Using SAS to Perform a Table Lookup

Last revised: 05/22/96

Table lookup is the technique you use to acquire additional information from an auxiliary source to supplement or replace information being processed in your SAS data set. To illustrate, consider the data below which was taken from the Places Rated Almanac, published by Rand McNally. 329 cities are rated. The first nine columns of the data contain the rating criteria used by Places Rated Almanac. The tenth column contains a numeric code identifying the city which was rated.

```
468  7339  618  970 2531 2560   237  859 5250   3
476  7908 1431  610 6883 3399  4655 1617 5864   4
659  8393 1853 1483 6558 3026  4496 2612 5727   5
520  5819  640  727 2444 2972   334 1018 5254   6
559  8288  621  514 2881 3144  2333 1117 5097   7
537  6487  965  706 4975 2945  1487 1280 5795   8
```

A second set of data illustrated below contains the same numeric code used above identifying the city which was ranked. The second column identifies the city and state which correspond to the numeric code.


```
134 Houma-Thibodaux, LA
 23 Aurora-Elgin, IL
  5 Albuquerque, NM
 80 Davenport-Rock Island-Moline, IA-IL
```

When you write a SAS program to retrieve the city and state information from the second data set in order to add this information to the first data set, this is known as table lookup.

Definition of Terms

Primary File	is the file for which you want to obtain auxiliary information - the first data set in the illustration above.
Lookup File	is an auxiliary file that is maintained separately from the primary file and is referenced for one or more of the observations of the primary file - the second data set in the illustration above.
Key Variable	is the variable(s) whose values are the common elements between the primary file and the lookup file - the numeric code representing the city in the illustration above.

The lookup file either can be part of the source code, stored in a SAS data set, or stored as a user-defined format.

Table Lookup Techniques

There are several table lookup techniques available in SAS which will be illustrated in this document:

- Match-Merging Two SAS Data Sets
- Direct Access
- Format Tables
- Arrays

SAS also provides a querying language called SQL which can perform table lookup but that technique will not be covered herein.

Match-Merging Two SAS Data Sets

Match-merging can be used in the example on the previous page to retrieve the city and state information from the lookup file in order to add this information to the primary file. Match-merging requires that both the primary file and the lookup file be SAS data sets. To merge the primary and lookup files, do the following:

- make sure both data sets are sorted by the values of the key variable
- merge the two data sets by the key variable to form a third data set

The SAS program below illustrates the process:

```
data places;
   infile "~/soc365/places.dat";
   input climate housing hlth_car crime transp educ arts rec econ
city_cod;
run;

data lookup;
   length city $ 50;
   infile "~/soc365/places.key" missover pad;
   input city_cod city $50.;
run;

proc sort data=places; by city_cod; run;
proc sort data=lookup; by city_cod; run;

data all;
   merge places lookup;
   by city_cod;
   drop city_cod;
run;
```

A partial listing of the resulting data set is shown below:

OBS	CLIMATE	HOUSING	HLTH_CAR	CRIME	TRANSP	EDUC	ARTS	RECON	CITY
1	521	6200	237	923	4031	2757	996	1405	7633	Abilene, TX
2	575	8138	1656	886	4883	2438	5564	2632	4350	Akron, OH
3	468	7339	618	970	2531	2560	237	859	5250	Albany, GA
4	476	7908	1431	610	6883	3399	4655	1617	5864	Albany-Schen, NY
5	659	8393	1853	1483	6558	3026	4496	2612	5727	Albuquerque, NM

Sometimes you need to *compute* the information you want to merge with the primary file. For example, suppose you have a data set which contains information on family members. Each record contains one person's data including a family identifier, the person's gender, and their age:

famid	sex	age
1	1	35
2	1	36
2	1	5
2	2	3
3	1	37
3	2	38
3	2	2
4	1	39
4	1	40
4	2	1
4	2	2
5	1	41
5	1	3
5	1	42
5	1	43
5	2	44
5	2	1
5	2	2

Suppose you need to append to each record the number of women within a family that are between the age of 16 and 66. In order to do match-merging, this supplemental information must be stored in a SAS data set. This can easily be accomplished by doing the following:

- create a new variable whose value is 1 if the individual is a woman and between the age of 16 and 66
- sort the SAS data set by the family identifier
- use one of SAS's procedures for computing descriptive statistics to add up the number of women in each household between 16 and 66 and output the information to a new SAS data set.

The SAS program below illustrates the process:

```
data workable;
   infile "~/dissert/families.dat";
   input famid sex age;
   if 16 < age < 66 and sex = 1 then wfemale=1;
      else wfemale=.;
run;

proc sort data=workable;
   by famid;
run;

 /*count the number of working women in the family */
proc means data=workable noprint;
   var wfemale;
   output out=count sum=numwf;
   by famid;
run;
```

A listing of the resulting data set is shown below:

OBS	FAMID	NUMWF
1	1	1
2	2	1
3	3	1
4	4	2
5	5	3

This SAS data set can now be used as the lookup file in the match-merge:

```
data two;
   merge workable count;
   by famid;
run;
```

The merged data set is shown below:

OBS	FAMID	SEX	AGE	WFEMALE	NUMWF
1	1	1	35	1	1
2	2	1	36	1	1
3	2	1	5	.	1
4	2	2	3	.	1
5	3	1	37	1	1
6	3	2	38	.	1
7	3	2	2	.	1
8	4	1	39	1	2
9	4	1	40	1	2
10	4	2	1	.	2
11	4	2	2	.	2
12	5	1	41	1	3
13	5	1	3	.	3
14	5	1	42	1	3
15	5	1	43	1	3
16	5	2	44	.	3
17	5	2	1	.	3
18	5	2	2	.	3

Table Lookup using Direct Access

Instead of searching a file sequentially until you find the match you need, you can access the lookup result directly based on its location in the lookup file. This technique is known as direct access. If your files are entry-sequenced, you can obtain your lookup results directly, based on the physical location of a given value of the key variable.

To perform a table lookup using direct access, you must be able to identify or assign a key variable in the primary file that indicates the record's location within the lookup file. In the above example, the variable CITY_COD in the PLACES SAS data set meets this requirement. To illustrate, suppose you need to identify the 10 top ranked cities under the category of education. For EDUCATION, the higher the value, the higher the ranking. So, first you need to sort the data set in descending order by EDUCATION:

```
proc sort data=places out=top_educ(obs=10);
   by descending educ;
```

`(obs=10)` is used in order to eliminate all but the first 10 observations. The resulting data set is shown below:

OBS	CITY_COD	EDUC
1	234	3781
2	34	3653
3	314	3635
4	131	3628
5	256	3582
6	243	3558
7	237	3544
8	193	3539
9	262	3530
10	288	3525

Now you are ready to perform the table lookup. First, read in the observations from the primary file:

```
data all;
   set top_educ;
```

Then, to access the correct observation from the lookup file, use the POINT= option in another SET statement. The POINT= option identifies the variable in the primary file (CITY_COD, in this example) whose value represents the location (by observation number) of the observation you want the SET statement to read.

```
   set lookup point=city_cod;
```

Only the necessary observations are read from the lookup data set with direct access unlike match-merging which reads all the records. Note also that the POINT= variable is *not* added to the output data set. Below is the entire program that performs the table lookup and the resulting data set:

```
data all;
   set top_educ;
   set lookup point=city_cod;
run;
```

OBS	CITY	EDUC

1	Philadelphia, PA-NJ	3781
2	Bergen-Passaic, NJ	3653
3	Washington, DC-MD-VA	3635
4	Hartford, CT	3628
5	Rochester, NY	3582
6	Providence, RI	3558
7	Pittsburgh, PA	3544
8	Middlesex-Somerset, Hunterdon, NJ	3539
9	St. Louis, MO-IL	3530
10	Springfield, MA	3525

Table Lookup using Format Tables

Table lookup can sometimes include data manipulation that involves checking data against a list or separating data into categories. Often format tables are used as the lookup file in these situations. The advantage of using format tables is that they can be created and stored separately from the program and data libraries in SAS, meaning they are accessible to any SAS program and can be altered without altering the SAS data set.

Checking Data Against a List

Often when you are processing a variable's data values, you may want to check those values against a list of entries. To illustrate, consider the variable for educational attainment in the PUMS 1990 data set. Educational attainment is coded into 18 categories, 0-17. Printing frequency tables involving this variable is not very meaningful unless you have a data dictionary handy:

EDUC	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	1	0.0	1	0.0
1	4	0.0	5	0.1
2	9	0.1	14	0.1
3	35	0.4	49	0.5
4	91	0.9	140	1.4
5	251	2.5	391	3.9
6	481	4.8	872	8.7
7	962	9.6	1834	18.3
8	1386	13.9	3220	32.2
9	1759	17.6	4979	49.8
10	1740	17.4	6719	67.2
11	1389	13.9	8108	81.1
12	973	9.7	9081	90.8
13	561	5.6	9642	96.4
14	233	2.3	9875	98.8
15	88	0.9	9963	99.6
16	26	0.3	9989	99.9
17	11	0.1	10000	100.0

Using a format table which contains the value labels for the educational attainment codes, would be a big improvement to the above table. This can be done with the list-checking operation following the steps below:

- create a format table whose values act as the lookup table
- check the primary file against the lookup file

- print out the results.

Use PROC FORMAT to create the lookup table:

```
proc format;
   value educ 0="< 3 yrs old"
1="no school"
2="nursery school"
3="kindergarten"
4="thru 4th grade"
5="thru 8th grade"
6="9th grade"
7="10th grade"
8="11th grade"
9="12th but nongrad"
10="H.S. Grad"
11="College,no degree"
12="Assoc.,occupat."
13="Assoc., academic."
14="Bachelor Degree"
15="Master Degree"
16="Profess. degree"
17="Doctorate degree";
run;
```

Then to check the primary file against the lookup file and print out the results, use a FORMAT statement to associate the format table with the educational attainment variable, EDUC:


```
proc freq;
   tables educ;
   format educ educ.;
run;
```

EDUC	Frequency	Percent	Cumulative Frequency	Cumulative Percent
< 3 yrs old	1	0.0	1	0.0
no school	4	0.0	5	0.1
nursery school	9	0.1	14	0.1
kindergarten	35	0.4	49	0.5
thru 4th grade	91	0.9	140	1.4
thru 8th grade	251	2.5	391	3.9
9th grade	481	4.8	872	8.7
10th grade	962	9.6	1834	18.3
11th grade	1386	13.9	3220	32.2
12th but nongrad	1759	17.6	4979	49.8
H.S. Grad	1740	17.4	6719	67.2
College,no degre	1389	13.9	8108	81.1
Assoc.,occupat.	973	9.7	9081	90.8
Assoc., academic	561	5.6	9642	96.4
Bachelor Degree	233	2.3	9875	98.8
Master Degree	88	0.9	9963	99.6
Profess. degree	26	0.3	9989	99.9
Doctorate degree	11	0.1	10000	100.0

Separating Data into Categories

You can also use format tables to categorize data values. This is sometimes a convenient

way of temporarily recoding values. Suppose you needed to recode the 18 educational attainment variables into four categories. You can use a VALUE statement in PROC FORMAT to categorize the values:

```
proc format;
   value 0-9="Did not graduate"
10="H.S. Grad"
11-13="Some college"
14-17="College Grad";
run;
```

Then to check the primary file against the lookup file and print out the results, use a FORMAT statement to associate the format table with the educational attainment variable, EDUC:

```
proc freq;
format yearsch educcat.;
tables educ;
run;
```

EDUC	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Did not graduate	4979	49.8	4979	49.8
H.S. Grad	1740	17.4	6719	67.2
Some college	2923	29.2	9642	96.4
College Grad	358	3.6	10000	100.0

Positional Lookup with Temporary Arrays

Arrays are appropriate for use in table lookups when:

- values to be returned can be identified positionally (that is, 1st item, 2nd, item, and so on)
- one or more numeric values identify the table value to be returned.

The values that make up the array (lookup table) can be hard coded in either the DATA step or stored in a SAS data set or an external file and loaded into array variables via a SET or INPUT statement. Both techniques will be illustrated.

The examples in this section make use of *temporary* arrays. To designate an array as temporary you use the keyword _TEMPORARY_ in the ARRAY statement instead of variable names as elements of the array. Temporary arrays yield the following advantages:

- elements of the array are not written to data sets
- each element requires about 40 bytes less memory than do DATA step variables used in arrays
- faster retrieval of values.

Using Arrays that are Hard Coded in the DATA Step

To illustrate this technique, consider a data set containing scores for various high school tests. Suppose you want to compare each student's performance on the Math, Science, and English exams (HSM, HSE, and HSS) with that of the school district's median scores for the three exams - 8.0 for the Math exam, 7.0 for the Science exam, and 7.5 for the English exam. Write the program following the steps below:

1. Define a temporary array to hold the district wide median scores:

```
array median {3} _temporary_ (8 7 7.5);
```

2. Define an array whose values will be HSM, HSE, and HSS for each observation:

```
array exams {3} hsm hss hse;
```

   The order in which the variables are specified is very important. They must correspond exactly to the order in which they were specified for the temporary array.

3. Define a third array whose values will contain the differences between the student's scores and the district median scores:

```
array diff {3};
```

   Note that no variables or numbers were specified as element of the array in the statement above. By default, SAS automatically creates variable names as elements. For this example, the variable names assigned are DIFF1, DIFF2, and DIFF3.

4. Now, all that is needed is a DO loop to compute the differences. The entire DATA step and a partial listing of the resulting data set are shown below:

```
data gpa;
   infile "gpa.dat";
   input gpa hsm hss hse satm satv sex;
   label gpa="College Grade Point Average"
         hsm="High School Math Exam"
         hss="High School Science Exam"
         hse="High School English Exam"
         satm="Math SAT Score"
         satv="Verbal SAT Score";
   array median {3} _temporary_ (8 7 7.5);
   array exams {3} hsm hss hse;
   array diff {3};
   do i=1 to 3;
      diff{i}=exams{i}-median{i};
   end;
   drop i;
run;
```

OBS	HSM	HSS	HSE	DIFF1	DIFF2	DIFF3
1	10	10	10	2	3	2.5
2	9	9	10	1	2	2.5
3	9	6	6	1	-1	-1.5
4	10	9	9	2	2	1.5
5	6	8	5	-2	1	-2.5

When the lookup operation depends on more than one factor, you can use a

multidimensional array. To illustrate, consider the example on the previous page but assume district median scores were computed separately for men and women:

	HSM	HSS	HSE
Women	Median(1,1)8.0	Median(1,2)7.5	Median(1,3)8.5
Men	Median(2,1)8.5	Median(2,2)7.5	Median(2,3)8.0

With two-dimensional arrays, the elements are placed in the array by filling each column within each row:

```
array median {2, 3} _temporary_ (8.0, 7.5, 8.5, 8.5, 7.5, 8.0);
```

The program is very similar to the one for the above example except that you need to use an IF statement within the DO loop to determine the value of SEX for the observation. Then depending on the value of SEX, the difference is computed. The entire DATA step and a partial listing of the resulting data set are shown below:

```
data gpa;
   infile "gpa.dat";
   input gpa hsm hss hse satm satv sex;
   label gpa="College Grade Point Average"
         hsm="High School Math Exam"
         hss="High School Science Exam"
         hse="High School English Exam"
         satm="Math SAT Score"
         satv="Verbal SAT Score";
   array median {2,3} _temporary_ (8,7.5,8.5,8.5,7.5,8);
   array exams {3} hsm hss hse;
   array diff {3};
   do i=1 to 3;
      if sex=1 then diff{i}=exams{i}-median{1,i};
         else diff{i}=exams{i}-median{2,i};
   end;
   drop i;
run;
```

SEX	HSM	HSS	HSE	DIFF1	DIFF2	DIFF3
1	10	10	10	2.0	2.5	1.5
2	9	9	10	0.5	1.5	2.0
1	9	6	6	1.0	-1.5	-2.5
2	10	9	9	1.5	1.5	1.0

Loading an ARRAY from a SAS Data Set

Array values can also be stored in a SAS data set or an external file. This is convenient when there are too many values to easily initialize in the ARRAY statement and/or the same values are used in many programs. To illustrate, assume you have a permanent SAS data set called CLASS which contains the following variables for a group of teenagers: HEIGHT, WEIGHT, AGE, and TYPE. TYPE represents body type and is made up of three categories: small, medium, or large (coded 1, 2, or 3). CLASS will be the primary file in the table lookup. Following is a listing of the data set:

OBS	HEIGHT	WEIGHT	AGE	TYPE
1	69	112.5	14	2
2	56	84.0	13	1
3	65	98.0	13	2
4	62	102.5	14	3
5	63	102.5	14	3
6	57	83.0	12	2
7	59	84.5	12	3
8	62	112.5	15	2
9	62	84.0	13	1
10	59	99.5	12	3
11	51	50.5	11	3
12	64	90.0	14	2
13	56	77.0	12	3
14	66	112.0	15	1
15	72	150.0	16	3
16	64	128.0	12	2
17	67	133.0	15	2
18	57	85.0	11	3
19	66	112.0	15	3

Suppose you want to determine how overweight the individual is based on a published table of recommended weights based on height and body type. You have this table stored in a permanent SAS data set called TABLE:

OBS	SMALL	MEDIUM	LARGE	HEIGHT
1	102.5	104	105.5	51
2	103.5	105	106.5	52
3	104.5	106	107.5	53
4	105.5	107	108.5	54
5	106.5	108	109.5	55
6	107.5	109	110.5	56
7	108.5	110	111.5	57
8	109.5	111	112.5	58
9	110.5	112	113.5	59
10	111.5	113	114.5	60
11	112.5	114	115.5	61
12	113.5	115	116.5	62
13	114.5	116	117.5	63
14	115.5	117	118.5	64
15	116.5	118	119.5	65
16	117.5	119	120.5	66
17	118.5	120	121.5	67
18	119.5	121	122.5	68
19	120.5	122	123.5	69
20	121.5	123	124.5	70
21	122.5	124	125.5	71
22	123.5	125	126.5	72
23	124.5	126	127.5	73
24	125.5	127	128.5	74
25	126.5	128	129.5	75

TABLE will be the lookup table in this example. Write the DATA step program following the steps below:

1. Define a temporary two-dimensional array to hold the recommended weights:

   ```
   array wt{51:75,3} _temporary_;
   ```

   The syntax `{51:75,3}` indicates that the array has elements numbered {51,1}, {52,1}, {53,1}, . . . , {75,3} instead of {1,1}, {2,1}, {3,1}, . . . 51- 75 represent the range in values for HEIGHT and 1 - 3 represent the range in values for body type in the lookup table.

2. All the values can be loaded into the array at once so to prevent SAS from reloading the values for each loop of the DATA step you need to add an IF condition to instruct SAS to load the array only the first time through the DATA step:

   ```
   if _n_ = 1 then do i=1 to 25;
   ```

   `i` is set equal to `1 to 25` because there are 25 weight values for each body type. You could also have specified `i=51 to 75`.

3. Use a SET statement to instruct SAS where to find the values for the array. Then specify assignment statements to load the elements of the array:

   ```
      set save.table;
      wt{height,1}=small;
      wt{height,2}=medium;
      wt{height,3}=large;
   end;
   ```

The `end;` statement signals the end of the DO loop.

4. Last, SET the primary file and compute the difference between the person's weight and their ideal weight:

```
set save.class;
diff=wt{height,type}-weight;
```

The variables HEIGHT and TYPE are used as pointers to the ARRAY.

The entire DATA step and a partial listing of the resulting data set are shown below:

```
data shape;

* load the lookup table into an array;
   array wt{51:75,3} _temporary_;
   if _n_ = 1 then do i=1 to 25;
      set save.table;
      wt{height,1}=small;
      wt{height,2}=medium;
      wt{height,3}=large;
   end;

* read in the class data;
   set save.class;
   diff=wt{height,type}-weight;
run;
```

OBS	TYPE	HEIGHT	WEIGHT	DIFF
1	2	69	112.5	9.5
2	1	56	84.0	23.5
3	2	65	98.0	20.0
4	3	62	102.5	14.0
5	3	63	102.5	15.0
6	2	57	83.0	27.0
7	3	59	84.5	29.0
8	2	62	112.5	2.5
9	1	62	84.0	29.5
10	3	59	99.5	14.0
11	3	51	50.5	55.0
12	2	64	90.0	27.0
13	3	56	77.0	33.5
14	1	66	112.0	5.5
15	3	72	150.0	-23.5
16	2	64	128.0	-11.0
17	2	67	133.0	-13.0
18	3	57	85.0	26.5
19	3	66	112.0	8.5