Paper 234-25

Using KEY= to Perform Table Look-up

Sandra Lynn Aker, ASG, Inc., Vernon Hills, IL

Introduction

This paper demonstrates the use of indexes to perform table look-up, the process of locating observations in a table file based on values in a master file. It begins by discussing how this is done with the traditional MATCH-MERGE, including a demonstration of tips and tricks for handling matches and nonmatches. It then compares several different techniques used to create an INDEX, explaining both the simple and composite, and shows how to use them in a MATCH-MERGE, and then in a KEY READ with the KEY= data set option. It explains the _IORC_ and _ERROR_ automatic variables and discusses how to use these to make the process more efficient. It continues with demonstrations of different ways to access a table, especially when the key values are not unique on either the table or the master file, and also shows how to process nonmatches accurately. It concludes with a demonstration of how to use a KEY READ to modify data sets, and finally, a discussion of the overall advantages and disadvantages of using indexes.

Simple Merge

This is the standard technique for comparing values where data is read from a data set, or from an external flat file, sorted by any number of variables, and merged by those same variables. Note that the ability to flag the data set from which the match occurs using the IN operator, 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 MERGE;
MERGE MASTER TABLE;
BY DEPT;
```

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 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 PRINTO NEW PRINT=NOMATCH;
PROC PRINT DATA=NOTTABLE;
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 a 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 MERGE;
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 MERGE;
   ELSE DO;
      COUNT + 1;
      PUT 'NO MATCH WITH TABLE ' DEPT=;
   END;
END;
IF EOF THEN
PUT 'NOMATCH COUNT ' COUNT COMMA5.;
```

Merge Using Indexes with 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 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 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 MERGE;
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 MERGE;
SET MASTER;
SET TABLE KEY=DEPTLINE/UNIQUE;
IF _IORC_ = 0;
```

Merge Using Indexes with Key Read By First LINE

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, and the variables input, 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 observation 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 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 MERGE;
ELSE
IF FLAG = 'N' THEN OUPUT NOMERGE;
```

Merge Using Indexes with Key Read By First LINE With a VSAM Table File

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.

In this example, on the first occurrence of LINE when a match with the key occurs, the remaining variables are 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 observation 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 again the variables are set to missing or blank when a nonmatch occurs.

```
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;
      PERCENT=.;
      PUT
      'NO MATCH WITH TABLE '
         DEPTLINE= PERCENT=;
      RETURN;
   END;
   INPUT @30 PERCENT 4.1;
END;
```

If you do not wish to process the nonmatches, then the following will input only those observations from the table where there is a match.

```
IF FIRST.LINE THEN DO;
   DEPTLINE = PUT(DEPT,Z2.)||LINE;
   INFILE FILEDATA VSAM KEY=DEPTLINE;
   IF _IORC_ = 0 THEN
      INPUT @30 PERCENT 4.1;
END;
```

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

These examples showed multiple observations on the master file and unique observations on the table file. If there are also unique observations 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 ITEM;
PROC SORT DATA=TABLE;
BY DEPT LINE ITEM;

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

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 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;
```

Updating Using Indexes with Key Read when the Files are Too Large for Temporary Work Space

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 update file.

This is in contrast to performing a KEY READ with a SET statement, which merges the master with the update, to create an output file with information from both. With KEY= on the MODIFY statement, only those observations where the key on the master file matches the key on the update are changed in place, while those that do not match remain as they were.

Note that when using the MODIFY statement with the KEY= option, it is best to have unique observations on both files. Also, note that the variable on the update file is renamed so that It can be uniquely referred to in the assignment statement.

Finally, 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.

```
PROC 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;
```

Guidelines for Indexing

The SAS® System will decide whether to use an index outside of a table lookup when it is referenced in a BY Statement, A WHERE clause or WHERE data set option, or an SQL join. Note, that an index is not used with a subsetting IF statement. Using the MSGLEVEL=I option, mentioned earlier, will display a message in the log to indicate if the index was used.

Therefore, when selecting a variable to be indexed, it is best to select one that is not only frequently queried with a table lookup, but also one that is often subset on with BY or WHERE. Also, an indexed variable should have values that identify small subsets of the data set, preferably where less than 25% of the observations have the same value as the index variable. Furthermore, an index should have a large number of distinct values, and should be evenly distributed throughout the data or, even better, the indexed variable should be sorted.

Other guidelines to follow are that the number of indexes should be kept small so as to reduce the resources necessary for storage and maintenance. Data sets that are indexed should be stable because resources are required to rebuild the index each time an observation is added or deleted. and, finally, small data sets with only a few pages of memory should not be indexed as any efficiencies would be minimal.

Advantages and Disadvantages of Indexing

The advantages of using an index for table lookup are that only the observations where a match occurs are read from the table, multiple values can be retrieved from the table, and the appropriate master observation is directly matched. This means that, if the guidelines are adhered to, access to the table is made significantly faster.

The disadvantages are that more CPU cycles and I/O operations are needed to create and maintain the index, and also that extra memory is needed to load the index pages when processing code, and to store the index on disk. Also, note that indexes cannot be used with a table file that is on tape.

Conclusion

This paper demonstrated the use of indexes to perform table look-up, with examples that showed the many ways processing can be reduced by the use of KEY= as opposed to the traditional MATCH-MERGE. It also provided additional tips and tricks that can make it's use more efficient and accurate, and concluded with its overall advantages and disadvantages, including guidelines for its use.

Author Information

Sandra Lynn Aker
ASG, Inc.
175 East Hawthorne Parkway
Suite 155
Vernon Hills, IL 60061
(847) 982-7084
SANDRA.L.AKER@chi.monsanto.com

Trademarks