NO MORE MERGE - *Alternative Table Lookup Techniques*

Written and Prepared by

destiny CORPORATION

ABSTRACT

This tutorial is designed to show you several techniques available for pulling together multiple sets of data for the purpose of combining or comparing their values.

INTRODUCTION

When you are first faced with the concept of pulling together mutliple sets of data, the primary method taught in SAS for many years is merge processing. While this is a very flexible form of combining and comparing data, it is not the only method. There are several other methods you should consider because each of them has their merits. We talk about five additional lookup methodologies in this paper, some of which avoid the need to sort your data. Hint: the first big benefit of reading this paper. These are:

- **Lookups Using Formats**
- **Lookups Using Indexes**
- **Lookups Using Arrays**
- **Lookups Using SQL Joins**
- **Lookups Using Macros**

We use the term "Table Lookup" instead of the word "Merge" to avoid confusion. For the following examples, most of our data has made it into the form of one or two SAS data sets with a key variable. The key variable is used to associate the two forms of data. A typical key would be social security number or employee ID. The examples we use assume two sets of data, EMPLOYEEs and MANAGERs associated together by a department number. The assumption is that only one manager oversees a department of multiple employees. The incoming data is not sorted.

The MANAGER information is:

```
DEPT    MANAGER     TITLE
401     J JONES     PRESIDENT
201     S SMITH     VICE PRESIDENT
301     R OSWALD    ADMINISTRATOR
101     B WILLIE    MANAGER
```

The EMPLOYEE information is:

```
DEPT    EMPLOYEE
401     JOHN DOE
201     SALLY MAY
301     FAT TUESDAY
401     PAULIE SURE
401     SALLY MUSTANG
401     JEFF WILEY
401     AGATHA HARLEY       and so on
```

The final result in most of the following examples will yield a data set similar to the following:

```
DEPT  EMPLOYEE        MANAGER    TITLE
401   JOHN DOE        J JONES    PRESIDENT
201   SALLY MAY       S SMITH    VICE PRESIDENT
301   FAT TUESDAY     R OSWALD   ADMINISTRATOR
401   PAULIE SURE     J JONES    PRESIDENT
401   SALLY MUSTANG   J JONES    PRESIDENT
401   JEFF WILEY      J JONES    PRESIDENT
401   AGATHA HARLEY   J JONES    PRESIDENT
201   JP JONES        S SMITH    VICE PRESIDENT
301   LARRY SMITH     R OSWALD   ADMINISTRATOR
301   STEVE PERRY     R OSWALD   ADMINISTRATOR
301   AL FRANKLIN     R OSWALD   ADMINISTRATOR
301   STEVE SMITH     R OSWALD   ADMINISTRATOR
301   JOHN HOWES      R OSWALD   ADMINISTRATOR
```

TRADITIONAL MERGE PROCESSING

Before we examine alternatives, let's look at a traditional merge. The values to retrieve during a merge operation can be stored in a SAS data set. A key is a unique value found in the data set and is used to determine the observation obtained by the merge. This form of MERGE is called MATCH MERGING.

MERGE statements are written in the data step with a BY statement. This combination of MERGE and BY statements reads and combines observations from two or more data sets based upon their key variable values.

MATCH MERGING uses a special data set option called (IN=variable). This structures a variable in the program data vector. It can be checked for its contribution to the data step. If it has a value of 1, the observation was contributed by that data set; if it has a value of 0, the observation was not.

Syntax

The syntax for data step merging is as follows:

```
DATA sasdataset;
   MERGE sasdataset1(IN=variable1)
         sasdataset2(IN=variable2);
   BY keyvariable(s);
RUN;
```

This assumes the incoming data sets are coming into the step organized by the key variable(s).

Guidelines

- All BY variables must exists in every data set and must be of the same type.
- Exact merging requires variables to not only be the same type, but the same length.

- Data sets referenced on the MERGE statement must be sorted according to the order of the variables in the BY statement. Data sets indexed on these variables is also acceptable.
- Observations are matched when values in the BY variables on an observation are the same.
- Possible selection statements against two data sets with IN= variables called A and B would be:
  - IF A;
  - IF B=1;
  - IF A AND B;
  - IF A OR B;
  - IF NOT(A AND B);

Example

PROGRAM EDITOR

```
proc sort data=saved.managers out=managers;
   by dept;
run;
proc sort data=saved.employee out=employee;
   by dept;
run;
data lookup;
   merge employee(in=a)
         managers(in=b);
   by dept;
   if a and b;
run;
proc print data=lookup;
run;
```

Benefits

- Multiple observations and BY values can be used and retrieved.
- The data set's maximize size is limited to DASD or disk space available.
- It is simple to look up observations when more than one variable is needed to match observations. Simply list those variables in the BY statement.
- Multiple data sets can be used, but each requires ordering according to the BY group variables.

Considerations

- The data sets inbound must be organized according to their BY variables.
- BY variables must match exactly between observations.
- BY variables must be present in every inbound data set.

Lookups Using Formats

The PROC FORMAT statement allows creation of user-defined formats for data. When a table lookup is required, pre-defined formats can be used to retrieve the appropriate data needed. Formats are temporary or permanent and are stored in a Formats catalog in a SAS data library.

Use the PROC FORMAT statement to define:

value formats	create labels for codes using either numeric or character values.
Picture formats	create templates for numeric values like 999-99-9999 for social security numbers.
Informats	control the reading and storing of data in data entry applications.

Syntax

The syntax for value formats is as follows:

```
PROC FORMAT options;
VALUE numfmt
value1 = 'formatted-value-1'
        value2 = 'formatted-value-2'
        valuen = 'formatted-value-n'
        OTHER  = 'formatted-value';
        VALUE numfmt
range1 = 'formatted-value-1'
        range2 = 'formatted-value-2'
        rangen = 'formatted-value-n'
        OTHER  = 'formatted-value';
        VALUE charfmt
'value1' = 'formatted-value-1'
        'value2' = 'formatted-value-2'
        'value3' = 'formatted-value-n'
        OTHER    = 'formatted-value';
RUN;
```

Guidelines

- Format names may be up to 8 characters long which includes using the $ as part of the name.
- Place character values in quotes for formatting. Numeric values do not require quotes but allow for ranges.
- Missing values can be formatted as well.
- The resulting label can be mixed case, but the value to be formatted, when character, is case sensitive.
- Character formats are designed for labeling character data.
- Numeric formats are designed for labeling numbers and ranges.
- Canned SAS format names are reserved. User defined format names must be different.
- All character formats must begin with a $ as part of the name.
- User-defined formats cannot end in a number, primarily because SAS looks at a number as the width of the formatted value.
- All formatted values result in quoted labels.
- A value is used to specify a user defined format.

- Character values to format can be up to 200 characters long.
- Values must be unique. Formatted values are sorted in ascending order.

The PUT function is typically used to test on the returned, formatted value of a variable. The syntax is:

```
PUT(argument,format.)
```

The following two data sets are using in all of the lookup examples.

Example

PROGRAM EDITOR

```
data manname;
   set saved.managers;
   rename dept   =start
          manager=label;
   fmtname = '$manname';
run;
proc format cntlin=manname;
run;
data mantitle;
   set saved.managers;
   rename dept   =start
          title  =label;
   fmtname = '$mantitl';
run;
proc format cntlin=mantitle;
run;
data lookup;
   set saved.employee;
   manager = put(dept,$manname.);
   title   = put(dept,$mantitl.);
run;
proc print data=lookup;
run;
```

Benefits

- After formats exist, they can be used in data steps or in procs.
- Values needing formatted labels can be in the form of a list, discrete, or a range.
- Since formatting values can change often, consider storing the values in a data set, instead of hard coding them. Use CNTLOUT= and CNTLIN= which specify data set information.
- Formatted look ups use a binary search, by default.
- Data step processing that uses the PUT function with a format is much more efficient in CPU utilization than a MERGE statement.

Considerations

- The entire format is loaded into memory, therefore the size may be limited by this resource.
- Only one value is retrieved when a format is matched.
- Only one value may be used to look up data at a time.
- A variable can take on only one format a time.

Lookups Using Indexes

Indexes may also be used to retrieve data from tables. Indexes are most often used when the table is too large to hold in memory, only a few values need to be retrieved, or a SET or MODIFY statement contains the KEY= option.

You may use indexes when performing an SQL join, using a BY statement variable list starting with a simple or composite primary key, or using a WHERE statement referencing a simple or composite primary key.

Syntax

The syntax for indexes usage is as follows:

```
SET sasdataset KEY=indexname;
SET sasdataset;
   BY indexname or variablesinindex;
SET sasdataset;
   WHERE indexname or variablesinindex;
```

Guidelines

- Indexes are relatively inefficient on small data sets.
- Create an index based on a variable with a large number of distinct values.
- Indexes should be used to retrieve a small subset of the data set.
- Keep the number of indexes small so disk storage and update costs are minimal.
- Indexes must conform to the assumptions concerning value distributions.
- Sort your data set in order of the most frequently used index.
- Use the MODIFY statement when the master data set is indexed based on the variables used for matching transactions.

Example

PROGRAM EDITOR

```
data managers(index=(dept=(dept)));
   set saved.managers;
run;
data lookup;
   set saved.employee;
   set managers key=dept;
run;
proc print data=lookup;
run;
```

Benefits

- Only the observations needed are read from the lookup data set.
- Multiple values are retrieved as a result of the lookup operation.
- The appropriate master observation is directly accessed.
- No additional disk space is required because updates are done in place with the MODIFY statement.

Considerations

- Increased resources required to store and maintain the index.

Lookups Using Arrays

Arrays can also be used for table lookups. Arrays are often used when the data to be retrieved can be identified positionally, e.g. the 1st item, 2nd item, etc..., or when the table value to be retrieved is identified by one or more numeric values.

The ARRAY(table) can be made up of values which are either hard coded in the data step or are stored into a data set or external file and then loaded into array variables via a set or input statement.

Use the ARRAY statement in the data step to define a set of variables to be processed in a similar manner.

Syntax

The syntax for ARRAY statement is as follows:

```
ARRAY arrayname{dimension} $ length elements
                                    initialvalues;
```

A temporary ARRAY is usually defined to list al potential values to compare data with. The ARRAY option used is _temporary_.

Guidelines

- Variables are created if they do not already exist in the program data vector.
- An ARRAY must be defined before the ARRAY name can be referenced.
- The ARRAY is designed to be of one type of variable, either all character elements or all numeric elements.
- You cannot execute an ARRAY statement.
- You cannot refer to ARRAY statements in other compile time statements.
- ARRAY statements do not become part of the output data set. They exist only while the data step is processing.
- A SET statement can be used to load values into ARRAY elements.

Example

PROGRAM EDITOR

```
data lookup(drop=i);
   array namearay{101:401,4} $ 10
   _temporary_;
   array titlaray{101:401,4} $ 15
   _temporary_;
   if _n_ = 1 then do i = 1 to manobs;
     set saved.managers nobs=manobs;
     namearay{input(dept,3.),i}=manager;
     titlaray{input(dept,3.),i}=title;
   end;
   set saved.employee;
   do i = 1 to manobs;
     manager=
     namearay{input(dept,3.),i};
     title  =
     titlaray{input(dept,3.),i};
     if manager ne ' ' then output;
   end;
run;
proc print;
run;
```

Benefits

- The exact position of values in the PDV can be used.
- Many values can be used to determine the ARRAY element number 's value to be retrieved.
- Calculations can be used to figure which element of the ARRAY is needed.
- The inbound data set does not have to be organized in any way.

Considerations

- ARRAYs are loaded into memory.
- ARRAY elements are pointed to with numbers. Character information cannot be used.
- The lookup operation results in only one value being retrieved.

Lookups Using SQL Joins

The PROC SQL statement does not require all data sets to have common variables in order to join them. Multiple tables may be joined to create a new data set. In many cases, data retrieval problems can be eliminated by using a join, subquery, or both.

Subqueries are used when more than one query is needed to achieve the desired results. Each subquery provides a subset of the table used in the query. While joins and subqueries are used in queries, a join is usually the most efficient process.

PROC SQL can also be used to create SQL views and SAS data sets.

Syntax

The syntax used for SQL joins is as follows:

```
PROC SQL;
   CREATE TABLE tablename as
   SELECT sasvariables
   FROM sasdataset1,sasdataset2
   WHERE sasdataset1variable(s)=
sasdataset2variable(s);
QUIT;
```

Guidelines

- Joins can be performed in many ways. An understanding of the different forms of joins available in SAS demonstrates the capability of an Inner Join, Outer Join, Full Join, Left Join, and a Right Join.
- The default Join as in the syntax above is an Inner Join.
- SQL Joins are table lookups that require exact matches across observations
- SQL can also be used to summarize data during the process
- SQL can also be used to return data in a sorted order during the process
- SQL can also be used to return statistics and subsetted data before or after the join process.
- Indexes, if available, are used appropriately.

Example

PROGRAM EDITOR

```
proc sql;
   create table lookup as
   select
   a.dept,a.employee,b.manager,b.title
   from saved.employee as a,
   saved.managers as b
   where a.dept=b.dept;
quit;
proc print data=lookup;
run;
```

Benefits

- Data sets need not be ordered any way when being read.
- SAS data sets, views, or hardcopy reports can be created.
- Many data sets can be pulled together without having the same variables in common in all data sets.

Considerations

- Only sixteen tables can be joined at one time.
- SQL joins require more resources than using MERGE statements.

Lookups Using Macros

Macros may also be used to call data from a table. Values can be placed into macro variables using a CALL SYMPUT function or a %LET statement. These macro variable values containing table information can then be called using a SYMGET function. The SYMGET function relates program data vector key variables to macro variables.

Syntax

The syntax used for Macro lookups is as follows:

```
DATA _NULL_;
   set or input section;
   CALL SYMPUT('characterprefix'!!
        keyvariable,TRIM(valuevariable(s));
RUN;
DATA RESULT;
   set or input section;
newvariable=SYMGET('characterprefix'!!valuevariable
(s));
RUN;
```

Guidelines

- Macro lookups require a key to assign and retrieve information.
- Macro variables load in memory.
- Consider using this method when there are a few different values across a large file and not many unique values matching with many unique values.

Example

PROGRAM EDITOR

```
data _null_;
   set saved.managers;
   call
   symput('dep'!!dept,manager!!title);
run;
data lookup;
   set saved.employee;
   manager=
   substr(symget('dep'!!dept),1,20);
   title  =
  substr(symget('dep'!!dept),21,20);
run;
proc print data=lookup;
run;
```

Benefits

- No sorting of data sets is needed prior to the lookup.
- Save processing time
- Match against single key variables or multiple key variables
- Retrieve single variable data values or multiple variable data values
- Lookup macro variables do not take up disk space.

Considerations

- Requires an exact match.
- Requires a good understanding of SAS macros, programming with functions and strings.
- Since macro variables load in memory, many of them may use up available memory. Consider increasing memory for macro variables with the...

OPTIONS MSYMTABMAX=value;

CONCLUSION

As we have seen, there are many additional methodologies available to pull different sets of data together. One form is not better than another. The goal of this paper was to make you aware of what they are. These are additional tools to be added to your SAS tool set for future programming endeavors.

ACKNOWLEDGEMENTS

Some topics compiled from:

Advanced Techniques and Efficiences, First Edition, Copyright 1995 by Destiny Corporation.

Advanced SAS Programming Techniques and Efficiencies, Copyright 1992 by SAS Institute Inc.

Author: Dana Rafiee, Trainer/Consultant

Destiny Corporation/DDISC Group Ltd. U.S.

1321 Silas Deane Highway #A

Wethersfield, CT 06109-4302 USA

Phone: 1-800-7TRAINING

1-860-721-1684

Fax: 1-860-721-9784