Table Lookups in the SAS® Data Step

Gary L. Katsanis, Blue Cross and Blue Shield of the Rochester Area, Rochester, NY

Introduction - What is a Table Lookup?

You have a sales file with one observation for each order and an address file, with one observation for each customer. To mail orders, you want to include address in the sales file. You sort both files, then merge them by customer, and keep all observations with information from the sales file, as shown in Example 1.

You have just performed a *table lookup*. Table lookup describes the process when you want to add information from some other location to every observation in a data set. You add the information according to some rule, usually based on the value of one or more variables in your data set.

Example 1: Match Merge

```
PROC SORT DATA=ORDER ;
   BY CUSTID ;
   RUN ;
PROC SORT DATA=ADDRESS ;
   BY CUSTID ;
   RUN ;
DATA BILLS ;
   MERGE ORDER(IN=KEEP)
      ADDRESS ;
   BY CUSTID ;
   IF KEEP ;
   RUN ;
```

The information you add needs to be stored in some form that the SAS System can use, such as a data set, format, or source code.

Table lookup is one of the most common operations you will perform when working with data stored in more than one file. Table lookups are so common for two reasons: first, your data may be coming to you from two separate sources; second, you might store data in separate data sets to avoid wasting space. In our example above, the sales file might have a lot of observations and few variables, so adding customer address could make the file many times larger.

Purpose of This Paper

Table lookup has received a lot of attention at SUGI in the past; (1), (2), (3) and (4) are only a few of the presentations on the subject. There hasn't been much change in techniques either, with the exception of the KEY= option, which I describe below. Craig Ray, Leonard Lutomski, and others have done an exceptional job of describing techniques and efficiencies, and I have very little to add.

However, everyone who uses the SAS System, or any data base manager, uses table lookup techniques eventually. The purpose of this paper is to help these SAS users recognize when they need to perform a table lookup, as well as to provide enough information on the common techniques that the user has a chance of picking an effective technique. Table lookup is not arcane, and every SAS user can benefit from knowing the basic techniques.

Table Lookup Terms

Table lookup is easier to discuss if we have a set of common terms. With minor modification, I'm using the terms presented in SAS Language and Procedures, Usage 2.[1]

Primary File - the data set to which you want to add information

Lookup Source - the location holding the information you want to add to the primary file; if this is a data set, it's called a *lookup file*

Key Variables - the variables that tell you which data from the lookup source you want to add to a particular observation in the primary file

[1] Chapter 10 of the SAS Language and Procedures, Usage 2 presents a thorough discussion of table lookup. It covers some techniques not presented here and is a very useful reference.

Lookup Result - the information, specific to one or more observations in the primary file, from the lookup source that you want to add to the primary file

Seek Operation - the process by which you find the lookup result based on the information in the primary file

In Example 1, the *primary file* is ORDER, the *lookup source* is ADDRESS, the *key variable* is CUSTID, the *lookup result* is the customer's address, and the *seek operation* is the match-merge performed based on the MERGE and BY statements.

Cautionary Notes

Make sure key variable values are unique in the lookup file.

Most table lookups are based on having only one answer to the seek operation. That is, there is only one observation in a lookup file for each value of the key variable, or only one appearance of each value of the key variable in your lookup source, if the source is not a data set. I will assume this in my discussion.

You should make sure that your table lookups follow this guideline, or you should program very carefully to work around it. If you have more than one observation with the same value of the key variable and different values for the lookup result, you may face some unpleasant results. In a match merge situation, observations from the primary file may be duplicated, which can corrupt totals; moreover, the value of the lookup result could be different from observation to observation. In a binary search or an index search, you will not duplicate observations in the primary file, but you will not know which lookup result will appear in the data set you create.

Make sure that the only variables that appear in both the primary and the lookup file are the key variables.

If the same variable appears in more than one data set, you will get one or the other in the data set you create, but you will not get both! In a match merge, you will get the one from the data set listed last, and in a binary search or an index search, you will get the one from the lookup file. You should either rename or drop the variables from one of the incoming data sets[2].

What Techniques Can You Use?

One of the first table lookup techniques that people master is the MERGE as shown in Example 1. However, there are a whole series of techniques you can use in the DATA step to perform table lookups. I will give examples of several other techniques, then discuss when you would want to use them.

Using Formats as a Lookup Source

Let's say you were working with the sales file above, but rather than needing a customer's address, you need just the customer's name. You could create a format from the customer file, which translates the customer's ID number to their name, then you can call that format in a DATA step or a PROC step to use the customer's name, as shown in Example 2.

Example 2: Creating and Using a Format

```
*  create the table lookup
   format, using the CNTLIN
   option of PROC FORMAT ;
DATA CNAMEFMT ;
   RETAIN FMTNAME "$TONAME" ;
   SET ADDRESS
   (KEEP=CUSTID NAME
    RENAME=(CUSTID=START
            NAME=LABEL));
   RUN ;
PROC FORMAT CNTLIN=CNAMEFMT ;
   RUN ;

DATA BILLS ;
   SET ORDER ;
   NAME = PUT(CUSTID,$TONAME.) ;
   RUN ;
```

In Example 2, I am using the CNTLIN option for the FORMAT procedure. This option lets you create a format directly from the contents of a SAS data set. You need to name the variables in your data set appropriately to use this feature. For more

[2] You will find a discussion of this in Chapter 4 of the SAS Language Guide (6).

information on creating formats, see (7,275-312). This process is much more efficient if you make $TONAME a permanent format. For more examples of formats used in table lookup, see (5,197-204).

Using Binary Search of a Lookup File

If your lookup source is a SAS data set, you can use a binary search for table lookup. In the DATA step, binary search requires you to use the POINT= option on the SET statement(6,485). POINT= instructs the SET statement to read observations from a file based on the sequence number of the observation.

In a binary search, you look for the value of the key variable in the middle of ever-smaller ranges of observations, until you find it.

1. The first range is the whole file.
2. By looking at the middle observation in the range, you either (a) find the value you need, or (b) find whether the value you need is larger than the middle or smaller that the middle.
3. If the value you need is larger than the value at the middle, your new range is the half of the old range with the larger values of the key variable.
4. If the value you need is smaller than the value in the middle, your new range is the half of the old range with the smaller values of the key variable.
5. You go back to step 2 until you find the value you were looking for.

A program that uses binary search appears in Example 3 at the end of this paper. Binary search is much more efficient if your permanent copy of the lookup file is sorted by the key variable (2,649). While you don't need to sort the primary file, my example runs much more efficiently if you do sort it by the key variable as well.

Using an Indexed Search of a Lookup File

An alternative to using binary search for table lookup is to create and use an index based on the key variable. This is much easier to program than a binary search, because the seek operation is built in and you just call it by using the KEY= option on the SET statement(8,43). To build an index, you can use the DATASETS procedure, as in Example 4 (7,264-265), or you can use the INDEX= data set option (8,82-83).

Example 4: Creating an Index

```
PROC DATASETS LIBRARY=BILLING ;
   MODIFY ADDRESS ;
   INDEX CREATE CUSTID ;
   RUN ;
   QUIT ;
```

To use the index for table lookup in a DATA step, you use two SET statements, the first for your primary file, and the second SET statement with the KEY= option for your lookup file. The first SET statement assigns a value to your key variable, and the second uses the value of the key variable to find the observation with that value from the lookup file, as in Example 5.

Example 5: Using an Index

```
DATA BILLS ;
   SET ORDER ;
   SET ADDRESS KEY=CUSTID ;
   RUN ;
```

There is no need to sort either file, although you can refine this technique by sorting the primary file and performing a seek operation only for the first observation for each value of the key variable.

Using Arrays as a Lookup Source

You could also use arrays as a lookup source. To do this, you need to find a way to identify the lookup result based on the key variable. The easiest way is to set up your array so that you get the lookup result when you use the value of the key variable as the array subscript. In Example 6, I'm assuming that there are 50 customers, and that CUSTID varies between 1 and 50.

This method is critically dependent on developing a way to perform the seek operation. In my example, I'm assuming that the key variable has acceptable values for array subscripts. If this is not the case, you will need to either (a) create a format that maps the key variable to the array subscript you need, or

Example 6: Using an Array

```
DATA BILLS ;
   LENGTH NAME1-NAME50 $30 ;
   ARRAY NAMES{50} NAME1-NAME50;
   IF _N_ EQ 1 THEN
      DO I = 1 TO 50 ;
      SET ADDRESS(KEEP=NAME) ;
      NAMES{I} = NAME ;
   END ;
   SET ORDER ;
   NAME = NAMES{CUSTID} ;
   RUN ;
```

(b) store the key variable in an array and use a sequential or binary search on it. See (5,204-210) for an example using a format.

Other Techniques

There are other table lookup techniques. Blocks of hard-coded information, such as:

```
WHEN(1) MONTH="January";
WHEN(2) MONTH="February";
 . . etc . .
```

are actually a form of table lookup. For small tables that change very infrequently, this is not necessarily a bad method. There are also more specialized techniques, like hashing (3), the Golden Section Search (2), and use of macro variables (4). Each has its benefits but is less widely used than the techniques I presented above.

Which Technique Should You Use?

If you haven't already spent some time looking into table lookup, you might be asking, "What's the matter with match merge? It works." The short answer is that there's nothing wrong with it. However, it might not be the fastest technique for your purpose.

Recently, I evaluated techniques my department uses to find names in a lookup file with 500,000 observations. Often, our primary file only had 100-200 observations. The technique we used was match merge. I found that the most efficient technique is indexing the file and using KEY= for our table lookup. Using an index saves 98.5% of the CPU time used by match merge.

Often efficiency is not an issue, when you write a program that will be used once. When you develop applications for frequent use, you need to pay more attention to efficiency, and selecting the right table lookup can be a big part of it. Leonard Lutomski (2) did extensive testing and benchmarking, and his findings may be useful when you choose a technique for table lookup.

You may want to choose by testing each approach in your application and seeing which one performs best. I usually narrow my choices by excluding techniques that don't work well in my circumstances. If I have a critical application and more than one technique that can work well, I test them all. Otherwise, I just pick the easiest to use.

In Table 1, I've listed some issues that you should consider when you need to perform a table lookup. You need to look at your application and identify the issues, such as:

- Do I have more than one key variable?
- Does my lookup result contain more than one variable?
- Are my primary and lookup files already sorted by the key variable?
- Will I use most of my lookup file? Only a small part of my lookup file?

In my example with the 500,000 observation name data set above, I identified the following issues:

1. We had one key variable and one variable in the lookup result.
2. We used a very small part of the lookup file at any time.
3. We could rearrange observations in the lookup file any way we wanted.
4. We had space for an index.
5. The primary file was not always sorted.

The best use of match merge is when both files are already sorted by the key variable and you will use all of the lookup file, especially if you have more than one key variable or more than one variable in your lookup result. That is clearly not the case here.

Of the other techniques, I considered using a format first. Since there is only one key variable and one variable in the lookup result, we did not have two of the major limitations on using formats. However, the number of observations in our lookup file was well beyond the practical limits of using a format.

Since arrays cannot handle large lookup sources, I considered binary search and using an index. Both are effective with small primary files and large lookup files. Binary search requires significant programming, and an index requires significant space. Since we would use the table lookup frequently in many programs and since we had the space to create an index, using an index and SET with the KEY= option for table lookups is our best choice.

Conclusion

Table lookup is a technique that SAS users apply very frequently to their work without always consciously recognizing what they are doing as table lookup. By recognizing when you need a table lookup and learning about table lookup techniques, you can quickly choose among the techniques to develop efficient applications. If you develop an application where efficiency is critical, you can narrow down your choices, then test each table lookup technique for optimal performance in your circumstances.

References

1 Ray, Craig, "A Comparison of Table Lookup Techniques," SAS Institute, Inc., *Proceedings of the Twelfth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc., 1987, pp 40-50.

2 Lutomski, Leonard S., "The Comparative Efficiency of Table Lookup Methods," SAS Institute, Inc., *Proceedings of the Thirteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc., 1988, pp 648-653.

3 Ray, Craig, "Implementation of a Hashing Routine in SAS Software," SAS Institute, Inc., *Proceedings of the Thirteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc., 1988, pp 1184-1190.

4 Ray, Craig and Howard Levine, "Efficient Use of Table Lookup Procedures," SAS Institute, Inc., *Proceedings of the Eleventh Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc., 1986, pp 718-723.

5 SAS Institute, Inc., *SAS Language and Procedures: Usage 2, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1991, 649 pp..

6 SAS Institute, Inc., *SAS Language Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1990, 1042 pp..

7 SAS Institute, Inc., *SAS Guide, Version 6, Third Edition*, Cary, NC: SAS Institute, Inc., 1990, 705 pp..

8 SAS Institute, Inc., *SAS Technical Report P222, Changes and Enhancements to Base SAS Software, Release 6.07*, Cary, NC: SAS Institute, Inc., 1991, pp 20, 82-83.

Table 1: Desirable and Undesirable Characteristics of Table Lookup Techniques

Desirable Characteristics	Match Merge	Format	Binary Search	Index Search	Array
Is it simple to program?	yes	yes	no	yes	no
Can it be added easily to an existing DATA step?	if files are already sorted	yes	no	yes	no
Is it easy to use lookup results with more than one variable?	yes	no	yes	yes	no
Is it easy to use multiple key variables?	yes	no	no	yes	no
Undesirable Characteristics					
Does it need a sorted primary file?	yes	no	no	no	no
Does it need a sorted lookup file?	yes	no	yes	no, but needs index	no
Is it slow if you use a small part of a large lookup file?	yes	no	no	no	yes
Is it slow if you use most or all of a large lookup file?	no	no	yes	yes	depends on logic of seek operation
Is there a practical limit on the size of the lookup source?	no	yes	no	no	yes

Other Comments

All Techniques: Sorting the primary file and performing the seek only when the value of the key variable changes can improve performance.

Formats: You can store the format in a permanent library to avoid overhead. If you have ranges of values for the key variable that should give the same lookup result, you can specify the range and save space.

Index: You can build more than one index for a single lookup file. Indexes take significant resources to create, and significant space to store.

Array: If your key variable is not a valid array subscript, you need to build an efficient seek operation, or the array technique will not give you good results; see above.

Example 3 - Binary Search

```
DATA BILLS ;

   * ORDER is the primary file and does not have to be sorted, but this
     is more efficient if it is sorted by CUSTID ;
   SET ORDER;

   * START and END are the numbers of the first and last observation in
     the range you're looking at, while POINTER points to the middle of
     the range ;
   START = 0 ;
   END = CNUM ;
   POINTER=INT((END+START)/2) ;

   * if your last seek operation found the same key variable, the seek
     loop will not run at all, and the lookup result from the last seek
     will be used ;
   DO WHILE(CUSTID NE SEEKID) ;
      * the only way START will be greater than END is if the last seek
        had START equal to END and was also unsuccessful ;
      IF START GT END THEN LEAVE ;
      * read the lookup file according to the latest value of POINTER,
        note that ADDRESS must be sorted by CUSTID! ;
      SET ADDRESS(RENAME=(CUSTID=SEEKID)
         KEEP=CUSTID RESULT)
         KEY=POINTER NOBS=CNUM ;
      * if you found a key larger than the one you wanted then use
        POINTER-1 as the END of the new range,otherwise, use POINTER as
        the START of the new range;
      IF CUSTID LT SEEKID THEN END = POINTER-1 ;
      ELSE IF CUSTID GT SEEKID THEN START = POINTER+1 ;
      * reset POINTER to the middle of the range ;
      POINTER=INT((END+START)/2) ;
   END ;

   IF CUSTID NE SEEKID THEN DO ;
      PUT 'Customer not found' CUSTID= ;
      * set the lookup result to missing, to avoid bad data;
      RESULT = ' ' ;
   END ;

   RUN ;
```