn._maxlengthsv;
quit;

************************************************************;
** Apply the new character variable storage lengths to    **;
** the output data set. Also remove the informats from    **;
** the character variables as they would not longer make  **;
** sense with the storage length. If there was a user     **;
** defined format applied to the variable, it will be     **;
** updated with a generic character variable format.      **;
************************************************************;
data &dsout.;
   length
      %do i=1 %to &numvars.;
           &&name&i.
         %if &&type&i.=char %then
         %do; $&&maxlength&i. %end;
             %else
         %do; &&maxlength&i. %end;
        %end;
   ;
   set &dsin.;
   format
      %do i=1 %to &numvars.;
           %if &&type&i.=char %then
             %do;
```

```
                &&name&i. $&&maxlength&i...
              %end;
         %end;
   ;
   informat
      %do i=1 %to &numvars.;
           %if &&type&i.=char %then
             %do;
                &&name&i.
             %end;
         %end;
   ;
run;

*************************************************************;
** Examine the input and output data set attributes to   **;
** ensure you are getting what you are expecting.        **;
*************************************************************;
proc contents data=&dsin.;
title1 "Contents of &dsin. Before Greenspace";
run;

proc contents data=&dsout.;
title1 "Contents of &dsin. After Greenspace";
run;

*************************************************************;
** Reset the titles and footnotes.                       **;
*************************************************************;
title1;
footnote1;

%mend greenspace;
```

CONCLUSION

Knowing your data attributes is important in addition to knowing your data content. By storing SAS data with appropriate variable attributes, poor performance and confusion can be avoided. There are times when this method could cause chaos. For example, if you need your variable attributes to be consistent as new data is added or removed, this approach may not be the best.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Brian Varney:
Experis
5220 Lovers Lane, Suite 200
Portage, Michigan 49002
Office: (269) 553-5185
Fax: (269) 553-5101
E-mail: brian.varney@experis.com
Web: www.experis.us/analytics