# Data Transposition with Proc Report

**Lingqun Liu**
**The University of Michigan, Ann Arbor**

## Abstract

Transposition is a very common data manipulation task. With SAS, data transposition can be carried out in more than one way. Data Step and Proc Transpose are the two most common methods. Besides these two, this paper also introduces a third way, PROC REPORT, which SAS users can use to transpose data, in some circumstances, more efficiently than the other two methods. In appendix 2, I use Proc Report to solve some data transposition problems posted on SAS_L website.

## Keywords

*Data Transposition, Data Step, Proc Transpose, Proc Report, SAS_L*

## Introduction

This paper includes the following parts: (1) Data transposition; (2) Data Step and data transposition; (3) Proc Transpose; (4) Proc Report and data transposition. There also are two appendixes.

## I. Data Transposition

Data transposition is the process of restructuring values of a SAS data set by turning selected variables into observations. They are commonly needed to support certain analyses that require particular data structures.

Basically there are two types of data arrangement: longitudinal structure and latitudinal structure. (See Figure 1.1 and 1.2) In the former arrangement, values of all measures on each object are stored in the same variable. There will be multiple rows for each object. The key will be *object* and *measure*. In the latter, values of each measure are stored in a separate variable. There is only one observation for each object. Variable *Object* is the key itself. Most data transpositions switch data between these two structures.

**Figure 1.1: Longitudinal Structure**

| *object* | *measure* | *value* |
|---|---|---|
| A | ma | 12 |
| A | mb | 32 |
| A | mc | 12 |
| B | ma | 11 |
| B | mb | 33 |
| B | mc | 14 |
| … … | | |

**Figure 1.2: Latitudinal Structure**

| *object* | *ma* | *mb* | *mc* |
|---|---|---|---|
| A | 12 | 32 | 12 |
| B | 11 | 33 | 14 |
| … … | | | |

## II. Data Step and Data transposition

Lots of SAS users frequently write Data Step code to transpose their data sets. That might be because they are more comfortable with Data Step programming, or they like to perform extra data manipulation tasks within the same code, or they think they would be able to have a "better" control of the process.

Here is an example from work by the University of Michigan Kidney Epidemiology and Cost Center (UM-KECC). UM-KECC creates Dialysis Facility Reports (DFRs) annually for all dialysis service providers in the United States. Research data about the providers are identified and linked by their Medicare Provider Numbers in various data sources. Because a facility may change its provider number due to ownership changing or other reasons, a facility may have more than one provider number associated with it during a 4 year DFR period. In order to create a facility level data file we need to map all related provider numbers to their main provider numbers. Figure 2.1 shows a simplified facility data set. In this data set, variable *provnum* contains the main numbers and *altprovnum* contains all the related numbers. To have one observation for each main provider requires a data transposition process. Let's see how a Data Step can help here.

**Figure 2.1 PROVNUM_LONG**

| | provnum | state | altprovnum |
|---|---|---|---|
| 1 | 262319 | MO | 260020 |
| 2 | 262319 | MO | 262319 |
| 3 | 262320 | MO | 262320 |
| 4 | 262320 | MO | 263300 |
| 5 | 262325 | MO | 260137 |
| 6 | 262325 | MO | 262325 |
| 7 | 262325 | MO | 262339 |
| 8 | 262326 | MO | 262326 |
| 9 | 262326 | MO | 263302 |
| 10 | 262337 | MO | 260040 |
| 11 | 262337 | MO | 262337 |
| 12 | 162500 | IA | 162500 |
| 13 | 162501 | IA | 162501 |

First, the input data has to be sorted or indexed appropriately (or grouped). If it's not sorted and not indexed, the **NOTSORTED** option is required in the BY statement. This is always true whenever a BY statement is present in your SAS code. Second, ARRAY and RETAIN statements are used to store and process data values. Third, the BY statement and the SAS automatic variables FIRST.*variable* and LAST.*variable* are used to control the process.

Here is the Data Step code:

**CODE 2.1**

```
data provnum_lat;
set provnum_long;
by provnum notsorted; 1
retain prov1-prov3; 2
array prov{3} $ ;
if first.provnum then do; 3
   c = 1;
   do i = 1 to 3;
      prov{i}=' ';
   end;
end;
else c+1;
if c le 3 then
prov{c} = altprovnum;
if last.provnum then do;3
  multi  = (c gt 1);
  output;
end;
drop altprovnum c i;
run;
```

The output of **code 2.1** is shown in **Figure 2.2** below.

**Figure 2.2 PROVNUM_LAT**

| | provnum | state | prov1 | prov2 | prov3 | muti |
|---|---|---|---|---|---|---|
| 1 | 262319 | MO | 260020 | 262319 | | 1 |
| 2 | 262320 | MO | 262320 | 263300 | | 1 |
| 3 | 262325 | MO | 260137 | 262325 | 262339 | 1 |
| 4 | 262326 | MO | 262326 | 263302 | | 1 |
| 5 | 262337 | MO | 260040 | 262337 | | 1 |
| 6 | 162500 | IA | 162500 | | | 0 |
| 7 | 162501 | IA | 162501 | | | 0 |

Transposing PROVNUM_LAT backward to PROVNUM_LONG with a Data Step is a straightforward task (and not shown here).

You may have noticed that PROVNUM_LONG has a slightly different structure than that of **Figure 1.1**. There

are repeated measurements for a provider and *altprovnum* is the only measure. If there are multiple variables (measures) with repeated measurements, we just need a little revision to the Data Step code listed above. (See **APPENDIX 1)**

## III. PROC TRANSPOSE

As SAS Documentation says, "The TRANSPOSE procedure can often eliminate the need to write a lengthy Data Step to achieve the same result." In deed, to reshape PROVNUM_LONG to PROVIDER_LAT with Proc Transpose, you just need the following lines.

**CODE 3.1**

```
proc transpose data=provnum_long
   out=provnum_lat
   (where=(prov1 ne '') drop=_:)
   prefix = prov;
by provnum notsorted;
var altprovnum;
copy state;
run;
```

However, you may still need an extra Data Step to process extra variables.

```
data provider_lat;
set provider_lat;
if prov2 ne '' then multi = 1;
else multi = 0;
run;
```

We can also transpose data set PROVIDER_LAT to data set PROVIDER_LONG with Proc Transpose.

**CODE 3.2 (reverse)**

```
proc transpose data=provider_lat
               out=provider_long
(drop=_:   where=(altprovnum   ne
'')   rename=(col1=altprovnum));
by provnum state;
var prov1-prov3;
run;
```

When using the Data Step to transpose a data set, you have to know the maximum value of numbers of observations in by groups. In other words, you have to know how many variables will be created in the output dataset (for example, 3 variables in code 2.1) However, in code 3.1, Proc Transpose will take care of this.

To take full advantage of Proc Transpose, we have to understand its statements and options and how they work.

The Proc Transpose syntax is

```
PROC TRANSPOSE
<DATA=input-data-set>
<OUT=output-data-set>
<PREFIX=prefix>
<NAME=name> <LABEL=label> <LET>;

BY <DESCENDING> variable-1
<...<DESCENDING> variable-n>
<NOTSORTED>;

COPY variable(s);
ID variable;
IDLABEL variable;
VAR variable(s);
```

The **PROC TRANSPOSE** statement specifies the input and output data sets and lets you provide optional values for naming and labeling the output data set. (Although input and output data set names are optional, it's a good programming practice to specify them.)

The **PREFIX=** option specifies a prefix to name newly created variables in the output data set. It can work together with a ID variable. If the **ID** statement, which specifies a variable to name the transposed variables, is present, then the combinations of the value of PREFIX= and values of the ID variable will be the names of transposed variables in the

output data set. If the ID variable has duplicated values, the **LET** option will keep the last occurrence of each ID value within the data set (or BY group). Values of the **IDLABEL** variable will label transposed variables in the output data set. The **NAME=** and **LABEL=** options let you rename the **_NAME_** and **_LABEL_** variables which contain, respectively, names and their labels of variables listed in the VAR statement.

The **VAR** statement lists variables to transpose. If no VAR statement is present, by default, all numeric variables except the ones listed in the BY, COPY and ID statements are variables to transpose.

The **BY** statement lists variables that will be used to form BY groups. Proc Transpose doesn't transpose data by groups. It produces one observation for each BY group for each variable to be transposed. If the **notsorted** option is used, the index on the BY variables will be ignored.

**CODE 3.3 (BY)**

```
proc transpose data=provider
               out=provider_tran
                 (drop=_:)
               prefix = prov;
by provnum state notsorted;
var altprovnum;
run;
```

We can revise **CODE 3.1** because *state* can be used as a group variable. The revised code (see **CODE 3.3**) eliminates the usage of the **COPY** statement and the WHERE= data set option which was used to remove the extra observations created due to the use of COPY statement. Without the WHERE= option, the output of **CODE 3.1** would look like the one shown in Figure 3.1.

If a variable is not listed in any of these statements: BY, VAR, ID, IDLABEL, it will be dropped from the output data set. Using the COPY statement lets users save these variables in the output data set for further processing. However the way in which Proc Transpose "copies" variables can be puzzling.

**Figure 3.1 Without WHERE= option**

| | provnum | stat | _NAME_ | prov1 | prov2 | prov3 |
|---|---|---|---|---|---|---|
| 1 | 162500 | IA | altprovnum | 162500 | | |
| 2 | 162501 | IA | altprovnum | 162501 | | |
| 3 | 262319 | MO | altprovnum | 260020 | 262319 | |
| 4 | 262319 | MO | | | | |
| 5 | 262320 | MO | altprovnum | 262320 | 263300 | |
| 6 | 262320 | MO | | | | |
| 7 | 262325 | MO | altprovnum | 262325 | 262339 | 260137 |
| 8 | 262325 | MO | | | | |
| 9 | 262325 | MO | | | | |
| 10 | 262326 | MO | altprovnum | 262326 | 263302 | |
| 11 | 262326 | MO | | | | |
| 12 | 262337 | MO | altprovnum | 260040 | 262337 | |
| 13 | 262337 | MO | | | | |

SAS Documentation says the COPY statement **"**copies variables directly from the input data set to the output data set without transposing them." To better understand how it works, **CODE 3.4** tries to simulate the COPY process.

**CODE 3.4 (COPY Simulation)**

```
data
provider_long(index=(provnum));
input
provnum $ state $ altprovnum $;
cards;
 262319     MO      260020
 262319     MO      262319
 262320     MO      262320
 262325     MO      262325
 262325     MO      262339
 262326     MO      262326
 262326     MO      263302
 262337     MO      260040
 262337     MO      262337
 162500     IA      162500
 162501     IA      162501
 262320     MO      263300
 262325     MO      260137
run;

proc transpose
data=provider_long
```

```
   out=provnum_lat
   prefix = prov;
by provnum ;
var altprovnum;
copy state;
run;
*1 subset TOCOPY;
data tocopy;
set provider_long;
by provnum;
keep provnum state;
run;
*2 NOCOPY: transpose without
COPY;
proc transpose
data=provider_long
   out=nocopy
   prefix = prov;
by provnum ;
var altprovnum;
/*copy state; */
run;

* 3. Pad observations into
NOCOPY for each group;
data pad_missing;
set tocopy;
by provnum;
keep provnum;
if not first.provnum then
output;
run;
data nocopy_paded;
set nocopy pad_missing;
by provnum;
run;
* 4. One-to-one merge;
option mergenoby=warn;
data final;
merge tocopy nocopy_paded;
run;
* Check;
proc compare
   data=final
   compare=provnum_lat ;
run;
```

First, a subset of the input data set is created in which only variables listed in the BY and COPY statements are kept. Let's name it TOCOPY. Second, Proc Transpose carries out the same transposition without the COPY statement, which creates the output data set NOCOPY. Third, if TOCOPY and NOCOPY have the same number of observations for each BY group, then a one-to-one merge is performed to produce the final output data set. If not, fill the one that has fewer observations with observations containing only missing values to make sure there are the same number of observations for each BY group in these two data sets before the one-to-one merge is performed to create the final output file.

In this simulation code, because the data set NOCOPY has fewer rows than the data set TOCOPY, the data set PAD_MISSING is created to make sure the data set NOCOPY_PADDED has the same number of observations as the data set TOCOPY. Then a one-to-one merge is performed on them to mimic the result of **Code 3.1** (See **Figure 3.1**).

Proc Transpose is an interesting procedure. It is a powerful data transposition tool. It has a simple syntax with just a few options. However, the structure of the output data set is determined by the characteristics of the input data set, the combinations of the Proc Transpose statements and options users used, and the way in which Proc Transpose works them together to construct the output data set. Some complex data transpositions involve multitasks of Proc Transpose and the syntax is sort of puzzling to some users. Proc Transpose requires quite a bit of practice to produce the users' expected results. That may encourage lots of users to carry out their data transposition jobs with other approaches instead of Proc Transpose. For some data transposition jobs, Proc Report can be a much easier alternative.

## IV. Data Transposition with Proc Report (and more)

Proc Report is designed to generate data reports. It allows users to include statistics and to compute new variables. Proc Report provides an ACROSS column type that could be used (along with the OUT= option) to transpose numeric variables. For example, the following code will transpose the data set shown in **Figure 1.1** to the data set in **Figure 1.2**. (My solutions with Proc Report to some data transposision problems posted on SAS_L website can be found in **APPENDIX 2**)

```
proc report data=long
    out=lat(drop=_break_) nowd;
column object measure, value;
define object/group noprint;
define measure/across noprint;
define value/mean noprint;
quit;
```

To transpose data with Proc Report, the group variables don't have to be sorted or indexed in the input data set. The structure of the output data set is straightforwardly under the user's control via the COLUMN and DEFINE statements. Keep in mind there are some limitations to using Proc Report to transpose data. First, it can only transpose data from the LONG structure to the LAT structure. Second, all variables to be transposed must be numeric variables. That's because all variables under the ACROSS variable have to be defined as analysis variables, and only numeric variables can be defined as analysis variables. Also, there should be only one row for each combination of GROUP variables and ACROSS variables. That will let us safely use the MEAN or SUM operations on variables to be transposed. If not, Proc Report will not complain, but the result will be different. Proc Report also lets users create new variables with the column option COMPUTED and the COMPUTE block.

Here is another example. As mentioned above, UM-KECC links various ESRD data sources such as Medicare claims, medical evidence, patient events and death notification files with Medicare provider numbers to create yearly DFRs. During the process, we need to count the usage of each provider number in these data sets each year. **Figure 4.1** is a simplified provider number usage file. For facility level analyses, we need to transpose it into a facility level data set. The output data set has one row for each facility and contains variables: *provnum, clm_2002-clm_2005, me_2002-me2005, evt_2002-evt2005, dn_2002-dn2005.*

**Figure 4.1**

```
data provnum_usage
    (index=(py=(provnum year)));
input
provnum $ year clm me evt dn;
cards;
112222 2003 1233 103 303 2
111111 2002 1232 102 302 3
112222 2002 1232 102 302 4
111111 2004 1234 104 304 4
112222 2005 1235 105 305 5
111111 2003 1233 103 303 3
113333 2002 1232 102 302 2
113333 2004 1234 104 304 5
113333 2003 1233 103 303 1
113333 2005 1235 105 305 1
111111 2005 1235 105 305 0
run;
```

In order to illustrate and compare the different methods of data transposition with SAS, we list more than one way to transpose this to a facility level data set. **Code 4.1, 4.2, 4.3** and **4.4** exercise Data Step or/and Proc Transpose techniques. **Code 4.5** brings into play

Proc Report as an unconventional data transposition approach.

**Code 4.1** also illustrates when multiple Proc Transpose steps, one for each variable, are needed to accomplish complicated data transposition.

**CODE 4.1 (Proc Transpose only)**

```
* use Proc Transpose only;
proc transpose
  data=provnum_usage
  out=provnum_usage1(drop=_:)
  prefix=clm_;
by provnum;
id year;
var clm;
copy year me evt dn;
proc transpose
  data=provnum_usage1
  out =provnum_usage2(drop=_:)
  prefix=me_;
by provnum;
id year;
var me ;
copy year clm: evt dn;
proc transpose
  data=provnum_usage2
  out =provnum_usage3(drop=_:)
  prefix=evt_;
by provnum;
id year;
var evt ;
copy year clm: me: dn;

proc transpose
  data=provnum_usage3
  out =provnum_usage4(drop=_:)
  prefix=dn_;
by provnum;
id year;
var dn;
copy clm: me: evt:;
run;
```

**CODE 4.2 (lengthy Data Step)**

This code uses a lengthy Data Step, with similar techniques to **CODE 2.1**, to carry out the same job.

```
%macro assign(year);
      clm_&year = clm;
      me_&year  = me;
      evt_&year = evt;
      dn_&year  = dn;
%mend;

data all;
set provnum_usage;
by provnum year;

array outvars[*]
    clm_2002-clm_2005
    me_2002 -me_2005
    evt_2002-evt_2005
    dn_2002 -dn_2005;
retain
    clm_: me_:
    evt_: dn_:;
if first.provnum then do;
      do i=1 to 16 ;
         outvars[i]=.;
      end;
end;
if year=2002 then do;
      %assign(2002); end;
else if year=2003 then do;
      %assign(2003); end;
else if year=2004 then do;
      %assign(2004); end;
else if year=2005 then do;
      %assign(2005); end;
if last.provnum then output;
drop clm me evt dn i year;
run;
```

**CODE 4.3 (simple Data Step)**

To get the exact same result data set, this code uses a much simpler Data Step with different techniques than **Code 4.2** based on the characteristics of this transposition. A macro is also utilized to set data set options for input data sets.

```
* use data step;
%macro subset(year);
provnum_usage (Where =
(year=&year)
rename= (clm=clm_&year
         me=me_&year
         evt=evt_&year
         dn=dn_&year))
%mend;
```

```
data all;
merge
   %subset(2002)
   %subset(2003)
   %subset(2004)
   %subset(2005);
by provnum;
drop year;
run;
```

**CODE 4.4 (Data Step & Proc Tran)**

```
*Data Step and Proc Tranpose;
proc transpose
  data=provnum_usage
  out =provnum_usage_t;
by provnum year;
run;

data provnum_usage_d;
set provnum_usage_t;
id=compress(_name_||'_'||year);
run;

proc transpose
  data=provnum_usage_d
  out =provnum_usage_lat3
       (drop=_:);
by provnum;
id id;
var col1;
run;
```

**CODE 4.5 (Proc Report)**

```
* use proc report;
%macro ren(vars,ids);
%let
v_ct=%eval(%sysfunc(countc("&var
s", ' '))+1);
%let
i_ct=%eval(%sysfunc(countc("&ids
", ' '))+1);
%put &v_ct &i_ct;
%do i=1 %to &i_ct;
%let id=%scan(&ids,&i);
%do v=1 %to &v_ct;
%let c=%eval((&i-1)*&v_ct+&v+1);
%let var=%scan(&vars,&v);
_c&c._ = &var._&id
%end; %end;
%mend;
proc report data=provnum_usage
nowindows out=all_rpt
  (drop=_break_  rename=(
%ren(clm me evt dn,2002 2003
2004 2005)));
column provnum year,(clm me evt
dn) ;
define provnum/group noprint;
define year/across noprint;
define clm/sum noprint;
define me/sum noprint;
define evt/sum noprint;
define dn/sum noprint;
quit;
```

## Conclusion

The Data Step has an unlimited capability for data manipulation; it can handle all kinds of data transpositions, simple or complex. That potential originates in the powerful SAS data step language which is a well-designed data processing tool. Proc Transpose is designed to perform typical jobs of data transformation. Like all other SAS procedures, it might not cover all situations users may encounter in the real world. (See examples in the appendixes) Proc Report is intended to generate various data reports with selected statistics. It can be an alternative to transposing some data sets, although its use can be limited. Users can always choose appropriate methods to get the job done according to the formats of their input and output data sets and their acquaintance with each technique. As illustrated in code 4.4, using both Data Step and Proc Transpose together may sometimes be the best choice. But as shown in code 4.5, using Proc Report has proved a good solution for users who are familiar with Proc Report. (See attachment for more data transposition examples from SAS_L website).

**APPENDIX 1**

## Multiple Rows on Multiple Measures

- **Data Step**

```
* A1D1: long to lat;
data long;
input object$ measure$ value;
cards;
A    ma  12
A    ma  13
A    mb  32
A    mb  32
A    mc  12
B    ma  11
B    mb  33
B    mc  14
run;

data lat;
set long;
by object measure;
retain ma1 ma2 mb1 mb2 mc1 mc2;
* store repeated values;
array ma{2} ;
array mb{2} ;
array mc{2} ;
if first.object then do;
* initialization ;
      ma1=.;ma2=.;
      mb1=.;mb2=.;
      mc1=.;mc2=.;
end;
if first.measure then ct=1;
else ct+1;
if ct<3 then do;
      if      measure='ma' then
              ma[ct]=value;
      else if measure='mb' then
              mb[ct]=value;
      else if measure='mc' then
              mc[ct]=value;
end;
if last.object then output;
drop measure value ct;
run;
```

A1D1 output:

```
object ma1 ma2 mb1 mb2 mc1 mc2
A      12  13  32  32  12   .
B      11   .  33   .  14   .
```

```
* A1D2: lat to long;
data long2;
set lat;
array values{*} ma1 ma2 mb1 mb2
mc1 mc2;
do i=1 to 6;
      if values[i] then do;
            if i <3 then
            measure='ma';
            else if i <5 then
            measure='mb';
            else measure='mc';
         vaule=values[i];
         output;
      end;
end;
drop i ma1 ma2 mb1 mb2 mc1 mc2;
run;
```

- **Proc Transpose**

1. Can't transpose data LONG directly to data LAT.
2. Can transpose LONG to a similar structure with multiple steps.

A1T1 output:

```
object    ma    mb    mc
  A       12    32    12
  A       13    32     .
  B       11    33    14
  B        .     .     .
```

```
* A1T1: long to lat;
proc transpose data=long
out=lat_t1 (drop=_:);
by object measure;
run;
proc transpose data=lat_t
out=lat_t2  (drop=_:);
by object ;
id measure;
run;
```

**APPENDIX 2**

### Data Transposition problems on SAS_L

**POST 1:** (February 14, 2005)

http://listserv.uga.edu/cgi-bin/wa?A2=ind0502B&L=sas-l&P=R38365

Sent: Monday, February 14, 2005 3:05 PM
To: SAS-L@LISTSERV.UGA.EDU
Subject: Data Transpose question

Hello everybody,

I have a data set that looks like this, and has values of missing or greater than zero for weeklysales 1-7, Key2 has values from 1-8, Key1 has values from 1-5:

Key1 Key2 WeeklySales1..... WeeklySales7
1 1
1 2
2 7

I would like the final dataset to look like below, with 5 rows representing 5 values for Key1:

Key1 Weeklysales1Key2=1..... Weeklysales7Key2=1 WeeklySales1Key2=2....

where the column names would be a combination of WeeklySales1-7 and Key2 (ranges from 1-8).
Thanks for your help.
Regards,
Shukla

**CODE A2.1 (Solution with Proc Report)**

```
data sale;
input key1 key2 sale1- sale7;
cards;
1 1 1 2 3 4 5 6 7
2 2 1 2 3 4 5 6 7
3 3 1 2 3 4 5 6 7
4 4 1 2 3 4 5 6 7
5 5 1 2 3 4 5 6 7
5 6 1 2 3 4 5 6 7
5 7 1 2 3 4 5 6 7
5 8 1 2 3 4 5 6 7
run;

proc report data=sale out=sale_r
(rename=(%ren(sale1 sale2 sale3
sale4 sale5 sale6 sale7, 1 2 3 4
5 6 7 8))) nowd ;
column key1 key2,(sale1-sale7);
define key1/group noprint;
define key2/across noprint;
define sale1/sum noprint;
define sale2/sum noprint;
define sale3/sum noprint;
define sale4/sum noprint;
define sale5/sum noprint;
define sale7/sum noprint;
define sale7/sum noprint;
quit;
```

**POST 2:** (15 Mar 2005)

http://listserv.uga.edu/cgi-bin/wa?A2=ind0503C&L=sas-l&P=R2050

Actually, I have several variables for each question, so the original data like below:

```
data sample;
input id ques $ answ1 answ2;
cards;
1 q1 3 12
1 q2 4 13
1 q3 3 15
2 q1 4 11
2 q2 4 14
2 q3 3 13
3 q1 3 15
3 q2 2 16
3 q3 4 17
;
```

Then I want the final data format like:

| id | answ1_q1 | answ1_q2 | answ1_q3 | answ2_q1 | answ2_q2 | answ2_q3 |
|---|---|---|---|---|---|---|
| 1 | 3 | 4 | 3 | 12 | 13 | 15 |
| 2 | 4 | 4 | 3 | 11 | 14 | 13 |
| 3 | 3 | 2 | 4 | 15 | 16 | 17 |

;

Can SAS recode the data to above format? I did try adding 'answ1' and 'answ2' after the statement 'var', but all data only use q1, q2, q3 as the top variable name so each id repeated twice for answ1 and answ2, like:

```
id q1  q2 q3
1  3   4   3
1  12 13  15
2  4   4  3
2  11 14 13
;
```

Thanks for the help.
Jane

**CODE A2.2R (with Proc Report)**

```
data sample;
 input id ques $ answ1 answ2;
 cards;
 1 q1 3 12
 1 q2 4 13
 1 q3 3 15
 2 q1 4 11
 2 q2 4 14
 2 q3 3 13
 3 q1 3 15
 3 q2 2 16
 3 q3 4 17
;

proc report data=sample
out=sample_r(rename=(%ren(answ1
answ2,q1 q2 q3))) nowd;
column id ques,(answ1-answ2);
define id/group noprint;
define ques/across noprint;
define answ1/sum noprint;
define answ2/sum noprint;
quit;
```

**CODE A2.2TT (with Proc Transpose)**

```
proc transpose data=sample
out=tran1(drop=_name_)
```

```
 prefix=answ1_;
   by id;
   id ques;
   var answ1 ;
   copy answ2;
run;
proc transpose data=tran1
out=sample_t(drop=_name_)
prefix=answ2_q;
   by id;
   var answ2 ;
   copy answ1_:;
run;
```

**POST 3:** (9 Jul 2006)
http://listserv.uga.edu/cgi-bin/wa?A2=ind0607D&L=sas-l&P=R30418

Hi everyone - I really hope someone can help me with the following problem. I have a list of teams playing each other and I want to convert the table I have - which is in the form of a Team1 identifier, Team2 identifier followed by the number of times that Team1 have beaten Team2(T1Win), the number of times Team2 has beaten Team1 (T2Win) and finally the number of times that the games have finished in a tie/draw (Tie12). I show an example here:

```
data initial;
input Team1 $ Team2 $ T1Win T2Win Tie12;
datalines;
A B 5 6 2
A C 4 7 3
B C 3 7 2
E A 4 3 1
E D 7 6 4
;
```

In my real dataset I have over 100 teams rather than just the 5 shown here (A-E). What I am trying to do is to transpose the data somehow so that what I have is a decomposition of the data into individual lines, 3 lines per one line in the above data, where all of the team dentifiers become variables, team1 winning is signified with a 1, a losing team by -1 and if it is a tie we put a 0.5 in each of those teams columns, a final column then shows how many times that result has occurred. Much easier to show what I am hoping to transform the above into (I just show the decomposition of the first 3 of the lines from data set initial to save space):

```
 data required;
 input A B C D E Counter;
 datalines;
 1 -1 0 0 0 5
 -1 1 0 0 0 6
 0.5 0.5 0 0 0 2
 1 0 -1 0 0 4
 -1 0 1 0 0 7
 0.5 0 0.5 0 0 3
 0 1 -1 0 0 3
 0 -1 1 0 0 7
 0 0.5 0.5 0 0 2
 ;
run;
```

Just to highlight again - my actual dataset has over 100 teams, otherwise given the time I have been trying to do this I would have done it manually!
Hope someone can help me achieve what I am trying to do!
Andrew

**CODE A2.3TD (method 1)**

```
* THIS ONE IS A LITTLE BIT
COMPLEX. We will do it with both
DATA STEP and PROC TRAN.;
data team;
input team1 $ team2$ t1win t2win
tie12;
cards;
A B 5 6 2
A C 4 7 3
B C 3 7 2
E A 4 3 1
E D 7 6 4
;
run;

* easy to understand;
proc transpose data=team
out=tran1(rename=(col1=counter))
;
by team1 team2 ;
run;

data step1;
set tran1;
group=ceil(_n_/3);
if _name_= 't1win' then do;
v1=1;v2=-1;end;
else if _name_= 't2win' then do;
v1=-1;v2=1;end;
else do;v1=.5;v2=.5;end;
rename=compbl('V1='||team1||'
'||'V2='||team2);
if mod(_n_,3) = 0 then
call ymput(compress('r'||group),
rename);
drop _name_ rename team1 team2 ;
run;

* rename variables and
concatenate them group by group;
%macro ren(group);
data final;
set
%do i=1 %to &group;
step1 (where=(group=&i)
rename=(&&r&i))
%end;
drop group; run;
%mend;
```

**CODE A2.3TTDT (method 2)**

```
proc transpose data=team
out=tran1 name=pair
prefx=counter;
by group;
```

```
copy team: v:;
run;

proc transpose data=tran1
out=tran2(drop=id2 id3
index=(pair))
name=tm prefix=id;
by group;
var team1 team2;
copy pair counter1 v:;
run;

data score ;
pair='t1win'; v1=1; v2=-1;v3=.5;
output;
pair='t2win'; v1=-1; v2=1;v3=.5;
output;
run;

data tm_score ;
merge tran_2 score;
by pair;
run;

proc transpose data=tm_score
out=final ;
by group;
var v:;
id id1;
copy counter1;
run;
```

## REFERENCE

SAS Institute Inc., *Base SAS(R) 9.1.3 Procedures Guide*, 2004, Cary, NC, USA.

## ACKNOLWGEMENT

I'd like to thank Sandy Callard, Richard Eikstadt, Jeffrey Pearson and Randy Webb for their valuable comments, suggestions and support.

## CONTACT

Questions and comments are welcomed. Please feel free to contact the author:

**Lingqun Liu**
Kidney Epidemiology and Cost Center
University of Michigan
315 W. Huron, Suite 240
Ann Arbor, MI 48103
Phone: (734) 998-6609
Fax: (734) 998-6620
Email:lqliu@umich.eud