Paper 064-2009

Proc Format, a Speedy Alternative to Sort Sort Merge

Jenine Milum, Wachovia a Wells Fargo Company, Charlotte, NC

ABSTRACT

Many users of SAS System software, especially those working with large datasets, are often confronted with processing time challenges. How can one reduce the amount of CPU required to retrieve specific data? In this paper, an "outside the box" approach using a matching method utilizing Proc Format replaces the CPU heavy Sort/Sort/Merge. It is ideal for situations when a key from one file is needed to extract data from another file. It is more apparently useful when at least one of the files is quite large. This method has been proven time and again to decrease CPU by 70%-80% and is effective on all platforms utilizing Base SAS.

INTRODUCTION

Users of SAS System software are often confronted with the challenge of retrieving specific information from very large datasets. How can the desired information be extracted effectively while reducing the amount of time required retrieving the data? This paper attempts to use a matching method utilizing Proc Format to replace the CPU heavy Sort/Sort/Merge. It is ideal for situations when a key from one file is needed to extract data from another file. It is most useful when at least one of the files is quite large. This method has proven to reduce CPU by 70% - 80%.

TRADITIONAL SORT/SORT/MERGE

To appreciate the effectiveness of the Proc Format method presented in this paper, it helps to discuss the sort/sort/merge which it's replacing. It also gives an excellent baseline for desired output and CPU times.

```
proc sort data=largefile;
  by keyvar;
run;

proc sort data=keyfile(keep=keyvar) nodupkey;
  by keyvar;
run;

data match; merge largefile(in=large)
                  keyfile(in=key);
  by keyvar;
  if key and large;
run;
```

Sort/Sort/Merge is used here when key values from one file are needed to extract records from another file containing the same key, or BY variable. Suppose you have a small file (keyfile) that contains a list of Social Security numbers (the key) of individuals in a department at your company. These Social Security numbers need to be matched to another file (largefile) representing the whole company to extract additional information on each individual. You want only the records that can be found in both files.

For a clean merge, both data sets have to be sorted. Note that both files have to be processed twice, once in the sort and once in the merge step each. If either or both of the files are large, CPU time can be considerable.

There are other basic concerns when running any Sort/Sort/Merge. Are there any duplicate records in either dataset being used in the merge? Is the merge logic being handled properly so that required data from matching data sets will not be accidentally overlaid? These are no longer issues with the Proc Format Method.

UNDERSTANDING FORMATS

Before utilizing the Proc Format method this paper focuses on, a basic understanding of the "parts" of a format is necessary. Take a basic format, we'll call $CTYST., that links Cities and States. Using the "fmtlib" option of Proc Format will reveal the metadata surrounding this existing format.

```
proc format library=work fmtlib;
  select $key;
run;
```

The result shows several of the critical pieces we'll be using to create a format out of one of the KEYFILE dataset we use in the above Sort/Merge. The value of the FORMAT is $CTYST. START is the value of the variable used to match, in this case the city name. Variable END is not necessary since we are not working with numeric ranges. LABEL represents the value START which will be assigned in the instance of a match, in this case, the state.

FORMAT NAME: $CTYST LENGTH: 1 NUMBER OF VALUES: 4
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 1 FUZZ: 0

START	END	LABEL (VER. 8.2 03JAN2005:15:45:32)
Atlanta	Atlanta	Georgia
Boise	Boise	Idaho
Columbus	Columbus	Ohio
Portsmouth	Portsmouth	Virginia

Working with these 3 metadata items in a standard format, we can create a format from the KEYFILE dataset that we can utilize for matching purposes. When a record has a match to the value in START, the value in LABEL is linked to it for additional use.

Understanding these basics about formatting allows for a slick trick utilizing the selection ability (I'm not sure what "keying ability" really means here – is there a more straightforward term you can use?) of Proc Format.

SPEEDY METHOD USING PROC FORMAT

```
data key; set keyfile(keep=keyvar);
  /* These variables translate to the FORMAT values in the metadata */
  fmtname = '$key';
  label   = '*';
  rename keyvar = start;
run;

proc sort data=key nodupkey;
  by start;
run;

proc format cntlin=key;
run;

data match; set largefile;
  if put(keyvar,$key.) = '*';
run;
```

In the first data step, a SAS data set needs to be created from the input file with the required valid format variables names LABEL, START, and FMTNAME. Doing so prepares the information to be turned into a format.

- START is set to the variable assigned as key.
- FMTNAME becomes the format name to be referenced later. (formatting naming conventions do not allow for a FMTNAME to end in a numeric value)
- LABEL becomes the symbol that the desired key values are associated with. In this case asterisk (*).

The LABEL is assigned an asterisk (*) as the formatted value, which acts as the hook into the bigger file. Using this simple line of code in any data step within your program will select those records that match the formatted values.

```
if put(key,$key.)='*';
```

The variable name KEY can be any name in a dataset as long as the values in the variable will have matches to the values in the format. Yes, you can use this line of code to select these matches anywhere in your program.

It is very important that the symbol assigned to LABEL will never have an occurrence in the key character string of the master file. Otherwise, an unwanted match will result. The asterisk (*) symbol works in most situations as it is rarely contained in character variable values.

This pre-format data set needs to be sorted and any duplicates eliminated. SAS formats will not allow duplicate values. Running Proc Format with duplicate values will create the SAS system error `"ERROR: This range is repeated, or values overlap:"` and processing will be halted. (Note: there are special situations in which you can force SAS to accept overlapping format values, but this isn't the right application for it.)

To actually create a working format, execute PROC FORMAT using the CNTLIN=option using the sorted dataset Key as the input control dataset. It converts the data stored in the pre-format SAS data set (Key) to a SAS format and adds it to the format catalog in the library *work*.

Once a format is created, it can be used anywhere else in the program for selecting matches to the key. In essence, the assignment statement gives the value of LABEL, in this case asterisk (*), to a matching key. This in turn can be used for additional coding. In the above scripts, it is used to select records matching the key.

Using the PROC FORMAT method, only one file is processed twice,
It's the smaller file and only one variable is needed from it.
The biggest savings in CPU is that the sorting
of the large file is NOT required.

COMPARING RESULTS

For the non-believers, the results below show several different methods for comparison. The same input files were used in all examples. All tests were run using the same UNIX Sun platform. The key file had 730 observations. The larger file had 1.5 million records. For these examples, the following CPU times were recorded.

Formatted Table

	CPU	% Time Reduction
Sort/Merge	42.823 seconds	-
Indexing	38.000 seconds	73%
Proc Format Method	10.267 seconds	76%

ADDITIONAL USES

MERGING USING MORE THAN 1 VARIABLE

Frequently, merging requires more than one variable. While still using the Proc Format methodology, two additional solutions provide the desired results. One method involves concatenating the matching variables. The 2nd method suggests creating additional formats or key variables for matching. The following approaches allow you to select records by matching Cities and State.

THE VARIABLE CONCATENATION FORMAT APPRAOCH

Concatenating the variables you would normally sort by and then merge by creates just one unique variable by which a format can be created. Then this single variable can be used to create a format by which is used to select records. Taking the value of CITY as *Atlanta* and the value of STATE as *Georgia*, the value of the new variable would be *AtlantaGeorgia*.

```
data key;
  set keyfile(keep=city state);
  start = trim(city)||trim(state);
  fmtname = '$ctyst';
  label = '*';
run;

proc sort data=key nodupkey;
  by start;
run;

proc format cntlin=key;
```

```
run;

data matchfile;
  set largefile;
  if put(trim(city)||trim(state),$ctyst.) = '*';
run;
```

THE MULITIPLE FORMAT APPRAOCH

You can use one data step to create multiple pre formatted datasets. Notice that START and FMTNAME are assigned, then output. The 2nd format variable name is being reset, and then output. You can really do this up to as many key variables as necessary. You will need one Format SAS procedure to create each of the key formats. Once the formats are created, simply use them in any future data step to identify matching records.

```
data keycity keystate;
  set keyfile(keep=City State);
  label = '*';
  start = City;
  fmtname = '$city';
  output keycity;
  start = State;
  fmtname = '$state';
  output keystate;
run;

proc sort data=keycity nodupkey;
  by start;
run;

proc sort data=keystate nodupkey;
  by start;
run;

proc format cntlin=keycity;
run;

proc format cntlin=keystate; run;

data matchfile; set largefile;
  if put(city,$city.)   = '*' and
     put(state,$state.) = '*';
run;
```

ADDITIONAL TIPS AND SUGGESTIONS

- Make the LABEL more meaningful.

  (in the first data step)

  ```
  label = 'Match';
  ```

- Use multiple formats to include or exclude records

  ```
  data newfile; set bigfile;
    if put(key1,$keyone.) = '*' and put(key2,$keytwo.) ne '*';
  run;
  ```

- Create one format with multiple values.

  (in the first data step)

  ```
  if key = 'one' then label = 'one';
  if key = 'two' then label = 'two';
  ```

(then in the last data step)

```
data newfileone newfiletwo; set bigfile;
  if put(key,$key.) = 'one' then output newfileone; else
  if put(key,$key.) = 'two' then output newfiletwo; else
  delete;
run;
```

- Utilize the Format in a WHERE clause.

```
data newfile ;
  merge bigfileone(where=(put(key,$key.) = 'one'))
        bigfiletwo(where=(put(key,$key.) = 'two'));
  by newvar ;
run ;
```

REFERENCES

Rick Aster and Rhena Seidman, Professional SAS Programming Secrets

ACKNOWLEDGMENTS

I would like to thank Carla Mast for her excellent editing skills.

CONTACT INFORMATION

Jenine Milum
Wachovia a Wells Fargo Company
Work Phone: (704) 383-7377
Email: Jenine.Milum@Wachovia.com