

Enhancing the Multiple Match-Merge Solution

Bill Horman, A.T. Kearney, Inc.

Background

The multiple match-merge is a manipulation of data that SAS[®] cannot accomplish with just one MERGE statement. It involves the creation of all combinations of records for a given value of a merge variable. For example, if data set A has 3 observations with a merge variable value of IL and data set B has 5 observations with the same merge variable value, the resultant data set will have 15 (3*5) records with the merge variable value IL. A multiple match-merge may or may not output records that do not have any matching values of the merge variable in the other merged data set. In the more common application of a multiple match-merge, records that do not match are not output to the final data set. This paper and its source code will be dealing with just the common application, illustrating the savings in computer resources achieved by using that source code.

In one rather large application, I had been working with Terry Lessmann's paper from SUGI 1990. However, an attempt to create a large multiple match-merged data set continued to fail due to the existence of intermediate data sets. It became necessary, due to space limitations, to create the data set in just one data step.

Objectives

With one data step as the primary goal, other goals were created to help reduce the number of alternatives tested. A secondary goal was to create the final data set while expending fewer resources, primarily CPU seconds and real minutes. The third goal was to create a solution that could be adapted as a macro. Finally, the solution should be able to simply handle applications where unmatched records are added to the final data set. If they could be achieved, larger problems could be solved without the usually inevitable involvement of work-around design and programming.

Approach

The search involved as many alternative and creative methods of data manipulation that could be created within a several hour time budget. Testing followed, using the %MACMERG macro as the standard. Of the two alternatives that achieved the primary goal, one had used fewer CPU seconds and appeared quicker.

More rigorous testing was attempted on our IBM 4381 mainframe running version 5.18 under VM/CMS. However, control was difficult. The two input data steps would vary greatly from program to program and from run to run depending, of course, on the load on the mainframe. Still, CPU second savings were in the 60-70% range.

In order to regain control, testing was moved to a 386 PC running version 6.03 of base SAS. Time then became the sole measure of resource efficiency, with tests parallel to the mainframe tests generating 50% savings.

Findings

The source code included at the end of this paper was able to achieve the first three goals of the multiple match-merge solution. It duplicated the content of the %MACMERG macro in every test, while using only one DATA step. All tests generated a time savings of at least 20%, and more importantly, all tests showed an increase in net savings as data set size increased and as combinations per merge variable increased. Detailed resource savings are included after the source code. It is also easily adaptable as a macro since the source code needs to know only the name of &MERGVAR.

The key to the source code is the SET statement with the POINT= option. It allows us the ability to randomly access the SECOND data set, but at a small price. With the POINT= option, variables that mimic FIRST.&MERGVAR indicators from DATA FIRST must be created. Those variables allow the use of pointers to control which record from data set SECOND we are accessing. The three pointers know where we've been, where we are, and where we are going. A loop is created from where we've been to one record shy of where we are going at each record of the FIRST data set when the merge variable changes. It is then just a matter of running the loop at any subsequent record of the FIRST data set with that value of the merge variable to collect every combination.

The efficiency shown so far can be extended to normal merge situations. In limited testing this code has outperformed not only the MERGE statement but other alternative merge solutions. There is, however, one restriction to outperforming "Being Dynamic". The huge data set, which plays the role of the FIRST data set, must exist in the order of the merge variable.

Benefits

In a production environment the resource savings may be of greatest benefit, but in a large multiple match-merge the lack of intermediate data sets will be treasured. Although the testing has been limited to small data sets (less than 50000 records output), the findings of reduced resource utilization should apply up into the million plus record range.

This source code can also be expanded in many ways. The most obvious expansion is to use it as a macro in other programs, including programs where the intent is to do a multiple match-merge of a data set with itself. Only in a macro could an expansion of the number of &MERGVARs be accomplished. It is also probable that an expansion on the number of data sets randomly accessed (SET THIRD POINT=, etc.) could only occur within a macro.

The only expansion that is difficult is the inclusion of unmatched records. It is possible, but it is at this time choppy and not thoroughly tested.

Bibliography

Lessmann, Terry. "A Solution to Unique Match-Merge Situations." SAS Users Group International Conference Proceedings, 1990, pp. 910-915.

Stinson, Warren E. and Fisher Jr., Philip S. "Being Dynamic." SAS Users Group International Conference Proceedings, 1991, pp. 148-151.

Trademark Notice

*SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

Author

Bill Horman
A.T. Kearney, Inc.
222 S. Riverside Plaza
Chicago, IL 60606

Enhancing the Multiple Match-Merge Solution Source Code

```

* Comments follow the SAS statement ;

%LET MERGVAR=STCNTY;
    * This version can only use one merge variable ;

DATA SECOND;
    INFILE SECDATA;
    INPUT STCNTY $ ZIP $;

PROC SORT; BY &MERGVAR;

DATA FIRST;
    INFILE FIRDATA;
    INPUT STCNTY $ NAME $;

PROC SORT; BY &MERGVAR;

DATA ALL; SET FIRST;
    * BY statements are not allowed with the POINT= option;
    IF _N_=1 THEN HLDIT=&MERGVAR;
    * Create HLDIT variable that has the same length and ;
    * type as &MERGVAR. Without this statement, HLDIT ;
    * would be created numeric during the RETAIN statement;
    RETAIN OLDPTR NEWPTR PTR 1 HLDIT;
    DROP HLDIT CHKIT OLDPTR NEWPTR;
    * POINT= and NOBS= variables are not added to new data sets ;
    IF &MERGVAR NE HLDIT THEN DO;
    * We are now on a FIRST.&MERGVAR record ;
        HLDIT=&MERGVAR;
        PTR=NEWPTR;
        OLDPTR=NEWPTR;
        * Because of the change in &MERGVAR, the pointers are ;
        * reset beyond the last group of matched records ;
    END;
    DO UNTIL ((&MERGVAR LT CHKIT) OR (PTR GT NUM));
        SET SECOND(RENAME=(&MERGVAR=CHKIT)) POINT=PTR NOBS=NUM;
        IF &MERGVAR=CHKIT THEN DO;
            *** Subsetting IF statements and new variables go here ***;
            OUTPUT;
            ***;
            PTR=PTR+1;
            * Since we matched, increment PTR to check the next record ;
        END;
        IF &MERGVAR GT CHKIT THEN PTR=PTR+1;
        * SECOND's record does not match, so check it's next record ;
    END;
    * SECOND's value of &MERGVAR exceeds FIRST's ;
    * or we have reached the EOF of SECOND ;
    NEWPTR=PTR;
    * NEWPTR becomes the location to start searching ;
    * if &MERGVAR changes ;
    PTR=OLDPTR;
    * PTR is reset to the last value that could have matched ;

PROC SORT; BY &MERGVAR;

PROC PRINT;
    TITLE 'FINAL OUTPUT OF MERGED DATA SETS';

```

Enhancing the Multiple Match-Merge Solution Detailed Time Savings

FIRST Rec	SECOND Rec	FINAL Rec	Max Sets	Combin Ave	One Step Time	%MACMERG Time	% Diff	Net Diff
7200	3600	7200	2	1	160.80	271.00	40.66	110.20
16200	3600	16200	2	1	661.20	836.20	20.93	175.00
600	600	1200	2	2	25.00	63.00	60.32	38.00
1200	1200	2400	2	2	43.00	99.00	56.57	56.00
1800	1800	3600	2	2	61.80	129.00	52.09	67.20
3600	3600	7200	2	2	118.20	232.20	49.10	114.00
3600	3600	7200	2	2	118.20	232.20	49.10	114.00
7200	3600	14400	2	4	231.00	356.40	35.19	125.40
10800	3600	21600	2	6	342.00	588.00	41.84	246.00
14400	3600	28800	2	8	466.20	716.20	34.91	250.00
7200	3600	14400	2	4	231.00	356.40	35.19	125.40
10800	5400	32400	3	6	451.20	789.60	42.86	338.40
14400	7200	57600	4	8	745.20	1414.00	47.30	668.80
7200	7200	28800	4	4	378.00	811.80	53.44	433.80
14400	7200	57600	4	8	745.20	1414.00	47.30	668.80