BACK FROM THE FUTURE: DIFFERENT WAYS TO TRANSFER THE VALUES FROM LATER TO EARLIER VISITS

Alexander M. Feigin, Ph.D., Synthes USA, Inc., West Chester, PA  feiginy@aol.com

ABSTRACT

In order to bring the values from earlier observations to later ones, SAS language has the perfect, easy-to-use tools, like RETAIN statement or LAG function. Yet often it is a common task for a SAS programmer in Clinical Trials to do just the reverse: bring the values backward, from later to earlier ones. In this presentation we compare different ways to perform this task. We offer the following approaches: 1) creating separate data set for each visit and merging them together; 2) sorting by visit in descending order and retaining the value from the first visit, which is actually the latest one; 3) transferring the values at later visit into macro variables with call symput routine and extracting these values at any visit using symget function; 4) selecting and placing together all necessary variables from all necessary visits with Proc SQL. We demonstrate that in spite of the fact that approaches 3 provides us with the shortest code, it is not usable for large data sets, while approach 2 is efficient, short enough and does not require knowledge of SAS macro language or PROC SQL.

*  *  *

Introduction to the task

It is a common task for a SAS Programmer in clinical trials to write a statement to be executed for a certain visit based on the value(s) of some variable(s) at the later visit(s). For example, the request is to select observations for Visit1 for patients with systolic blood pressure above 150 mm who had systolic blood pressure below 110 mm at Visit3. The data are presented in data set ONE.

```sas
data one;
  input subject visit systbp;
  cards;
  1 1 155
  1 2 140
  1 3 108
  2 1 120
  2 2 134
  2 3 106
  3 1 152
  3 2 143
  3 3 103
  4 1 155
  4 2 141
  4 3 127
```
Possible solutions

1. Creating separate data set for each visit and merging them together.

```sas
data vis1 vis3;
set one;
  if visit=1 and systbp gt 150 then output vis1;
  if visit=3 and systbp lt 110 then output vis3;
run;

proc print data=vis3;run;
```

```sas
proc sort data=vis1;
  by subject;
run;
```

```sas
proc sort data=vis3;
  by subject;
run;
```

```sas
data final1;
merge vis1(in=a) vis3(in=b rename=(systbp=systbp3) drop=visit);
  by subject;
  if a and b;
run;
```

```sas
proc print;run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>subject</th>
<th>visit</th>
<th>systbp</th>
<th>systbp3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>155</td>
<td>108</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>152</td>
<td>103</td>
</tr>
</tbody>
</table>

2. Sorting by visit in descending order and retaining the value from the first visit, which is actually the latest one.

```sas
proc sort data=one out=two;
  by subject DESCENDING visit;
run;
```
Now it is easy to put systolic blood pressure from Visit3 to Visit1 just by retaining the value at Visit3.

```
data two;
set two;
retain syst3;
by subject DESCENDING visit;
if visit=3 then syst3=.;
if visit=3 and first.subject and systbp lt 110 then syst3=systbp;
if visit=1 and systbp gt 150 and syst3 ne . ;
run;
```

```
proc print;run;
```

If all we need is to keep the observations from Visit1 as is on the base of conditions realized in later visits then both approaches are kind of clumsy because they need sorting the data sets, creating new variables, retaining, merging, etc. Instead of getting into all these troubles we would rather use Macro language approach, in particular combination of Call Symput routine and Symget function.

3. Transferring the values at later visit into macro variables with call symput routine and extracting these values at any visit using symget function.

```
data null_;
set one;
if visit=3 then call symput('vis3'||compress(subject),compress(systbp));
run;
```

The code above creates as many macro variables as many observations with VISIT=3
we have in data set ONE. The name of each of these macro variables is a composite of character string 'vis3' and a value of variable SUBJECT. The value of each of these macro variables is a value of variable SYSTBP in a respective observation, where SUBJECT has the same value as in macro variable name and VISIT = 3. Now, all we need is to bring this value into the observation for the same SUBJECT for visit=1. SYMGET function is exactly what we need for this purpose. SYMGET uses the same combination of 'vis3' and value of variable SUBJECT in order to refer to a certain macro variable.

So, in the data step below, for each observation the value of a relevant macro variable (which is actually the value of SYSTBP at Visit3 for this patient) is brought in for estimating whether the condition "input(put(symget('vis3'||compress(subject)),3.),8.) lt 110" is true or false. The combination of INPUT and PUT functions is used to convert the character value of macro variable into numeric one.

``` SAS
data three;
  set one;
  if visit=1 and systbp gt 150 and input(put(symget('vis3'||compress(subject)),3.),8.) lt 110;
run;

proc print;run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>subject</th>
<th>visit</th>
<th>systbp</th>
<th>syst3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>155</td>
<td>108</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>152</td>
<td>103</td>
</tr>
</tbody>
</table>

This approach can be especially useful in case we need to select our observations at Visit1 on the base of information from more than one later visit. For example, the request is to select observations for Visit1 for patients with systolic blood pressure above 150 mm who had systolic blood pressure below 110 mm at Visit3 and below 142 mm at Visit2.

``` SAS
data _null_;  
  set one;
  call symput('viso'||compress(visit)||compress(subject),compress(systbp));
run;

data four;
  set one;
  if visit=1 and systbp gt 150 and input(put(symget('viso3'||compress(subject)),3.),8.) lt 110 and input(put(symget('viso2'||compress(subject)),3.),8.) lt 142 ;
run;

proc print;run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>subject</th>
<th>visit</th>
<th>systbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>155</td>
</tr>
</tbody>
</table>

The SYMGET function is really flexible. It can have as a part of its argument either the name of the variable used in call symput expression or just the value of this variable in quotes.
The data step with `DATA _ NULL_` is not necessary because the `call symput` expression can be placed at the original data step.

4. Selecting and placing together all necessary variables from all necessary visits with `Proc SQL`.

Also elegant seems the use of PROC SQL:

```
proc sql;
create table testura as
select prima.subject, prima.visit,prima.systbp from one as prima, one as secunda where
prima.subject=secunda.subject and
prima.visit=1 and secunda.visit=3 and
prima.systbp gt 150 and secunda.systbp lt 110;
quity;
```

proc print;run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>subject</th>
<th>visit</th>
<th>systbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>155</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>152</td>
</tr>
</tbody>
</table>

**How to choose which approach is better? What are the criteria for good and bad?**

When we compare all four approaches from elegancy point of view, the macro variables and PROC SQL ones seem much more elegant then the first two. Are they also the most efficient?

In order to answer this question lets use the code which can creates the original data set (TEST) with different number of observations and, then apply all four approaches to all these data sest. Our points of interest are real time and CPU time necessary to run each type of code on data sets of different size.

The code which creates the data sets is following:

```
%macro test(num)
data test;
do subject=1 to &num;
do visit=1 to 5;
   if mod((subject+visit),4)=0 then systbp=160;
   else if mod((subject+visit),5)=0 then systbp=140;
   else systbp=105;
   output;
end;
run;
%mend;
%test(1000);
%test(10000);
```
The task is: select all observations for Visit1 when systolic blood pressure was greater than 150, if at Visit3 it was less than 110 and at Visit5 it was greater than 110.

Again, we have four versions of code to perform the task:

1) Creating separate data set for each visit and merging them together

```plaintext
data vis1 vis3 vis5;
set test;
if visit=1 and systbp gt 150 then output vis1;
if visit=3 and systbp lt 110 then output vis3;
if visit=5 and systbp gt 110 then output vis5;
run;
```

```plaintext
proc sort data=vis1;
by subject;
run;
```

```plaintext
proc sort data=vis3;
by subject;
run;
```

```plaintext
proc sort data=vis5;
by subject;
run;
```

```plaintext
data final1;
merge vis1(in=a) vis3(in=b keep= subject) vis5(in=c keep= subject) ;
by subject;
if a=1 and b=1 and c=1;
run;
```

2) Sorting by visit in descending order and retaining the value from the first visit, which is actually the latest one.

```plaintext
proc sort data=test out=two;
by subject DESCENDING visit;
run;
```

```plaintext
data final2;
set two;
retain syst3 syst5;
by subject DESCENDING visit;
if visit=3 then syst3=.;
if visit=5 then syst5=.;
if visit=5 and systbp gt 110 then syst5=systbp;
if visit=3 and systbp lt 110 then syst3=systbp;
```
if visit=1 and systbp gt 150 and syst3 ne . and syst5 ne .;
run;

3) Transferring the values at later visit into macro variables with `call symput` routine and extracting these values at any visit using `symget` function.

```plaintext
data null;
set test;
call symput('viso'||compress(visit)||'_'||compress(subject), compress (systbp));
run;

data final3;
set test;
if visit=1 and systbp gt 150 and input(put(symget('viso3'||'_'||compress(subject)),3.),8.) lt 110 and input(put(symget('viso5'||'_'||compress(subject)),3.),8.) gt 110;
run;
```

4) Selecting and placing together all necessary variables from all necessary visits with `Proc SQL`.

```plaintext
proc sql;
create table testura as
select prima.subject, prima.visit,prima.systbp from test as prima , test as secunda , test as terzia
where
    prima.subject=secunda.subject and prima.subject=terzia.subject and
    prima.visit=1 and secunda.visit=3 and terzia.visit=5 and
    prima.systbp gt 150 and secunda.systbp lt 110 and terzia.systbp gt 110;
quit;
```

The values for real and CPU time are presented at Table1.

Table1.

<table>
<thead>
<tr>
<th>Code</th>
<th>n=5,000</th>
<th>N=50,000</th>
<th>N=500,000</th>
<th>N=5,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real Time</td>
<td>CPU Time</td>
<td>Real Time</td>
<td>CPU Time</td>
</tr>
<tr>
<td>1. Separate data</td>
<td>0.27 sec</td>
<td>0.13 sec</td>
<td>0.11 sec</td>
<td>0.11 sec</td>
</tr>
<tr>
<td>sets</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Sorting DESCENDING</td>
<td>0.04 sec</td>
<td>0.04 sec</td>
<td>0.1 sec</td>
<td>0.1 sec</td>
</tr>
<tr>
<td>3. Macro approach</td>
<td>0.12 sec</td>
<td>0.12 sec</td>
<td>13 sec</td>
<td>13 sec</td>
</tr>
<tr>
<td></td>
<td>0.01 sec</td>
<td>0.01 sec</td>
<td>0.04 sec</td>
<td>0.04 sec</td>
</tr>
<tr>
<td>4. SQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Conclusion**

From these results we can conclude that approach 3, with transferring the values at later visit into macro variables, is not usable for large data sets at all while any of the rest three can be used successfully and the choice between them is rather a matter of personal preference of a SAS programmer than anything else. Obviously, approach 2, with sorting by visit in descending order and retaining the value from the *first visit*, provides the programmer a short and efficient way to perform the task and does not require knowledge of SAS macro language or PROC SQL.

**AKNOWLEDGEMENT**

The author would like to express his gratefulness to Dr. Jeff Stein, Synthes, USA for the fruitful discussion of this paper.