To Merge or Not to Merge: These are the Answers

John L. Badolato, Walsh America/PMSI, Newtown, PA

Abstract

The MERGE statement is a very useful tool for combining data from different sources. You may, however, encounter instances where MERGE limitations are exposed, especially when processing large data sets. Newer versions of the SAS® system have incorporated alternatives to the MERGE statement which can combine data using less computer resources to circumvent these limitations.

In this paper, I will explore the alternative merging techniques I have used to combine data between large data sets. These techniques include PROC SQL, PROC FORMAT with the CNTUN option and indexed data sets. Each technique has its own particular benefits for certain types of merging situations.

All the examples in this paper deal with the efficient and practical uses of these techniques for combining pharmacy terminal prescription data for the pharmaceutical industry, however, the techniques are useful for any audience.

Introduction

Using the SAS System as a Database Management System (DBMS), I have had numerous opportunities to combine data from different sources into a single data set of more useful information. The MERGE statement has served this function very well in most situations. However, there are some situations where limitations of the MERGE statement have hampered me. In order to accomplish goal of merging data, I needed to find alternatives techniques to MERGE.

Large Data sets

The problems I have encountered typically involved large data sets (millions of rows, over one hundred variables) where the data needed to be merged. The major problem has been computer resources even though our computer is a mainframe computer. Before two data sets can be merged using the MERGE statement, both data sets are required to be sorted. This is a very expensive operation and sometimes it is practically impossible to sort the data sets.

One option I have tried is to use SYNCSORT® before reading the data into the SAS system. This overcomes the problem of sorting within the SAS system, however, a second problem immediately becomes a reality. This deals with the fact that MERGE requires the data to be read first into a SAS data set and then merged. If the data are too large to be read into the SAS system, then you must use DATA _NULL_ to read, combine and write the data in the same step and the MERGE statement cannot be used in this fashion.

Outer Joins

Another problem was encountered during a redeployment of our sales force. I needed to find all possible and reasonable matches between sales representatives and newly defined sales territories. This required that I combine all sales representatives with all territories and only retain the combinations which a person could feasibly work based on travel distances. PROC SQL provides this capability.

Lookup Tables

Many situations also require merging data from more than one data set with a large master data set, each using a separate BY variable. With the MERGE statement, the data set would need to be sorted before each individual data set can be merged. This can be a cumbersome problem which wastes valuable computer resources. This task is very easy to accomplish using indexed data sets.

Data

The data I most often use is call pharmacy terminal data. It is the prescriptions that individual physicians write and are dispensed through retail pharmacies throughout the U.S. As you can expect, this is a very large amount of data and there are multiple merges which need to be made to the data to create better information. An example of the data is shown in Table 1.

Table 1. Sample data for RXS data set

<table>
<thead>
<tr>
<th>DOC_ID</th>
<th>PROD_ID</th>
<th>TABS1-TABS24</th>
</tr>
</thead>
<tbody>
<tr>
<td>01234</td>
<td>A</td>
<td>12 14 ... 16</td>
</tr>
<tr>
<td>23456</td>
<td>A</td>
<td>18 24 ... 19</td>
</tr>
<tr>
<td>45678</td>
<td>B</td>
<td>25 29 ... 35</td>
</tr>
<tr>
<td>67890</td>
<td>B</td>
<td>15 17 ... 14</td>
</tr>
</tbody>
</table>
All data which is not numeric is represented by codes. The codes identify items such as physician id and product. Physician characteristics such as specialty, number of sales calls and samples are very useful pieces of information which could be merged. Product specific information is also very important to be combined, such as cost per tablet or number of tablets taken per day (signa).

In order to combine these pieces of data with the prescription data, I found the following techniques to be very useful and they do not use excessive computer resources.

**Technique #1**

**PROC FORMAT CNTLIN Option**

The first technique I found is to use PROC FORMAT with the CNTLIN option to create a format from a SAS data set. This method requires that you create a data set with special variable names. The variables required are:

- **START**: Merging variable name
- **LABEL**: Information you wish to merge
- **TYPE**: C = Character, N = Numeric
- **FMTNAME**: Name for format

Another useful variable which can be included is:

- **HLO**: Indicates High, Low, Other

The HLO variable is used to indicate when the special values of HIGH, LOW or OTHER appear in the range.

An example of when this technique was useful is when accessing signa values for products to create a patient days of therapy value from number of tablets. For instance, if product A is taken one tablet per day and product B is taken two tablets per day, then it takes twice as many B tablets to equal the same patient days of therapy as product A. This example takes into account both the large data set problem as well as lookup tables.

The lookup table includes the product codes and signa. Table 2 shows sample data for this data set.

**Table 2. Sample data for LOOKUP data set**

<table>
<thead>
<tr>
<th>PROD_ID</th>
<th>SIGNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.0</td>
</tr>
<tr>
<td>B</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**PUT Function**

You must also use the PUT function to access the data from the format using the merge variable as the key. The PUT function allows you to create a character variable which will contain the desired value. If the value you are trying to access is a numeric, you must also use the INPUT function to translate the SAS character value into a SAS numeric value. The following SAS code performs a lookup table search and accesses the numeric value of signa.

```sas
DATA FMT;
  SET LOOKUP;
  LENGTH START FMTNAME $ 8 ;
  LENGTH LABEL $ 4 ;
  START = PROD_ID ;
  LABEL = PUT(SIGNA,4.1) ;
  FMTNAME = 'SIGNA' ;
  TYPE = 'C' ;

PROC FORMAT CNTLIN = FMT ;

DATA PDT ;
  SET RXS ;
  LENGTH SIGNA 8 ;
  SIGNA = INPUT(PUT(PROD_ID,$SIGNA.),4.1) ;
  ARRAY TABS(24) TABS1-TABS24 ;
  ARRAY PDT(24) PDT1-PDT24 ;
  DO I = 1 TO 24;
    PDT(I) = TABS(I) + SIGNA ;
  END ;

PROC APPEND BASE = FMT DATA = FMT2 ;
```

**Missing Lookup Values**

A potential problem with this method is that other products may appear in your input file for which you have not defined a signa value. If this occurs, you will obtain a missing value for the signa value. Or worse yet, if your product codes are numeric, the format will just pass the numeric value into the SIGNA variable. This will lead to corrupted data.

To bypass this problem, you should also use the HLO variable in the FMT data set. If the following code is added after the DATA FMT step, all products which do not have a specified signa will then have a signa of 1.0 applied.

```sas
DATA FMT2 ;
  SET FMT ;
  START = 'OTHER' ;
  LABEL = ' 1.0' ;
  OUTPUT ;
  STOP ;

PROC APPEND BASE = FMT DATA = FMT2 ;
```
Technique #2
Indexed Data sets

When a lookup table has a large number of rows, the PROC FORMAT technique runs into limitations since the formats are stored in memory as opposed to disk. Because of this physical limitation, another technique was needed to combine physician level information with the prescription data. For this task, millions of rows of prescription data was to be combined with information from over 1 million physicians. This number of lookup items led to using indexed data sets.

Indexed data sets allow random access to any record based on the value of a key variable. Since the access is random, the entire indexed data set is not required to be accessed record by record and you only use the limited number of records you really need. Also, since you do not need to use the MERGE statement, the data set which requires the access does not need to be sorted which is an expensive operation. The index creation does require additional computer resources, but these resources are well spent if you do not need to read a data set, sort it, and then merge it.

The physician information was read into a SAS data set with an index on DOC_ID. There are multiple methods you can use to create an index, including a DATA statement option and PROC DATASETS. If you know that an index will be required when creating the data set, use the DATA statement option. If the data set is already created and stored, you should use the PROC DATASETS method.

Creating an Index

The only change one must make to create an index is to add the INDEX option onto the DATA statement. The following code creates an index on DOC_ID at the same time the data set is being created.

```sas
DATA DOCTORS (INDEX=(DOC_ID));
  INFILE INDD;
  INPUT DOC_ID $CHAR5.
  SPEC $CHAR3.
  CALLS 5.
  SAMPLES 5;

If your data set is already created, the index can be created by the following code.

```sas
PROC DATASETS;
  MODIFY DOCTORS;
  INDEX CREATE DOC_ID ;
```

SET Statement KEY Option

To randomly access the doctor records, you need to use the SET statement with the KEY option. The KEY variable must be defined with the value you wish to match on. If a match is found, all data in the record is then available for further processing. The code to match the physician information with the prescription data follows.

```sas
DATA RXS ;
  SET RXS (KEEP = DOC_ID TABS1-TABS24);
  SET DOCTORS KEY = DOC_ID ;

If a match is found for doctor, then all information from the DOCTORS data set is brought into the Program Data Vector (PDV) and is available for processing. If a match is not found, you must program around this to clear the previous matched information from the PDV.

Program Data Vector (PDV)

The PDV must be understood before you should attempt to use the random access through the SET statement. The PDV sets aside space for each variable which is in the input data sets or is created in the DATA STEP. To illustrate the potential problem, let's assume the following data for the DOCTORS data set and use the data originally shown for the RXS data set.

Table 3. Sample data for DOCTORS data set.

<table>
<thead>
<tr>
<th>DOC_ID</th>
<th>SPEC</th>
<th>CALLS</th>
<th>SAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>01234</td>
<td>FP</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>45678</td>
<td>GP</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>67890</td>
<td>IM</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

Notice that DOC_ID 23456 is not included in the DOCTORS data set but is in the RXS data set. As we step through the RXS input data, we will look at what happens to the PDV.

Record #1 before SET DOCTORS KEY = DOC_ID ;

```sas
DOC_ID = 01234
SPEC = ""
CALLS = .
SAMPLES = .
```

This doctor is found in the DOCTORS data set so the PDV values are changed to:

```sas
DOC_ID = 01234
SPEC = 'FP'
CALLS = 5
SAMPLES = 24
```
Record #2 before SET DOCTORS KEY = DOC_ID;

DOC_ID = 23456
SPEC = 'FP'
CALLS = 5
SAMPLES = 24

Notice that the values for variables from the DOCTORS data set have not changed from the previous record although the value of DOC_ID has changed. When the SET statement attempts to access the DOCTORS data set and no match is found, one would expect that the variables from the DOCTORS data set would be set to missing. This is not the case! The variables retain the values from the previous match found. This will lead to errors in the data. You should always check the return code from the SET statement and if the record is not found, you must set all the variables you are receiving from the indexed data set to missing.

Technique #3
PROC SQL

PROC SQL is another technique to merge data sets without using the MERGE statement. To use PROC SQL, the data sets must be sorted by the variable to be merged. Although the code is very simple to perform a straight merge, you do not gain much in processing time or coding since coding a MERGE statement is very simple also. The following two sets of code perform the same function in about the same processing time.

DATA RXS;
  MERGE RXS DOCTORS;
  BY DOC_ID;
PROC SQL;
  CREATE VIEW TEMP AS
  SELECT * FROM RXS, DOCTORS WHERE RXS.DOC_ID = DOCTORS.DOC_ID ;
DATA RXS;
  SET TEMP ;

One particular feature of PROC SQL is very helpful for performing an outer join. An outer join is a merge in which all combinations of data set 1 with data set 2 are created. This feature was handy for the following task. After a redeployment of the sales force, our sales representatives needed to be assigned to the newly defined sales territories. Since many sales representatives live close to one another, some methodology was needed to find all possible assignments for each sales rep.

The two data sets I had for this process both had longitude and latitude variables from which I could pinpoint where the sales representative lives and the center of the sales territory. From these two pieces of information, I could then select only those combinations of sales representative and sales territory where the sales rep lives less than 100 miles from the sales territory. This would have been very difficult to program using normal SAS statements. However, PROC SQL makes this process a very simple programming exercise. If you do not include a WHERE clause in the SQL, then you will create all possible combinations.

PROC SQL:
  CREATE VIEW TEMP
  SELECT * FROM REPS, TERRS ;

This creates a combination of sales reps with all sales territories. Nothing further is required. Next, you use a DATA step to access the VIEW TEMP and then use a subsetting IF to select only those records where the distance between is less than 100 miles.

This functionality alone is worth the effort to get to know PROC SQL.

CONCLUSIONS

I have used and will continue to use all three techniques I have described above. These techniques are very useful especially for processing large data sets where sorting is an expensive or sometimes impossible task. Technique #1, PROC FORMAT is very good when you need to perform a lookup to obtain 1 value from a relatively small data set. Technique #2, Indexed Data sets is an excellent method when you need to combine data from two relatively large sources and when you need multiple data items from a single data set. This method also allows you to access data from more than one lookup source. For example when you need to combine physician and product specific information with prescription level data. Technique #3, PROC SQL can be used for outer joins which the MERGE statement cannot process. It is also a very good method to use for those who have more experience in SQL than in SAS programming. This method will make those people more comfortable using the SAS system and allow them to access all the other valuable tools SAS brings to the table.

For further information, please contact the author at:

John L. Badolato
Walsh America/PMSI
105 Terry Drive
Suite 118
Newtown, PA 18940
References

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA

Katsanis, Gary L., Table Lookups in the SAS Data Step, Proceedings of the Seventeenth Annual SAS Users Group International Conference, 1364-1370

Weiss, Phillip J., Set, Merge, Update or Modify? (And What about Indexing?), Proceedings of the Seventeenth Annual SAS Users Group International Conference, 127-132

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