TRANSFORMING MULTIPLE-RECORD DATA INTO SINGLE-RECORD FORMAT WHEN NUMBER OF VARIABLES IS LARGE.

David Izrael, Abt Associates Inc., Cambridge, MA
David Russo, Independent Consultant

ABSTRACT

In one large survey project each record in the raw data set contains information from one survey form. At some point, Adenormalization@ of the data needs to be done transposing a multiple-record-per-object data set to a one-record-per-object data set that retains all raw variables from all data sources. The large dimension of the original data set requires an approach that can:

a) carry out the denormalization in a reasonable amount of time;
b) handle the indexing and standardization of variables: names automatically;
c) easily adapt as the set of variables changes over time.

One approach uses a two-step PROC TRANSPOSE. This approach deals elegantly with issues b) and c) , but it takes a substantial amount of time to run. The second approach uses a driving macro based on the RETAIN statement. Although this method requires some maintenance, it runs much more quickly than the TRANSPOSE method.

INTRODUCTION

We have been involved in a survey that periodically collects many items of information from each of many respondents. Some of the items may come from multiple sources, and these items must be combined into a single record. A key step in the process creates distinct variable names that indicate both the nature of the item and the source of the data. The large number of variables makes it advantageous to automate this
step.

Our approach exploits the structure that the survey has imposed on the names of the variables. Many variables belong to groups; their names share a prefix and are distinguished by a digit that follows the prefix. For example, within the group of variables with prefix PFX1 five month variables (text components of dates) might be

PFX11_MO, PFX12_MO, PFX13_MO, PFX14_MO, PFX15_MO (1)

The prefixes may have different lengths, so the suffixes may have different lengths and structure as well. Thus, depending on the length of the prefix, the variables in (1) above may look like:

PFX11_M, PFX12_M, PFX13_M, PFX14_M, PFX15_M (2)

or even

PFX11M, PFX12M, PFX13M, PFX14M, PFX15M (3)

Below, we refer to these variables as Type 1 variables.

The second type (Type 2) of variables includes all plain Ascalar-like@ variables. The survey record also contains a case ID variable (denoted by CASEID in excerpts below) and the variable SEQ_NUM, which, in our multi-record case, distinguishes the survey forms (i.e., respondents) for a given CASEID.

In processing the data we need to do calculations across all the survey forms for a given CASEID and then across all the CASEID=s. In other words, the input data set has to be transposed, creating a record with unique CASEID that contains values from all SEQ_NUM. The newly created Type 1 variables must bear the same prefix and primary index as the original ones . Also, variables of both types must be assigned a secondary index that reflects their original SEQ_NUM. Because the prefixes have different lengths, the code implementing the transpose must control the length and the structure of the derived variables. Thus, the set of variables (1) would lead to the following set of transposed variables (assuming that maximum SEQ_NUM is 5):

<u>Length of PFX1 = 3:</u>

PFX11_MO1, ..., PFX11_MO5, PFX12_MO1, ... , PFX15_MO5 (4)

<u>Length of PFX1 = 4:</u>

PFX11_M1, ..., PFX11_M5, PFX12_M1, ... , PFX15_M5 (5)

<u>Length of PFX1 = 5:</u>

PFX11M1, ..., PFX11M5, PFX12M1, ... , PFX15M5 (6)

The first challenge in automating the building of derived variables is to accommodate the addition of new variables to the original data set. Dimensions of the data themselves pose a challenge. Around 60,000 records with 600 variables impose a strict execution time constraint on the implemented code.

DOUBLE PROC TRANSPOSE

The main idea in this approach is to transpose the raw variables of each unique length and type separately, so that all variables in the resulting data set do not have the same length and type. The natural tool for such a task is PROC TRANSPOSE with some pre- and postprocessing.

The following code implements this idea. Because we deal we deal with character raw variables, the presented macro omits any mentioning of TYPE of variables but it can easily be modified take TYPE into account.

```
*****************************************;
* Double Transpose;
*****************************************;

proc contents data=inp_data          ❶
              out=temp
              noprint;
run;


proc sort data=temp;                 ❷
     where upcase(name) notin
('CASEID','SEQ_NUM');
     by length name;
run;


data _null_;                         ❸
set temp end=eof;
by length;
if first.length then do;
    numlens+1;          /* increment
            number of unique lengths*/
call
symput('LENGTH'||left(put(numlens,2.)),
trim(left(put(length,3.))));
                      end;
if eof then call
symput('NUMLENS',trim(left(put(numlens,
2.))));
run;

%MACRO TRANSP;

%DO I = 1 %TO &NUMLENS;

data _null_;
set temp;
where length = &&length&i ;          ❹
 file "varlist.prv" lrecl=120 flowover;
if upcase(name) notin
('CASEID','SEQ_NUM') then put name +1
@;
run;

proc transpose data=inp_data         ❺
 out=inp_&I;
by CASEID SEQ_NUM;
var
%include "varlist.prv";
run;

data inp_&I;
length _NAME_ $8;
set inp_&I;
_NAME_=trim(substr(_NAME_,1,7))||
  put(SEQ_NUM,1.);                   ❻

run;

proc transpose data=inp_&I           ❼
out=inp_&I(drop=_name_);
by CASEID;
var col1;
run;

%END;


data output;                         ❽
  merge
  %do i = 1 %to &numlens;
   inp_&i
  %end;;
  by CASEID ;
   run;

%mend transp;

%transp;
```

❶ Generate list of variables for the following PROC TRANSPOSE.

❷ Group together all variables with the same length .

❸ Create global macro variables that give the number of different unique variable lengths (&NUMLENS) and each of the lengths (&LENGTH1, &LENGTH2, etc.).

❹ Create ASCII file that contains names of variables of length &&LENGTH&I.

❺ Transpose variables indicated in the ASCII file. Created by PROC TRANSPOSE, variable COL1 contains values of transposed variables. The variable _NAME_ has the names of the variables transposed in this iteration.

❻ Append the sequence number to the values of the _NAME_ variable, thus creating the secondary index in the names of variables which will be used in the next transpose.

❼ A final transpose for iteration &I.

❽ Merge transposed data sets for the unique variable lengths.

This approach avoids the need for a programmer to manually modify code when the set of raw variables changes. To use this code for different types (e.g., NUMERIC, DATA) of variables, one can introduce the variable TYPE into BY statement in ❶,❷,❸. WHERE clause in ❹ must also include TYPE.

However, the CPU time and work space requirements eventually became so substantial that this part of the whole code was a serious bottleneck. The number of iterations (i.e., number of unique lengths of variables) in our case is 11. In each iteration, the first PROC TRANSPOSE creates tens of million of records, and the second PROC TRANSPOSE literally spends hours to process them. Also, the accumulation of work data sets from each iteration required about gigabyte of precious disk space.

Nonetheless, the double-transpose method could be successfully used in three cases: a) when the computer's resources are less limited than the programmer's, b) when the set of raw variables changes frequently and dimension of the original data set is not very large, and c) when the number of distinct lengths of variables (i. e., the number of iterations) is small.

In our case, we have decided to switch to ....

OLD RELIABLE RETAIN

The second approach makes straightforward use of the RETAIN statement. A close examination of variable names allowed us to build macros that perform the following functions:

a) assigning values of raw variables to derived variables along with control over their lengths
b) maintaining the original indexes and creating the secondary ones

```
************************************
*MACRO INIT
************************************;

 %macro init;
      %do i=1 %to 5;
         var1&i    =' ';
         var2&i    =' ';
```

c) initializing derived variables
d) keeping derived variables
e) retaining derived variables.

The macros operate on sets of variables that share the same prefix (e.g., the month, day, and year components of a date, along with the associated flags). The following macros implement those functions. In our case both primary and secondary indexes have 5 as a maximum value.

```
*****************************************
* MACRO KEEP
*****************************************;

%macro keep;                                  ❶

    keep =     caseid
           %do i=1 %to 5;                     ❷
                var1&i
                var2&i
               ........

            %do j=1 %to 5;                    ❸
                  Pfx1&i_mo&j
                  Pfx1&i_da&j
                  Pfx1&i_yr&j
                  ............
           %end;
         %end;
%mend;
```

❶ Macro %KEEP keeps the derived variables in the resulting data set.

❷ This %DO loop keeps derived Type 2 variables (only the secondary index is assigned).

❸ This %DO loop keeps derived Type 1 variables: text components of dates and special flags that supplement the dates (both indexes are assigned).

All transposed variables must be listed in this macro. Thus, unlike in the first method, the programmer must type in all derived variables (a straightforward task with any good editor). Similarly, variables must be added or removed manually.

The above note applies also to the following %INIT and %RETAIN macros.

```
          ..............

       %do j=1 %to 5;
             Pfx1&i_mo&j  =' ';
             Pfx1&i_da&j  =' ';
             Pfx1&i_yr&j  =' ';
             ...................
       %end;
```

```
     %end;
%mend;
```

Macro %INIT assigns the initial ›missing‹ values to the transposed variables. It has the same structure as the macro %KEEP.

```
*****************************************
*MACRO RETAIN
*****************************************;

%macro retain;

    retain
          %do i=1 %to 5;
                var1&i
                var2&i
                ......

            %do j=1 %to 5;
                  Pfx1&i_mo&j
                  Pfx1&i_da&j
                  Pfx1&i_yr&j
                  ............

            %end;
          %end;

%mend;
```

Macro %RETAIN retains the transposed variables in the resulting data set. Its structure is identical to that of macro %KEEP.

Now, we declare arrays for assigning values to derived variables of Type 2.

```
*****************************************
*MACRO ARRAY1
*****************************************;

%macro array1 (pref, l);
   %if %length(&pref)<8 %then  %do;  ❶

        %let namarr =&pref._ ;
        %let varout =&pref;
                              %end;
   %else %do;
        %let nam=%substr(&pref,1,7); ❷
        %let namarr=&nam._ ;
        %let varout=&nam;
          %end;
      %let ty =ty_ ;
    %end;

    %else %if %length(&pref)=4 and   ❷
(&und gt) %then %do;
      %let mc = _m;
      %let dc = _d;
      %let yc = _y;
   array &namarr {5} $ &l
       &varout.1 &varout.2 &varout.3
        &varout.4 &varout.5;

%mend;
```

Macro %ARRAY1 prepares an array for a further assignment of Type 2 variables. It has two parameters: Apref@ (prefix of the variable, which in case of Type 2 just means the whole variable name) and Al@ (length of the variable). ❶ and ❷ determine the name of the array from the length of the variable.

We now assign values to the derived variables of Type 2.

```
************************************
* MACRO ASSIGN1
************************************;

%macro assign1(pref, in);
      %if %length(&pref)<8 %then  %do;
        %let namarr =&pref._ ;
                              %end;
   %else %do;
        %let nam=%substr(&pref,1,7) ;
        %let namarr=&nam._ ;
          %end;

  &namarr.[&in] = &pref;
%mend;
```

Macro %ASSIGN1 assigns values of raw Type 2 variables to the derived variables through the array created by macro %ARRAY1.

We declare some arrays for a further assigning of values to derived variables of Type 1.

```
*****************************************
MACRO ARRDATE
*****************************************;

%macro arrdate(pref,flag=,op=,und=);

 %if %length(&pref)<=3 %then %do;    ❶
      %let mc = _mo;
      %let dc = _da;
      %let yc = _yr;
      %let opp =op_ ;

      %let ty = ty;
      %let opp = op;
   %end;
    %else %if %length(&pref)=4 %then ❸
     %do;
       %let mc = mo;
       %let dc = da;
       %let yc = yr;
```

```
    %end;
    %else %if %length(&pref)=5 %then ❹
 %do;
       %let mc = m;
       %let dc = d;
       %let yc = y;
     %end;


%do i=1 %to 5;

 array &pref&i._d{5} $ 2              ❺
  &pref&i&dc.1 &pref&i&dc.2
&pref&i&dc.3
  &pref&i&dc.4 &pref&i&dc.5;

 array &pref&i._m{5} $ 2
  &pref&i&mc.1 &pref&i&mc.2
&pref&i&mc.3
  &pref&i&mc.4 &pref&i&mc.5;

 array &pref&i._y{5} $ 4
  &pref&i&yc.1 &pref&i&yc.2
&pref&i&yc.3
  &pref&i&yc.4 &pref&i&yc.5;

     %if &flag=yes %then  %do;
 array &pref&i._t{5} $ 1
  &pref&i&ty.1 &pref&i&ty.2
&pref&i&ty.3
  &pref&i&ty.4 &pref&i&ty.5;
                          %end;


     %if &op=yes %then  %do;
  array &pref&i._o{5} $ 1
       &pref&i&opp.1 &pref&i&opp.2
&pref&i&opp.3
       &pref&i&opp.4 &pref&i&opp.5;
                          %end;

 %end;
%mend;
```

Macro %ARRDATE prepares arrays of components of dates (Type 1 variables), along with their supplemental flags
(FLAG and OP in our case). Parameter UND controls whether the underscore should be inserted before the suffix. Those arrays will be used in the further assignments

```
%include 'retain.inc';
%include 'init.inc';                  ❶
%include 'arrays.inc';
%include 'assign.inc';

data rslt_ds(%keep); /* resulting data
                        set */
                                      ❷

%arrdate(Pfx1,flag=yes,op=yes,und=yes);
```

to the derived variables.
❶,❷,❸, and ❹ assign suffixes to the array name, depending on the length of the prefix. Suffixes are assigned to the date components and flags mentioned above.
❺ declares arrays for the date components and for the flags, if any.

We assign components of dates and special flags:

```
*****************************************
* MACRO ASSIGNDT
*****************************************;

%macro assigndt(pref,in,flag=,op=);

    %do i=1 %to 5;

   %if %length(&pref)<5 %then %do;  ❶
    &pref&i._d[&in] =&pref&i._da;
    &pref&i._m[&in] =&pref&i._mo;
    &pref&i._y[&in] =&pref&i._yr;
                           %end;
   %else %do;
     &pref&i._d[&in] =&pref&i.d;    ❷
     &pref&i._m[&in] =&pref&i.m;
     &pref&i._y[&in] =&pref&i.y;
         %end;

      %if &flag=yes %then %do;      ❸
    &pref&i._t[&in] =&pref&i.ty_;
                           %end;
      %if  &op =yes %then %do;
    &pref&i._o[&in] =&pref&i.op_;
                           %end;
     %end;
%mend;
```

❶ and ❷ assign values of original date components, depending on the length of the prefix; ❸ does the same operation but for flags, if any.

And now it is time to assemble the parts and make them work.

```
*****************************************
* Main Code
*****************************************;

%include 'keep.inc';
%arrdate(Pfx2,flag=yes,op=yes,und=yes);
.....................................
.....................................

   %array1(var1,12);                  ❸
   %array1(var2,2);
   ..............
   %retain;
set orig_ds;   /* original data set */
by caseid;
```

```
if first.caseid then do;
                  %init;              ❹
                    s=0;
                      end;
                    s+1;

%assigndt(Pfx1,s,flag=yes,op=yes);    ❺
%assigndt(Pfx2,s,flag=yes,op=yes);
..................................
%assign1(var1,s);                     ❻
%assign1(var2,s);
.................

if last.caseid then output;
run;
```

❶ Include all described macros.
❷ Run macro ARRDATE. All existing prefixes must be listed here.
❸ Run macro ARRAY1. All existing variables of Type 2 must be listed here.
❹ Initialize variables and creates count on SEQ_NUM.
❺ Assign values of original dates components and flags. All variables of Type 1 must be listed here.
❻ Assign values of original variables of Type 2. All variables of Type 2 must be listed here.

Preparation of these codes and macros may seem tedious; but this is generally a one-time effort that, with a good editor, does not take much time for typing. Our set of variables is comparatively stable. Around five new prefixes arrive every half a year. So modification of macros does not present a problem either.

RETAIN method has dramatic advantages over the Double Transpose one. It runs several minutes against several hours needed for Double Transpose, and requires little disk space because everything runs in memory.

CONCLUSION

A programmer should decide on a case-by-case basis which method to apply when transposing records with a large number of variables. The decision should take into account the dimension of the original data set, the structure of the variables names, and available resources.

ACKNOWLEDGMENTS

The authors would like to thank David C. Hoaglin of Abt Associates Inc., Cambridge, Massachusetts for his comments and assistance in developing this paper.