

Using KEY= to Perform Table Look-up Sandra Lynn Aker, ASG, Inc.

Introduction

This paper demonstrates the use of indexes to perform table look-up, a technique whereby values stored in a master file can be compared to values stored in a table. This technique does not use the traditional *match-merge*, but instead is contrasted against it for advantages and disadvantages, most importantly for efficiencies. There are other alternatives to matching values without using the MERGE statement, and for an overview, the author recommends the SAS® training publication referenced at the end.

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).

```
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;
```

```
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 COMMA5.;
```

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.

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 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 = 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 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 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 MERGE;
SET MASTER;
SET TABLE KEY=DEPTLINE/UNIQUE;
IF _IORC_ = 0;

```

Being more creative and efficient, the next example only accesses the table on the first occurrence of line, then turns on a flag when a match occurs and retains it across all items 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. All observations are output, to one file when the flag is turned on, and to another when the flag is turned off. Because of the BY processing in this example, the master file must be sorted or indexed prior to the table look-up, while again the table does not. It is important to note in this example that when a task follows a nonmatch, the _ERROR_ automatic variable must be reset to 0, because a nonmatch is treated as an error which causes SAS to set _ERROR_ to 1, and print an error message on the log each time a nonmatch is encountered up to the limit in the ERROR= option. Also, note when a nonmatch occurs, that record would be propagated with variable values from the last match, and to avoid this the variables are set to blank or missing.

```

PROC SORT DATA=MASTER;
BY DEPT LINE ITEM;

```

```

DATA MERGE NOMERGE;
SET MASTER(KEEP=DEPT LINE ITEM SALES);
BY DEPT LINE ITEM;
RETAIN FLAG;
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 MERGE;
ELSE
IF FLAG = 'N' THEN OUPUT NOMERGE;

```

An even more creative and efficient method would not only access the table on the first occurrence of line, and only when a match occurs, but also avoid putting the table to a data set. This can be achieved if the table resides on an indexed VSAM file, and in this example, on the first occurrence of line when a match with the key occurs, the remaining variable is read from the VSAM table file and the input is retained across all items in that line. When a match does not occur the _ERROR_ automatic variable is set to zero to continue to the PUT statement where the missing record is printed, and then to the RETURN statement which takes processing to the top of the data step. Note the creation of DEPTLINE to be used as the key for the VSAM file, and that because the variable DEPT is numeric the PUT function is used to retain leading zeroes. Finally, note that a more simple process without the need for printing of nonmatches is given.

```

PROC SORT DATA=MASTER;
BY DEPT LINE ITEM;

DATA MERGE;
SET MASTER(KEEP=DEPT LINE ITEM SALES);
BY DEPT LINE ITEM;
RETAIN PERCENT;
IF FIRST.LINE THEN DO;
  DEPTLINE = PUT(DEPT,Z2.))||LINE;
  INFILE FILEDATA VSAM KEY=DEPTLINE;
  INPUT @;
  IF _IORC_ NE 0 THEN DO;
    _ERROR_ = 0;
    PUT 'NO MATCH WITH TABLE ' DEPTLINE=;
    RETURN;
  END;
  INPUT @30 PERCENT 4.1;
END;

IF _IORC_ = 0 THEN DO;
  INPUT @30 PERCENT 4.1;
END;

```

These examples showed multiple records on the master file and unique records on the table file. If there are also unique records on the master file, and both files are sorted in the order of the key variable, it is faster to not use the UNIQUE option to allow sequential processing.

```

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

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

```

If there are unique records on the master file and multiple records on the table file a DO loop is needed, otherwise only the first matching key value from the table file is 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 of blank. Also note that unique is not used with the `KEY=` option as this produces an infinite loop. Finally, note that the files do not need to be sorted, although processing is faster if they are.

```
DATA MERGE;
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;
```

If the data sets are large, modifying them in place would save a significant amount of temporary work space by avoiding the creation of additional copies. This is accomplished by performing a *key read* with the `MODIFY` statement, which updates or replaces existing variables on the master file with those from the table file, in contrast to performing a *key read* with a `set` statement which merges the master with the table. Only those records on the master file whose key matches the key on the table file are changed, while those that do not match remain as they were. In this example, the percents are updated on the table file, and not on the master file because the `MODIFY` statement can not be used when there are duplicate keys. Note that the variable being replaced is renamed so that it will be unique in the assignment. Also, note that the master file is backed up with the `COPY` procedure, because should a system abort or interrupt occur the result would be an unrecoverable loss of data.

```
DATA COPY IN=SASDATA.TABLE OUT=BACKUP.TABLE;

DATA SASDATA.TABLE;
SET SASDATA.NEWTABLE(RENAME=(PERCENT=NEW_PCNT)
                      KEEP=DEPT LINE PERCENT);
MODIFY SASDATA.TABLE KEY=DEPTLINE;
PERCENT = NEW_PCNT;
```

Conclusion

This paper, with its examples, compared table look-up using *match-merge* with the technique of using indexes. The examples demonstrate the many ways processing is reduced by the use of `KEY=` as opposed to the use of the traditional `SORT/MERGE` statements, and also show additional tips that can make it's use 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)
 Kim L. Kolbe Ritzow, "Randomly Accessing Files Using the `KEY=` Option", *Proceedings of the New York Area SAS Users' Group*, VOL 11, NO 3, September 1996

Trademarks

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