Paper 195-27

Table Look-up Using Techniques Other Than the Matched Merge DATA Step

Sandra Lynn Aker, Alysandra Lynn, Inc., Bensenville, IL

Introduction

This paper demonstrates different techniques whereby values stored in a master file can be compared to values stored in a table, an operation referred to as table look-up. Using real-life code, from a retail-sales forecasting project, these techniques are compared and contrasted against the traditional match-merge for advantages and disadvantages, most importantly for efficiencies and limitations. Additionally, tips and tricks are presented with each technique to show how to maximize their effectiveness. The techniques include Indexes, SQL, Arrays, and Formats.

Simple Merge

This technique for comparing values is the standard, and one that is very simple, and yet very powerful. Data is read from a data set or from an external flat file into a data set, and then read again from any number of the same. These files are sorted by any number of variables, and then merged together by those same variables. The ability to flag the data set from which the match occurs using the IN operator, one of the strengths of this technique, allows data manipulation contingent on the match/ nonmatch.

```
DATA MASTER;
SET SASDATA.MASTER
   (KEEP=DEPT ITEM SALES);

DATA TABLE;
INFILE FILEDATA;
INPUT   @1 DEPT 2. @30 PERCENT 4.1;

PROC SORT DATA=MASTER;
BY DEPT;
PROC SORT DATA=TABLE;
BY DEPT;

DATA MATCH;
MERGE MASTER(IN=A) TABLE(IN=B);
BY DEPT;
IF A AND B
```

Merge with Output of Nonmatches Using PROC PRINTO

Note in this example that the first PROC PRINTO designates a new output file where the nonmatches are printed, and the second PROC PRINTO returns printing to the standard output file.

```
DATA MATCH NOMATCH;
MERGE MASTER(IN-A) TABLE(IN-B);
BY DEPT;
IF A THEN DO;
   IF B THEN OUTPUT MATCH;
   ELSE OUTPUT NOMATCH;
END;
PROC PRINTO NEW PRINT=NOTTABLE;
PROC PRINT DATA=NOMATCH;
PROC PRINTO;
```

Merge with Output of Nonmatches Using FILE PRINT and PUT Statement

Note in this example, an output file is created where the nonmatches are placed to be printed or viewed without the need for another data set and a PROC PRINT procedure.

Also note that the PUT statement is used to print the nonmatching observations and their total on the log. (PUT _ALL_= would print all the variables without specifying them).

```
FILENAME REPORT filespec;

DATA MATCH;
MERGE
MASTER(IN=A KEEP=DEPT ITEM SALES)
         END=EOF
TABLE(IN=B KEEP=DEPT PERCENT);
BY DEPT;
FILE REPORT NOTITLES;
IF A THEN DO
   IF B THEN OUTPUT MATCH;
   ELSE DO;
      COUNT + 1;
      PUT 'NO MATCH WITH TABLE ' DEPT=;
   END;
END;
IF EOF THEN
PUT 'NOMATCH COUNT ' COUNT COMMA5.;
```

Merge Using Indexes with the Merge Statement

Although it is not efficient to create an index solely for a match-merge, because the resources will be increased if the data is not in sort order, an index allows table look-up with the MERGE statement without first using a PROC SORT.

Note that in this example the value of the system option MSGLEVEL= is changed to **I** to display index usage information in the log, and the option UNIQUE is used in the table file, but not in the master where the key variable has duplicate values.

Also note that both the INDEX= data set option and the PROC DATASETS procedure are used to create the indexes. Using the INDEX= data set option tested slightly faster than using PROC DATASETS, however, the advantage of using the latter is the ability to gage whether or not there is enough space in the library to create the index before using it. Finally, note that this shows examples of simple indexes, where variables are indexed independently of one another.

```
OPTIONS MSGLEVEL =  I;

DATA MASTER(INDEX=(DEPT));
SET SASDATA.MASTER
        (KEEP=DEPT ITEM SALES);

DATA TABLE;
INFILE FILEDATA;
INPUT @1 DEPT 2. @30 PERCENT 4.1;

PROC DATASETS LIBRARY=WORK;
MODIFY TABLE;
INDEX CREATE DEPT / UNIQUE;

DATA MATCH;
MERGE MASTER(IN=A) TABLE(IN=B);
BY DEPT;
IF A AND B;
```

Merge Using Indexes with Key-Read

A better use of the index would be to do a table look-up with a key-read, particularly if the table is large and relatively few variables need to be retrieved. Here the SET statement uses the KEY= option and the automatic variable _IORC_ with a return code of zero (0) to determine when a match has occurred.

Note that the system option MSGLEVEL= does not need to be set to **I** because the use of an index is explicitly requested, and that only the table file needs to be indexed. Also note that this example uses a composite index where a unique name refers to two or more variables which together make up the index.

```
DATA MASTER;
SET SASDATA.MASTER
        (KEEP=DEPT LINE ITEM SALES);

DATA TABLE
    (INDEX=(DEPTLINE=(DEPT LINE)/UNIQUE));
INFILE FILEDATA;
INPUT   @1 DEPT 2.
        @3 LINE $1.
        @30 PERCENT 4.1;

DATA MATCH;
SET MASTER;
SET TABLE KEY=DEPTLINE/UNIQUE;
IF _IORC_ = 0;
```

Merge Using Indexes with Key-Read By First LINE

Being more creative and efficient, in the next example, the table is only accessed on the first occurrence of a line, and a flag is turned on when a match occurs. Then the flag, and the variables that are input, are retained across all item records in that line. When a match does not occur, the _ERROR_ automatic variable is set to zero to continue processing and the flag is turned off.

Note that all observations are output, to one file when the flag is turned on, and to another when the flag is turned off. This is equivalent in match-merge logic to using the IN=A operator for the master file, and IN=B for the table file and then selecting IF A THEN OUTPUT MATCH and IF A AND NOT B THEN OUTPUT NOMATCH.

It is important to note in this example that when a task follows a nonmatch, the _ERROR_ automatic variable must be reset to 0. This is because a nonmatch is treated as an error which causes SAS® to set _ERROR_ to 1, and to print an error message on the log each time a nonmatch is encountered, up to the limit in the ERROR= option. Also, note that when a nonmatch occurs, that observation would be propagated with variable values from the last match, and to avoid this, the variables are set to blank or missing. Finally, note that because of the BY processing, the master file also must be sorted or indexed prior to the table look-up.

```
PROC SORT DATA=MASTER;
BY DEPT LINE ITEM;

DATA MATCH NOMATCH;
SET MASTER
        (KEEP=DEPT LINE ITEM SALES);
BY DEPT LINE ITEM;
RETAIN FLAG PERCENT;
IF FIRST.LINE THEN DO;
    SET TABLE(KEEP=DEPT LINE PERCENT)
            KEY=DEPTLINE/UNIQUE;
    IF _IORC_ NE 0 THEN DO;
        _ERROR_ = 0;
        FLAG = 'N';
        PERCENT = .;
    END;
    ELSE FLAG = 'Y';
END;
IF FLAG = ‘Y’ THEN OUTPUT MATCH;
ELSE
IF FLAG = ‘N’ THEN OUPUT NOMATCH;
```

Merge Using Indexes with Key-Read Master File has Unique Observations and Table File Has Multiples

If there are unique observations on the master file and multiple observations on the table file, a DO loop is needed, otherwise only the first matching key value from the table file would be retrieved.

Note that again it is necessary to reset the _ERROR_ automatic variable to 0, and that because only matches are kept, it is not necessary to set variables to missing or blank. Also note that UNIQUE is not used with the KEY= option, as this would produce an infinite loop. Finally, note that the files do not need to be sorted, although processing is faster if they are.

```
DATA MATCH;
SET MASTER(KEEP=DEPT LINE SALES);
DO UNTIL (_IORC_ NE 0);
    SET TABLE
        (KEEP = STORE DEPT LINE PERCENT)
         KEY=DEPTLINE;
    ERROR_= 0;
    IF _IORC_ = 0 THEN OUTPUT;
END;
```

Merge Using Indexes with Key-Read
Both Files have Unique Observations and Are Sorted

The prior examples showed multiple observations on one file, the master file or the table file, and unique observations on the other. If there are unique observations on both the master and the table, and both files are sorted in the order of the key variable, it is faster not to use the UNIQUE option to allow sequential processing.

```
PROC SORT DATA=MASTER;
BY DEPT LINE MONTH;
PROC SORT DATA=TABLE;
BY DEPT LINE;

DATA MATCH;
SET MASTER(KEEP=DEPT LINE SALES);
SET TABLE(KEEP = DEPT LINE PERCENT)
      KEY=DEPTLINE;
IF _IORC_ = 0;
```

PROC SQL

If, on the other hand, there are multiple records with the same key variable(s) on both the master file and the table file, table look-up cannot be performed with a key-read. Although it can be performed using a match-merge, a message will appear on the log that the **merge statement has one or more data sets with repeats of by values.** Furthermore, instead of merging every key variable on the master file with every matching key variable on the table file, the first key variable on the master will be matched with the first matching key variable on the table. Then the second will match with the second, the third with the third, and so on, until all of the key variables on the master file have been matched.

MASTER FILE

```
STOREA  MISSES  BLOUSES  $200,000
STOREB  MISSES  BLOUSES  $300,000
STOREC  MISSES  BLOUSES  $400,000
STORED  MISSES  BLOUSES  $500,000
```

TABLE FILE

```
STOREA  MISSES  BLOUSES  QRT1   20.2%
STOREB  MISSES  BLOUSES QRT2  30.3%
STOREC  MISSES  BLOUSES QRT3  40.4%
STORED  MISSES  BLOUSES QRT4   50.5%
```

Performing a match-merge on the department/line, MISSES/BLOUSES, using the above master and table files would produce the following file, which is not the intent:

MATCH-MERGE FILE

```
STOREA  MISSES  BLOUSES  $200,000  QRT1  20.2%
STOREB  MISSES  BLOUSES  $300,000  QRT2  30.3%
STOREC  MISSES  BLOUSES  $400,000  QRT3  40.4%
STORED  MISSES  BLOUSES  $500,000  QRT4  50.5%
```

However, using PROC SQL to join these files on the department/line, MISSES/BLOUSES, would produce the following intended file.

PROC SQL FILE

```
STOREA  MISSES  BLOUSES $200,000  QRT1  20.2%
STOREA  MISSES  BLOUSES $300,000  QRT2  30.3%
STOREA  MISSES  BLOUSES $400,000  QRT3  40.4%
STOREA  MISSES  BLOUSES $500,000  QRT4  50.5%
```
Etc.

This is because the SQL procedure can combine every record from the master file with every matching record form the table file. So that, in the above example, the first record of the master file combines with the first, second, third, and fourth matching records of the table file, and so on. It should be noted that the SQL procedure refers to the table look-up as a join, the files as tables, the records as rows, and the variable as columns.

The CREATE statement names the table to be created. The SELECT statement names the columns to be selected, (Note that the '*' selects all columns. This statement also specifies from what data sets the columns will be selected.) The WHERE statement specifies the matching column values which relate the tables together to produce the join. The ORDER statement sorts the file, as with PROC SORT. The QUIT statement stops the procedure from processing

```
PROC SQL
CREATE TABLE join AS
SELECT *
   FROM master as m, table as t
   WHERE m.dept =t.dept AND m.line=t.line
ORDER BY dept line;
QUIT;
```

The preceding example is equivalent, in a match-merge, to using the IN=A operator for the master file and the IN=B operator for the table file, and then selecting IF A AND B. If there might be rows on the master for which there are not matches on the table, the FROM statement would include a LEFT JOIN and ON, and the WHERE statement would not be used. This is the equivalent to selecting IF A.

```
SELECT *
   FROM master as m LEFT JOIN table as t
   ON m.dept=t.dep AND  m.line=t.line
```

Similarly, a RIGHT JOIN would resolve rows on the table for which there are not matches on the master, and would be equivalent to selecting IF B.

```
SELECT *
   FROM master as m RIGHT JOIN table as t
   ON m.dept=t.dep AND  m.line=t.line
```

Unfortunately, it is not possible to create two output tables with PROC SQL, so that the equivalent to selecting IF A AND NOT B would have to be performed separately as

```
PROC SQL
SELECT *
   FROM master as m
   WHERE not exists
      (SELECT *
      FROM table as t
      WHERE m.dept =t.dept AND m.line=t.line);
QUIT;
```

Arrays

Here, the table has one record for each department and line, but instead of the different percents being in rows, they are in twelve columns, one for each month. This example shows an easy way to read them, and the use of simple arrays to change the percent value for later multiplication. Note that in the first ARRAY statement, the order of the percents is changed so that in processing, PCNT(1) is P(7), PCNT(2) is P(8), etc. This is done in order to change the month that starts the processing to July.

```
            DATA TABLE(KEEP=DEPT LINE PCNT1-PCNT12);
            INFILE FILEDATA;
            INPUT
/* no spaces between the columns would be (4.1) */
            @1 DEPT 2. @3 LINE $1. @30 (P1-P12) (4.1 + 4);
I* start the processing with July */
            ARRAY P (12) P7-P12 P1-P6;
            ARRAY PCNT (12);
/* remove the decimal point */
            DO J = 1 TO 12;
               PCNT(J) = P(J)/100;
            END;
```

The purpose of the following program is to provide a beginning point for planning sales based on last year for next year with estimates where there is no previous sales history. The estimate is made by multiplying for each department/line/item, the expected percent sales increase (from the table), with the annual sales forecast. This is calculated as the total sales across all months (from the master) divided by total percents across all months (from the table), totaled when sales in those months are not zero. The problem is that the table file has months as columns, where the master file has months as rows. This first example transposes the table file to percents by months as rows, and merges the table and master files together. The annual forecast by department/line/item is calculated in a separate data step, and merged back with the master file to estimate sales when sales are zero.

```
/* read the master file */
            DATA MASTER;
            SET SASDATA.MASTER
            (KEEP=DEPT LINE ITEM MONTH SALES);
/* read  the table file */
            DATA TABLE(KEEP=DEPT LINE PCNT1-PCNT12);
            INFILE FILEDATA;
            INPUT
            @1 DEPT 2. @3 LINE $1. @30 (P1-P12) (4.1 + 4);
            ARRAY P (12);
            ARRAY PCNT (12);
/* remove the decimal point */
            DO J = 1 TO 12;
            PCNT(J) = P(J)/100;
            END;
/* sort the master and table files */
            PROC SORT DATA=MASTER;
            BY DEPT LINE MONTH;
            PROC SORT DATA=TABLE;
            BY DEPT LINE;
/* flip the table file */
            DATA FLIPTBLE;
            (KEEP=DEPT LINE MONTH PERCENT);
            SET TABLE;
            BY DEPT LINE;
            ARRAY PCNT (12);
/* flip the percents as columns to rows */
            DO J = 1 TO 12;
               PERCENT = PCNT(J);
               MONTH = J;
               OUTPUT;
            END;
/* merge the master file and flipped table file */
            DATA SALES;
            MERGE MASTER(IN=A) FLIPTBLE(IN=B);
            BY DEPT LINE MONTH;
            IF A AND B;
/* sort the merged sales file */
            PROC SORT DATA=SALES;
            BY DEPT LINE ITEM MONTH;
/*create a  forecast computation file */
            DATA FORECAST
            (KEEP=DEPT LINE ITEM ANN_FCST);
            SET SALES;
            BY DEPT LINE ITEM MONTH;
/* initialize the percent sum and sales sum */
            IF FIRST.ITEM THEN DO;
               SUMPCNT = 0;
               SUMSALE = 0;
            END;
/* sum percent and sales across months */
/* when sales are not 0 */
            IF SALES NE 0 THEN DO;
               SUMPCNT + PERCENT;
               SUMSALE + SALES;
            END;
/* at last month compute annual forecast and output */
            IF LAST.ITEM THEN DO;
            ANN_FCST = SUMSALE / SUMPCT;
               OUTPUT;
            END;
/*create the final sales file */
            DATA COMPUTE
            (KEEP=DEPT LINE ITEM SALES ANN_FCST);
/* merge the forecast file back with the sales file */
            MERGE SALES(IN=A) FORECAST(IN=B);
            BY DEPT LINE ITEM;
/* compute sales when sales are 0 */
            IF  SALES = 0 THEN
                SALES = ANN_FCST * PERCENT;
```

This can be made much simpler and more efficient by transposing the master file to months as columns, performing a table look-up of the table file with a key-read, and then performing all of the calculations within arrays in a single data step. Note that the table look-up is performed at the first occurrence of line in the master file, and only when a match occurs are the percents input, recalculated and retained across all items. Note also that because the table resides on an indexed VSAM file, the key-read does not require putting the table to a data set.

```
/* read and sort the master file */
            DATA MASTER;
            SET SASDATA.MASTER
            (KEEP=DEPT LINE ITEM MONTH SALES);
            PROC SORT DATA=MASTER;
            BY DEPT LINE ITEM MONTH;
```

```
/* flip the master file */
             DATA FLIPMAST
             (KEEP=DEPT LINE ITEM SLS1-SLS12);
             SET MASTER;
             BY DEPT LINE ITEM MONTH;
             ARRAY SLS (12);
             RETAIN SLS1-SLS12;
/* flip the sales as rows to columns */
             J = MONTH;
             SLS(J) = SALES;
             IF LAST.ITEM THEN OUTPUT;
/* merge the flipped master file and table file */
             DATA SALES
             (KEEP= DEPT LINE ITEM
                          SLS1-SLS12 PCNT1-PCNT12);
             SET FLIPMAST;
             BY DEPT LINE ITEM;
             ARRAY P (12);
             ARRAY PCNT (12);
/* retain the percents across all items */
             RETAIN PCNT1-PCNT12;
/* perform at the first line */
             IF FIRST.LINE THEN DO;
/* concatenate dept and line */
                 DEPTLINE = PUT(DEPT,Z2.)||LINE;
/* define deptline as a key */
                 INFILE FILEDATA VSAM KEY=DEPTLINE;
/* read the table file and hold the record */
                 INPUT @;
/* if a match occurs
                 IF _IORC_ EQ 0 THEN DO;
/* input the percents * /
                     INPUT @30 (P1-P12) (4.1 + 4);
/* remove the decimal */
                     DO J = 1 TO 12;
                         PCNT(J) = P(J)/100;
                     END;
                 END;
             END;
/* create  the final sales file */
             DATA COMPUTE
             (KEEP= DEPT LINE ITEM
                          SLS1-SLS12 ANN_FCST);
/*set the sales file */
             SET SALES;
             BY DEPT LINE ITEM;
             ARRAY SLS (12);
             ARRAY PCNT (12);
/*initialize  the percent sum and sales sum */
             SUMPCNT = 0;
             SUMSALE = 0;
/* sum sales and percents across months */
/* when sales are not 0 */
             DO J = 1 TO 12;
                 IF SLS(J) NE 0 THEN DO;
                     SUMSALE + SLS(J);
                     SUMPCNT  + PCNT(J);
                 END;
             END;
/* compute the annual forecast *\
              ANN_FCST = SUMSALE / SUMPCNT;
/* compute sales when sales are 0 *\
             DO K = 1 TO 12;
                 IF SLS(K) = 0
                 THEN SLS(K) = ANN_FCST * PCNT(K);
             END;
```

It should be noted that this example tested much faster than the first as the I/O was reduced with the use of the arrays.

Formats

The final task in this process is to select only certain departments for which to plan sales. Such a selection can be made with an IF statement or WHERE clause if the selection list is small, otherwise a table would be used to identify the appropriate selections. This example shows making the selection using match-merge.

```
DATA MASTER;
SET SASDATA.MASTER
(KEEP=DEPT LINE ITEM MONTH SALES);

DATA TABLE;
INFILE FILEDATA;
INPUT @1 DEPT 2. @6 DEPTNAME $20.;

PROC SORT DATA=MASTER;
BY DEPT;
PROC SORT DATA=TABLE;
BY DEPT;

DATA MERGE;
MERGE MASTER(IN=A) TABLE(IN=B);
BY DEPT;
IF A AND B;
```

If the table were in a PROC FORMAT procedure, as in the following example, the selection could be made much more economically with the WHERE= data set option using the PUT function. Further the name from the table would still be available as a variable with another PUT function.

```
PROC FORMAT;
VALUE SELECTDP
01  = 'BED AND BATH'
02  = 'ELECTRONICS'
03  = 'JEWELRY'
.....rest of departments
OTHER = 'MISTAKE';

DATA MASTER;
SET SASDATA.MASTER
(KEEP=DEPT LINE ITEM MONTH SALES
WHERE=(PUT(DEPT,SELECTDP.) NE 'MISTAKE'));
DEPTNAME = PUT(DEPT,SELECTDP.)
```

To avoid hardcoding this format procedure in every program where it might be used, it could be permanently stored in a data library in a catalog. The data library is SASLIB, referring to the DD statement, and the catalog name is SASFMTS. If a catalog is not named, the format is stored in the FORMATS catalog. This example puts a format to a catalog in a permanent library in an MVS environment. Note the FMTSEARCH system option which is necessary to identify the catalog to be searched if it is not named FORMATS. This option must also be used in the programs that will use the permanent format. In the FORMATS statement, note the use of the FMTLIB option which prints the contents of the catalog, and the PAGE option which prints information about each format in the catalog on a separate page.

```
//SASLIB DD DSN=T.SASLIB.FORMATS,DISP=(,CATLG)
//DCB=(BLKSIZE=23040,LRECL=23040,
//          DSORG=PS,RECFM=FS)
//SYSIN   DD * OPTIONS FMTSEARCH=(SASLIB.SASFMTS);

          PROC FORMAT
          LIBRARY=SASLIB.SASFMTS  FMTLIB PAGE;
          VALUE SELECTDP
          01  = 'BED AND BATH'
          02  = 'ELECTRONICS'
          .....rest of departments
          OTHER = 'MISTAKE';
```

Conclusion

This paper, with its examples, compared table look-up using match-merge with techniques using Indexes, SQL, Arrays, and Formats. The examples not only demonstrate how processing is reduced by the use of these other techniques as opposed to the use of the traditional match-merge statements, but also show additional tips that can make them, as well as match-merge, even more efficient.

References

SAS Institute, Inc., (1992), **Advanced SAS Programming Techniques and Efficiencies Course Notes**, Cary, NC,

Trademarks

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

Author Information

Sandra Lynn Aker

Alysandra Lynn, Inc.
110 East George Street #307
Bensenville, IL 60106
630-766-5339