Transposing Tables from Long to Wide: A Novel Approach Using Hash Objects

Joseph Hinson, Merck Sharp & Dohme Corp., Rahway, NJ
Changhong Shi, Merck Sharp & Dohme Corp., Rahway, NJ

ABSTRACT

Transposing tables is often a necessity in data analysis. In clinical studies, some data, for example laboratory data, are collected in a longitudinal manner. Yet a horizontal form of such data may be more suited for statistical analysis. SAS® provides the TRANSPOSE procedure for such purposes, but this approach can be quite challenging. SAS® 9 introduced the use of hash objects for the purpose of providing fast table lookups and merging without the need for pre-sorting data. In this paper, we explored an entirely different aspect of the technique - the ability of hash objects to look at a whole table as a matrix in a DATA step rather than observation-by-observation. This allowed us to easily rearrange data in a table.

INTRODUCTION

Statistical programmers often encounter situations where a data table requires restructuring. Clinical data management systems typically rely on "normalized" (vertical) relational data structure for optimal data management. Yet various statistical procedures require that information being analyzed be on the same observation, the dependent variable, as well as the independent variables. In other situations, the ease of coding is greatly influenced by the structure of the data table whether horizontal or vertical. Thus, table transposition has become an essential part of everyday SAS® programming.

Two main approaches have been available for transposing data with SAS®:

(a) the use of PROC TRANSPOSE, and
(b) DATA step programming

Also, programmers have used the SUMMARY procedure to transpose data, but for most situations, PROC TRANSPOSE offers a quick and straightforward solution to data restructuring. However to the novice programmer, the use of the PROC TRANSPOSE could prove quite challenging with an unpredictable outcome.

If the goal of the programmer is greater flexibility, then DATA step programming is usually the method of choice for rearranging data.

We hereby present yet another flexible approach which relies on a technique in SAS® - hash objects programming, which could lead to greater ease in data table restructuring.

HASH OBJECTS

Hash objects are memory-resident tables with efficient data look-up methods. They are part of the SAS® DATA step Component Objects. The term "hashing" simply refers to the mathematical algorithms responsible for the highly-efficient direct data lookup. These memory-resident tables are considered "objects" because they possess associated methods and attributes. For instance, if an "object" called "dm" is created, one of its several methods would be "dm.find()", and one of its attributes would be "dm.item_size". One can create several hash objects within a single DATA step with each object having its own associated methods and attributes. For example, dm.find() can be used to obtain the age of a subject, vs.find() to pull out that subject's blood pressure, cm.check() to see if the subject was taking prohibited medication, pv.add() to include the subject in the list of protocol violators, and rand.num_items to determine the total number of randomized subjects. However, the advantages of using hash objects are mainly seen in fast table look-ups and speedy sort-less merging.

The speed associated with hash table look-ups is primarily because the objects are RAM memory-resident thereby avoiding slow disk-based information access. The hashing process further provides efficiency through a direct-addressing of data elements, and since every data value in a hash table has an associated key, data retrieval can be completed in a random-access manner. It is this last property that is exploited in this paper for table transposition.

TRANSPOSING VIA HASH TABLES

The DATA step, with its traditional implicit loop, operates on data observation in a linear fashion. With hash objects, one can access observations back and forth, and in no particular order. The table as a whole, like a matrix, becomes available for manipulation. Any data element from a vertical table structure can be instructed, via hash keys, to go to
any position within a new horizontal table formation. The unique keys make sure the right data element goes to the right place in the new "wide" table. Additionally, hash tables automatically select the unique variables as row elements (unless specified, as in SAS® v9.2 and v9.3, duplicate keys are not allowed in hash tables, thus, enforcing referential integrity). This restriction is rather exploited in this paper to make transposition possible.

**TABLE TRANSPOSITION AND COLUMN NAMES**

Problematic to any table transposition method is how to make accurate DATA VALUES from the vertical table into column VARIABLE NAMES in the transposed horizontal structure. We found that using certain SAS® data access functions made such conversions straightforward.

**DATA ACCESS FUNCTIONS**

SAS® provides several data access functions (which are also available for SAS® Component Language or SCL programming). The functions used in this paper are:

- **OPEN()** - opens a data set and creates an empty data set data vector (DDV)
- **FETCHOBS()** - retrieves an observation from the opened data set and places it in the DDV
- **VARNUM()** - determines the variable position number
- **GETVARC()** - obtains the current value of a character variable from the DDV
- **GETVARN()** - obtains the current value of a numeric variable from the DDV
- **CLOSE()** - closes the data set

With the assistance of the Macro facility, the GETVARC() and GETVARN() functions provided a convenient way to convert character data values to column variable names. Once the character value of a variable is obtained, it is a simple matter to assign a data value to the new variable as shown below:

```sas
%let tag = %sysfunc(getvarc(dataset_id_number, target_variable));
%let value = %sysfunc(getvarn(dataset_id_number, target_variable));
&tag. = &value.;
```

**Explanation:**

First, a data access function, **getvarc**, is used to fetch the character value of the desired variable ("target_variable"). This character data would be transformed into a variable name. The dataset_id_number identifies the proper data set and is issued when that data set is opened with the **OPEN** data access function.

The **getvarn** function is used to obtain the numerical value from the target numerical variable, and the value assigned to the text (**&tag**) obtained with the **getvarc** function.

By assigning a value to **&tag**, a new variable (**&tag**) automatically gets created.

**THE STRATEGY FOR TRANSPOSING**

The entire effort of transposing with hash objects can be considered a two-step strategy:

A. create an empty hash table shell resembling the desired wide structure
B. fill the new wide hash table shell with data from the source long table, while creating new column names from source text data being transposed

Let's consider the clinical study example:

**Example-1: Transposing Laboratory Data (LB domain):**

```sas
data lbtable;
  length subjid $6 labtest $3 ;
  infile datalines;
  input subjid $ labtest $ result;
```

A. CREATING THE EMPTY WIDE HASH OBJECT

The process involves the creation of macro variables for the new rows and columns of the transposed table:

1. Create a macro variable, &longtable, for the table to be transposed (e.g., LBTABLE).
2. Assign a variable to form the new columns: &colvar.
3. Assign a variable for the new rows: &rowvar.
4. Assign a variable to hold the data values: &datavar.
5. Additional macro variables are created to provide the number of observations, variable lists for hash objects, list for call missing function, and attrib statement. Some lists require quotes, comma separations, both, or neither:
   - Total number of observations to transpose: &obsn.
   - Variable list suitable for the ATTRIB statement: &coltext.
   - Variable list suitable for hash object DefineData method: &colnames.
   - Variable list suitable for Call Missing function: &collist.
6. Initialize variables &rowvar, &colvar, &coltext, to be used by hash object, with the "Call Missing" function.
7. Create an empty hash object, "wide", with key as &rowvar, and data elements as &colnames.

Empty wide hash table created (table has no rows yet):

```
<table>
<thead>
<tr>
<th>subj</th>
<th>ALT</th>
<th>AST</th>
<th>CPK</th>
<th>GGT</th>
<th>LDH</th>
<th>RBC</th>
<th>WBC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

B. FILLING THE EMPTY WIDE HASH OBJECT WITH DATA

This second step uses the unique keys of the hash object to direct data to the newly transposed columns. Data access functions are used to open data set, load observations, and retrieve data values.
Variable position numbers (obtained with VARNUM function) are used by the GETVARC and GETVARN functions to extract data from particular variables of the current observation.

The transposed columns get new variable names created from data values from the rows of the original long table. The new column variables are then assigned data values.

**STEPS:**

**Using data access function to pull data from the long table:**

A. **Using data access function to pull data from the long table:**

1. Open the &longtable data set and get its ID number, &dsid, and also create an empty DDV, using the OPEN data access function. (Other functions would also need this ID number to access the data set's data).

2. Fetch one observation (determined by &counter) from the data set and place into DDV, using the FETCHOBS data access function with &dsid and &counter as arguments.

3. Determine the variable position number, &xrow, of the row variable, using the VARNUM function with arguments &dsid and &rowvar.

4. Determine the variable position number, &xcol, of the column variable: using the VARNUM function with &dsid and &colvar as arguments.

5. Determine the variable position number, &xval, of the data variable using the VARNUM function with &dsid and &datavar as arguments.

6. Use the GETVARC function with &dsid and &xrow as arguments to obtain the current row label, &row, to use as a hash object key, &rowvar, by making &rowvar equal to "&row".

B. **Putting data into the hash table, WIDE:**

7. Use the hash FIND method and the current key value, &rowvar, to retrieve any existing data from the hash table, WIDE (this step is important for preserving data already in the hash table).

8. Retrieve current variable values to use as new data for updating the hash table, wide:

   &col is obtained with the GETVARC function with arguments &dsid and &xcol, and &val is obtained with the GETVARN function, using arguments &dsid, and &xval.

9. Convert column data into a new variable name and assign a data value: by making &col equal to &val.

10. Update row in hash table with all the new information: by using the REPLACE hash method.

11. Reset all current variables (provided by &collist) to missing:

   (this step is essential for preventing the carrying over of data, in cases of subsequent missing values).

12. Re-initialize the column variable &colvar with "call missing".

13. Repeat for other observations by going back to step (2).

14. Close the data set with the CLOSE data access function.

15. Save the completely-filled hash table into a data set called "widetable1", using the hash OUTPUT method.

---

Output 1. Transposed Laboratory Data

<table>
<thead>
<tr>
<th>ViewTable: Work.Widetable1</th>
</tr>
</thead>
<tbody>
<tr>
<td>subj</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
OTHER CLINICAL STUDY EXAMPLES

The full macro version, %long2wide, is based on the above algorithm but provides variable lengths and types as macro parameters, and also uses logic to detect variable types in order to automatically choose between GETVARC() and GETVARN() functions for obtaining variable values.

Parameters for macro %long2wide():

- **Longtable**: Long dataset to transpose
- **Rowvar**: Variable for rows in the transposed table
- **Stayvar**: Variable that stays in position (not transposed)
- **Colvar**: Variable to provide column names in transposed table.
- **Datavar**: Values for transposed variables,
- **Ctxtlen**: The length of the column variables in the transposed table (example: 6 for numeric, $15 for character)
- **staycol**: Flag (Y or N) for whether long table has variable that should not be transposed
- **outtable**: Name of transposed dataset (wide)

EXAMPLE-2: Transposing Vital Signs Data (VS domain) - An Example with Missing Values:

```sas
data vstable;
  length VSTESTCD $8 VSORRES 8 VSORRESU $10 SUBJID $6;
  format VSORRES 6.1;
  infile datalines;
  input VSTESTCD $ VSORRES VSORRESU $ SUBJID $;
  datalines;
    WEIGHT 103.1 kg   158712
    SYSBP   153   mmHg 158712
    RESP   20   breaths/min   158712
    DIABP   86   mmHg 158770
    PULSE   67   beats/min 158770
    WEIGHT   87.6  kg   158770
    DIABP   83   mmHg 158770
    WAIST   93   cm    158770
    BMI    22.1  kg/m2 159255
    DIABP   70   mmHg 159255
    TEMP  36.8   C    159255
    SYSBP  127   mmHg 159255
    WAIST   78   cm   159255
    WEIGHT   51.7  kg   159255
    RESP   18   breaths/min 159255
    BMI    40.7  kg/m2 158719
    HEIGHT  184    cm 158719
    WEIGHT   137.8 kg 158719
    PULSE   64   beats/min 158719
    SYSBP  136   mmHg 158719
    DIABP   82   mmHg 158719
    SYSBP  126   mmHg 158764
    WEIGH T   83.8  kg 158764
    DIABP   65   mmHg 158764
    PULSE  100   beats/min 158764
    SYSBP  125   mmHg 158764
    WAIST  109    cm 158764
  ;
run;
```
Transposing Tables from Long to Wide: A Novel Approach Using Hash Objects - continued

```%
long2wide(
    longtable=vstable,
    rowvar=SUBJID,
    stayvar=,
    colvar=VSTESTCD,
    datavar=VSORRES,
    ctxtlen=8,
    staycol=N,
    outtable=widetable2
);
```

THE RANDOM FILLING OF THE HASH TABLE ILLUSTRATED:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Result</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEIGHT</td>
<td>103.1 kg</td>
<td></td>
</tr>
<tr>
<td>RESP</td>
<td>20 breaths/min</td>
<td></td>
</tr>
<tr>
<td>DIAP</td>
<td>86 mmHg</td>
<td></td>
</tr>
<tr>
<td>PULSE</td>
<td>67 beats/min</td>
<td></td>
</tr>
</tbody>
</table>

**VIEWTABLE: Work:hash_filling_step_1**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103.1</td>
</tr>
</tbody>
</table>

**VIEWTABLE: Work:hash_filling_step_2**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103.1</td>
</tr>
</tbody>
</table>

**VIEWTABLE: Work:hash_filling_step_3**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103.1</td>
</tr>
</tbody>
</table>

**VIEWTABLE: Work:hash_filling_step_4**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103.1</td>
</tr>
</tbody>
</table>

**VIEWTABLE: Work:hash_filling_step_5**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103.1</td>
</tr>
</tbody>
</table>

(Ex extera)

Transposing Vital Sign Values

**VIEWTABLE: Work.Widetable2**

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>BMI</th>
<th>DIAP</th>
<th>HEIGHT</th>
<th>PULSE</th>
<th>RESP</th>
<th>SYSBP</th>
<th>TEMP</th>
<th>WAIST</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>66</td>
<td>.</td>
<td>20</td>
<td>153</td>
<td>.</td>
<td>.</td>
<td>103.1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>64</td>
<td>.</td>
<td>136</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>137.8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>65</td>
<td>.</td>
<td>100</td>
<td>125</td>
<td>.</td>
<td>.</td>
<td>83.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>83</td>
<td>.</td>
<td>67</td>
<td>.</td>
<td>.</td>
<td>93</td>
<td>87.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>.</td>
<td>18</td>
<td>127</td>
<td>36.8</td>
<td>78</td>
<td>51.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output 2. Fully Transposed Vital Signs Table (sorted)
EXAMPLE-3: Transposing Subject Characteristics Data (SC domain)
- An Example with Character Categorical Data:

```sas
data estable;
  set SUBID $ SCTEST $ SCORES $;
  inclue data lines;
  input SUBID $ SCTEST $ SCORES $;
  datalines;
  50019 Osteoarthritis N
  50019 Nephropathy N
  50019 Retinopathy N
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy Y
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy Y
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy N
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy N
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy N
  50020 Osteoarthritis N
  50020 Nephropathy N
  50020 Retinopathy N
  50020 Osteoarthritis N
  50020 Nephro
```

```sas
p途中ners=estable,
  robear=SUBID,
  stravar=|
  colvar=SCTEST,
  declear-SCORES,
  stvall-N,
  outtable=estable0
```

`; run;`
Output 3. Transposed Subject Characteristics Data

<table>
<thead>
<tr>
<th>SUBJ.</th>
<th>Nephropathy</th>
<th>Neuropathy</th>
<th>Osteomyelitis</th>
<th>Retinopathy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
<td>N</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>N</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>N</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE-4: **Transposing Study Medicine Data (SM domain)**

thereby Creating a Day-Level Table for Subjects ("data journal"):

```r
data metafile:

```longside
```r
search SUBJID DOSEDAY FILLSTAKEN @ FILLBOTTLE 
variable names:
input SUBJID DOSEDAY: date, PILLETAKEN FillBOTTLE & @;
format DOSEDAY dates:

datalines:
```longside
data:

<table>
<thead>
<tr>
<th>DOSEDAY</th>
<th>FILLSTAKEN</th>
<th>FILLBOTTLE</th>
<th>SUBJID</th>
</tr>
</thead>
<tbody>
<tr>
<td>21FEB2011</td>
<td>372</td>
<td>21FEB2011</td>
<td>E</td>
</tr>
<tr>
<td>21FEB2011</td>
<td>372</td>
<td>21FEB2011</td>
<td>E</td>
</tr>
<tr>
<td>21FEB2011</td>
<td>372</td>
<td>21FEB2011</td>
<td>E</td>
</tr>
<tr>
<td>21FEB2011</td>
<td>372</td>
<td>21FEB2011</td>
<td>E</td>
</tr>
</tbody>
</table>
```

```r
longside
```longside
Transposing Tables from Long to Wide: A Novel Approach Using Hash Objects - continued

Output 4. Restructured Day-Level Study Medicine Table

CONCLUSION

This paper has demonstrated that with a little reliance on data access functions and macro variables, it is feasible to use hash objects to transpose data. Although the first three examples could easily have been done with PROC TRANSPOSE, the fourth example, restructuring the study medicine data, requires a more sophisticated approach such as provided by the use of hash objects. The study medicine data set, which includes subject identification, dates of visits, start dates and stop dates for drug intake, number of pills taken and from which pill bottle, etcetera, is often used to evaluate study medicine compliance as part of protocol violation assessment. The subject’s compliance is typically assessed by counting the number of compliant days which is defined as the intake of a particular number of pills taken from specific pill bottles. To accomplish such assessments, clinical programmers frequently find the need to create several sub-tables before applying a suitable programming logic to compute compliance. Having, for each patient, a day-level record of drug intake that includes the date, the medication bottle labels, and pill quantity presented on the same observation, compliance programming becomes straightforward. It is expected that such use of hash objects will be extended to transposing tables from wide to long structures.

REFERENCES


ACKNOWLEDGMENTS

The authors would like to thank their management and colleagues who provided valuable input and comments on this paper.

CONTACT INFORMATION

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

Joseph Hinson*
Merck & Co., Inc.
RY34-A320
P.O. Box 2000
Rahway, NJ 07065
Phone: 732-594-7789
E-mail: joseph.hinson@merck.com

Changhong Shi
Merck & Co., Inc.
RY34-A3093S
Rahway, NJ 07065
Phone: 732-594-1383
E-mail: changhong_shi@merck.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

* Joseph Hinson is working for Merck under a contract with Agile-1, Torrance, CA, 90504