Paper 86-27

Powerful Techniques For Data Processing Using Formats

Tim Muir, Compaq Computer Corporation, Littleton, MA

Abstract

This paper demonstrates the creation formats and techniques to use them when processing large files.

Introduction

Switching industries from clinical trials where small data sets (<1K observations) are typically used to database marketing, in which observations run into the millions is quite an experience. The amount of time it takes to handle these large datasets can seem like forever, if you're not careful. Reading in millions of observations from each data set, sorting, then merging could take forever at times. Sure, I could go down the hall and pick up a brand new top-of-the-line WildFire GS360 AlphaServer from the Compaq company store, but that would be too easy. This paper will show where performance gains can be obtained using formats in place of sorting/merging for sub setting purposes and creating new variables, and how easily these formats can be created.

Approaches

It occurred that the typical approach to pulling specific contact records, or creating data sets for data mining, was to subset records in one data set meeting criteria, sorting, then ultimately match merging with the contact file. This frequently required several sorts and match merges along the way. The amount of records read from data sets would first have to be reduced. There was no sense reading the complete file, millions of records, every time the file was accessed. "The WHERE statement is, according to the SAS LANGUAGE REFERENCE GUIDE, a more efficient technique… because the WHERE statement selects observations before they are brought into the program data vector".

Dynamically creating the comparisons using macro variables worked ONLY when there were fewer than some 32,000 comparisons. This would not work. There had to be a method that could replace the use of DATA step MERGES to subset data or merge records for a larger subset than 32,000.

Maybe formats will work…

At wits end, formats were given a shot, but not without fearing again reaching another limitation of some type. To test this out, one of the datasets previously used for match merging was used to create a temporary CNTLIN dataset. This datasets will be used to create a format, which can then in turn be used in a WHERE statement to subset a large data set. To create a dataset that can be loaded in a format, at minimum, variables FMTNAME, START, TYPE, and LABEL are required. Using an arbitrary format name, and a variable previously used for match merging as the START variable, All that remained was the label value, to which all would be set to 'Y'.

Let's do it!

You may have been lost by that last paragraph so I'll tell ya what. I'll show you some code for creating the formats, and show you examples of how to use the method.

Scenario

Before jumping into the code, lets define specifically what we want to accomplish. Let’s say we want to identify customers who have an AlphaServer model 4100® in the Southeast region, so we can send them information on possible hardware upgrade paths or newer models.

Code for creating a format

In the step below, we keep records For customers with AlphaServer 4100®. Variables FMTNAME, START, TYPE, and LABEL are created.

```
*--- Create CNTLIN dataset ---;
data a4100(keep=fmtname type start
                    label hlo);
  set customer.parts;
  where model=’AlphaServer 4100’;
  fmtname=’keep’;
  start=customer_id;
  label=’Y’;
  if eof then do;
    hlo='O';
    label="N";
    output;
  end;
run;
```

At this point we use a PROC FORMAT statement specifying the a4100 data set as the CNTLIN data set.

```
*--- Load dataset into format ---;
proc format library=work
              cntlin=a4100;
run;
```

Following execution of the PROC FORMAT code we will have a format named $keep. Putting a customer_id through this format will result in either a
Y or an N. We will be using it to pull records with a Y, customers who have a 4100.

In data step FUNC43 below, we create a file containing specific types of contacts, say someone from the purchasing department. We will create a format to use as an additional sub setting criteria.

```
*--- dataset containing contact ---;
data func43(keep=customer_id);
  set y;
  where (source_expire_date eq . ) &
           (functional_title_code_1 eq '43');
run;
```

We will use the same approach to load this data set into a format, but since this technique will be used repeatedly, a macro was created to handle the work. It would be to much work to add (type) code each time you wanted to make a format, unless the format required more customization. Generally we’re going to use existing values in data sets to create formats.

```
*---load agents into a format ---;
%makefmt(dsin=func43,
              fmtname=purchase,
              start=customer_id,
              lv=Y,
             other=N);
```

The macro takes the following parameters:

DSIN – name of the input data set.
FMTNAME – name of the format
START – name of the variable to use
  for the start value.
LV – Label value, if a LABEL variable
  is not specified.
LABEL – name of the variable to use
  for the label value.
OTHER – Label value for any other start
  Values encountered.

Creating a typical format

The last piece of sub setting code we want to create is a format for identifying Southeast region contacts. To do this, we'll just map zip codes to regions using permanent data set REGIONS.

```
*--- load region into a format ---*;
%makefmt(dsin=regions,
                fmtname=zip2reg,
                start=zip,
                label=region,
                other=UNK);
```

Once the above macro call is executed, we will have all the pieces we need to reach our objective without having used sorting/merging. All that remains to be done is to execute the following data step, using the formats to subset the contacts data set.

```
*--- create final data set ---*;
data contacts;
  set contacts.contacts;
  where put(customer_id,$keep.)='Y'
    and  put(customer_id,$purchase.)='Y'
    and  put(zip, $zip2reg.)='Southeast';
run;
```

When executed, only records meeting the where conditions are read into the program data vector (PDV), reducing execution time and direct I/O.

Macro Code

```
%macro makefmt ( dslib=work, dsin=,
   fmtname=, start=, label=, other=, lv=);

  *--- build format data set ---*;
  data cntlin(keep=fmtname type start
                         label fuzz hlo);
    length start $&slen label $&max;
    set &dslib..&dsin end=eof;
    retain fmtname "&fmtname" type
            "&type" fuzz 0;
      start=&start;
    %if %nrbquote(&lv) ne %then %do;
      label="&lv";
    %end;
    %else %do;
      label=&label;
    %end;
    output;
    *--- if value for other, compile ---*;
    %if %nrbquote(&other) ne %then %do;
      if eof then do;
        hlo='O';
        label="&other";
        output;
      end;
    %end;
  run;

  *--- load format data set;
  proc format cntlin=cntlin;
  run;

  *--- clean up ---*;
  proc datasets nolist;
    delete cntlin;
  quit;
%mend;
```

Conclusion

This paper describes a quick and efficient method for creating SAS formats from SAS data sets and using those formats to reduce i/o processing time on larger files.

Contact Information

Tim Muir
Compaq Computer Corporation
153 Taylor Street TAY2/B6
Littleton MA 01460
978-264-5137
timothy.muir@compaq.com