

An Approach to Creating Archives That Minimizes Storage Requirements

Ruben Chiflikyan, RTI International, Research Triangle Park, NC
Mila Chiflikyan, RTI International, Research Triangle Park, NC
George Grubbs, RTI International, Research Triangle Park, NC

ABSTRACT

Many research studies use datasets that are available and processed on a regular basis such as daily, weekly, or monthly. They also require the capability to recreate analyses and reports based on datasets for a certain date or date range, thus requiring that all datasets be archived over time. Often the datasets are stored on labeled external media, and the data manually retrieved and loaded from these media when needed. When stored in this manner, these archives can grow to be hundreds of gigabytes. This approach is workable, but is subject to loss of data should the media be misplaced or deteriorates over time, and it takes time to locate, retrieve, and load the desired datasets. Some studies have data values that vary very little from dataset to dataset. For these studies, we offer a novel archiving method that preserves the capability of recreating analyses and reports for any specific dataset or set of datasets and minimizes storage requirements making it feasible to maintain the archive in an online database. The key to the method is to store only unique sets of values along with date-time fields that are used when recreating a set of values for a particular date. We describe the procedure used to create storage-saving archives and provide guidelines on savings gained as a function of dataset size and percentage of identical values, along with the processing time cost of reconstructing datasets for date ranges from their baseline datasets.

INTRODUCTION

Many organizations collect and process data on a regular basis such as daily, weekly, or monthly. Various types of reports are required to be created using data gathered at various times and that requires that all datasets generated during certain periods of time have to be present. The authors in [1] have automated the process of creating SAS® output files generated from longitudinal studies and renaming these copies that includes a number in to keep track of previous executions. However, in this process how to archive SAS datasets was not considered. Often the datasets are stored on external media which creates additional difficulties when retrieving data from them. Usually this direct approach of copying every bit of data into archives becomes unacceptable, especially when datasets grow to hundreds of gigabytes and even over a terabyte. While this approach is workable, it is subject to loss of data should the media be misplaced or deteriorates over time, and it takes time to locate, retrieve, and load the desired datasets. Some studies have data values that vary very little from dataset to dataset. For these studies, we offer a novel archiving method that preserves the capability of recreating analyses and reports for any specific dataset or set of datasets and minimizes storage requirements making it feasible to maintain the archive in an online database. The key to the method is to store only unique sets of values along with date-time fields that are used when recreating a set of values for a particular date. We describe the procedure used to create storage-saving archives and provide an analysis of savings gained as a function of dataset size and percentage of identical values, along with the processing time cost of reconstructing datasets for date ranges from their baseline datasets.

THE DESCRIPTION OF THE PROGRAM

The following program demonstrates how this approach works. In this example, we assume that SAS datasets arrive on daily basis. They could have a different number of variables and observations.

```
libname ttt 'C:\SESUG09\Testing'; * the folder, where all datasets will be kept;  
%let ds=t; * the prefix for datasets' names;
```

In this situation, the datasets have date stamp in the form yyyyymmdd, which is arbitrary and could change in the way as needed. For this example the first dataset has the following form:

```
* the very first dataset;
```

```

data ttt.t_20091025;
infile datalines;
input id1 $ id2 $ var1 var2;
datalines;
1 1 10 100
2 1 20 200
3 1 30 300
4 1 40 400
;

* the dataset generated on the second day;
data ttt.t_20091026;
infile datalines;
input id1 $ id2 $ var1 var2 var3;
datalines;
1 1 11 100 1000
2 1 20 200 2000
3 1 31 300 3000
5 1 41 400 5000
6 1 41 400 6000
;

* the dataset generated on the third day;
data ttt.t_20091027;
infile datalines;
input id1 $ id2 $ var1 var2;
datalines;
1 1 10 100
2 1 20 200
3 1 30 300
4 1 40 400
;

```

As can be seen, the three datasets above are illustrating input for each of three consecutive days, October 25, 26, and 27 2009. We want to demonstrate how to create and keep a history table for these three datasets, and how to retrieve data by request for any specific date.

CREATION OF THE INITIAL HISTORICAL DATASETS

First, we need to create the historical dataset on the basis of the ttt.t_20091025 dataset. To do this, we need to create an identifier that will uniquely identify the observations in the dataset. This could be one or any combination of native ids. In this specific case, we decided to use the id1 SAS variable as the unique id. This new variable will be added to each dataset below. Note that it also has to have a unique name.

```
%let ids=id1; * the newly created identifier;
```

Since this dataset was created on Oct 25, 2009, we create a macro variable &dt as follows:

```
%let dtt=25OCT09;
```

Note that the same date format is used when creating SAS dates from the system using %let ddt=&sysdate; statement.

To create the SAS date, we use the following datastep:

```

data _null_;
call symput('dt',put(input("&dtt",date7.),8.));
run;

```

The following datastep creates the &reptime macro variable that is exactly in the yyyyymmdd format:

```

data _null_;
length y $4 m d $2 reptime $8;
y=left(trim(year(&dt)));

```

```

m=left(trim(month(&dt)));
d=left(trim(day(&dt)));

if substr(m,2,1)=' ' then m='0' || substr(m,1,1);
if substr(d,2,1)=' ' then d='0' || substr(d,1,1);
redate=left(trim(y||m||d));
call symput('redate',redate);
run;

```

In the dataset below, we create the history datasets ttt.hist_&ds and the auxiliary ttt.&ds._yest that will be updated with each new arrival of daily datasets.

Note that these datasets contain all the original variables plus four new variables as follows: id is the unique observation identifier, defined above, dob is the variable that will contain the date value when the specific observation was created, current is the variable that contains today's date, and dod is the variable with the date that the specific observation was not found in the new arrival.

```

data ttt.&ds._yest ttt.hist_&ds;
length id $30;
set ttt.&ds._&redate;
id=trim(left(&ids)); * the newly created identifier;
dob=&dt; * date when the observation was created;
current=&dt; * date with today's date ;
dod=0; * date when the appearance of this specific observation was terminated;
run;

```

We have to take into account that the number of variables can be different from day-to-day. To handle this situation, we need to save the variable name for each dataset. The next dataset uses the OPEN function and creates the all_names dataset with one variable name that contains all the names for variables that were present in the historical dataset.

```

data all_names;
length name $30;
drop dsid i num rc;
dsid=open("ttt.&ds._&redate","i");
num=attrn(dsid,"nvars");
do i=1 to num; *num is the number of vars;
name=varname(dsid,i);
output;
end;
rc=close(dsid);
run;

```

After that, a macro variable &allvars containing all variable names is created from the SQL statement below:

```

proc sql;
select name into :allvars
separated by ' '
from all_names
;
quit;

```

Then, another history table that contains the variable names associated with each date is created as follows:

```

data ttt.all_vars_hist;
length allvars $1000;
date=&dt;
allvars="&allvars";
run;

```

To be able to create the archival system, we need to have online the following three datasets: ttt.hist_&ds for keeping all data in the historical dataset, ttt.&ds._yest for keeping data in the dataset one day before, and ttt.all_vars_hist for keeping track of the specific composition of variable names for each daily dataset. The program above should be run only once on the first day when the historical dataset is created. We show below how these datasets are updated.

UPDATING ON THE REGULAR BASIS OF THE HISTORICAL DATASETS

In this example, the dataset named "ttt_20091026" arrived on Oct 26, 2009. Again, we need to explicitly identify the unique identifier for observations. It should be the same as the one used in the initial historical dataset created above.

```
%let ids=id1; * the newly created identifier;
```

Since this dataset was created on Oct 26, 2009, we create a macro variable &dt as follows:

```
%let dt=26OCT09;
```

In the situation where the database may come automatically, one can use the '%let dt=&sysdate;' statement.

To have today's and yesterday's dates as SAS dates, we use the following statements:

```
data _null_;
call symput('dt',put(input("&dt",date7.),8.));
call symput('last',put(input("&dt",date7.),8.)-1);
run;
```

The following dataset creates today's date in the yyyymmdd format:

```
data _null_;
length y $4 m d $2 today $8;
y=left(trim(year(&dt)));
m=left(trim(month(&dt)));
d=left(trim(day(&dt)));

if substr(m,2,1)=' ' then m='0' || substr(m,1,1);
if substr(d,2,1)=' ' then d='0' || substr(d,1,1);
today=left(trim(y||m||d));
call symput('tday',today);
call symput('redate',today);
run;
```

The subsequent three dataset steps and the proc SQL create the temporary dataset corresponding to the specific dataset generated today and use it for updating of the existing history table "ttt.all_vars_hist". It contains the current date variable and the character variable &allvars, which contains all variables names (see above SQL select statement for details).

```
* creation of the dataset with the list of all variables' names;
data all_names;
length name $30;
drop dsid i num rc;
dsid=open("ttt.&ds._&tday","i");
num=attrn(dsid,"nvars");
do i=1 to num; *num is the number of vars;
name=varname(dsid,i);
output;
end;
rc=close(dsid);
run;
```

```
* creation of the &allvarscur macro variable with variables' names;
proc sql;
select name into :allvarscur
separated by ' '
from all_names
;
quit;
```

```

* creation of the variable names' composition;
  data all_vars_cur;
  length allvars $1000;
  date=&dt;
  allvars="&allvarscur";
  run;

* updating of the historical ttt.all_vars_hist table;
  data ttt.all_vars_hist;
  set ttt.all_vars_hist all_vars_cur;
  run;

```

For further transformations, we need to create the &vrs macro variable and calculate the number of variables in the current dataset:

```

proc sql;
  select name into :vrs
  separated by ' '
  from all_names
  ;

  select count(name) into :numobs
  from all_names
  ;
quit;

```

The id, dob, current and dod variables are added to the current dataset in the same way as was done above.

```

data ttt.&ds._today;
  length id $30;
  set ttt.&ds._&tday;
  id=&ids;
  dob=&dt;
  current=&dt;
  dod=0;
  run;

```

Finally, the newly created dataset is appended to the already existing history table.

```

data hist_add;
  set ttt.hist_&ds ttt.&ds._today;
  run;

proc sort data=hist_add; *sorting;
  by id descending dob;
  run;

```

The macro below identifies those observations with the same id that have at least one distinct value in one of the variables. To do this, we use the LAG function that compares values for each pair of observations that have the same id. In case this situation is encountered, we assign dod=&dt, which means that this specific observation does not exist anymore.

```

%let and=and;
%macro chk1;
  data new;
  set hist_add;
  if id=lag(id)
  and ^
  (
  %do I=1 %to &numobs;
  %if &I=&numobs %then %do; %let and=; %end;
  %scan(&vrs,&I,' ')=lag(%scan(&vrs,&I,' ')) &and
  %end; * comparing 2 consecutive values of the same
  Variable having the same identifier;

```

```

    )
    and dod=0
    then dod=&dt;
    run;
%mend chk1;
%chk1;

* recreating the previous order of observation;
proc sort data=new out=new1;
by id dob;
run;

* deleting observations that are identical. Note that in this case the observations
with earlier date remain in the dataset;
proc sort data=new1 out=new2 nodupkey;
by &ids &vrs dod;
run;

* assigning today's date the current variable;
data hist_add_ndp;
set new2;
current=&dt;
run;

```

There could be a situation where some ids do not exist in today's dataset, while they were present in yesterday's dataset.

```

* creating dataset with ids, which ceased in today's dataset;
proc sql;
create table dd as
select id
from ttt.&ds._yest
where id not in (select id from ttt.&ds._today)
;

* creating a macro variable with list of values for this type of ids;
select id into :idsa
separated by ' '
from dd
;

* finding the number of ids of this type;
select count(id) into: dnum
from dd
;
quit;

```

Next, for those observations that do not exist in today's dataset, we assign `dod=&t if dod=0`.

```

* dod=0 condition below is required no to overwrite already created dod;
%macro ddd;
data ttt.hist_&ds;
set hist_add_ndp;
current=&dt;
%if &dnum > 0 %then
%do;
%do I=1 %to &dnum;
if id="%scan(&idsa, &I, ' ')" and dod=0 then dod=&dt;
%end;
%end;
run;

%mend ddd;
%ddd;

```

Finally, we assign today's dataset as it was created yesterday to be able to implement the same logic above.

```
data ttt.&ds._yest;  
set ttt.&ds._today;  
run;
```

Now, we run this program for the second time to include the dataset of October 27, 2009. Below, we will demonstrate how to restore any dataset that was used to create the final dataset.

RESTORING OF THE PARTICULAR DATASET FROM HISTORICAL DATASET

First, we need to provide the date for which the restoration will be done. For example, in this case it will be

```
%let dtt=11FEB09;
```

The dataset below creates the corresponding SAS date.

```
data _null_;  
call symput('dt',put(input("&dtt",date7.),8.));  
run;
```

Additionally we need to identify, which variables were present at that time. The dataset below selects the observation with the same date from the historical dataset ttt.all_vars_hist.

```
data current;  
set ttt.all_vars_hist;  
if date=&dt;  
run;
```

* This dataset creates a macro variable that contains the list of all variable names that were present at that time;

```
data _null_;  
set current;  
call symput('keep',allvars);  
run;
```

The dataset below creates the corresponding date in the yyyymmdd format which we use as a datestamp in dataset's names.

```
data _null_;  
length y $4 m d $2 today $8;  
y=left(trim(year(&dt)));  
m=left(trim(month(&dt)));  
d=left(trim(day(&dt)));  
  
if substr(m,2,1)=' ' then m='0'||substr(m,1,1);  
if substr(d,2,1)=' ' then d='0'||substr(d,1,1);  
recday=left(trim(y||m||d));  
call symput('recdate',recday);  
run;
```

The dataset below restores the required dataset.

```
data ttt.rcnstr_&ds._&recdate(keep=&keep);  
set ttt.hist_&ds;  
if dod>0 then  
do;  
if dob <= &dt <= current and dod > &dt;  
end;
```

```

else
do;
  if dob <= &dt <= current;
end;
run;

```

Finally, to compare the accuracy of restoration, we run the proc COMP procedure.

```

proc compare base=ttt.&ds._&reccdate
             compare=ttt.rcnstr_&ds._&reccdate
             out=result outnoequal outbase outcomp outdif
             noprint;
             id &ids;
run;

```

USAGE

This approach could be very effective for small- and medium-size datasets. The current system date can be used in the same way instead of using the hard coded date above. The efficiency of using this technique for large datasets requires additional investigation. Note that the lower the percentage of the daily new additions, the higher the efficiency of this approach. Note also that along with the historical dataset, we need to keep two other auxiliary datasets as follows: the dataset that was created yesterday, and a small historical dataset that contains information about variable composition.

CONCLUSION

In this paper, we offered a novel archiving method that preserves the capability of recreating analyses and reports for any specific dataset or set of datasets which minimizes storage requirements making it feasible to maintain the archive in an online database. The key to the method is to store only unique sets of values along with date-time fields that are used when recreating a set of values for a particular date. We described the procedure used to create storage-saving archives and provided guidelines on savings gained as a function of dataset size and percentage of identical values, along with the processing time cost of reconstructing datasets for date ranges from their baseline datasets.

REFERENCES

1. Box, Jim and Morris, Joey, Backups at the Click of the Mouse (Paper CC-04), Proceedings of PharmaSUG 2004, May 3-26, 2004, San Diego, California.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The electronic copy of this paper is available. Please, contact the authors at:

Ruben Chiflikyan
 RTI International
 3040 Cornwallis Road, PO Box 12194
 Research Triangle Park, NC, 27709-2194
 Work Phone: (919)541-6064
 Fax: (919)541-6178
 E-mail: rchiflikyan@rti.org
 Web: <http://www.rti.org>

Mila Chiflikyan
 RTI International
 3040 Cornwallis Road, PO Box 12194
 Research Triangle Park, NC, 27709-2194

Work Phone: (919)316-3360
Fax: (919)541-6178
E-mail: milachif@rti.org
Web: <http://www.rti.org>

George Grubbs
RTI International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work Phone: (919)990-8397
Fax: (919)541-6178
E-mail: ggrubbs@rti.org
Web: <http://www.rti.org>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.