Paper 260-29

Starts and Stops: Processing Episode Data With Beginning and Ending Dates

Mike Rhoads, Westat, Rockville, MD

ABSTRACT

Regardless of the industry you work in, you will probably encounter data involving things that begin and end on particular dates. People open and close banking and credit card accounts, enroll in and graduate from schools, are hired for and resign from jobs, are admitted to and discharged from hospitals. Data files for these types of data typically include fields for person ID, beginning date, ending date, and various characteristics of the episode.

Answering questions involving this kind of data can be challenging for even experienced SAS® programmers, to say nothing of beginners. How many days was this person employed during the previous year? Which records in this file have time periods that are inconsistent with each other? Using data files with dates of insurance coverage and hospital stays, what insurance coverage does each patient have during each stay?

This tutorial will provide examples of how SAS DATA steps and PROC SQL can be used to solve typical programming problems related to episode data. The language constructs used in the examples will be explained so that relatively junior programmers should be able to follow and benefit from the examples. However, the techniques presented should also be useful to those with more SAS experience.

INTRODUCTION

This paper is somewhat different from most SAS conference tutorials. The typical approach is to focus on a set of SAS tools: perhaps a particular PROC, set of functions, or debugging methods. Since one of the ways we improve as SAS programmers is to expand our knowledge of the tools that are available to us, such presentations can be extremely useful.

However, merely having a wide repertoire of tools is not in itself enough to make someone a successful programmer, whether using SAS or other computer software. We need to be able to approach real-world problems in a systematic fashion, consider what tools are appropriate for solving them, and then design and develop a program that meets the specifications. (Needless to say, documentation, testing, and debugging are also critical parts of this process.)

The main portion of this paper starts by taking a look at two small data sets. We will then work through two examples of typical tasks we might need to perform using such data. For each example, we begin by stating the task, summarizing the output that is required, and listing some points that need to be considered when attacking the problem. We then describe a basic approach for solving the problem. Finally, we present and discuss the SAS code needed to implement our solution. The code discussion includes brief descriptions of SAS tools and techniques used that may be unfamiliar to a significant number of SAS programmers. The first example is one that can be solved with a single SAS step, while the second requires multiple steps and considerably more code.

I decided to focus on episode data for a number of reasons. I've had considerable personal experience over the years dealing with health insurance and medical expenditures data. While there are many people who work in this area, I also realized that data involving periods of time are common in many other industries as well. Questions involving this type of data are also quite common on SAS-L. For the uninitiated, these problems can often be difficult to tackle. SAS, however, does offer a wide range of tools that can be used to come up with creative and understandable solutions to such problems. Hopefully this paper, by systematically working through examples of such problems and their solutions, will enable you to more easily handle similar problems that you face in the future.

A FEW QUICK NOTES ON SAS DATES

This paper is certainly not intended to be a comprehensive discussion of all the tools that SAS has available for handling dates. You can get a good general introduction to this subject in the "Dates, Times, and Intervals" chapter of *SAS Language Reference: Concepts*. A few of the many excellent SAS conference papers discussing related issues and techniques are listed in the References section at the end of this paper. However, I decided to include a very brief introduction here for the benefit of anyone who has not worked with dates in SAS before (and as a refresher for others).

SAS does not actually have a built-in "date" type variable. Typically, however, a date value is stored in a single numeric SAS variable, and the many date-related functions, informats and formats assume that this is the case. The date value is an

integer that represents the number of days since January 1, 1960. So, January 1, 1960 is represented by a value of 0. January 2, 1960 has a value of 1, while December 31, 1959 is represented as -1.

This representation has several important consequences for solving the examples we will see later in this paper. First, it is easy to compare two dates, since the later date will have the larger numeric value. Second, you can use date values in basic arithmetic operations (addition and subtraction). You can subtract one date value from another to get the number of days between them. You can also add or subtract a numeric value (number of days) from a date value, resulting in a new date value which is that number of days later or earlier than the initial date.

You can use "constant" date values in your SAS programs, just as you can specify numeric or character constants (literals). You write a date constant by specifying the day of the month, followed by the 3-letter month abbreviation and the year. (This corresponds to the DATE. informat.) Enclose the result in quotes, immediately followed by a D to indicate that the string is a date constant rather than a string constant. For example, '1JAN1960'D. You may use single or double quotes, and the D and the month abbreviation are case-insensitive. If you include only two digits for the year (not recommended), the value of the YEARCUTOFF system option is used to determine the century.

Always assign a format when you are working with variables holding date values, so the date will be displayed in a meaningful way. SAS offers a wide range of date formats (and if none of these meet your needs you can take advantage of date-time "directives" in PROC FORMAT to completely customize the display of your date values). Good general choices include YYMMDD10 and DATE9. Be sure to specify a length with the format, since many of the SAS defaults display 2-digit rather than 4-digit years. To save some space in this paper and the accompanying slides, I decided to use MMDDYY5 as the format for my date values. This displays numeric month and day values separated by a slash (e.g. 12/31 for December 31). The year is not shown, which obviously only makes sense if you are certain that all the dates you are working with are within the same year.

THE DATA

The data for our examples come from a fictitious study of the health insurance coverage of 8 people during 2002. Each of the 8 persons in the study has one record in the Demographics data set, whose variables are PersID, Gender, and Age. The second data set is InsCovData, which contains one record for each reported health insurance plan. The variables on InsCovData are PersID, PlanNum (a sequential identifier within PersID), PlanType (1=Medicare, 2=Medicaid, 3=Military, 4=Private), PlanStart, and PlanEnd. Persons with no health insurance coverage during 2002 (such as person #7 in our data) do not have any records in InsCovData. Plan records may overlap, and there may be periods that are not covered by any plan record.

PersID	PlanNum	PlanType	PlanStart	PlanEnd
1	1	Medicare	01/01	12/31
2	1	Medicaid	01/01	01/31
2	2	Medicaid	09/01	11/30
3	1	Private	01/01	12/31
3	2	Medicare	10/01	12/31
4	1	Military	07/01	12/31
5	1	Private	01/01	10/31
5	2	Private	10/01	11/30
5	3	Private	12/01	12/31
6	1	Private	01/01	08/22
6	2	Medicaid	11/01	12/31
8	1	Military	01/01	02/15
8	2	Private	02/01	06/30
8	3	Medicaid	09/01	11/30
8	4	Private	12/15	12/31
8	5	Private	12/17	12/31

To keep the programs in the remainder of the paper from getting too complicated, we will specify that our data have the following characteristics:

1. For each record, PlanEnd is greater than or equal to PlanStart.
2. All values for PlanStart and PlanEnd are in 2002.
3. There are no missing values in the data.
4. Both data sets are sorted by PersID.
5. On InsCovData, plan records for each person are ordered by PlanStart and PlanEnd.

EXAMPLE 1 -- SUMMARIZING EPISODES

THE TASK: For each person in the study, determine how many days during 2002 they had some health insurance coverage, and how many days they had no coverage.

THE OUTPUT: We just need summary counts at the person level, so the output file should have one record for each person in the study. Keep the PERSID variable to identify the person, and create two new variables: DaysCovered and DaysUncovered.

PersID	DaysCovered	DaysUncovered
1	365	0
…	…	…
7	0	365
8	289	76

POINTS TO CONSIDER

- Persons who did not have any health insurance during 2002 will not have any records in the InsCovData file. Therefore, we also need to use the Demographics file so these persons are included in the output file.
- At any given time, a person may be covered by more than one insurance plan. Therefore, we cannot simply add the number of days covered by each plan to get a total DaysCovered for the year. We have to make sure a particular day of coverage is counted only once.

THE APPROACH

- In a DATA step, MERGE the Demographics file and the InsCovData file by PersID.
- Divide the logic of the step into three phases: (1) a group of statements to be executed when beginning to process the input records for a person, (2) a group of statements to be executed for each incoming insurance coverage record, and (3) a group of statements to be executed after processing all the records for a person.
- Create a temporary variable, LatestCoverageDate, so we can avoid double-counting any coverage periods.

1.1 Preliminaries

```
%LET FirstDateOfInterest = '01JAN2002'D;
%LET LastDateOfInterest  = '31DEC2002'D;
DATA AnyCoverageSummary
  (KEEP = PersID DaysCovered DaysUncovered
  LABEL =
  "1 record for each person in the study.");
MERGE MYLIB.Demographics
      MYLIB.InsCovData (IN=In_InsCov);
BY PersID;
LABEL
  DaysCovered = '# days covered'
  DaysUncovered =
    '# days with no insurance coverage'
  LatestCoverageDate =
    'Latest coverage date so far for this
person (temp var)'
;
LENGTH DaysCovered DaysUncovered 3;
FORMAT LatestCoverageDate MMDDYY5.;
RETAIN DaysCovered LatestCoverageDate;
```

- Our program will need to know the starting and ending dates of the period we are concerned with (calendar year 2002 in this example). In SAS, we use **date constants** to express fixed dates. These are written in the form '01JAN2002'D, where the part between the quotes is a date written in the standard SAS format, and the "D" at the end tells SAS that this is a date constant and not a character string.
- We could put these date constants deep in the middle of our program, but since we may need to use them more than once, and because we may need to change them in a later run of the program, we assigned them to **macro variables** with a **%LET statement**. When a macro variable is used later in the program, SAS replaces the macro variable reference with its value.
- The **DATA statement** starts our DATA step and creates the SAS data set AnyCoverageSummary. We also use two **data set options**: **KEEP=** to list the variables that should go into the data set, and **LABEL=** to provide additional documentation for it.
- The **MERGE statement** and **BY statement** tell SAS to combine data from the Demographics and InsCovData files, using the variable PersID to link the files. The **IN= data set option** tells SAS to set the value of In_InsCov to 1 if an InsCovData record has been read for this BY group, and to 0 if it has not.
- The **LABEL**, **LENGTH**, and **FORMAT** statements assign various attributes to our variables. Be sure to always

assign an appropriate date format to all date variables! Otherwise the date will print out as a difficult-to-interpret number.

- The **RETAIN** statement tells SAS that we want to hold on to the values of DaysCovered and LatestCoverageDate. Without this statement, SAS would reset them to missing each time it started a new iteration of the DATA step.

1.2 Beginning-of-person processing

```
IF FIRST.PersID THEN DO;
  DaysCovered = 0;
  LatestCoverageDate = .;
END;
```

- Because we listed PersID in our BY statement, SAS automatically creates the temporary variables **FIRST.PersID** and **LAST.PersID**, which we can use in the DATA step. We have a group of one or more records for each person. When we are at the first record for a person SAS sets FIRST.PersID to 1. If it is not the first record in the group, SAS sets FIRST.PersID to 0. LAST.PersID is similar, except that it is set to 1 for the last record in the group rather than the first.
- So, we can use an **IF … THEN** to execute one or more statements only when we are at the beginning of a person. We want to start DaysCovered at 0 so that we can add to it as we move through the records for this person. We are using the LatestCoverageDate variable to hold the latest insurance coverage date we have found for a person. We want to set this to missing here so we don't carry over the value from the previous person in the file.

1.3 Coverage record processing

```
IF In_InsCov THEN DO;
  IF PlanStart > LatestCoverageDate THEN DO;
    DaysCovered = DaysCovered +
      (PlanEnd - PlanStart + 1);
  END;
  ELSE IF PlanEnd > LatestCoverageDate THEN DO;
    DaysCovered = DaysCovered +
      (PlanEnd - (LatestCoverageDate + 1) + 1);
  END;
  LatestCoverageDate =
    MAX(LatestCoverageDate,PlanEnd);
END;
```

- We now want to process the insurance coverage record. The test IF In_InsCov (equivalent to IF In_InsCov = 1) makes sure that we have an insurance coverage record to process – if the current person did not have any insurance coverage during 2002, he will not have a record in that file. (The code would actually work correctly without this test, although including it makes the program clearer.)
- Now, we compare the PlanStart and PlanEnd of this plan to the latest coverage date we have previously encountered for this person, in order to determine how many coverage days from this plan are "new" and thus should be added to the total days of coverage for the year.
- If this plan started after the latest previously encountered coverage date (IF PlanStart > LatestCoverageDate), we should add all of the coverage days from this plan to the total.
- Since we are counting a person as "covered" on both the starting and ending dates of coverage, the days of coverage for the plan is PlanEnd – PlanStart + 1. Thus, if both the starting and ending dates were the same, we would consider the person covered for one day.
- For plans that don't meet this first test, we check whether the end date for the plan is after the latest previously encountered coverage date (If PlanEnd > LatestCoverageDate).
- If this is the case, the period from the day AFTER the latest previous coverage date through the end date for this period must be added to the total coverage days (PlanEnd - (LatestCoverageDate + 1) + 1).
- Otherwise, this plan must have ended prior to the latest previously encountered coverage date, and is not contributing any new days of coverage.
- Finally, we set the LatestCoverageDate to the later of its previous value and the ending date for this plan, using the **MAX function**.

1.4 End-of-record processing

```
IF LAST.PersID THEN DO;
  DaysUncovered = (&LastDateOfInterest -
&FirstDateOfInterest + 1) - DaysCovered;
  OUTPUT;
END;
RUN;
```

- We use the **automatic variable LAST.PersID** to test whether we are at the end of the group of records for this person.
- We have already determined DaysCovered by adding to it as we go. We can get the number of days without coverage by subtracting DaysCovered from the total number of days in the period (the year 2002).
- For the starting and ending dates of the entire period, we use the **macro variables** that we set earlier (&FirstDateOfInterest and &LastDateOfInterest).
- Finally, we write out the record for this person with the **OUTPUT statement**.

EXAMPLE 2 -- CONSTRUCTING A COVERAGE TIMELINE

THE TASK: Construct a health insurance timeline for each person for 2002, so we can easily tell how many plans of each type he or she was covered by during each distinct period of time.

THE OUTPUT: The output file should contain the variables PersID, PeriodStart, PeriodEnd, and a set of variables denoting the numbers of plans of each type the person was covered by during the period (MedicarePlanCount, MedicaidPlanCount, MilitaryPlanCount, PrivatePlanCount). For persons who had the same insurance coverage during the entire year, we will only need one record in the timeline file. For those whose coverage changed at some point, we will need one record for each distinct period of coverage – that is, each period where at least one of the count variables is different from the previous period.

SAMPLE INPUT (PersID #5)

PlanType	PlanStart	PlanEnd
Private	01/01	10/31
Private	10/01	11/30
Private	12/01	12/31

SAMPLE OUTPUT (PersID #5)

Period Start	Period End	Medicare PlanCount	Medicaid PlanCount	Military PlanCount	Private PlanCount
01/01	09/30	0	0	0	1
10/01	10/31	0	0	0	2
11/01	12/31	0	0	0	1

POINTS TO CONSIDER

- We need a new record in the timeline file each time the person's coverage status changes. The coverage status potentially changes on the day a plan starts, and on the day after a plan ends.
- Sometimes one plan ending and another plan starting does not indicate a coverage status change. For instance, coverage under one private plan may end on one day, and coverage under a different private plan may start the next day. Both before and after this change, the person was covered by one private plan.

THE APPROACH

- Start by getting all of the potential status change dates for each person into a temporary file. These include the first day of the year, the starting date of each plan, and the day after each plan ends.
- Sort this temporary file by person and date, and eliminate duplicate records.
- We want to get the ending date for each status period, which is the day before the next period starts. We can do this in a DATA step by doing a "lookahead merge" of the starting date file constructed above with itself.
- Now we can use PROC SQL to join the status period file we just created with our original insurance coverage file to get counts by plan type for each status period.

- Finally, we use a DATA step to combine consecutive records with the same coverage into a single record.

2.1 Get status change dates

```
DATA StatusChangeDates
  (KEEP = PersID StatusChangeDate);
MERGE MYLIB.Demographics
      MYLIB.InsCovData (IN=In_InsCov);
BY PersID;
LABEL
  StatusChangeDate =
    'Date insurance status potentially changed'
;
LENGTH StatusChangeDate 6;
FORMAT StatusChangeDate MMDDYY5.;
* Make sure we put out the first date of the
  period of interest;
IF FIRST.PersID THEN DO;
  StatusChangeDate = &FirstDateOfInterest;
  OUTPUT;
END;
IF In_InsCov THEN DO;
  * Output a record using start date for this
    plan as StatusChangeDate;
  StatusChangeDate = PlanStart;
  OUTPUT;
  * Output a record using day after this plan
    ended as StatusChangeDate
    (unless plan ran through end of period);
  IF PlanEnd ^= &LastDateOfInterest THEN DO;
    StatusChangeDate = PlanEnd + 1;
    OUTPUT;
  END;
END;
RUN;
```

- We only need two variables in the output data set from this step – the person ID (PersID) and the potential coverage status change date (StatusChangeDate).
- Most of the information we need comes from InsCovData. However, we need to merge this file with the Demographics file so we don't leave out persons with no coverage.
- In the final timeline file, the first record for each person needs to start at the beginning of the period (January 1, 2002). Therefore, as we begin to process each person (IF FIRST.PersID), we create and write out a record that has StatusChangeDate set to the macro variable &FirstDateOfInterest.
- Now, we want to process each plan coming from InsCovData. We check the IN= variable first (IF In_InsCov) to be sure that we have a record from that file (in other words, not a person that has no coverage).
- In most cases, we want to write out two records for each plan: one using PlanStart as the StatusChangeDate, and the other using PlanEnd + 1 as the StatusChangeDate. Before creating and writing the second record, however, we check to be sure that the plan's coverage does not run through the end of the year (IF PlanEnd ^= &LastDateOfInterest). If it does, PlanEnd + 1 would be outside the period of interest, and we do not want a record for it in StatusChangeDates.

2.2 Sort by person/date and eliminate duplicates

```
PROC SORT  DATA=StatusChangeDates
  OUT=SortedStatusChangeDates  NODUPKEY;
BY PersID StatusChangeDate;
RUN;
```

- You need to be careful when using the **NODUPKEY** and **NODUPREC** options in PROC SORT to eliminate duplicates from a file. You can rely on things going as you expect when all of the variables on the file are part of the sort key (listed on the BY statement).

2.3 Get status period starting and ending dates

```
OPTIONS MERGENOBY = NOWARN;
DATA TimelineDates
  (KEEP = PersID PeriodStart PeriodEnd);
MERGE
  SortedStatusChangeDates (RENAME =
    (StatusChangeDate=PeriodStart))
  SortedStatusChangeDates (FIRSTOBS=2
    RENAME = (PersID=NextPersID
    StatusChangeDate=NextStartDate))
;
* BY statement intentionally omitted;
LABEL
  PeriodStart = 'Period start date'
  PeriodEnd   = 'Period end date'
;
LENGTH PeriodStart PeriodEnd 6;
FORMAT PeriodStart PeriodEnd MMDDYY5.;
IF PersID = NextPersID
  THEN PeriodEnd = NextStartDate - 1;
ELSE   PeriodEnd = &LastDateOfInterest;
```

- For this step, we want to use our file of status change dates to "slice" the period of interest (2002) into a set of time periods, each of which uses one status change date as its starting date, and the day before the next status change date as its ending date. The output data set, TimelineDates, should contain the variables PersID, PeriodStart, and PeriodEnd.
- So, in order to construct the output record whose beginning date is the status change date from the **current** record, we need the status change date from the **next** record for that person to determine the ending date of the current period.
- A good way of solving such problems in SAS is with a **lookahead merge**, which merges two instances of the same dataset. Specify the **FIRSTOBS=2** data set option on the second instance, so the second record from instance 2 of the data set is read at the same time as the first record from instance 1, and so on through the file. The table below illustrates this, with the shading within the body showing how a record is first read from instance 2 of the input data set, and then again from instance 1 the next time through:

```
RUN;
OPTIONS MERGENOBY = ERROR;
```

	Instance 1		Instance 2	
_N_	PersID	Status ChangeDate	PersID	Status ChangeDate
1	1	01/01	2	01/01
2	2	01/01	2	02/01
3	2	02/01	2	09/01
4	2	09/01	2	12/01
5	2	12/01	3	01/01
…	…	…	…	…
24	8	12/15	8	12/17
25	8	12/17	.	.

- The lookahead merge works by merging each record in the data set with the one that follows it, rather than by using one or more variables to link the files. Normally when doing a SAS DATA step merge, we do want to specify the variables that link the records in the different files, which we indicate by using a BY statement. A common error in SAS programs is to accidentally forget to put in the BY statement when doing a merge.
- For that reason, SAS provides the system option **MERGENOBY**. You can use this to have SAS issue a warning or error message if a DATA step uses a MERGE statement without a BY statement. Here, since we are deliberately leaving out the BY statement, we specify **OPTIONS MERGENOBY=NOWARN** to tell SAS (and anyone who reads the program) that the omission is intentional. At the end of the step, we set MERGENOBY to ERROR so that SAS will stop processing if we unintentionally leave out the BY statement in a later DATA step merge.
- When we merge a data set with itself, both instances of the data set start out with the same variable names. In order to use both sets of variables, we use the **RENAME= data set option** on each input data set. The effect of this is shown below:

Instance 1	Instance 2
PersID	~~PersID~~ NextPersID
~~StatusChangeDate~~ PeriodStart	~~StatusChangeDate~~ NextStartDate

- Once we set up the lookahead merge and rename variables, the logic is very simple. As long as the "current" and "next" status change dates are for the same person (PersID = NextPersID), we want to set the end of the period to the day before the next period starts (NextStartDate – 1). When this condition is not true, it means that we are at the last record for this person, and the ending date for this last period should be set to the last date of the period of interest.(PeriodEnd = &LastDateOfInterest).

2.4 Get counts by plan type for each period

```
PROC SQL;
CREATE TABLE UncollapsedInsuranceTimeline
AS
SELECT
  TL.PersID,
  PeriodStart,
  PeriodEnd,
  SUM
```

- Now we need to join the file of timeline dates we just constructed with our original file of coverage dates for each insurance plan. **PROC SQL** is a good tool for situations such as this, where the join must deal with ranges of values – here, starting and ending dates – rather than single values.
- We use **CREATE TABLE** to create our output table, UncollapsedInsuranceTimeline, using the specifications in

```
    (CASE PlanType WHEN 1 THEN 1 ELSE 0 END)
      AS MedicarePlanCount,

  SUM
    (CASE PlanType WHEN 2 THEN 1 ELSE 0 END)
      AS MedicaidPlanCount,
  SUM
    (CASE PlanType WHEN 3 THEN 1 ELSE 0 END)
      AS MilitaryPlanCount,
  SUM
    (CASE PlanType WHEN 4 THEN 1 ELSE 0 END)
      AS PrivatePlanCount
FROM
  TimelineDates AS TL
  LEFT JOIN MYLIB.InsCovData AS PLAN
  ON TL.PersID = PLAN.PersID
  AND NOT
    (PeriodStart > PlanEnd
    OR PlanStart > PeriodEnd)
GROUP BY TL.PersID, PeriodStart, PeriodEnd
ORDER BY TL.PersID, PeriodStart, PeriodEnd
;
QUIT;
```

the SELECT clause.

- The **SELECT clause** begins by listing the variables we want in our output table. The first three are PersID, PeriodStart, and PeriodEnd. Since more than one of the incoming tables have a PersID column, we must use the **alias** for one of the tables to indicate which PersID we want. (TL is assigned as an alias for the TimelineDates table in the FROM clause below.) Note that this is unlike SAS DATA step processing, where columns from different data sets but having the same name are combined into a single Program Data Vector variable.
- We want our table to contain, for each period, a count of how many insurance plans of each type are in effect (the variables MedicarePlanCount, MedicaidPlanCount, MilitaryPlanCount, and PrivatePlanCount). We will come back to the code for these variables after we look at the FROM and GROUP BY clauses.
- The **FROM clause** tells PROC SQL what tables are going to be used and how to join them. The two tables used here (there could be more than two) are TimelineDates (to which we assign the alias TL) and MYLIB.InsCovData (to which we assign the alias PLAN).
- The **LEFT JOIN** means that we want to keep all of the rows from the left-hand table coming in (TimelineDates), even if they do not match up to anything on the right side. This is unlike a SAS DATA step merge, where all of the rows are kept by default. We want to do a left join because we don't want to lose persons with no insurance coverage (who don't have any records in InsCovData).
- The **ON clause** tell s PROC SQL what conditions have to be met in order for rows from the two incoming tables to be joined.
- The first condition to be met, of course, is that the records must be for the same person: TL.PersID = PLAN.PersID. Note how we use the table alias values to indicate which table we are talking about.
- Even with records for the same person, we only want to join records where the time periods overlap. It is actually a little clearer to specify when the records don't overlap: they don't overlap when the "TL" record starts after the "PLAN" record ends (PeriodStart > PlanEnd), or vice versa (PlanStart > PeriodEnd).
- So, you can think about the ON clause as saying to join a pair of records when (1) they are for the same person, and (2) it's not the case that they don't overlap.
- The **GROUP BY clause** indicates that the result should have only one row for each combination of the listed variables: here, PersID, PeriodStart, and PeriodEnd. For our task, we want to wind up with just a single output record for each timeline period (defined by PeriodStart and PeriodEnd). Without the GROUP BY clause, there would be one row produced for each insurance plan that overlapped the period.
- The **ORDER BY clause** specifies the order for the rows in our output table. Here, we want them ordered by PersID, PeriodStart, and PeriodEnd.
- Now let's go back to the list of variables at the beginning of the **SELECT**. Notice that there are two types of variables listed here. First, we have the same columns that were listed in the GROUP BY: TL.PersID, PeriodStart, and PeriodEnd. The remaining variables, the ones that hold the counts of the different types of plan, are specified using the **SUM aggregate function**. This causes the expression within parentheses to be summed across all of the rows in the group. (Other aggregate functions in SQL include COUNT, MIN, MEAN, and MAX.) The **AS** at the end gives the computed column a name.
- The argument to the SUM function is a **CASE expression**, which always begins with CASE and ends with END. This serves the same function in SQL that a SELECT block or a series of IF-THEN-ELSEs does in a DATA step.
- To get the MedicarePlanCount variable, we look at the

PlanType variable for each pair of joined records in the group. If it has a value of 1, which indicates Medicare, count a 1 for that pair, else count a 0. Summed across all of the plan records that overlap this timeline period, that gives us a count of the Medicare plans in effect. (In actual practice, for insurance types such as Medicare, there should not be more than one record for any given time period.)

- We then do the same thing for the other three "plan count" variables, checking for the corresponding value of PlanType in each case.
- Finally, think for a minute about what happens for the person with no insurance coverage. There will be a record in the result set for that person, since we were careful to do a LEFT JOIN. Since there are no corresponding rows in InsCovData, the variables from that table, such as PlanType, will be missing (NULL in SQL parlance). Therefore, since PlanType is not a 1, 2, 3, or 4, none of the count variables will get incremented, and each will wind up with a value of zero.

2.5 Combine consecutive records with identical coverage

```
DATA MYLIB.FinalInsuranceTimeline;
SET UncollapsedInsuranceTimeline;
BY PersID MedicarePlanCount MedicaidPlanCount
  MilitaryPlanCount PrivatePlanCount NOTSORTED;
LABEL HoldPeriodStart =
  'Start date for first record in set (temp)';
FORMAT HoldPeriodStart MMDDYY5.;
DROP HoldPeriodStart;
RETAIN HoldPeriodStart;
IF FIRST.PrivatePlanCount THEN DO;
  HoldPeriodStart = PeriodStart;
END;
IF LAST.PrivatePlanCount THEN DO;
  PeriodStart = HoldPeriodStart;
  OUTPUT;
END;
RUN;
```

- We are very close to what we want, but not quite there. We may have situations where two adjacent timeline periods for the same person have the same set of counts for each plan type: this will happen when a person's coverage for one plan stops, and the next day a different plan of the same type goes into effect. We can see this with person #5 in our file. One private plan ends on 11/30, and another starts on 12/01:

PlanType	PlanStart	PlanEnd
Private	01/01	10/31
Private	10/01	11/30
Private	12/01	12/31

- When we look at the UncollapsedInsuranceTimeline records for this person, we see that the last two records have identical values for all the counts (hence the same "coverage status"):

Period Start	Period End	Medicare PlanCount	Medicaid PlanCount	Military PlanCount	Private PlanCount
01/01	09/30	0	0	0	1
10/01	10/31	0	0	0	2
11/01	11/30	0	0	0	1
12/01	12/31	0	0	0	1

- What we want to do is to identify all adjacent groups of records for the same person where all the counts are identical. For each group, we then want to put out a single record, using the starting date of the first record in the group for PeriodStart, and the ending date of the last record in the group for Period End. Records that do not have the same pattern of count variables as the previous or next record will go out unchanged: we can simplify the code by thinking of these as single-record groups. For person #5, we thus want to have 3 records in the output file rather than 4:

Period Start	Period End	Medicare PlanCount	Medicaid PlanCount	Military PlanCount	Private PlanCount
01/01	09/30	0	0	0	1
10/01	10/31	0	0	0	2
11/01	12/31	0	0	0	1

- As we saw earlier, **FIRST and LAST variables** are a great

DATA step tool for identifying and processing groups of records. Here we specify PersID and the four plan count variables in the BY statement, since we want to group all consecutive records where there are no changes in this set of variables.

- Note that we do NOT specify PeriodStart and PeriodEnd in the BY statement, even though these are part of the sort order of the file. We omit these because their values are always different from one record to the next, and so they would prevent us from identifying the groups that we want.
- Since we are leaving PeriodStart and PeriodEnd off of the BY statement, the incoming records are not necessarily sorted in the order of the BY statement variables. For instance, the 3rd record for our person #5 would actually sort before the 2nd record, if we ignore the dates. Normally the BY statement requires the input data set(s) to be sorted, but here we can use the **NOTSORTED option** on the BY statement to tell SAS that we just want to group records, and so we are interested in when the BY variables change whether the sort order is correct or not. This option is not allowed in situations where more than one input data set is involved.
- We create and RETAIN the variable HoldPeriodStart. Since we don't want it in our output data set, we also list it in a **DROP statement**.
- We identify the beginning of each group by checking the variable FIRST.PrivatePlanCount. Since PrivatePlanCount is the rightmost variable on the BY statement, this will be set to one whenever any of the BY variables changed its value from the previous record to the current one. At the start of a group, we want to save the PeriodStart value for this record into HoldPeriodStart, so that we can use it when we get to the end of the group.
- Similarly, we identify the last record in each group by checking the value of LAST.PrivatePlanCount. At that point we write out a record, using PeriodEnd from the current record and PeriodStart from the first record in the group (by putting the retained value of HoldPeriodStart back into PeriodStart).
- Note that we don't do anything with records that are in the MIDDLE of a group (both FIRST.PrivatePlanCount and LAST.PrivatePlanCount are zero).

CONCLUSION

Problems involving episode data can be among the most challenging to face us as SAS professionals. As we have seen, however, SAS provides a wide variety of techniques that you can use to conquer such tasks. Start by understanding the problem, the data set(s) that you will be using, and exactly what your final results need to be. Then come up with an overall approach, taking into account the tools that you have available and any special aspects of your problem that need to be considered. Once you've gotten to this point, you can develop, test and debug your solution.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

REFERENCES

Chakravarthy, Venky (2002), " Have a Strange DATE? Create your own INFORMAT to Deal with Her," *Proceedings of the Twenty-Seventh Annual SAS® Users Group International Conference.*
www2.sas.com/proceedings/sugi27/p101-27.pdf

Gilsen, Bruce (2002), "Date Handling in the SAS System," *NorthEast SAS Users Group 15th Annual Conference Proceedings,* 200-209.
www.nesug.org/Proceedings/nesug02/bt/bt003.pdf

Karp, Andrew (2000), "Working With SAS Date and Time Functions," *NorthEast SAS Users Group 13th Annual Conference Proceedings,* 197-203.
www.nesug.org/Proceedings/nesug00/bt/bt3007.pdf

Satchi, Thiru, and Edgar Mounib (2001), "Processing Dates in Health Care Enrollment Data," *NorthEast SAS Users Group 14th Annual Conference Proceedings,* 335-337.
www.nesug.org/Proceedings/nesug01/cc/cc4017.pdf

Whitlock, H. Ian (1999), "Managing the INTNX Function," *Proceedings of the 1999 SouthEast SAS Users Group Conference.*

CONTACT INFORMATION

If you have any questions or comments about the paper, you can reach the author at:

Mike Rhoads
Westat
1650 Research Blvd.
Rockville, MD 20850
RhoadsM1@Westat.com

APPENDIX

This appendix consolidates into one SAS program all of the code needed to create the sample data sets and run the two examples.

```
/***** ASSIGN LIBNAMES, CREATE FORMATS AND DATASETS *****/

LIBNAME  MYLIB     "C:\";
LIBNAME  LIBRARY   "C:\";

PROC FORMAT  LIBRARY=LIBRARY;
VALUE PlanTypF
  1 = 'Medicare'
  2 = 'Medicaid'
  3 = 'Military'
  4 = 'Private'
;
VALUE GenderF
  1 = 'Male'
  2 = 'Female'
;
RUN;

DATA MYLIB.Demographics (LABEL =
  "Demographic data.  One record per person.  Unique ID is PersID.");
LABEL
  PersID     = 'Person ID'
  Gender     = 'Gender'
  Age        = 'Age'
;
LENGTH  PersID 4  Gender Age 3;
FORMAT  Gender GenderF.;
INPUT PersID Gender Age;
CARDS;
1 2 72
2 1 46
3 1 65
4 2 23
```

```
5 1 27
6 1 44
7 2 25
8 1 30
RUN;

DATA MYLIB.InsCovData (LABEL =
  "Insurance coverage data.  One record for each reported period of
 coverage.  Unique ID is PersID/PlanNum.");
LABEL
  PersID     = 'Person ID'
  PlanNum    = 'Plan number'
  PlanType   = 'Type of plan'
  PlanStart  = 'Start date of this plan'
  PlanEnd    = 'Ending date of this plan'
;
LENGTH  PersID 4  PlanNum PlanType 3  PlanStart PlanEnd 6;
INFORMAT PlanStart PlanEnd YYMMDD10.;
FORMAT  PlanType PlanTypF.  PlanStart PlanEnd MMDDYY5.;
INPUT PersID PlanNum  PlanType PlanStart PlanEnd;
CARDS;
 1 1 1 2002-01-01 2002-12-31
 2 1 2 2002-01-01 2002-01-31
 2 2 2 2002-09-01 2002-11-30
 3 1 4 2002-01-01 2002-12-31
 3 2 1 2002-10-01 2002-12-31
 4 1 3 2002-07-01 2002-12-31
 5 1 4 2002-01-01 2002-10-31
 5 2 4 2002-10-01 2002-11-30
 5 3 4 2002-12-01 2002-12-31
 6 1 4 2002-01-01 2002-08-22
 6 2 2 2002-11-01 2002-12-31
 8 1 3 2002-01-01 2002-02-15
 8 2 4 2002-02-01 2002-06-30
 8 3 2 2002-09-01 2002-11-30
 8 4 4 2002-12-15 2002-12-31
 8 5 4 2002-12-17 2002-12-31
RUN;

/***** EXAMPLE 1 -- SUMMARIZING EPISODES *****/

/* 1.1  Preliminaries */
%LET FirstDateOfInterest = '01JAN2002'D;
%LET LastDateOfInterest  = '31DEC2002'D;
DATA AnyCoverageSummary
  (KEEP = PersID DaysCovered DaysUncovered
  LABEL =
  "1 record for each person in the study.");
MERGE MYLIB.Demographics
      MYLIB.InsCovData (IN=In_InsCov);
BY PersID;
LABEL
  DaysCovered = '# days covered'
  DaysUncovered =
    '# days with no insurance coverage'
  LatestCoverageDate =
    'Latest coverage date so far for this person (temp var)'
;
LENGTH DaysCovered DaysUncovered 3;
FORMAT LatestCoverageDate MMDDYY5.;
RETAIN DaysCovered LatestCoverageDate;
/* 1.2  Beginning-of-person processing */
IF FIRST.PersID THEN DO;
  DaysCovered = 0;
  LatestCoverageDate = .;
END;
/* 1.3  Coverage record processing */
IF In_InsCov THEN DO;
```

```
  IF PlanStart > LatestCoverageDate THEN DO;
    DaysCovered = DaysCovered +
      (PlanEnd - PlanStart + 1);
  END;
  ELSE IF PlanEnd > LatestCoverageDate THEN DO;
    DaysCovered = DaysCovered +
      (PlanEnd - (LatestCoverageDate + 1) + 1);
  END;
  LatestCoverageDate =
    MAX(LatestCoverageDate,PlanEnd);
END;
/* 1.4  End-of-record processing */
IF LAST.PersID THEN DO;
  DaysUncovered = (&LastDateOfInterest - &FirstDateOfInterest + 1) - DaysCovered;
  OUTPUT;
END;
RUN;

/***** EXAMPLE 2 -- CONSTRUCTING A COVERAGE TIMELINE *****/

/* 2.1  Get status change dates */
DATA StatusChangeDates
  (KEEP = PersID StatusChangeDate);
MERGE MYLIB.Demographics
      MYLIB.InsCovData (IN=In_InsCov);
BY PersID;
LABEL
  StatusChangeDate =
    'Date insurance status potentially changed'
;
LENGTH StatusChangeDate 6;
FORMAT StatusChangeDate MMDDYY5.;
* Make sure we put out the first date of the
  period of interest;
IF FIRST.PersID THEN DO;
  StatusChangeDate = &FirstDateOfInterest;
  OUTPUT;
END;
IF In_InsCov THEN DO;
  * Output a record using start date for this
    plan as StatusChangeDate;
  StatusChangeDate = PlanStart;
  OUTPUT;
  * Output a record using day after this plan
    ended as StatusChangeDate
    (unless plan ran through end of period);
  IF PlanEnd ^= &LastDateOfInterest THEN DO;
    StatusChangeDate = PlanEnd + 1;
    OUTPUT;
  END;
END;
RUN;

/* 2.2  Sort by person/date and eliminate duplicates */
PROC SORT  DATA=StatusChangeDates
  OUT=SortedStatusChangeDates  NODUPKEY;
BY PersID StatusChangeDate;
RUN;

/* 2.3  Get status period starting and ending dates */
OPTIONS MERGENOBY = NOWARN;
DATA TimelineDates
  (KEEP = PersID PeriodStart PeriodEnd);
MERGE
  SortedStatusChangeDates (RENAME =
    (StatusChangeDate=PeriodStart))
  SortedStatusChangeDates (FIRSTOBS=2
    RENAME = (PersID=NextPersID
```

```
    StatusChangeDate=NextStartDate))
;
* BY statement intentionally omitted;
LABEL
  PeriodStart = 'Period start date'
  PeriodEnd   = 'Period end date'
;
LENGTH PeriodStart PeriodEnd 6;
FORMAT PeriodStart PeriodEnd MMDDYY5.;
IF PersID = NextPersID
  THEN PeriodEnd = NextStartDate - 1;
ELSE   PeriodEnd = &LastDateOfInterest;
RUN;
OPTIONS MERGENOBY = ERROR;

/* 2.4  Get counts by plan type for each period */
PROC SQL;
CREATE TABLE UncollapsedInsuranceTimeline
AS
SELECT
  TL.PersID,
  PeriodStart,
  PeriodEnd,
  SUM
    (CASE PlanType WHEN 1 THEN 1 ELSE 0 END)
      AS MedicarePlanCount,

  SUM
    (CASE PlanType WHEN 2 THEN 1 ELSE 0 END)
      AS MedicaidPlanCount,
  SUM
    (CASE PlanType WHEN 3 THEN 1 ELSE 0 END)
      AS MilitaryPlanCount,
  SUM
    (CASE PlanType WHEN 4 THEN 1 ELSE 0 END)
      AS PrivatePlanCount
FROM
  TimelineDates AS TL
  LEFT JOIN MYLIB.InsCovData AS PLAN
  ON TL.PersID = PLAN.PersID
  AND NOT
    (PeriodStart > PlanEnd
    OR PlanStart > PeriodEnd)
GROUP BY TL.PersID, PeriodStart, PeriodEnd
ORDER BY TL.PersID, PeriodStart, PeriodEnd
;
QUIT;

/* 2.5  Combine consecutive records with identical coverage */
DATA MYLIB.FinalInsuranceTimeline;
SET UncollapsedInsuranceTimeline;
BY PersID MedicarePlanCount MedicaidPlanCount
  MilitaryPlanCount PrivatePlanCount NOTSORTED;
LABEL HoldPeriodStart =
  'Start date for first record in set (temp)';
FORMAT HoldPeriodStart MMDDYY5.;
DROP HoldPeriodStart;
RETAIN HoldPeriodStart;
IF FIRST.PrivatePlanCount THEN DO;
  HoldPeriodStart = PeriodStart;
END;
IF LAST.PrivatePlanCount THEN DO;
  PeriodStart = HoldPeriodStart;
  OUTPUT;
END;
RUN;
```