

Table Look-up using Indexes, Arrays, Formats, and SQL without using Matched Merge Data Steps Sandra Lynn Aker, ASG, Inc.

Introduction

This paper presents four 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. These techniques do not use the traditional *match-merge*, but instead are contrasted against it for advantages and disadvantages, most importantly for efficiencies. The author recommends the SAS® training publication referenced at the end for further comparison of these alternatives to matching values without using the MERGE statement, including CPU and IO usage.

Match-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 from any number of the same, sorted by any number of variables, and merged by those same variables.

```
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 MERGE;
MERGE MASTER TABLE;
BY DEPT;
```

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. Note in the example on the right, an output file is created where the nonmatches are placed to be printed or viewed without the need for a data set and a PROC PRINT procedure. Also note that the PUT statement is used to print the nonmatching records and their total on the log. (PUT _ALL_= would print all the variables without specifying them).

<pre>DATA MERGE NOTTABLE; MERGE MASTER(IN=A) TABLE(IN=B); BY DEPT; IF A THEN DO; IF B THEN OUTPUT MERGE; ELSE OUTPUT NOTTABLE; END; PROC PRINT DATA=NOTTABLE;</pre>	<pre>DATA MERGE; MERGE MASTER(IN=A KEEP=DEPT ITEM SALES) END=EOF TABLE (IN=B KEEP=DEPT PERCENT); BY DEPT; FILE REPORT NOTTILES; IF A THEN DO; IF B THEN OUTPUT MERGE; ELSE DO; COUNT + 1; PUT 'NO MATCH WITH TABLE ' DEPT=; END; END; IF EOF THEN PUT 'NOMATCH COUNT ' COUNT COMMAS.;</pre>
---	--

Indexes

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 the SORT

procedure. Note that in this example the value of the system option MSGLEVEL= is changed to 1 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 gauge 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 = 1;

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

```

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 to determine when a match has occurred. Note that the system option MSGLEVEL= does not need to be set to 1 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 MERGE;
SET MASTER;
SET TABLE KEY=DEPTLINE/UNIQUE;
IF _IORC_ = 0;

```

PROC SQL

If there are multiple records with the same key variable(s), table look-up cannot be performed with a *key read*, and although it can be performed using *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, until all of the key variables on the master file have been matched.

MASTER FILE	TABLE FILE
FOX VALLEY ELECTRONICS TELEVISIONS 20000	ELECTRONICS TELEVISIONS QRT1 20.2
RANDHURST ELECTRONICS TELEVISIONS 30000	ELECTRONICS TELEVISIONS QRT2 30.3
WOODFIELD ELECTRONICS TELEVISIONS 40000	ELECTRONICS TELEVISIONS QRT3 40.4
STRATFORD ELECTRONICS TELEVISIONS 50000	ELECTRONICS TELEVISIONS QRT4 50.5

Therefore, merging on department/line would NOT produce the desired file.

MERGE FILE

```
FOX VALLEY ELECTRONICS TELEVISIONS 20000 QRT1 20.2
RANDHURST ELECTRONICS TELEVISIONS 30000 QRT2 30.3
WOODFIELD ELECTRONICS TELEVISIONS 40000 QRT3 40.4
STRATFORD ELECTRONICS TELEVISIONS 50000 QRT4 50.5
```

However, joining on department/line would produce the desired file.

PROC SQL FILE

```
FOX VALLEY ELECTRONICS TELEVISIONS 20000 QRT1 20.2
FOX VALLEY ELECTRONICS TELEVISIONS 30000 QRT2 30.3
FOX VALLEY ELECTRONICS TELEVISIONS 40000 QRT3 40.4
FOX VALLEY ELECTRONICS TELEVISIONS 50000 QRT3 50.5
```

This is because the SQL procedure can combine every record from the master file with every matching record from 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, with * selecting all columns. This statement also specifies what data sets will be selected from. The WHERE statement specifies the matching values which relate the tables together. 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. The ORDER statement sorts the file. Note the QUIT statement which stops the SQL 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;
```

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

Arrays

Here, the table has one record for department and line, and twelve columns with percents, one for each month. This example shows an easy way to read them, and 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(6), PCNT(2) is P(7), etc. This was done to change what month which starts the season.

```
DATA TABLE(KEEP=DEPT LINE PCNT1-PCNT12);
INFILE FILEDATA;
INPUT @1 DEPT 2. @3 LINE $1. @30 (P1-P12) (4.1 + 4); /* no spaces between is (4.1) */
ARRAY P (12) P6-P12 P1-P5;
ARRAY PCNT (12);
DO J = 1 TO 12;
PCNT(J) = P(J)/100;
END;
```

The purpose of this 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 percent (from this table), with the annual forecast, which is the total sales across all months (from the master) divided by total percents across all months (from this table), totaled when sales in those months are not zero. The problem is that this table file has months as columns, where the master has months as rows. This 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.

```

DATA MASTER;                                /* read the master file */
SET SASDATA.MASTER(KEEP=DEPT LINE ITEM MONTH SALES);

DATA TABLE(KEEP=DEPT LINE PCNT1-PCNT12);    /* read the table file */
INFILE FILEDATA;
INPUT @1 DEPT 2 @3 LINE $1. @30 (P1-P12) (4.1 + 4);
ARRAY P (12);
ARRAY PCNT (12);
DO J = 1 TO 12;                             /* remove the decimal point */
    PCNT(J) = P(J)/100;
END;

PROC SORT DATA=MASTER;                     /* sort the master file */
BY DEPT LINE MONTH;
PROC SORT DATA=TABLE;                     /* sort the table file */
BY DEPT LINE;

DATA FLIPTBLE(KEEP=DEPT LINE MONTH PERCENT); /* flip the table file */
SET TABLE;
BY DEPT LINE;
ARRAY PCNT (12);
DO J = 1 TO 12;                             /* flip the percents as columns to rows */
    PERCENT = PCNT(J);
    MONTH = J;
    OUTPUT;
END;

DATA MERGE;                                /* merge the flipped table file
MERGE MASTER(IN=A) FLIPTBLE(IN=B);          with the master file */
BY DEPT LINE MONTH;
IF A AND B;

PROC SORT DATA=MERGE;                     /* sort the merged file */
BY DEPT LINE ITEM MONTH;

DATA FORECAST(KEEP=DEPT LINE ITEM ANN_FCST);
SET MERGE;
BY DEPT LINE ITEM MONTH;
IF FIRST.ITEM THEN DO;                     /* initialize the percent sum and sale sum */
    SUMPCNT = 0;
    SUMSALE = 0;
END;
IF SALES NE 0 THEN DO;                     /* sum percent and sales across months
    SUMPCNT + PERCENT;                     when sales are not 0 */
    SUMSALE + SALES;
END;
IF LAST.ITEM THEN DO;                      /* at last month compute annual forecast and output */
    ANN_FCST = SUMSALE / SUMPCNT;
    OUTPUT;
END;

DATA COMPUTE(KEEP=DEPT LINE ITEM SALES ANN_FCST); /* merge the forecast file
MERGE MERGE(IN=A) SUMMS(IN=B);             with the flipped table/master */
BY DEPT LINE ITEM;
IF SALES = 0 THEN SALES = ANN_FCST * PERCENT; /* compute sales when sales are 0 */

```

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.

```

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

DATA FLIPMAST(KEEP=DEPT LINE ITEM SLS1-SLS12);   /* flip the master file */
SET MASTER;
BY DEPT LINE ITEM MONTH;
ARRAY SLS (12);
RETAIN SLS1-SLS12;
J = MONTH;                                     /* flip the sales as rows to columns */
SLS(J) = SALES;
IF LAST.ITEM THEN OUTPUT;

DATA MERGE(KEEP=DEPT LINE ITEM SLS1-SLS12 PCNT1-PCNT12); /* merge the flipped master file
SET FLIPMAST;                                     with the table file */
BY DEPT LINE ITEM;
ARRAY P (12);
ARRAY PCNT (12);
RETAIN PCNT1-PCNT12;                           /* retain the percents across all items */
IF FIRST.LINE THEN DO;                         /* perform at the first line */
  DEPTLINE = PUT(DEPT,Z2.)||LINE;               /* concatenate dept and line */
  INFILE FILEDATA VSAM KEY=DEPTLINE;           /* define deptline as a key */
  INPUT @;                                     /* read the table file and hold the record */
  IF _IORC_ EQ 0 THEN DO;                       /* if a match occurs
    INPUT @30 (P1-P12) (4.1 + 4);               input the percents */
    DO J = 1 TO 12;
      PCNT(J) = P(J)/100;                       /* remove the decimal */
    END;
  END;
END;

DATA COMPUTE(KEEP=DEPT LINE ITEM SLS1-SLS12 ANN_FCST); /*perform the computations */
SET MERGE;
BY DEPT LINE ITEM;
ARRAY SLS (12);
ARRAY PCNT (12);
SUMPCNT = 0;                                   /* initialize the percent sum and sales sum */
SUMSALE = 0;
DO J = 1 TO 12;
  IF SLS(J) NE 0 THEN DO;                       /* sum sales and percents across months
    SUMSALE + SLS(J);                           when sales are not 0 */
    SUMPCNT + PCNT(J);
  END;
END;
ANN_FCST = SUMSALE / SUMPCNT;                 /* compute the annual forecast */
DO K = 1 TO 12;
  IF SLS(K) = 0 THEN SLS(K) = ANN_FCST * PCNT(K); /* compute sales when sales are 0 */
END;

```

Formats

The final task in this program 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),UNIT=3380,SPACE=(CYL,(10,3)),
// 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'
03 = 'JEWELRY'
.....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 SORT/MERGE statements, but also show additional tips that can make them as well as *match-merge* even more efficient.

Author Information

Sandra Lynn Aker
Program Chair, Wisconsin-Illinois SAS User Group

ASG, Inc.
175 East Hawthorne
Suite 155
Vernon Hills, IL 60061
(847) 982-7084
slaker@monsanto.com

References

SAS Institute, Inc., (1992), **Advanced SAS Programming Techniques and Efficiencies Course Notes**, Cary, NC, (order #P58230)

Trademarks

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