**Paper 070-2011**

# Storage Space and Processing Time Comparisons between Horizontal and Vertical Longitudinal Data

John Hennessey, Social Security Administration, Woodlawn, MD

## ABSTRACT

A longitudinal, horizontal person-record flat file, with 7,152 variables, is converted to a SAS library of 15 datasets. One of the datasets contains the static variables. The other fourteen datasets are vertical longitudinal event history datasets for each of the longitudinal variables. The size of the resulting SAS library is 18% of the size of the original flat file. Further, the processing times to answer standard longitudinal questions are significantly shorter than the time to generate the same results from the horizontal file, even starting as a SAS dataset. Enterprise Guide and custom code are used to convert the horizontal file to vertical event history files and back to horizontal files.

## INTRODUCTION

Many longitudinal data files in the Office of Research, Evaluation, and Statistics (ORES) of the Social Security Administration (SSA) are stored in the classical way, as horizontal person-record flat files with one record per person. One such file contains 14 longitudinal variables that are stored horizontally, with values var1 through var504 for each variable. They represent the values of the given variable, month by month, starting in 1/1974 – the beginning of the SSI Program – to the current month. The remaining variables are blank, leaving room for future months.

The file also contains a number of static variables, such as Social Security Number (SSN), sex, race, etc. Including the static variables, each individual record contains over 7,100 variables. Many variables are discrete and repeated for each month with a small number of value changes over the 504 positions. Because the data are stored in a horizontal, fixed record length flat file, the 504 values of the 14 longitudinal variables reflects the longest possible string of values for all of the variables. Thus, many of the values in any record are missing and many values are the same for a long string of months.

As one might suspect, datasets maintained by SSA to administer the programs range from large to extra-large. It would not be unusual for ORES to request an extract from the dataset described above, to be used for research or calculations of descriptive statistics, to contain 20 to 40 million records. Sometimes datasets with 80 to 100 million records are required. Therefore, storage space and processing time is critical to productivity.

Having spent much of my 36 year career dealing with longitudinal data, and having read Cody's book, "Longitudinal Data and SAS", I am familiar with the structure of a vertical event history file. I also became aware that vertical event records allow for SAS Procs to replace arrays and do loops and more primitive coding within a Data Step.

Finally, I have become an Enterprise Guide convert. In earlier days, I cut my teeth on FORTRAN and then moved on to become a SAS code expert, typing thousands of lines of SAS code. Now that Enterprise Guide will do it for me with a few clicks of the mouse, why would I type it myself? And, when I need to insert a few lines of my own code to customize the Enterprise Guide code, I can do it.

I mention Enterprise Guide because it, just like most point & click systems, does not like wide files, i.e. files with a large number of variables. When I click on the icon for the SAS dataset version of the horizontal file described above, it takes about 3 minutes for Enterprise Guide to open the dataset or to open a point & click PROC FREQ session. It takes that long for Enterprise Guide to set up the 7,100 variables for use in the point & click session. The vertical event history SAS dataset for each of the longitudinal variables has only three columns – the Social Security Number, the new value of the variable, and the date of the change to the new value. When I click on the icon for this vertical event history dataset, it opens immediately.

The factors listed above motivated me to compare the storage space and the processing time for typical calculations on both the horizontal person-record file and the library of vertical datasets. My findings and lessons learned follow.

### DISCLAIMER

For security and privacy reasons, I have chosen a dataset that represents no particular population. I have changed the variable names. So, the results are not meaningful in any way. The results are displayed for illustrative purposes only.

## DATA DESCRIPTION

I happened to have an extract of the horizontal flat file described above from an earlier project. It consisted of 16,542 person-records with 7,152 variables. Part of one person-record is shown below in Table 1. It displays the values for STATUS261 through STATUS279 and FEDMONEY261 through FEDMONEY271. Note that it records many repetitive values of each of the variables, STATUS and FEDMONEY, month after month.

**Table 1: Partial Person-Record from Horizontal File**

| Obs | STATUS261 | STATUS262 | STATUS263 | STATUS264 | STATUS265 | STATUS266 | STATUS267 | STATUS268 | STATUS269 | STATUS270 | STATUS271 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | C01 | N44 | N44 | N44 | N44 | N44 | N44 | N44 | N44 | N44 |

| Obs | STATUS272 | STATUS273 | STATUS274 | STATUS275 | STATUS276 | STATUS277 | STATUS278 | STATUS279 | FEDMONEY261 | FEDMONEY262 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | N44 | N44 | N44 | T31 | T31 | T31 | T31 | T31 | 0 | 145 |

| Obs | FEDMONEY263 | FEDMONEY264 | FEDMONEY265 | FEDMONEY266 | FEDMONEY267 | FEDMONEY268 | FEDMONEY269 | FEDMONEY270 | FEDMONEY271 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

We first create one vertical SAS dataset for all of the variables the entire original horizontal dataset. Next, the repetitive values are removed. A record is kept when there is a CHANGE to the value of the variable. Also, the missing values are not recorded – only the CHANGE to a missing value. The code below does this task. It is a straightforward application of PROC TRANSPOSE followed by the use of the lag function in a DATA STEP to eliminate the repetitions. The int, mod, and mdy functions are used to convert the subscript of the longitudinal variables, var1 through var504 to dates.

```
Proc Transpose data=NTIDSCRM.SSILNGT(obs=max) out=work.SSI_Vertical;
    by SSN;
    var _all_;
run;


data work.SSI_Vertical2;
    retain SSN date variable col1;
    set SSI_Vertical;
    k=substr(_name_,anydigit(_name_),3);
    year=int((k-1)/12) + 1974;
    month=mod((k-1),12) +1;
    date=mdy(month,1,year);
    Variable=substr(_name_,1,anydigit(_name_)-1);
    if lag(SSN)=SSN and
                   lag(substr(_name_,1,anydigit(_name_)-1)) =
                   substr(_name_,1,anydigit(_name_)-1) and
                   lag(col1)=col1
    then delete;
    format date mmyys7.;
    drop year month k _name_;
run;
```

Table 2 contains a sample of the resulting vertical table. The full table contains the changes to ALL of the longitudinal variables for ALL people. Below, the variables FEDMONEY and STATUS are listed for one individual as an example. (The dataset also contains a variable for the person's SSN. It is left out for privacy reasons.) It is clear that the number of records needed to record vertically the changes to these two

**Table 2: Partial List of Records for One Person in the Vertical Dataset**

| date | variable | col1 |
|---|---|---|

| date | variable | col1 |
|---|---|---|
| 01/1974 | FEDMONY | 0 |
| 10/1995 | FEDMONY | 145 |
| 12/1995 | FEDMONY | 0 |
| 02/2002 | FEDMONY | 1081 |
| 03/2002 | FEDMONY | 363 |
| 05/2002 | FEDMONY | 0 |
| 01/1974 | STATUS | |
| 10/1995 | STATUS | C01 |
| 12/1995 | STATUS | N44 |
| 12/1996 | STATUS | T31 |
| 04/1997 | STATUS | C01 |
| 10/2001 | STATUS | N01 |
| 11/2001 | STATUS | E02 |
| 12/2001 | STATUS | C01 |
| 05/2002 | STATUS | N01 |
| 05/2003 | STATUS | T31 |

longitudinal variables is greatly reduced from the original 504 variables stored horizontally. However, the date now becomes a variable instead of a suffix for the variable name. Also, at this point, all of the variables in the entire horizontal record have been transposed to a vertical structure. And so, the middle column, variable, is created by PROC TRANSPOSE to identify the variable that is displayed in each record. We have removed the suffix. Finally, the SSN is repeated for each record (but not displayed here).

## WHY BREAK UP THIS DATASET?

Since we have added variables, date and variable name, to the dataset and dropped some of the records, it is not clear whether the vertical dataset is smaller than the horizontal one. In addition, PROC TRANSPOSE converts all of the variable values to character with a length equal to the largest possible character string of the entire dataset and the entire list of variables. By breaking up the dataset into a dataset for each of the longitudinal variables, we can eliminate the "variable" column, convert the numeric variables back to numeric, and squeeze each variable down to its minimal length.

**Table 1: Separated and Minimized Datasets**

| date | Fedmoney |
|---|---|
| 01/1974 | $0 |
| 10/1995 | $145 |
| 12/1995 | $0 |
| 02/2002 | $1,081 |
| 03/2002 | $363 |
| 05/2002 | $0 |

| date | col1 |
|---|---|
| 01/1974 | |
| 10/1995 | C01 |
| 12/1995 | N44 |
| 12/1996 | T31 |
| 04/1997 | C01 |
| 10/2001 | N01 |
| 11/2001 | E02 |
| 12/2001 | C01 |
| 05/2002 | N01 |
| 05/2003 | T31 |

The length of the date variable in both datasets above has been changed from 8 to 4. FEDMONEY has been changed from character of length 12 back to numeric with a length of 4. Col1 has been changed from a length of 12 to 3.

## SIZE COMPARISON

The size of the original flat file is 346 MB. The size of the horizontal SAS Dataset is 538 MB. The size of the compressed SAS horizontal file is 51 MB. The dramatic drop in size is due to the large number of blank and repeated values in the longitudinal variables. The total size of the entire SAS library of separate vertical data tables and the one horizontal dataset of static variables is 64 MB. This library is only 18% of the size of the flat file and is only slightly larger than the compressed horizontal file. The vertical files only have 3 variables, SSN, DATE, and VALUE. Therefore, compression actually increases the size of some of them. So, the question remains: Why break up the dataset?

## COMPUTATIONAL EFFICIENCY

The answer is: For a large decrease in storage space as compared to the flat file and for **computational efficiency**. Many of the calculations that are needed to answer classical longitudinal questions are easier to program in SAS for a vertical event history file than for a horizontal file. The main reason is that the records are now event-records, as opposed to person-records. Since the events are the units of analysis, not people, the power of various SAS PROCS can now be employed.

## DATA MANAGEMENT EXAMPLE

One example of this efficiency relates to data management. In a number of our situations, the events of interest are contained in several administrative files. If one extracts these events into a person-record horizontal file, one then has to write complicated code containing arrays and do-loop logic to intertwine the events into chronological order. On the other hand, if one generates vertical event history files from each data source, one only needs to stack all of the events together and then use PROC SORT to sort by SSN and date.

This data management example was the main reason to shift the horizontal data to vertical, after actually coding the arrays and do-loops for many years. But, after I did so, I then read the book by Cody, “Longitudinal Data and SAS”, and realized that there are many more reasons to leave the data in vertical form. I will admit that there is a learning curve as you change the way you think about using the vertical datasets, but, after some time, one begins to profit from the advantages.

To prepare for this presentation, I asked one of my “horizontally-oriented” research colleagues to list a few questions that he recalls addressing in his last paper which used the horizontal dataset above. I present here a few of his questions as examples of the difference between the horizontal approach and the vertical approach.

## QUESTION 1: HOW MANY PEOPLE IN THE DATA FILE WERE IN THE SSI PROGRAM ON DEC. 2001?

In this case, the question is very easy to answer with the horizontal file. One first computes the number of months from 1/1974 to 12/2001 = 336. One can simply run a PROC FREQ on the variable, STATUS336, and look at the count for the value C01.

It is also quite easy to compute with the vertical file. The event records for each individual tell us when the value of the variable changes. So, if we select the last record for each individual on or before Dec. 2001, that will be the value on that date. The following code would work.

```
data work.try1/view=work.try1;
    set VERT_SSI.STATUS_SQZ(where=(date <= '15dec2001'd));
    by SSN date;
    if last.SSN;
run;
```

We then run a PROC FREQ on that data view and obtain the result. So, it is easy enough in both cases.

**Results:** Processing time for the horizontal dataset: 58 seconds
Processing time for the vertical dataset: 57 seconds

Vertical wins by a hair! Essentially, it is a tie.

## QUESTION 2: WHAT WAS THE FIRST AND LAST MONTH OF SSI BENEFITS FOR EACH PERSON?

The computation for this question is complicated by the fact that the SSI benefit is a combination of a state benefit and a federal benefit. For the horizontal file, I need to set up arrays and a do loop to compute totmoney(k) = statemoney(k) + fedmoney(k); locate the first nonzero value and the last nonzero value; convert the suffix on the variable name in each case to the corresponding dates.

For the two vertical files, one for statemoney and one for fedmoney, since I am using Enterprise Guide, I use point & click in SAS Enterprise Guide to generate the following two PROC SQL codes:

```
PROC SQL;
 CREATE TABLE WORK.First_Fedmoney AS SELECT DISTINCT QUERY_FOR_FEDMONEY_SQZ2.SSN
FORMAT=$9.,
      (MIN(QUERY_FOR_FEDMONEY_SQZ2.date)) FORMAT=MMYYS7. AS 'First Fedmoney'n
 FROM VERT_SSI.FEDMONEY_SQZ2 AS QUERY_FOR_FEDMONEY_SQZ2
 WHERE QUERY_FOR_FEDMONEY_SQZ2.Fedmoney > 0
 GROUP BY QUERY_FOR_FEDMONEY_SQZ2.SSN
 ORDER BY QUERY_FOR_FEDMONEY_SQZ2.SSN;
QUIT;

PROC SQL;
 CREATE TABLE WORK.LAST_FEDMONEY AS SELECT QUERY_FOR_FEDMONEY_SQZ2.SSN FORMAT=$9.,
      (MAX(QUERY_FOR_FEDMONEY_SQZ2.date)) FORMAT=MMYYS7. AS MAX_OF_date,
      (intnx('month', calculated max_of_date,-1)) FORMAT=MMYYS7.0 AS 'Last Fedmoney'n
 FROM VERT_SSI.FEDMONEY_SQZ2 AS QUERY_FOR_FEDMONEY_SQZ2
 WHERE QUERY_FOR_FEDMONEY_SQZ2.Fedmoney = 0
 GROUP BY QUERY_FOR_FEDMONEY_SQZ2.SSN
 ORDER BY QUERY_FOR_FEDMONEY_SQZ2.SSN;
QUIT;
```

The first PROC SQL selects, for each person, the records where Fedmoney >0. It then computes the min date. The second PROC SQL selects only those records where the Fedmoney = 0. It then computes the max date and subtracts 1 month – the last month that the Fedmoney is NOT 0. This is then done for the Statemoney vertical table. We then take the lowest and highest for each person, using another PROC SQL task.

As mentioned above, it is a different way of thinking, but, there a several advantages:

1. The code is relatively straightforward, once one becomes familiar with manipulating vertical event history data, which only records CHANGES in variables. It avoids some complicated array and do-loop logic.
2. The computational time is significantly smaller for the vertical version. This is largely due to the fact that we go directly to the tables which only contain fedmoney and statemoney. All other variables have been separated out to other data tables. Also, SAS only has to read records of CHANGES, not month-to-month values and look for changes. This saves a lot of I/O time.

**Results:** Processing time for the horizontal dataset: 58 seconds
Processing time for the vertical dataset: 9.7 seconds ---a Clear Winner!

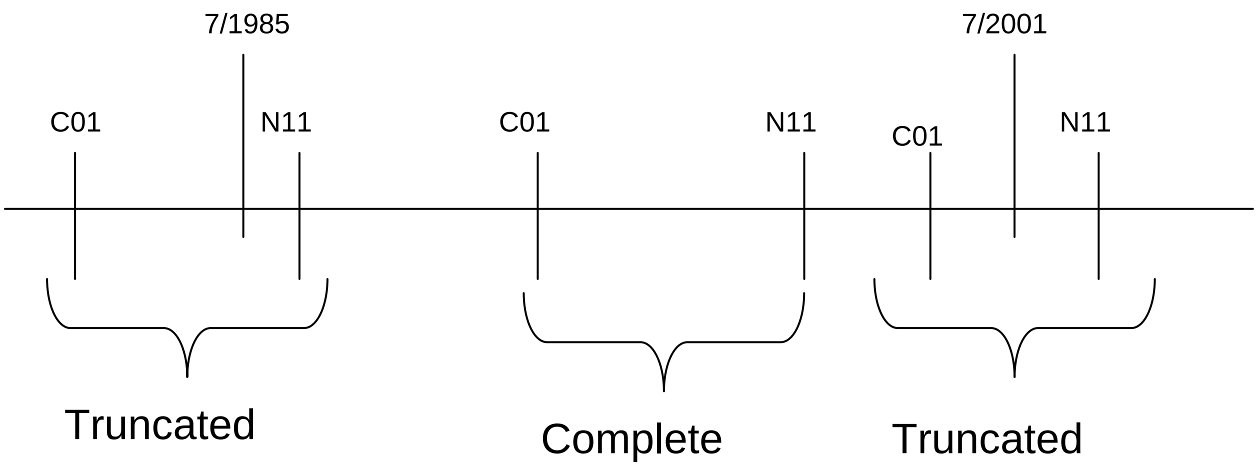
## QUESTION 3: HOW MANY PAYMENT MONTHS OCCURED BETWEEN JULY 1985 AND JULY 2001?

For the horizontal file, I once again, need to set up arrays and a do loop to check through the variables, STATUS1 – STATUS504, for the code, C01, which indicates that the person is in "Current Pay Status" for that month. We would limit the search to start at STATUS(k), where k represents the month for 7/1985 and ends with the value of k which represents 7/2001. This is not hard to do. However, as above, it requires moving across the entire data set and, even though we do not have to read every unnecessary variable, it still takes time to read the values for ALL months.

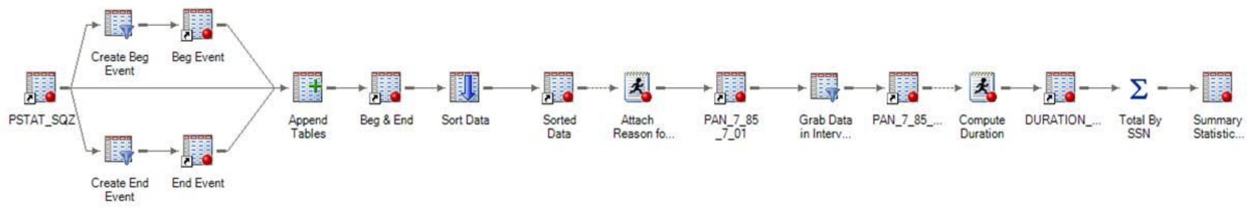
For the vertical file, we need only access the vertical file that contains the STATUS event history. Since the consecutive records contain the dates for when a CHANGE in STATUS occurs, we basically need only subtract the dates for the record where STATUS = 'C01' from the date of the next change – the record right below. The lag function works nicely in this case. The code below essentially accomplishes this task:

```
data work.duration_STATUS;
    set WORK.PAN_7_85_7_01_ONLY;
    by SSN;
    lag_date=lag(date);
    format lag_date mmyys7.;
    if not first.SSN then dur=intck('month',lag_date, date);
    if lag(col1) ='C01';
    if dur=. then dur=0;
run;
```

There is one complicating factor, however, which is presented in the diagram below:



Since we are counting payment status between 7/1985 and 7/2001, we could have a situation where an episode of months in pay status straddles the start or end of the endpoints of the window of time. Thus, the actual time in pay status for that episode is truncated by the beginning or end of the episode. This issue is resolved by inserting two new records for each person - a beginning event with a date of 7/1985 and an end event with a date of 7/2001. For each of these two new events, set STATUS to the value in the data record immediately above (to the left in the diagram) – the actual value at each endpoint. Once again, this can be done with the lag function. The following Process Flow Diagram from Enterprise Guide accomplishes the task:



Below is the code generated by Enterprise Guide with the "extra" code deleted.

```
PROC SQL;
 CREATE TABLE WORK.pan_only AS SELECT DISTINCT PSTAT_SQZ.scrampan FORMAT=$9.,
     ('XB?') AS col1,
     ('1jul1985'd) FORMAT=MMYYS7.0 AS date
 FROM VERT_SSI.PSTAT_SQZ AS PSTAT_SQZ
 ORDER BY PSTAT_SQZ.scrampan;
QUIT;

PROC SQL;
 CREATE TABLE WORK.pan_only_end AS SELECT DISTINCT PSTAT_SQZ.scrampan FORMAT=$9.,
     ('XE?') AS col1,
     ('1jul2001'd) FORMAT=MMYYS7.0 AS date
 FROM VERT_SSI.PSTAT_SQZ AS PSTAT_SQZ
 ORDER BY PSTAT_SQZ.scrampan;
QUIT;

PROC SQL;
CREATE TABLE WORK.PAN_Beg_End AS
SELECT * FROM WORK.PAN_ONLY
 OUTER UNION CORR
SELECT * FROM WORK.PAN_ONLY_END
 OUTER UNION CORR
SELECT * FROM VERT_SSI.PSTAT_SQZ
;
Quit;

PROC SORT DATA=WORK.PAN_BEG_END
    OUT=WORK.PAN_BEG_END_Sort(LABEL="Sorted WORK.PAN_BEG_END")
    ;
    BY scrampan date col1;
RUN;

data work.pan_7_85_7_01;
    set WORK.PAN_BEG_END_SORT;
    lag = substr(lag(col1),1,1);
    if col1='XB?' then col1='XB'||lag;
    else if col1='XE?' then col1='XE'||lag;
    drop lag;
run;

PROC SQL;
 CREATE TABLE WORK.PAN_7_85_7_01_Only AS SELECT PAN_7_85_7_01.scrampan FORMAT=$9.,
     PAN_7_85_7_01.col1,
     PAN_7_85_7_01.date FORMAT=MMYYS7.
 FROM WORK.PAN_7_85_7_01 AS PAN_7_85_7_01
 WHERE PAN_7_85_7_01.date BETWEEN '1jul1985'd AND '15jul2001'd
 ORDER BY PAN_7_85_7_01.scrampan, PAN_7_85_7_01.date;
QUIT;

data work.duration_PSTAT;
    set WORK.PAN_7_85_7_01_ONLY;
    by scrampan;
    lag_date=lag(date);
    format lag_date mmyys7.;
    if not first.scrampan then dur=intck('month',lag_date, date);
    if lag(col1) in  ('C01','XBC', 'XEC');
    if dur=. then dur=0;
run;

PROC SORT
    DATA=WORK.DURATION_PSTAT(KEEP=dur scrampan)
    OUT=WORK.SORTTempTableSorted
    ;
    BY scrampan;
RUN;

PROC MEANS DATA=WORK.SORTTempTableSorted
    NOPRINT
    CHARTYPE
    VARDEF=DF

           SUM
           N      ;
    VAR dur;
    BY scrampan;
OUTPUT     OUT=WORK.DURATION_Total(LABEL="Summary Statistics for
WORK.DURATION_PSTAT")

           SUM()=
           N()=

    / AUTONAME AUTOLABEL INHERIT
    ;
RUN;

PROC SQL;
 CREATE TABLE WORK.BetweenDates AS SELECT PSTAT_SQZ.scrampan FORMAT=$9.,
     PSTAT_SQZ.date FORMAT=MMYYS7.,
     PSTAT_SQZ.col1 FORMAT=$3.
 FROM VERT_SSI.PSTAT_SQZ AS PSTAT_SQZ
 WHERE PSTAT_SQZ.date BETWEEN '15jul1985'd AND '15jul2001'd
 ORDER BY PSTAT_SQZ.scrampan, PSTAT_SQZ.date;
QUIT;

data work.duration2;
    set WORK.BETWEENDATES;
    by scrampan;
    lag_date=lag(date);
    format lag_date mmyys7.;
    if not first.scrampan then dur=intck('month',lag_date, date);
    if lag(col1) = 'C01';
    if dur=. then dur=0;
run;

PROC SORT
    DATA=WORK.DURATION2(KEEP=dur scrampan)
    OUT=WORK.SORTTempTableSorted
    ;
    BY scrampan;
RUN;

PROC MEANS DATA=WORK.SORTTempTableSorted
    NOPRINT
    CHARTYPE
    VARDEF=DF

           SUM NONOBS    ;
    VAR dur;
    BY scrampan;

OUTPUT     OUT=WORK.MEANSummaryStatsDURATION2(LABEL="Summary Statistics for
WORK.DURATION2"
    drop=_type_
    )
```

```
            SUM()=

        / AUTONAME AUTOLABEL INHERIT
        ;
RUN;
RUN; QUIT;

PROC SQL;
 CREATE TABLE WORK.Query_for_Summary_Statistics_for AS SELECT
MEANSUMMARYSTATSDURATION2.scrampan FORMAT=$9.,
     MEANSUMMARYSTATSDURATION2._FREQ_,
     MEANSUMMARYSTATSDURATION2.dur_Sum,
     (( CASE WHEN 1 <= MEANSUMMARYSTATSDURATION2.dur_Sum THEN 1 ELSE
MEANSUMMARYSTATSDURATION2.dur_Sum END )) AS Recode_dur_Sum
 FROM WORK.MEANSUMMARYSTATSDURATION2 AS MEANSUMMARYSTATSDURATION2;
QUIT;
```

Since we are only accessing records which represent the CHANGE in status, as opposed to the value for each month, the computation time using the vertical file is significantly shorter than the time for the horizontal file.

| | | |
|---|---|---|
| **Results:** | Processing time for the horizontal dataset: | 59 seconds |
| | Processing time for the vertical dataset: | 6.6 seconds ---another clear winner! |

## USING PROC EXPAND TO RE-GENERATE THE HORIZONTAL FILE

If there is a reason to regenerate the horizontal file for STATUS, or any other variable, one can create the month-tp-month horizontal time series record using PROC EXPAND. One can ask proc expand to read the vertical file and, for each person, generate the time series of the value for STATUS for each month. Since PROC EXPAND only allows the variable to be numeric, one can create an integer variable which takes on specific integer values for each possible code value of STATUS. Then, choose the step function option so that PROC EXPAND does not interpolate between months. Format the resulting dataset with the correct STATUS codes for each integer value and you will have the horizontal, month-to-month values regenerated.

I tested to see if, for question 3, re-generating the horizontal file and counting the months in current pay status horizontally, as described earlier was as efficient as the calculations with the vertical file. It was not.

| | | |
|---|---|---|
| **Results:** | Processing time for the horizontal dataset: | 59 seconds |
| | Processing time for the vertical dataset: | 6.6 seconds |
| | Processing time for the PROC EXPAND Approach: | 16 seconds -- Not bad either! |

## CONCLUSION

The results presented above provide evidence that one should consider storing longitudinal data vertically as an event history, not horizontally as a time series. The result will be a savings in time AND space. If, for some reason, one needs the time series, one can use PROC EXPAND to generate it. There is a learning curve to the conversion. One must learn how to think differently about the algorithms that answer standard longitudinal questions. But, these new algorithms are more efficient and run in significantly less time than the code that answers the same question and runs against the horizontal time series file. In particular, the ability to easily interweave event histories about different events of interest by simply appending and sorting by person and date is well worth the effort. Further, since the vertical event history dataset is based on the key focus of analysis – events – the standard SAS PROCS can be used to process the data, instead of a data step with arrays and do-loops. Thus, the power of SAS can be better utilized to obtain the desired results.

## RECOMMENDED READING

Cody, Longitudinal Data and SAS

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John Hennessey
Office of Research, Evaluation, and Statistics
The Social Security Administration
6401 Security Blvd.
4-C-15 Operations Bldg.
Woodlawn, MD 21235
Work Phone: (410) 965-0102
E-Mail: John.C.Hennessey@ssa.gov