Paper 101-26

Merging Disk Data sets To Large Unsorted Tape Data sets without a Sort Step, Using Indexed SAS Files

William E Benjamin Jr, Phoenix, AZ

ABSTRACT

At times external flags or data need to be added to large unsorted tape files. The data may come from different systems or time periods, but need to be added to be able to represent relationships not available alone. Tape files with several million records and nearly a thousand variables take a long time to sort, if they can be sorted. Indexed SAS files can be used to find a sparse set of records that require updating and apply data to the output records, by reading data from the indexed disk file without sorting the tape file. This is, of course, slower than a merge but it can usually be faster than a sort. In this way data from any disk based file can be added to the tape data set. This is a process available in Base SAS to all levels of skill.

INTRODUCTION

Often large unsorted tape files pose problems. They are big, single user, sequential, slow, and at times difficult to sort. Large files strain system limits and challenge you to become more creative than the system. This technique of merging data sets avoids some of the problems associated with large tape data sets.

BACKGROUND

Many systems still use tape files to archive data for long periods. Some legacy systems can output daily files with millions of large records. Using these files after they have been converted to SAS files with hundreds of variables can be expensive and time consuming.

Sorting files consume system resources in addition to the records on the file. Most sorts read all file records into memory, or onto disk reserves, as a part of the sort process before being able to rearrange the records, changing the order for the sorted output file. When the size of the tape file exceeds the amount of available disk resources for the sort routine the file becomes unsortable. Additionally, some files approaching the system limit sizes can be sorted, but require long sort execution times. Some can exceed several hours of system resources.

PROBLEM

There are many times that small amounts of data need to be merged onto records on large unsorted tape files, or a limited number of records need to be extracted from the file. One case is when time related data is created. For instance, in an inventory application, a storage bin location may change. Another case is when records are on a list the records need to be extracted for further processing and data needs to be added to the tape or updated. But, if the record is not on the list the new fields need to be added with default or null values. Most programmers are accustomed to merging sorted files. SAS software makes this process relatively simple, FOR SORTED FILES! But, large unsorted tape files may take several hours to read once, without including the time to sort the file and rewrite the tape.

SOLUTION

A simple one-pass method is needed to read and update the unsorted file. When a smaller disk based file needs to be processed against the large tape file, it is possible to be able to use a feature of Base SAS, called indexing, to apply updates to the tape file. Each record from the tape file is checked against records on the indexed SAS disk file and processed accordingly. Indexed SAS files can be read using a key found on the tape file and data from both files can be processed. The indexed read returns an error code that should be checked to determine if there is a record on both files. This provides a way determine whether or not to apply data from the disk file or apply default values.

STEP ONE – CREATE AN INDEXED FILE

An index can only be created for a disk file, but can be created in any of several methods. Two methods are the SAS Dataset procedure, and the SAS SQL procedure index options, but the one method I like the best is the INDEX= option of the SAS DATA step. One of the drawbacks of using files with millions of records is that every thing your program does is repeated millions of times. Therefore, you must be careful to use the fewest number of steps and the fastest types of commands, or the job costs begin to soar. The object of using SAS indexed files, is to use the speed of the index's binary search to retrieve records from the disk file, to search for matches. The following example shows how to create an index when the file is created.

```
Data SASFile (index=(Part_no=(Part_no)));

      Infile cards;

      Input @01      Part_no       $char4.
            @06      newbin        $char2.
            @09      morevars      $char30.
      ;

*Part_no Bin_no   Cost      Description;
*2345678901234567890123456789012345678901234567;
cards;
0001 35 any other var data 1
0003 43 any other var data 2
0006 10 any other var data 3
;;;;
Run;
proc print ;

OBS    PART_NO    NEWBIN          MOREVARS

1      0001        35      any other var data 1
2      0003        43      any other var data 2
3      0006        10      any other var data 3
```

The file SASFile is created with an index called Part_no. In this case the file has three records and three variables.

TEST DATA FILE:

The following is code to create a small disk file to illustrate the technique. This process is most effective on the large unsortable tape files, but will work on any file.

```
Data partfile ;

      Infile cards;
      Input @01      Part_no            $char4.
            @11      bin_no             $char2.
            @18      price              5.2
            @29      desc               $char30.
      ;

*Part_no Bin_no   Cost      Description;
*234567890123456789012345678901234567890123456 7;
cards;
0001      15      5.43      Hammer
0012      17      7.17      Screw Driver no2
0003      23      8.12      Screw Driver no4
0004      42      9.43      Screw Driver no9
0005      15      7.43      Rip Saw
0006      36      2.50      Flashlight
0017      93      6.75      Lantern no3
0008      47      8.31      Shovel
0009      53      9.77      Rake
0010      60      5.33      Pick
0011      16      8.78      Axe
0012      33      9.89      Drop Cloth
;;;;
Run;

proc print ;


OBS  PART_NO  BIN_NO    PRICE          DESC

1    0001        15      5.43    Hammer
2    0012        17      7.17    Screw Driver  no2
3    0003        23      8.12    Screw Driver  no4
4    0004        42      9.43    Screw Driver  no9
5    0005        15      7.43    Rip Saw
6    0006        36      2.50    Flashlight
7    0017        93      6.75    Lantern no3
8    0008        47      8.31    Shovel
9    0009        53      9.77    Rake
10   0010        60      5.33    Pick
11   0011        16      8.78    Axe
12   0012        33      9.89    Drop Cloth
```

The file created by the preceding code is a file called “Partfile”. The file has no index and has three variables. But, it could have several hundred variables. In this case, the records are not in sorted order by the variable Part_no.

SLOW SOLUTION

Note that the index key can be part numbers or customer numbers or any record identifying information. These values are values that you may expect to find on the tape file, and may occur more than one time or in any order, on the tape file. The other variables in the indexed file may or may not be part of the updates required for the output file. This data may be actual values to add to the output file or codes to trigger internal processing or formatting. Here you can get creative. In a process such as this, you are expecting the large percentage of records to NOT be found by the index, and should code accordingly. The following code will work to merge the values from the indexed file with the data listed at the above.

```
Data Merged;
        Set partfile;

        Set SASFile KEY=Part_no/UNIQUE;

      If (_iorc_  eq 0) then do;
                Bin_No = newbin;
                output  Merged;
         end;

      If (_iorc_ ne 0) then output Merged;

      _error_ = 0;
      _iorc_ = 0;

      Drop newbin morevars;
Run;

proc print ;

OBS  PART_NO  BIN_NO    PRICE    DESC
1    0001      35        5.43    Hammer
2    0012      17        7.17    Screw Driver no2
3    0003      43        8.12    Screw Driver no4
4    0004      42        9.43    Screw Driver no9
5    0005      15        7.43    Rip Saw
6    0006      10        2.50    Flashlight
7    0017      93        6.75    Lantern no3
8    0008      47        8.31    Shovel
9    0009      53        9.77    Rake
10   0010      60        5.33    Pick
11   0011      16        8.78    Axe
12   0012      33        9.89    Drop Cloth
```

NOTES

The ‘/UNIQUE’ option on the Set statement causes the index processing to start at the beginning of the index for every input record tested, instead of continuing with the next record in the indexed file. This code will work, but not very fast, for the following reasons:

1. There will be two or more SAS files open as input files. The output Program Data Vector will, by default, contain ALL variables in both files. One optimization is to only include on the set statement, the needed variables from the indexed disk file. This will limit input to the data step and speed execution.
2. The code above has two ‘IF’ statements, each of which tests the value of the Input / Output Return Code (a system automatic variable called _iorc_). Since the test is either to match a value or not match a value, then only an ‘If….then….Else’ construct is required to test the values.
3. Within the ‘If … Then … Else’ the test case that will trigger most should be first , i.e. after the ‘Then’ clause, to speed processing.
4. Every instruction has system overhead associated with the instruction. So by combining tasks within fewer instructions the system overhead can be reduced. Even a small reduction can add up when done millions of times.
5. By testing for a specific value of the _iorc_ variable the user assumes that the value will always have the same meaning, This may not always be the case. The recommended method is to test for the SAS mnemonic using the ‘%SYSRC’ system macro. The value _SOK is for a successful completion of a task, and the proper syntax for the call is %SYSRC(_SOK).

FASTER SOLUTION

Note that the following code incorporates the changes above to enhance the program and speed of execution. Even though it may not be noticeable for small files like these, the speed of execution should be faster for the same tasks using the second method when millions of records are processed.

```
Data Merged2;

          Set partfile;

          Set SASFile
                (keep=Part_no newbin)
                KEY=Part_no/UNIQUE;

          If  (_iorc_ ne %sysrc(_sok))
               Then Do;
                  * Bin_no ok on From.Tape;
                  Output Merged2;
                  _iorc_ = 0;
                  _error_ = 0;
               End;
               Else Do;
                    Bin_no = newbin;
                    Output Merged2;
               End;
      Drop newbin;
Run;

proc print ;

OBS  PART_NO  BIN_NO    PRICE    DESC
1    0001      35       5.43    Hammer
2    0012      17       7.17    Screw Driver no2
3    0003      43       8.12    Screw Driver no4
4    0004      42       9.43    Screw Driver no9
5    0005      15       7.43    Rip Saw
6    0006      10       2.50    Flashlight
7    0017      93       6.75    Lantern no3
8    0008      47       8.31    Shovel
9    0009      53       9.77    Rake
10   0010      60       5.33    Pick
11   0011      16       8.78    Axe
12   0012      33       9.89    Drop Cloth
```

While the placement of, (or need for) the ‘Output’ statements within DO … END blocks, are for illustrative purposes, this construct allows other code such as setting a default value for new variables. Also, the only value of _iorc_ that is being tested positively, is if the record was found in the indexed file. This code considers no other errors to be meaningful. If your code needs to test for other conditions be sure to include them as needed. Also note that the system automatic variables _iorc_ and _error_ need to be reset to zero before the next record is read from the indexed file.

CONCLUSION

The main idea is that the procedure described here will merge two files when one file is not, or cannot, be sorted. The use of indexed files can be a useful tool that will allow the resourceful programmer to process an unsorted tape file and be able to merge data to the file.

REFERENCES

SAS Institute Inc. (1990), SAS Language: Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), SAS Macro Language: Reference, First Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

William E Benjamin Jr
1531 W Michigan Ave
Phoenix AZ, 85023
E-mail: WmEBenjaminJr3@juno.com