Paper 95-26

How Many Observations Are In My Data Set?

Jack Hamilton, First Health, West Sacramento, California

ABSTRACT

This paper presents a macro which returns the number of observations in a SAS data set or view, with an optional WHERE clause, and an additional macro which indicates only whether the data set or view is empty.

INTRODUCTION

It sometimes happens that a SAS program needs to know how many observations are in a SAS data set. The traditional, and fastest, method is to use the NOBS= option on a SET statement, but this method does not always return the correct number. This paper describes some of the problems with the NOBS= solution, and presents a macro as an alternative solution.

NOBS= WORKS WITH AN ORDINARY DATA SET

Suppose you have an ordinary data set, one that you've just created:

```
data row1col1;
   a = 12;
   output;
run;
```

If you use the NOBS= option on a SET statement, you can find the number of observations in the data set:

```
data _null_;
   put nobs=;
   stop;
   set row1col1 nobs=nobs;
run;
```

prints

```
NOBS=1
```

THE PROBLEM

The problem is that the NOBS= option doesn't produce correct results for all types of data sets.

NOBS= DOESN'T WORK WITH AN EDITED DATA SET

A data set which has been edited in place may not return the correct number:

```
data delobs;
   a = 12;
   output;
run;
data delobs;
   modify delobs;
   remove;
run;
data _null_;
   put nobs=;
   stop;
   set delobs nobs=nobs;
run;
```

will print NOBS=0 on OpenVMS (SAS 6.12), which is correct, but NOBS=1 on Windows (SAS 6.12) and Unix (SAS 8.1), which is incorrect.

NOBS= DOESN'T WORK WITH A DATA STEP VIEW

The NOBS= option also doesn't work with a data step view:

```
data datav1 / view=datav1;
   set row1col1;
run;
data _null_;
   put nobs=;
   stop;
   set datav1 nobs=nobs;
run;
```

prints

```
NOBS=2147483647
```

Plausible, if you happen to have a really large dataset, but incorrect.

NOBS= DOESN'T WORK IN VARIOUS OTHER CASES

NOBS= also doesn't work with transport data sets, SQL views (including database passthrough views), and in some other cases.

SO WHAT?

Use of the NOBS= option is OK if you just created the data set earlier in the same program, or if you have some other way of knowing for certain how the data set was created and possibly modified..

On the other hand, if you don't know how the data set was created or how it might have been manipulated, it's not safe to use NOBS=. In particular, if you're writing a general purpose program or macro that might be used by anyone on an arbitrary data set, you should not use NOBS= to count observations.

This problem was not obvious in the past, when views and transport data sets weren't common. But these days, a data set might come from anywhere – you might be dealing with a "real" data set, or a view, or an external database, or a real data set on a different platform.

A SOLUTION

One solution to this problem (the solution I present here) is to create a general-purpose macro which returns the number of observations in a data set, regardless of how it was created. The macros uses the data set information functions, new in late releases of SAS version 6, to provide the information needed.

The steps are:

1. Find out whether SAS knows how many observations there are,
2. If it does know, there's a function which returns the correct count. Use it and you're done.
3. If SAS doesn't know how many observations there are, iterate through the data set and count.

An advantage of this approach is that it supports where clauses, which NOBS= does not. Another advantage is that it is implemented entirely in the macro language, and will not create a step boundary in the calling program. The primary disadvantage is that it can be slow for large datasets.

Highlights of the code are shown below, and the complete macro is given at the end of the paper. A special case macro, MTANYOBS, is also shown; it checks whether there are any observations in the dataset, without counting them.

MAKE SURE THE DATA SET EXISTS

Use the OPEN function to make sure the data set exists; if it doesn't, or can't be opened, return a missing value:

```
%let DSID = %sysfunc(open(&DATA., IS));
%if &DSID = 0 %then
   %do;
   %put %sysfunc(sysmsg());
   %let counted = .;
   %goto mexit;
   %end;
```

The OPEN function returns an internal pointer to the data set if the open succeeded, or 0 if the open failed.

In the case of failure, use the SYSMSG function to get explanatory text, set the return value to missing, and go to the exit.

If the open succeeded, the dataset pointer is stored in DSID.

DOES SAS KNOW THE ANSWER WITHOUT COUNTING?

If SAS knows how many observations are in the data set, and if there's no WHERE clause, you can get the answer directly:

```
%let anobs  = %sysfunc(attrn(&DSID, ANOBS));
%let whstmt = %sysfunc(attrn(&DSID, WHSTMT));

%if &anobs = 1 & &whstmt = 0 %then
         %do;
         %let counted =
              %sysfunc(attrn(&DSID, NLOBS));
         %end;
```

The ATTRN function returns the value of a numeric attribute of a data set. The ANOBS attribute is 1 if SAS knows the number of observations in the data set specified by DSID, and 0 if it doesn't. The WHSTMT attribute is 0 if no where clauses are active, and non-zero if there are active where clauses.

DOES SAS HAVE TO ITERATE?

If SAS doesn't know the number of observations, or if you're using a WHERE clause, you can obtain the answer by iterating through the data set. This can be expensive, but it is your only reliable choice.

The code is simple; just loop and increment a counter:

```
%let counted = 0;
%do %while (%sysfunc(fetch(&DSID)) = 0);
   %let counted = %eval(&counted. + 1);
%end;
```

The Fetch function obtains the next observation in the referenced dataset.

EXAMPLES OF USAGE

The MTCNTOBS macro returns a number (as a string), and can be used anywhere a number might occur.

USAGE WITHOUT A WHERE CLAUSE

```
put "There are %MTCNTOBS(data=testdata) rows
in your table.";
```

USAGE WITH A WHERE CLAUSE

```
data _null_;
%let obscount = %MTCNTOBS(data=skiing
            (where=(xc='Y' and open ='Y')));
select (&obscount);
   when (0)
      put 'No XC resorts are open.';
   when (1)
      put '1 XC resort is open.';
   otherwise
   put "&OBSCOUNT. XC resorts are open.";
end;
run;
```

PRINTING A "NO OBSERVATIONS" PAGE

It's often desired to print out a special page if there are no observations in a data set (if there are no records, PROC PRINT and other reporting procedures will print nothing). The MTANYOBS macro can be used to do this:

```
title 'Listing of Errors';
proc print data=errors;
run;

data _null_;
   if %MTANYOBS(data=errors) = 0 then
      do;
      file print;
      put "No errors were found.";
      end;
   stop;
run;
```

If there were errors, the PROC PRINT will execute but the PUT statement in the data null will not. If there were no errors, the PUT will execute but the PROC PRINT will not.

Two things to note here:

1. The same title statements will be used in either case.

2. This special case can be made simpler; you don't need to iterate through observations 2 through 10,000,000 after you've found the first observation. A revised macro, MTANYOBS, is found at the end of this paper.

Q & A

WHY DOES ITERATION TAKE SO LONG?

It appears to be a problem with the macro language rather than the FETCH function. The increment statement, which requires converting a string to a number, adding, and converting back to a string, is slow. Running the FETCH function plus a counter against a million record dataset takes 671 seconds on a test machine (Windows NT). A simple loop counting from 1 to 1,000,000 takes 452 seconds on the same machine. In other words, two-thirds of the time is spent on simple integer arithmetic.

WHAT'S AN ALTERNATIVE?

One alternative is to do the counting in a data step. This is more work to program, but is faster to execute. A data step which uses the same counting technique as the MTCNTOBS macro takes 16 seconds, as opposed to 671 seconds in the macro. If you're dealing with large datasets, it might be worth your time to avoid using macros, even at the cost of creating additional step boundaries.

Here's the code used in a data step:

```
16   data _null_;
17      dsid = open('testdata
                    (where=(class=3))', 'is');
18      do while (fetch(dsid, 'noset') = 0);
19         i + 1;
20      end;
21      put i=;
22      rc = close(dsid);
23      stop;
24   *****; run;

i=249974
NOTE: DATA statement used:
      real time           15.93 seconds
```

It would not be difficult to "macrotize" this code so that it could be easily included in the middle of a data step; the primary difficulty would be ensuring that there is no variable name collision. It would also be easy to write a macro which creates a stand-alone data step which saves its results into a macro variable. These macros are left as an exercise for the reader (there's not room for them in this paper).

WHY NOT USE THE SET STATEMENT IN A DATA STEP?

Another iterative solution in the data step might be:

```
data _null_;
   do while (not nomore);
      set testdata (where=(class=3))
          end=nomore;
      i + 1;
   end;
   put i=;
   stop;
*****; run;
```

Unfortunately, this prints

```
i=1000000
```

The END= option doesn't seem to take the WHERE clause into account.

There are two other problems with the SET solution:

1. It causes data movement, so you might see a speed decrease for a data set with many variables, or large character variables.

2. It brings variables into the data step, and those variables might interfere with something else you're doing.

COULD I USE SQL?

PROC SQL provides another alternative for counting observations. In theory, it could be faster than other methods if the WHERE clauses is on indexed variables.

Here's sample code:

```
proc sql noprint;
   select  count(*)
   into    :OBSCOUNT
   from    testdata
   where   class = 3;
quit;
%put Count=&OBSCOUNT.;
```

WHY NOT USE THE NLOBSF DATA SET ATTRIBUTE TO OBTAIN THE COUNT?

The documentation for the NLOBSF option says that NLOBSF

> specifies the number of logical observations (those not marked for deletion) by forcing a read of each observation and taking the FIRSTOBS and OBS system options, and the WHERE clauses into account. Tip: Passing NLOBSF to ATTRN requires the engine to read every observation from the data set that matches the WHERE clause. Based on the file type and size, this can be a time-consuming process.

That sounds like what we want, without the overhead of counting in a macro.

Unfortunately, this option is not suitable, for two reasons:

1. It's not available in SAS version 6, and

2. It doesn't return the correct answer.

Here's an example:

```
97   %macro check;
98
99      %let dsid =
%sysfunc(open(sasuser.iris, IS));
100     %if &DSID = 0 %then
101        %put %sysfunc(sysmsg());
102
103     %let nlobs = %sysfunc(attrn(&dsid,
NLOBS));
104     %put nlobs=&nlobs.;
105     %let nlobsf = %sysfunc(attrn(&dsid,
NLOBSF));
106     %put nlobsf=&nlobsf.;
107
108     %let rc = %sysfunc(close(&dsid));
109
110  %mend;
111
112  %check;
nlobs=150
nlobsf=-1
```

OBTAINING A COPY OF THE MACROS

The complete text of the macros is shown at the end of this paper. I recommend that you type it in yourself as an aid to understanding, but it may also be downloaded from one of these sources:

```
<www.libname.com>
<www.sconsig.com>
<www.excursive.com/sas/sas.html>
```

An updated version of this paper may be available after SUGI.

CONTACT INFORMATION

Jack Hamilton
First Health
West Sacramento, California 95605 USA
(916) 374-3833
jackhamilton@firsthealth.com

TEXT OF THE MTCNTOBS MACRO

```
%macro MTCNTOBS(data=_last_);

   %* This macro returns the number of observations in a data set,
      or . if the data set does not exist or cannot be opened.

      - It first opens the data set.  An error message is returned
        and processing stops if the dataset cannot be opened.
      - It next checks the values of the data set attributes
        ANOBS (does SAS know how many observations there are?) and
        WHSTMT (is a where statement in effect?).
      - If SAS knows the number of observations and there is no
        where clause, the value of the data set attribute NLOBS
        (number of logical observations) is returned.
      - If SAS does not know the number of observations (perhaps
        this is a view or transport data set) or if a where clause
        is in effect, the macro iterates through the data set
        in order to count the number of observations.

      The value returned is a whole number if the data set exists,
      or a period (the default missing value) if the data set
      cannot be opened.

      This macro requires the data set information functions,
      which are available in SAS version 6.09 and greater.  ;

   %* By Jack Hamilton, First Health, January 2001.  ;

   %local dsid anobs whstmt counted rc;
   %let DSID = %sysfunc(open(&DATA., IS));
   %if &DSID = 0 %then
      %do;
      %put %sysfunc(sysmsg());
      %let counted = .;
      %goto mexit;
      %end;
   %else
      %do;
      %let anobs  = %sysfunc(attrn(&DSID, ANOBS));
      %let whstmt = %sysfunc(attrn(&DSID, WHSTMT));
      %end;

   %if &anobs = 1 & &whstmt = 0 %then
      %let counted = %sysfunc(attrn(&DSID, NLOBS));
   %else
      %do;
      %if %sysfunc(getoption(msglevel)) = I %then
         %put INFO: Observations in "&DATA." must be counted by iteration.;
      %let counted = 0;
      %do %while (%sysfunc(fetch(&DSID)) = 0);
         %let counted = %eval(&counted. + 1);
      %end;
      %end;

   %let rc = %sysfunc(close(&DSID));

%MEXIT:
&COUNTED.
%mend MTCNTOBS;
```

TEXT OF THE MTANYOBS MACRO

```
%macro mtanyobs(data=_last_);

   %* This macro returns 1 if there are any observations in a data set,
      0 if the data set is empty, or . if the data set does not exist or
      cannot be opened.

      - The macro first opens the data set.  An error message is displayed
        and processing stops if the dataset cannot be opened.
      - It next checks the values of the data set attributes
        ANOBS (does SAS know how many observations there are?) and
        WHSTMT (is a where statement in effect?).
      - If SAS knows the number of observations and there is no
        where clause, the data set attribute ANY is used to determine
        whether there are any observations.
      - If SAS does not know the number of observations (a view or transport
        data set) or if a where clause is in effect, the macro tries to read
        the first observation. If a record is found, the macro returns 1,
        otherwise it returns 0.

      This macro requires the data set information functions,
      which are available in SAS version 6.09 and greater.  ;

   %* By Jack Hamilton, First Health, January 2001.  ;

   %local dsid anobs whstmt hasobs rc;
   %let DSID = %sysfunc(open(&DATA., IS));
   %if &DSID = 0 %then
      %do;
      %put %sysfunc(sysmsg());
      %let hasobs = .;
      %goto mexit;
      %end;
   %else
      %do;
      %let anobs  = %sysfunc(attrn(&DSID, ANOBS));
      %let whstmt = %sysfunc(attrn(&DSID, WHSTMT));
      %end;

   %if &anobs = 1 & &whstmt = 0 %then
      %do;
      %let hasobs = %sysfunc(attrn(&DSID, ANY));
      %if &hasobs = -1 %then
         %let hasobs = 0;
      %end;
   %else
      %do;
      %if %sysfunc(getoption(msglevel)) = I %then
         %put INFO: First observation in "&DATA." must be fetched.;
      %let hasobs = 0;
      %if %sysfunc(fetch(&DSID)) = 0 %then
         %let hasobs = 1;
      %end;

   %let rc = %sysfunc(close(&DSID));

%MEXIT:
&HASOBS.
%mend mtanyobs;
```