Table Lookup in SAS

Weili Yang, Fang Chen, Liping Zhang, Wenyu Hu
Merck Research Labs, Merck & Co., Inc., Upper Gwynedd, PA

ABSTRACT

Table lookup or searching is a common task performed in SAS. Base SAS® Version 8, as well as earlier versions, offer many lookup methods such as SET with KEY=, arrays, SQL joins, formats, and MERGE with a BY statement. SAS Version 9 introduced the DATA Step hash object, which provides additional ways to perform table lookups without the need for sorting or indexing. This paper analyzes the programming details of the various table lookup methods and compares the advantages and disadvantages of each approach. The comparisons shown in this paper will aid programmers in selecting the optimal table lookup method to use, as dictated by their specific situation.

Keywords: Table Lookup, Set with KEY=, Array, Merge, SQL, Hash, Formats

INTRODUCTION

Searching for a value in one dataset when given a value from another dataset is one of the most frequent data processing operations. This value is often referred to as a "key". For example, given subject identification, one could retrieve the patient's demographics or other treatment related information from the dataset.

Examples of commonly used table lookup methods are:

- Join with PROC SQL
- MERGE with BY statement
- SET statement with KEY= option
- Positional Lookup with _Temporary_ Arrays
- Table Lookup using Formats
- Lookup with Hash Object

Some of these techniques use lookup values stored on disk, while others are designed to provide an in-memory lookup. In order to identify the best lookup mechanism, the programming details of each technique will be examined.

JOIN WITH PROC SQL

Using SQL, multiple datasets can be joined without having common variables across all datasets. Datasets do not have to be sorted or indexed.

Suppose a lookup dataset SV contains subject identification (*usubjid*), visit number (*visitnum*), visit date (*svstdtc*) and planned study day of visit (*visitdy*). The master dataset QS has subject identification (*usubjid*), visit number (*visitnum*) and the questions the subject answered at each visit. Example 1 demonstrates how to use SQL to merge *svstdtc* and *visitdy* into dataset QS based on *usubjid* and *visitnum*.

<u>Example 1:</u>

```
proc sql;
      create table qs_sql
      as select qs.usubjid, qs.visitnum, sv.svstdtc as qsvisdtc,
      sv.visitdy as qsvisdy, qs.qstestcd, qs.qstest,
      qs.qsorres, qs.qsstresc, qs.qsstresn
      from sdtm.qs qs left join sdtm.sv sv
      on qs.usubjid=sv.usubjid and
      qs.visitnum=sv.visitnum
      ;
quit;
```

In Example 1 above, the SQL code left join allows every record in the QS dataset to be included in the final dataset even if no matching *usubjid* and *visitnum* is found in SV.

MERGE WITH BY STATEMENT

Match-merging is a commonly used technique to combine observations from two or more datasets when the values of the BY variables are the same and requires the input datasets to be sorted or have an appropriate index. Example 2 demonstrates how to use this technique to achieve the same results as in Example 1.

Example 2:

```
proc sort data=sdtm.sv out=sv;
      by usubjid visitnum;
run;

proc sort data=sdtm.qs out=qs;
      by usubjid visitnum;
run;

data qs_merge;
      merge qs(in=a) sv;
      by usubjid visitnum;
      if a;
      qsvisdtc=svstdtc;
      qsvisdy=visitdy;
run;
```

The IN= dataset option is used here to include all observations from QS dataset.

SET STATEMENT WITH KEY = OPTION

Lookup operations can be performed using the KEY = option with one or more SET statements if the lookup dataset is indexed by unique key variables. No sorting or indexing is required on the master dataset. The composite key can include a combination of variables with either numeric or character types and multiple values can be returned. Example 3 shows another way to achieve the same result as the previous examples.

Example 3:

```
proc sql;
      create index patvis
            on sdtm.sv (usubjid, visitnum);
quit;

data qs;
      length qsvisdtc $19;
      set sdtm.qs;
      set sdtm.sv(keep=usubjid visitnum svstdtc visitdy)
      key=patvis/unique;
      if _iorc_=0 then do;
```

```
            qsvisdtc=svstdtc;
            qsvisdy=visitdy;
      end;
      else do;
            _error_=0;
            qsvisdtc='';
            qsvisdy=.;
      end;
run;
```

First, PROC SQL is used to create index *patvis* consisting of *usubjid* and *visitnum*. Second the index created is specified in the KEY= option in the SET statement of the lookup dataset. Note that only the lookup dataset needs to be indexed. Also note that the order of the datasets specified is important. The master dataset must be specified first, and the lookup dataset specified second. In order to match each of the duplicate patient visit records on the dataset QS with its corresponding matching record on the SV lookup dataset, the UNIQUE option must be specified. The UNIQUE option forces SAS to begin at the top of the lookup table each time it performs a lookup operation. If _IORC_ returns a value of 0, it indicates that SAS found a matching observation. To prevent a data error from being written to the log in the event that a match is not found, _error_ is set to 0.

POSITIONAL LOOKUP WITH _TEMPORARY_ ARRAY

Arrays can be used to perform a table lookup when the values to be returned are positionally identified and the key is a numeric value.

For example, a demography dataset, DEMO (shown in Figure 1), has the following variables: *subjid, weight, height, age* and *type*. Patients are assigned to types 1-4 based on their age and height. The IDEALWGT dataset (shown in Figure 2) is the lookup table which contains the ideal weight and height for types 1-4. The heights listed in the IDEALWGT dataset are even numbers only. For patients with an odd-numbered height, the next highest level of height is used to get the ideal weight.

Figure 1:

```
                                   Partial DEMO Dataset

OBS   SUBJID       AGE    HEIGHT    WEIGHT    TYPE

1    000100001     63      163      61.0        4
2    000100002     31      176      67.7        1
3    000100003     51      178      80.0        3
4    000100004     49      165      74.3        3
5    000100005     22      178      80.1        1
6    000100007     40      167      67.0        2
7    000100008     47      172      62.0        3
8    000100009     28      160      72.9        1
```

Figure 2:

```
                                 Partial IDEALWGT Dataset

OBS    TYPE1    TYPE2    TYPE3    TYPE4    HEIGHT
1      60       62       64       66       160
2      62       64       69       66       162
3      64       66       68       72       164
4      66       68       72       76       166
5      70       71       76       79       168
6      72       73       78       82       170
7      74       75       79       83       172
```

8	74	76	81	84	174

Example 4:

```
data wgt_array(keep=subjid weight height age sex type wgtdiff);

      array wgt{160:190, 4} _temporary_;
      if _n_=1 then do i=160 to 190 by 2;
            set idealwgt;
            array tmp{4} type1-type4;
            do j=1 to 4;
                  wgt{height, j}=tmp(j);
            end;
      end;

      set demo;
      if mod(height, 2) =1 then hgt_typ=height+1;
      else                      hgt_typ=height;
      if nmiss(wgt{hgt_typ, type}, weight)=0 then
            wgtdiff=wgt{hgt_typ, type}-weight;
run;
```

In Example 4, a temporary two-dimensional array is created to hold the suggested weight table. _N_=1 is used such that the array is only loaded the first time through the DATA step. The index variable i is used so that the SET statement is executed for each observation in the IDEALWGT dataset. The SET statement on the primary dataset DEMO computes the difference between patients' weight and ideal weight.

TABLE LOOKUP USING FORMATS

The FORMAT procedure can be used to define tables that store coded values and the corresponding definitions of the codes. These user-defined formats can be referenced when a table lookup operation is needed. Formats can be created by listing the value pairs or by generating them from an existing dataset.

In the following example, the DM dataset has subject identification (*usubjid*), age (*age*), sex (*sex*), treatment (*trtp*) and treatment start date (*rfstdtc*). It is of interest to get some subject-related information into dataset QS in order to perform some efficacy analyses.

Example 5:

```
data subj;
      keep start label fmtname type hlo;
      retain fmtname '$subj' type 'c';
      set sdtm.dm (rename=(usubjid=start)) end=last;
      label=put(sex, $char2.) || put(age, 3.)
            || put(trtp, $char50.) || put(rfstdtc, char19.);
      output;
      if last then do;
            hlo='O';
            label='';
            output;
      end;
run;

proc format cntlin=subj;
run;
```

```
data qs_format;
      set qs;
      sex=substr(put(usubjid, $subj.),1, 2);
      age=input(substr(put(usubjid, $subj.),3, 3),3.);
      trtp=substr(put(usubjid, $subj.),6, 50);
      rfstdtc=substr(put(usubjid, $subj.), 56, 19);
run;
```

In Example 5, a format is created using the lookup dataset DM. Note that hlo='O' is used to allow the capture of all non-matching *usubjid*'s from the QS. The CNTLIN=option is used to read the data and create the format. With formats, only one variable can be used for table lookup and only one value is returned. Thus, multiple PUT functions are required to construct the return value, and the value is later parsed to get subjects' demographic information.

LOOKUP WITH HASH OBJECT

When using DATA step hash object, neither dataset is required to be sorted or indexed. The key may be composite and may simultaneously consist of both numeric and character values and multiple data items can be stored per key.

The following example demonstrates the use of a hash object for table lookup using the same SV and QS datasets in the previous Example 1-3.

Example 6:

```
data qs_hash;
      length qsvisdtc $19;
      if _n_=1 then do;
            if 0 then
                  set sdtm.sv(keep=usubjid visitnum svstdtc visitdy);
            declare hash hh(dataset: "sdtm.sv");
            hh.definekey('usubjid', 'visitnum');
            hh.definedata('svstdtc', 'visitdy');
            hh.definedone();
      end;
      do until (eof2);
            set sdtm.qs end=eof2;
            rc=hh.find(key:usubjid, key:visitnum);
            if rc=0 then do;
                  qsvisdtc=svstdtc;
                  qsvisdy=visitdy;
            end;
            else do;
                  qsvisdtc='';
                  qsvisdy=.;
            end;
            output;
      end;
      stop;
run;
```

The above example first initializes the attributes of hash variables originating from an existing dataset by using a non-executing SET statement. Next, the composite key consisting of *usubjid* and *visitnum* is defined using DEFINEKEY. As a final step, the data elements *svstdtc* and *visitdy* are defined using DEFINEDATA. Multiple variables can be defined to be either character or numeric. The DEFINEDONE method must be called to complete the initialization of the hash object. The Find method is used to search for the composite key value of *usubjid* and *visitnum* in the dataset QS. If the search is successful, the return code rc is set to 0 and multiple numeric and/or character results can be returned.

Comparison of Lookup Methods

The table below summarizes the pros and cons of the various lookup methods discussed above.

Method	Pros	Cons
PROC SQL	Multiple datasets can be joined without having common variables in all datasets Datasets do not have to be indexed or sorted	Requires more resources than the DATA step with the MERGE statement for simple joins. Complex business logic is harder to incorporate into the joins
MERGE With BY	Multiple values can be returned Fast, sequential access of input datasets No limit to the size of the table, other than disk space	Datasets must be sorted or indexed based on the BY variables Must have exact match on the key values
Set with KEY=	Only records where a match occurs are read from the look-up file, thus processing time is reduced Multiple values can be returned Composite key could be used to look up data	An index on the lookup dataset is required Creating and maintaining an index uses resources
TEMPORARY Arrays	Use of multiple values to determine the array element to be returned Ability to use a non-sorted and non-indexed base dataset Fastest lookup method by using the key to access the data directly Does not require exact match to lookup the data	Memory requirements to load the entire array Return of only one single value from the lookup operation A contiguous chunk of memory is requested when the array is declared
FORMATS	Fast, binary search through lookup table	Memory requirements to load the entire format for the binary search Use of only one variable for the table lookup and only one value is returned Requires more memory than a hash object
Hash Objects	No sorting or index is required Key and data can be composed of multiple values of both character and numeric type Dynamically grow to fit as many records that fit into memory Provides in-memory data storage and retrieval Use the key for quick data retrieval	Hash object records must fit into memory _TEMPORARY_ array lookup faster than using a hash object

SUMMARY

This paper provides a variety of techniques for managing various table lookup problems. While there are many solutions to one lookup operation, one method may be more efficient than another depending on the size of the lookup table. Generally the larger the lookup dataset is in relation to the master table, the more the problem appears to be a match merge. Hash object is best used when the dataset for the hash table is relatively small when compared to the master dataset and it fits in memory. However, an array will be faster than a hash object or format if the Cons listed above can be accommodated. Depending on the size of the lookup table, many different approaches are possible.

REFERENCES

SAS Programming III: Advanced techniques and efficiencies

Jason Secosky, Janice Bloom (2006) Getting Started with the DATA Step Hash Object

Rob Rohrbough (2007) Table Lookups…You Want Performance?

TRADEMARKS

CONTACT INFORMATION

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

Weili Yang	Fang Chen	Liping Zhang	Wenyu Hu
UG 1D-88	UG 1D-88	UG 1CD-44	UG 1D-88
Merck Research Lab	Merck Research Lab	Merck Research Labs	Merck Research Labs
Merck Co., & Inc.	Merck Co., & Inc.	Merck Co., & Inc	Merck Co., & Inc
Upper Gwynedd, PA	Upper Gwynedd, PA	Upper Gwynedd, PA	Upper Gwynedd, PA
19454	19454	19454	19454
(267) 305-5383	(267) 305-6812	(267) 305-7980	(267) 305-6847
Weili_yang@merck.com	fang_chen1@merck.com	liping_zhang@merck.com	wenyu_hu@merck.com