Paper 200-26

A Meta-Data Approach to Problem Solving

John R. Gerlach, Maxim Group, Plymouth Meeting, PA
Cindy Garra, IMS Health, Plymouth Meeting, PA

Abstract

Assume you have a data file such that a record represents a collection of rates juxtaposed with their respective categories, as follows:

[*item*]<*category*><*rate*><*category*><*rate*>

Under more typical circumstances, the file would be structured such that the rates would be located in their appropriate field denoting the categories, which would become the variable identifiers. In this case, however, the value of the category denotes the appropriate variable that would contain the respective rate. Moreover, the categories are nominal; in fact, neither ordered nor sequential, hence, array processing is not readily apparent. Even worse, not all categories are represented for a given record. So, then, how do you read the data properly without using a myriad collection of IF statements accompanied by lots of hard-coded values?

Most discussion about meta-data concerns the access and use of the dynamic aspects of a SAS session; such as: the WORK library, user-define formats, and macro variables. This paper exploits the concept of meta-data and explains how to create needed information that easily solved the aforementioned problem.

Introduction

Since the advent of Dictionary tables, found in SAS 6.07, there's been much discussion about meta-data and its usefulness. Usually, however, this good topic emphasizes information about existing data sets, along with sundry information about the SAS session. In contrast, this paper discusses the creation of meta-data, as a means to an end, used for the specific purpose of solving an otherwise nasty data processing problem.

The Problem

Consider an input (text) data file consisting of seven rates (denoted by ‘xx.x’) juxtaposed with their respective categorical assignment, which would indicate the appropriate column name in a SAS table, as follows.

```
001  1001 xx.x  2002 xx.x  3003 xx.x
002  1001 xx.x  5005 xx.x
003  2002 xx.x  6006 xx.x  7007 xx.x
004  3003 xx.x  4004 xx.x  5005 xx.x
005  2002 xx.x  3003 xx.x
006  1001 xx.x  5005 xx.x  7007 xx.x
```

Notice that a given record contains only those categories that have non-missing values for a given item, that is, not all categories are represented in the record; consequently, the records vary in length and content. Also noteworthy, the categories are nominal, not ordinal, so using array vectors is not an acceptable solution. At least, however, the record layout does explicitly indicate the categories represented.

Typically, the data file would have a more conventional structure, as shown below. Notice that each record represents every category such that the data values are located in their appropriate columns (i.e., categories), including missing values. Of course, in this case, the categories are implied, rather than explicitly indicated, and known only through knowledge of the record layout.

```
001  xx.x xx.x xx.x   .    .    .    .
002  xx.x   .    .    .  xx.x   .    .
003    .  xx.x   .    .    .  xx.x xx.x
004    .    .  xx.x xx.x xx.x   .    .
005    .  xx.x xx.x   .    .    .    .
006  xx.x   .    .    .  xx.x   .  xx.x
```

Ultimately, after reading this data file, an abstract (contents) of the SAS table might look like the following. Thus, our purpose is to create a SAS table (not normalized) whose primary key is the column ITEM and its dependent columns, the rates, represent every category.

```
#      Variable       Type    Len     Label
-------------------------------------------------
1      ITEM           Char    3       Item #
2      R1001          Num     8       Rate 1001
3      R2002          Num     8       Rate 2002
4      R3003          Num     8       Rate 3003
5      R4004          Num     8       Rate 4004
6      R5005          Num     8       Rate 5005
7      R6006          Num     8       Rate 6006
8      R7007          Num     8       Rate 7007
```

Certainly, the more conventional file would be easier to read and process. In fact, the only task would be to supplant the missing values with zeros, as requested, which is easily accomplished by traversing an array inside a DO loop. However, let's assume that it’s Monday and we are given the unconventional raw data file for processing.

The Obvious Solution

Since the aforementioned data file contains only seven rates, the SAS solution would not require too much tedious effort, that is, enumerating the categories and specifying the logic needed to accomplish the task. One reasonable first step, upon reading the raw data, would be to create a normalized table, as the following listing clearly illustrates.

```
ITEM    CAT              VALUE
----------------------------
001     01001              99.9
001     02002              99.9
001     03003              99.9
002     01001              99.9
002     05005              99.9
        < etc. >
006     01001              99.9
006     05005              99.9
006     07007              99.9
```

Then, after obtaining the normalized table, employ another Data step that transposes the data in order to attain the desired results, as shown below.

```
ITEM R1001 R2002 R3003 R4004 R5005 R6006 R7007
------------------------------------------------
001   99.9  99.9  99.9  00.0  00.0  00.0  00.0
002   99.9  00.0  00.0  00.0  99.9  00.0  00.0
003   00.0  99.9  00.0  00.0  00.0  99.9  99.9
004   00.0  00.0  99.9  99.9  99.9  00.0  00.0
005   00.0  99.9  99.9  00.0  00.0  00.0  00.0
006   99.9  00.0  00.0  00.0  99.9  00.0  99.9
```

The following Data step accomplishes the second task that produces the desired non-normalized table. Notice the techniques used in the Data step, such as: by-group processing using FIRST / LAST logic; array processing, and the tedious logic needed to correctly populate the array vector.

```
data rates;
   array rates{*} r1001 r2002 r3003 r4004 r5005
      r6006 r7007;
   retain r1001 r2002 r3003 r4004 r5005
      r6006 r7007;
   set rates;
      by item;
   if first.item
      then do i = 1 to dim(rates);
         rates{i} = 0;
         end;
   select(cat);
      when('1001') r1001 = value;
      when('2002') r2002 = value;
      when('3003') r3003 = value;
      when('4004') r4004 = value;
      when('5005') r5001 = value;
      when('6006') r6002 = value;
      when('7007') r7003 = value;
      otherwise;
      end;
   if last.item then output;
   keep item r1001--r7007;
run;
```

By the way, notice that the TRANSPOSE procedure was not employed in order to get the desired results. Why, not? Because the procedure would not have preserved the categorical assignement of the respective values.

For example, Item 002 contains values for categories R1001 and R5005. However, after transposing the normalized table, the value for category R5005 has been misplaced, denoted by the variable R2, because the Transpose procedure left justifies the values, thereby misplacing the value for category R5005. In fact, the output data set does not contain enough columns to represent the categories, even, because there is no single item (observation) in the data that manifests all the categories. Consider the illustration, below.

```
proc transpose data=rates out=t_rates prefix=r;
   var value;
   by item;
run;
```

```
ITEM    R1      R2      R3
------------------------
001    99.9    99.9    99.9
002    99.9    99.9      .
003    99.9    99.9    99.9
004    99.9    99.9    99.9
005    99.9    99.9      .
006    99.9    99.9    99.9
```

So, given a normalized table, a single Data step correctly accomplishes the task of transposing the data, albeit using somewhat tedious code, especially in regards to the SELECT / WHEN statement.

That was the 'obvious' solution. We proceed now to the less obvious, more interesting solution that uses contrived meta-data.

The Meta-Data Solution

Imagine that the raw data file manifests *over one hundred* nominal categories, rather than only seven. Given the normalized data set attained from the initial Data step, then, certainly, the second Data step would require substantial enhancements, as follows.

```
data rates;
   array rates{*} < enumerated categories >;
   retain < enumerated categories >;
   set rates;
      by item;
   if first.item
      then do i = 1 to dim(rates);
         rates{i} = 0;
         end;
   select(cat);
      when('1001') r1001 = value;
      when('2002') r2002 = value;

      < WHEN clause for each category >

      otherwise;
      end;
   if last.item then output;
   keep item r1001 -- < last category >;
run;
```

Besides enumerating all the categories when defining the array vector and specifying the RETAIN statement, for proper initialization and retention, you would need a WHEN clause for every category in order to assign each category with their proper, respective value, if it exists. A nasty, tedious task, indeed. Also, keep in mind that the table (the deliverable) must contain suitable column names that denote all the categories. And, of course, those categories not represented in the original file become rates whose value is zero.

Consider the following idea:

Recognize the fact that there are *n* categories, known *a priori*, which are explicitly indicated in the data. Now, in the Data step, implement an array that represents all the categories by enumerating all the categories in the array definition. Then, for each item (observation), any **category / value** pair will have its proper location in the array vector, that is, assign the respective value to that *i*th element in the array.

But, how can you associate the nominal category (e.g., R1000) to the respective array element? Simply, create a format that maps the categories to a number, 1 to *n*, which functions **as the index of the array**, denoting the location in the vector (*i*th cell), thereby ensuring proper assignment.

So, create a format $CATF that maps each category to an ordinal number that becomes the index to an array, as follows.

```
proc format;
   value $catf
      '1001' = 1   '1002' = 2   '1003' = 3
      '1004' = 4   '1005' = 5  ...
      '1020' = 10  '1021' = 11  '1022' = 12
      '1023' = 13  ...
      '1047' = 37  '1048' = 38  '1049' = 39
           :      :      :      :      :
      '2030' = 86  '2040' = 87  '2100' = 84
           :      :      :      :      :
      '3100' = 92  '3120' = 93  '3230' = 94
           :      :      :      :      :
      '4100' = 100 '4120' = 101 '4130' = 102
      '4140' = 103 '5000' = 104 '6000' = 105
      '7000' = 106 '8000' = 107 '9000' = 108;
run;
```

But, even that's too tedious. Instead, let's contrive meta-data that affords the means to create the Control data set for creating the format.

Creating the Meta-data

In order to preserve the names of the categories found in the original data file, it is necessary to define the macro variable &CATS that represents the enumerated list of categories. Then, the following Data step uses the macro variable to define the array CATS.

```
%let cats = r1001-r1009 r1020-r1049
      r1100-r1109 r1120-r1149 r2000 r2020
      r2030 r2040 r2100 r2120 r2130 r2140
      r3000 r3020 r3030 r3040 r3100 r3120
      r3130 r3140 r4000 r4020 r4030 r4040
      r4100 r4120 r4130 r4140 r5000 r6000
      r7000 r8000 r9000;

data cats;
   array cats{*}$5 &cats.;
   stop;
run;
```

Notice that the table CATS is empty. Keep in mind that we are interested in meta-data, not the data set *per se*. And, like all SAS® tables, it contains descriptor information about itself. That is, even though the table has no rows, it contains over 100 columns (i.e.; R1001, R1002, ...,). The following SQL procedure, using the COLUMNS Dictionary table, obtains the meta-data needed to create the desired format.

```
proc sql;
   create table cats as
   select name
      from dictionary.columns
      where libname eq 'WORK'
             and memname eq 'CATS';
quit;
```

The table CATS contains over 100 observations having a single column, called NAME, as shown in this multi-column listing, below.

```
NAME       NAME       NAME       NAME       NAME
------------------------------------------------
R1000      R1040      R1130      R2100      R4000
R1002      R1041      R1131      R2120      R4020
R1003      R1042      R1132      R2130      R4030
 : :        : :        : :       R2140      R4040
R1020      R1100      R1140      R3000      R4100
R1021      R1101      R1141      R3020      R4120
R1022      R1102      R1142      R3030      R4130
 : :        : :        : :       R3040      R4140
R1030      R1120      R2000      R3100      R5000
R1031      R1121      R2020      R3120      R6000
R1032      R1122      R2030      R3130       : :
 : :        : :       R2040      R3140      R9000
```

Creating the format

Once the table CATS exists, a Data step easily converts it into a Control data set, used by the FORMAT procedure, shown below. Recall that the categories, as indicated in the data, consist of integers only. Hence, the SUBSTR function eliminates the R-prefix that is needed to make-up the column names, in compliance with the syntax rules for user-defined names in SAS.

```
data catf(drop=name);
   retain fmtname 'catf' type 'C';
   length start $5;
   set cats;
   start = substr(name,2);
   label + 1;
run;

proc format cntlin=catf fmtlib;
run;
```

The following fascimile of output from the FORMAT procedure shows how the label of the $CATF format, similar to a 3rd Normal form relational table, will afford the means to attain the proper index in an array that represents all the categories and stores the rates.

```
|--------------------------------------------|
| FORMAT NAME: $CATF    LENGTH: 3  VALUES: 108 |
|------------|--------------|------------------|
| START      | END          | LABEL            |
-------------|--------------|------------------|
| 1001       | 1001         | 1                |
| 1002       | 1002         | 2                |
|   :        |    :         | :                |
| 3100       | 3100         | 92               |
|   :        |    :         | :                |
| 8000       | 8000         | 107              |
| 9000       | 9000         | 108              |
|--------------------------------------------|
```

The Final Step

After creating the contrived meta-data and the $CATF format, a single Data step accomplishes the task, straight away, unlike the ‘obvious’ solution that requires prior processing of the data.

To illustrate the improved solution, let's use the sample data, discussed earlier, that contains only seven categories. The Data step defines an array that denotes the seven categories (R1001, ..., R7007), enumerated by the macro variable &CATS. Then, a DO-loop initializes the vector, assigning the value zero to each element. Because of the structure of the raw data, it is necessary to use the MISSOVER option and the trailing @-sign in order to read the data properly. After reading the ITEM datum (and holding the input pointer in the record), the second DO-loop proceeds to read the category and its associated rate, in a pair-wise fashion, until the IF statement forces a timely exit. For each instance of a category / rate pair, the format $CATF imputes the proper location in the array vector, as a function of the category value. Finally, an assignment statement stores the rate in the appropriate element of the array, indicating the category.

```
%let cats = r1001 r2002 r3003 r4004 r5005
   r6006 r7007;

<< Code that creates meta-data and format >>

data rates;
   length cat $4;
   array rates{*} &cats.;
   do i = 1 to dim(rates);
      rates{i} = 0;
      end;
   infile cards missover;
   input @1 item $3. @;
   do i = 1 to dim(rates);
      input cat $ rate @;
      if cat eq ''
         then leave;
      indx = input(put(cat,$catf.),3.);
      rates{indx} = rate;
      end;
   output;
   keep item &cats.;
cards;
001  1001 99.9  2002 99.9  3003 99.9
002  1001 99.9  5005 99.9
003  2002 99.9  6006 99.9  7007 99.9
004  3003 99.9  4004 99.9  5005 99.9
005  2002 99.9  3003 99.9
006  1001 99.9  5005 99.9  7007 99.9
;
```

Observe the results generated by the PRINT procedure.

```
ITEM  R1001 R2002 R3003 R4004 R5005 R6006 R7007
-----------------------------------------------
001    99.9  99.9  99.9   0.0   0.0   0.0   0.0
002    99.9   0.0   0.0   0.0  99.9   0.0   0.0
003     0.0  99.9   0.0   0.0   0.0  99.9  99.9
004     0.0   0.0  99.9  99.9  99.9   0.0   0.0
005     0.0  99.9  99.9   0.0   0.0   0.0   0.0
006    99.9   0.0   0.0   0.0  99.9   0.0  99.9
```

Certainly, the new solution is more concise and very robust, by creating contrived meta-data along with the idea of mapping a nominal value (category) to an ordinal value that serves as the location in an array vector.

The following list summarizes the solution:

- Define a macro variable enumerating the categories;
- Create an empty SAS table containing meta-data that represents the categories;
- Obtain the meta-data using Dictionary tables;
- Create a Control input data set needed to create a format that maps a category to an ordinal value;
- Create a format that maps each category to its proper location in an array that represents all categories ;
- Process the raw data file using the format;
- Produce a listing of the results.

Conclusion

Given a nasty data processing problem that requires substantial tedious effort, it may be useful to consider using meta-data, *even contrived*, as part of a good idea that greatly facilitates the task.

The real-world assignment discussed in this paper illustrates a creative, alternative solution from the more arduous, conventional approach. Indeed, the use of contrived meta-data to solve a data processing problem gives new meaning to the phrase, itself.

Author Information

John R. Gerlach
Maxim Group
Plymouth Meeting, PA
610.832.5493

Cindy Garra
IMS Health
Plymouth Meeting, PA
610.834.5171