Paper 2785-2015

Transpose Dataset by MERGE

Xia Ke Shan, 3GOLDEN Beijing Technologies Co. Ltd., Beijing, China
Matthew Kastin, I-Behavior, Inc., Louisville, Colorado
Arthur S. Tabachneck, Ph.D., AnalystFinder, Inc., Thornhill, Ontario Canada

ABSTRACT

Using proc transpose to make wide files wider requires running separate proc transpose steps for each variable that you want transposed, as well as a data step using a Merge statement to combine all of the transposed files. Additionally, if you want the variables in a specific order, an extra data step will be needed to rearrange the variable ordering.

INTRODUCTION

The present paper presents a method that accomplishes the task in a simpler manner, using less code, requiring fewer steps, and which will run n times faster than proc transpose (where n=the number of variables to be transposed).

AN EXAMPLE PROBLEM

We used the following data step to create our example data:

```
data data have;
  input subid date : date9. status : $3. type : 1.;
  format date date9.;
  cards;
1 28sep2003 Yes 1
1 25sep2004 Yes 3
2 28sep2003 Yes 4
2 26sep2006 No 2
2 14oct2007 No 3
;
run;
```

The file that we want to create is shown in table 1, below. As you can see, the task is to take a dataset that contains up to n number of records for each subid, with each record containing subid and three variables (namely, date, status and type), and transpose it into a dataset that contains one record for each subid, and variables to represent all n iterations of date, status and type.

subid	date1	status1	type1	date2	status2	type2	date3	status3	type3
1	28-Sep-03	Yes	1	25-Sep-04	Yes	3	.		.
2	28-Sep-03	Yes	4	26-Sep-06	No	2	14-Oct-07	No	3

Table 1. Desired Output

The task isn't very difficult to accomplish using proc transpose, but requires proc transpose steps for each of the three variables we want to transpose. For example, the following code could be used:

```
proc transpose data=have out=date(drop=_:) prefix=date;
  by subid;
  var date;
run;
```

```
proc transpose data=have out=status(drop=_:) prefix=status;
  by subid;
  var status ;
run;

proc transpose data=have out=type(drop=_:) prefix=type;
  by subid;
  var type ;
run;

data want;
  merge date status type;
  by subid;
run;
```

However, the above code will not accomplish the entire task, as the resulting transposed variables will be in a different order than that which was called for in the task's specifications. Specifically, with the above code, subid will be followed by all three dates, then all three status codes and, finally, by all three type codes. As such, an additional data step will be needed, namely:

```
data want;
  retain subid date1 status1 type1
               date2 status2 type2
               date3 status3 type3
;
  set want;
run;
```

Unfortunately the extra data step can't be written until we know the maximum iterations that a subid might have and, unless the maximum iterations holds constant for all subsequent datasets, the code will have to be modified each time it is run.

OUR PROPOSED ALTERNATIVE SOLUTION

The same final result can be obtained as follows. First, a data step is used to create an identity number for every record within each subid group:

```
data have;
  set have;
  by subid;
  if first.subid then n=0;
  n+1;
run;
```

Then, proc sql is used to create a macro variable that will contain the text which will be needed to merge the three files in a subsequent datastep:

```
proc sql noprint;
  select distinct catt('have (where=(n=',left(put(n,8.)),
         ') rename=(date=date',left(put(n,8.)),
         ' status=status',left(put(n,8.)),
         ' type=type',left(put(n,8.)),

         '))')
    into :mer separated by ' '
      from have
```

```
  ;
quit;
```

The proc sql code, shown above, will produce a macro variable called &mer, which will contain the following text:

```
have (where=(n=1) rename=(date=date1 status=status1 type=type1))
have (where=(n=2) rename=(date=date2 status=status2 type=type2))
have (where=(n=3) rename=(date=date3 status=status3 type=type3))
```

The code only accesses the unique values of the variable n, and uses those values to create the text that will be needed to accomplish the merge in the next datastep. That step, shown below, simply uses the macro variable &mer as a text substitution mechanism:

```
data want;
  merge &mer ;
  by subid;
  drop n;
run;
```

COMPARING THE TWO METHODS

Using proc transpose required 19% more code than using the combination of a datastep and proc sql. 341 characters as opposed to 405 characters may not seem like a lot, but it increases the risk of making typing errors and resulting in code that either doesn't run, or produces the wrong result.

The datasteps, in each method, required approximately the same amount of time for both methods. However, the proc sql step took approximately the same amount of time as just one of the proc transpose steps. As such, given that three variables had to be transposed, the proc transpose method took three times as long to complete. Obviously, the discrepancy will be greater if even more variables have to be transposed.

CONCLUSION

A method was proposed for making wide files wider by using a data preparation data step, proc sql, and a data step using a merge statement. That method was then compared with the traditional method of using proc transpose for such tasks.

The proposed method ran three times faster than the traditional proc transpose method, and required less code.

The only limitation of the proposed method is that the macro variable created by proc sql can only contain a maximum of 65,534 characters. While we doubt that anyone would need to transpose a file that would cause that limitation to be exceeded, it can be easily circumvented by using call execute as follows:

```
proc sort data=have(keep=n) out=temp nodupkey;
  by n;
run;

data _null_;
  set temp end=last;
  if _n_ eq 1 then call execute('data want; merge ');
  call execute(catt('have(where=(n=',left(put(n,8.)),
         ') rename=(date=date',left(put(n,8.)),
                    ' status=status',left(put(n,8.)),
                    ' type=type',left(put(n,8.)),'))'));
```

```
  if last then call execute(';by subid; drop n;run;');
run;
```

Using call execute, as shown above, would require slightly more code than using the traditional proc transpose method, but will run approximately 5 times faster for each variable that needs to be transposed. Thus, given the current example, the call execute method would have run approximately 3 times faster than the proposed method, and 15 times faster than the traditional proc transpose method.

CONTACT INFORMATION

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

Xia Ke Shan
3GOLDEN Beijing Technologies Co. Ltd.
Beijing, China
E-mail: keshan.xia@gmail.com

Matthew Kastin
i-behavior, Inc.
2051 Dogwood Street, Suite 220
Louisville, CO 80027
E-mail: matthew.kastin@gmail.com

Arthur Tabachneck, Ph.D., CEO
Analyst Finder, Inc.
80 Willowbrook Road
Thornhill, ON L3T 5K9 Canada
E-mail: art@analystfinder.com