

Methods of Producing Datasets with Unique Record per Id Number

Alexander Khartchenko, Management, Information & Analysis, Kaiser Permanente, Oakland, CA
Irina Tolstykh, Division of Research, Kaiser Permanente, Oakland, CA
Natalia Udaltsova, Division of Research, Kaiser Permanente, Oakland, CA

ABSTRACT

The task of producing a specific dataset with Unique Record Per Id (URPID) number on the basis of multi record information with duplicates collected from various sources in different formats appears to be frequently met, and at the same time important and non-trivial.

This paper describes few approaches that may be used to perform this task successfully. Using specific SAS functions Retain and Lag and procedures Freq, Transpose, Summary and SQL we demonstrate here how to aggregate information with a) large number of observations; b) big collection of variables (numerical and character), c) not clean initial data (duplicates, missing values, etc).

In our practice it is necessary to save time and memory space for the URPID. Therefore comparative characteristics of methods including CPU time, Memory usage, etc. are given as well.

INTRODUCTION

Efficiency in data processing methods is vital to success in preparing data sets for advanced statistical analysis. There is a wealth of useful “tips and tricks” described in SAS methodological literature. Some of these tips and tricks are well-known, for example: do not read a file unnecessarily; use a LENGTH statement, using 4 bytes instead of 8 default bytes very often reduces storage space and speed up data transfer; drop all unnecessary variables; reduce the number of sorts; use CLASS statement instead of BY statement in proc MEANS; use WHERE statement instead of IF statement selecting subset from a large file; use ELSE IF instead of multiple IF statements; use _NULL_ as a data set name when you only process records from the file but do not intend to keep resulting data set, and so on.

In our case we work with relatively large data sets – millions of observations, and hundreds of variables - and the data originally reaches us as a raw data flat file. Therefore, SAS data processing should start from a procedure such as, proc **Fslist** to see what category of data we are working and how it could be modified for further handling by SAS methods. After establishing the data layout, data processing starts.

It is rather common in epidemiological research as well as in management information forecasting and analysis tasks to have original input data with multiple numbers of records per patient, along with the need to collect specific information for output

usage or report. It can be the inpatient or outpatient information during 1-year time periods, or information about insurance plans and drug benefits.

PART 1

Assume that the original SAS data contains the following information: id number, date of record, and two variables with different types: numeric and character; so this dataset “old” has 4 variables: id, date, var_n, var_c. Notice, that id and date are given and not null, but values of var_n and var_c could be missing.

Approach 1

For the first task, we need to obtain URPID with last non-missing available information. Our output dataset “new” will include: id, var_n and date_n, var_c and date_c, where date_n and date_c are corresponding values of date. It would be preferable to know the length of character variable var_c - &len, so it could be presented by a macro variable, assignable by the following code:

```
proc contents data=old out=list  
(keep=name length) noprint; run;  
proc sql noprint;  
select length into :len from list  
where name='var_c';  
quit;  
%put length=&len;
```

First step – sorting

```
proc sort data=old; by id date; run;
```

Second step – construction of a new dataset with last record and collected information

```
data new(rename=(temp_c=var_c temp_n=var_n));  
set old; by id;  
length temp_c $&len.; format date_c mmdyy10.;  
length temp_n 8.; format date_n mmdyy10.;  
retain temp_c ` ` temp_n 0;  
retain date_c date_n .;  
if first.id then do; temp_c=` `; temp_n=0; end;  
if var_c ne ` ` then do;  
temp_c=var_c; date_c=date; end;
```

```

if var_n ne . then do;
temp_n=var_n; date_n=date; end;
drop date var_c var_n;
if last.id then output;
run;

```

Approach 2

If you have plenty of space and time then, probably, you would prefer to collect all non-missing information. Let's create URPID dataset with numerical variable new_n equal summary of all non-missing values of var_n and new comma delimited string variable new_c as the result of concatenation of all non-missing values var_c. In this situation final dataset will include also date of first record date_start and date of last record date_finish. So, dataset "new" will contents: id, date_start, date_finish, new_n, and new_c.

It is recommended for the string variable new_c to assign the length &lenmax, which is equal &len (length of var_c) multiply by &max (maximum number of records per id). For instance, the following code will perform this calculation:

```

proc sql noprint;
select max(count) into :max from
(select id, count(id) as count
from old group by id);
quit;
%put maximum=&max;
%let lenmax=%eval(&len*&max);
%put maximum length=&lenmax;

```

First step – sorting

```
proc sort data=old; by id date; run;
```

Second step – construction of new dataset with last record and all collected information.

```

data new(drop=date var_c var_n); set old; by id;
format date_start date_finish mmdyy10.;
length new_c $&lenmax.;
retain new_c ' 'new_n 0 date_start .;
if first.id then do;
temp_c=' '; temp_n=0; date_start=date; end;
if var_c ne ` ` then new_c=var_c||','||new_c;
if var_n ne . then new_n=var_n+new_n;
if last.id then do;
date_finish=date; output; end;
run;

```

Approach 3

Obviously, sometimes our task could be more specific and difficult. For example, we need to obtain URPID with the

highest-frequency value of variable – mode of variable. So, final dataset "new" will contain id, mode of numeric variable var_n (with name – new_n) and all counters for the distinct values of character variable var_c: count_val1, count_val2 and so on. Preliminary, we have to know all possible values for var_n and var_c and count them appropriately. Let's var_n be integer from 1 to 4 (if several values appear the same number of times – save the largest). Let's var_c has three distinct values: 'Val1', 'Val2', and 'Val3'.

First step – sorting

```
proc sort data=old; by id date; run;
```

Second step – construction of new dataset with last record and collected information.

```

data new(drop=var_c var_n i n1-n4 max);
set old; by id;
array cc{*} count1-count3;
array nn{*} n1-n4;
retain count1-count3 n1-n4 0;
if first.id then do;
do i=1 to dim(cc); cc{i}=0 end;
do i=1 to dim(nn); nn{i}=0 end; end;
if var_c='Val1' then count1+1;
if var_c='Val2' then count2+1;
if var_c='Val3' then count3+1;
do i=1 to 4; if var_n=i then nn{i}+1; end;
if last.id then do;
max=max(n1,n2,n3,n4);
do i=1 to 4; if max=nn{i} then new_n=i ; end;
output;
end;
run;

```

We can use a macro to produce calculations for each character value of var_c, and here it is:

```

proc sql noprint;
create table temp as
(select distinct var_c from old
where var_c ne ' ');
quit;

```

Dataset "temp" contains all distinct values of variable var_c and now we can generate all necessary macro variables.

```

data _null_; set temp end=last;
macroname=compress('value'||_N_);
countname=compress('count'||_N_);
call symput(macroname, var_c);
call
symput(countname, compress('count_'||var_c));
if last then call symput('dim',compress(_N_));
run;

```

Next loop will count values of character variable var_c one by one.

```
%macro count_char_value;
%do i=1 %to &dim;
retain &&count&i 0;
if first.id then &&count&i=0;
if var_c="&&value&i" then &&count&i=&&count&i+1;
%end;
%mend count_char_value;
```

Now, assume, that we would prefer to assign the most frequent value of character variable var_c to the new character variable new_c. All values of var_c appeared in alphabetic order. So, if there are more than one mode of variable var_c, then new_c will take first.

```
%macro assign_char_value;
maxc=&count1;
if maxc>0 then new_c="&value1";
%do i=2 %to &dim;
if &&count&i>maxc then do; maxc=&&count&i;
if maxc>0 then new_c="&&value&i"; end;
%end;
%mend assign_char_value;
```

Now it is time to call these macros inside the data step to create SAS dataset "new".

```
data new(drop=var_c var_n i n1-n4 max maxc);
set old; by id;
array nn{*} n1-n4; retain n1-n4 0;
if first.id then do;
do i=1 to dim(nn); nn{i}=0 end; end;
do i=1 to 4; if var_n=i then nn{i}+1; end;
length new_c $&len.; new_c=' ';
%count_char_value;
if last.id then do;
max=max(n1,n2,n3,n4);
do i=1 to 4; if max=nn{i} then new_n=i ; end;
%assign_char_value;
output;
end;
run;
```

Notice, that we can simply collect the information for more than one variable by using array statement during one data step. Notice also, that Lag and Dif functions could be used, if we don't have missing value of variable of interest.

PART 2

Here is one real-life example. Assume that we have multiple records per patient and wish to assign gender variable without mistake. All records have variable Gender as M, F; any other values are considered typo error. Let's count how many times

patient was identified as a male (variable sexm) and how many times patient was identified as a female (variable sexf). Based on that information, we will be able to assign gender logically. Let's observe and discuss several different methods to complete the same task.

Example1-1. We could use proc **Freq** and create a cross-table to see the most frequent appearances of Gender value.

```
proc freq data=start noprint;
tables id*gender/ out=fr(drop=percent);
run;
***real time 2.87 sec, cpu time 2.37 sec***;
```

Notice, that no sorting used before freq procedure, but the cross-table may require unpredictable space and variable percent is uninformative.

Example1-2. Let's use proc **Sort** first and create simple table in proc **Freq** for every id number.

```
proc sort data=start(keep=id gender); by id;
run;
***real time 1.00 sec, cpu time 0.57 sec***;
proc freq data=start noprint;
by id;
tables gender/ out=fr;
run;
***real time 2.71 sec, cpu time 2.67 sec ***;
```

There is no cross-table in this output, and dataset "fr" includes two informative variables: count and percent. If percent variable is equal 100, then it means that every time gender has the same value and there is no discrepancy in original data source. However procedure **Sort** could require additional space and extra time. To avoid this disadvantage it is recommended to keep only necessary variables.

Example1-3. We could use proc **SQL** to generate the same dataset.

```
proc sql noprint;
create table fr as
(select id, gender,
count(gender) as count
from start group by id,gender ) order by id;
quit;
***real time 3.12, cpu time 1.23***;
```

It is well known that SQL procedure has some strong points: it does not require input data to be sorted, does not need extra space, and performance time is pretty competitive. Also we are able to generate more variables in the same statement, such as total number of records per patient or maximum of count.

```
proc sql noprint;
create table fr as
```

```
(select *, sum(count) as total,
max(count) as max from
(select id, gender,
count(gender) as count
from start group by id,gender )
group by id) order by id;
quit;
```

To complete example 1 and create dataset “final” the following procedure could be used:

```
proc transpose data=fr out=final prefix=sex;
by id;
id gender;
var count; run;
```

Example2. As we stated above (Part1, Approach 3) the statement **Retain** also may be used to complete the same task. Please notice that proc **Sort** is required.

```
proc sort data=start; by id; run;
***real time - 2.00 sec, cpu time - 0.95 sec***;
data final(drop=gender); set start;
by id;
retain sexf sexm 0;
if first.id then do; sexf=0; sexm=0; end;
if gender='F' then sexf+1;
if gender='M' then sexm+1;
if last.id then output;
run;
***real time 0.42 sec, cpu time - 0.29 sec***;
```

We have already discussed the advantages of using data step to follow all variables of interest simultaneously. As one can see, the time usage is more than competitive, however preliminary sorting might be a problem for huge data sets with many variables.

Example3. One more procedure to use: proc **Summary**. Notice that for the character variable Gender we need to generate dummy variables sumf and summ before the summary step.

```
data start1; set start;
sumf=(gender='F');
summ=(gender='M');
run;
***real time 0.62 sec, cpu time 0.24 sec ***;
proc summary data=start1 nway missing;
class id;
var sumf summ;
output out=final sum=sexf sexm;
run;
***real time 1.23 sec, cpu time 1.15 sec***;
```

Procedure **Summary** as well as procedure **Means** are rather powerful and can generate output with all necessary variables in

a single step, with time demands that are competitive under other procedures without additional transposition. There are many examples in SAS literature when **SQL** and **Summary** will be efficient, methodologically speaking.

DISCUSSION

Economies of space and time have been achieved in generating URPID data sets consisting of millions of observations and hundreds of variables. Three coding approaches were illustrated to complete three different tasks.

Of the three coding examples, the proc **SQL** and **Summary** achieved the shortest computer run time. In actuality, depending on CPU performance, the three methods could be combined, accomplishing optimal savings in space and time.

ACKNOWLEDGMENTS

Special thanks go to Angelo Cosmides for smoothing out our prose.

REFERENCES

- [1] Western Users of SAS Software Tenth Annual Conference, WUSS 10, San Diego, September 4-6, 2002, Proceedings.
- [2] Applied Statistics and the SAS Programming Language, third edition, Ronald P. Cody and Jeffrey K. Smith, North-Holland, New York: 1991.
- [3] Longitudinal Data and SAS: A Programmer's Guide, Ronald P. Cody, SAS Publishing, 2001.

CONTACT INFORMATION

Alexander Khartchenko
Management, Information & Analysis
Kaiser Permanente Medical Group, Inc.
1950 Franklin Street, 17th Floor -Lakeside
Oakland, CA 94612
(510) 987-4696
Alexander.Khartchenko@kp.org

Irina Tolstykh
Division of Research
Kaiser Permanente Medical Group, Inc.
2000 Broadway
Oakland, CA 94612
(510) 891-3535
Irina.V.Tolstykh@kp.org

Natalia Udaltsova, Ph.D.
Division of Research
Kaiser Permanente Medical Group, Inc.
2000 Broadway
Oakland, CA 94612
(510) 891-3738
Natalia.Udaltsova@kp.org