Paper CC-05

IDENTIFYING, TRACKING, AND ANALYZING PATTERNS IN FINITE CONCURRENT AND SEQUENTIAL EVENTS USING SAS®

Vijayalakshmi Sampath, National Student Clearinghouse (NSC), Herndon, VA

ABSTRACT

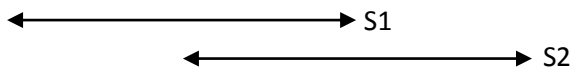
Two or more events are considered to be concurrent (simultaneous) if they happen at the same time. Historically, concurrent events have fascinated as well as posed challenges to researchers in the form of defining, interpreting, and measuring them. Be it in the field of Physics and the treatment of concurrent events as Objective (Newtonian classical physics and reference frame independent simultaneity) or subjective (Einstein's Special Theory of Relativity and reference frame dependent simultaneity), in the field of distributed computing where identifying simultaneous message communications between the systems is crucial for resource management, or even in the field of Education and Healthcare where tracking concurrent enrollments or eligibilities becomes essential for planning and policy analyses. Depending on the application area, the treatment of the source and the event may vary in terms of evaluating the concurrent events. This paper discusses, in general, the common factors to consider while attempting to identify a finite number of concurrent and sequential event patterns. It further presents generic SAS coding tips to track these patterns to quantify and analyze them. The paper concludes by presenting an example from an application area and how the concepts discussed apply to the particular example.

INTRODUCTION

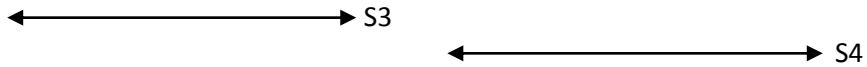
Following cohorts over a period of time is common in the field of medicine and the social sciences. In addition, analyzing events associated with cohorts very often pose challenges in the form of time range overlaps. As an example, enrollment managers at higher education institutions may be interested in tracking their students who are either concurrently enrolled in other institutions or transfer to other institutions. They would also be interested in analyzing the academic outcomes for such students as compared to the regular population of students. Even in the field of science and engineering many scientists have proposed numerous ways to track and analyze physical events occurring at the same time (whether instantaneous or over a period of time). Two events can be concurrent or consecutive and by introducing a new dimension, namely the source of the event, the complexities of analyzing the events become multifold.

Concurrent vs. Sequential events: Two or more events corresponding to a **single entity** may be termed as concurrent if the duration of one event is contained in the duration of the second event (i.e. overlapping time periods). On the other hand, they are considered sequential if the individual time periods don't overlap.

Consider the time periods of two events at sources S1 and S2 that are concurrent:



Whereas the following time periods of the events at sources S3 and S4 are sequential:



In attempting to distinguish between concurrent and sequential events it is required to: define the underlying event of interest, the units or entities for which these events are being evaluated, various sources where these events occur, and the time periods during which events occur. The two cases above are the simplest versions of concurrent and sequential events. In many practical situations there are

three or more events and the interplay between the duration and the source of these events render the analysis very complex.

The following are two areas where such events frequently occur and are a source of interest to researchers.

Attribute	Healthcare (Goal: Determine overall service utilization)	Post-Secondary Education (Goal: Analyze students outcomes such as success rate and graduation)
Event	Hospital Inpatient admissions	Enrollments at higher education Institutions
Unit/Entity	Patients	Students
Source	Hospitals/Healthcare Facilities	Universities/Colleges
Time Period	Dates of admissions and discharges	Academic (Enrollment) Terms

Defining and differentiating between concurrent and sequential events is entirely up to researchers and the area of application. This paper merely draws attention to the various factors to be considered and presents some ideas on how the events may be identified and tracked. Since the topic itself is subjective and, given that there is no concrete literature available, this paper is by no means an exhaustive treatment of the topic. In the next section a few sample scenarios and the corresponding analytical approaches that may be employed are presented.

TRACKING MULTIPLE EVENTS FOR ENTITIES

An event by itself has no analytical significance; it needs to be quantified. Delineating the quantifiable components of an event early on is very crucial to tracking multiple events. Some of the components of interest may be: how many events are concurrent and how many are sequential, what is the duration of concurrency (overlap), how many sources are involved under each type of event, etc.

While identifying and tracking the events, the following assumptions are made in this paper (which are not always necessary and may be relaxed or modified depending on the application area):

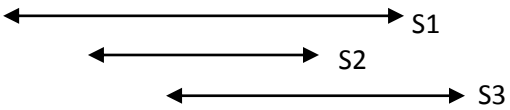
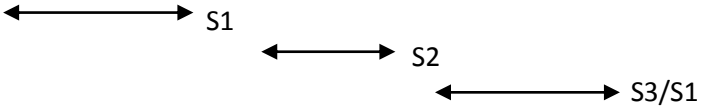
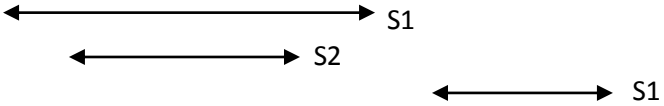
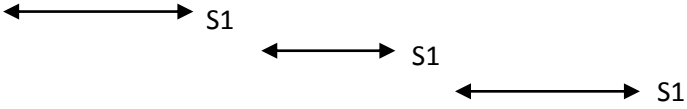
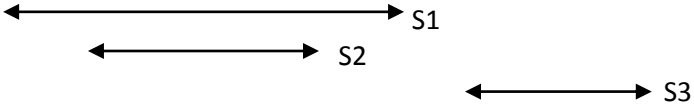
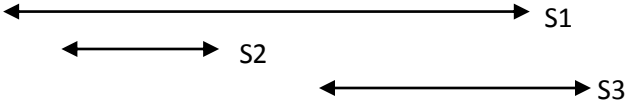
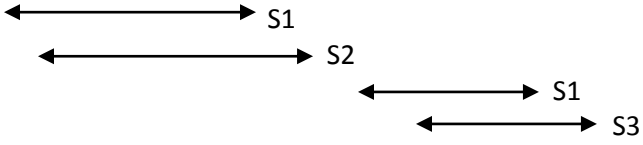
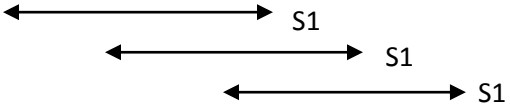
i) Events correspond to a single entity (or unit) and represent a single process (requirement for analysis, 2 different kinds of events cannot be analyzed under the same model).
ii) There is a finite # of events for every entity (computational ease and to reflect real life situations).
iii) Once an event is concurrent with another event it cannot be sequential with a third one (policy logic in application area considered in this paper).
iv) Two non-overlapping events at the same source are not considered sequential (same as in iii).
v) If an event is concurrent with more than one event the one with the longest duration of overlap is considered (same reason as in iii).

Figure 1 (on page 3) shows sample scenarios involving multiple events. Based on the assumptions above an event is classified as concurrent, sequential, or neither. In addition to classifying the events additional tracking variables to be used in subsequent analysis may be calculated. These variables fall under two broad categories:

i) **Event** level: Longest duration of overlap for an event, list of concurrent sources, etc.
ii) **Entity** level: Did an entity have at least one concurrent/sequential event, how many total concurrent/sequential sources are associated with an entity, what is the overall maximum # of days of overlap in the event history, etc.

In trying to evaluate such variables we must note that every event needs to be compared with every other event, one at a time, in the entire event history for an entity. The next section presents some SAS® coding tips and sample code to achieve this.

Figure 1. Sample Scenarios of Multiple Events for an Entity

Scenario	Events	# Concurrent	# Sequential
1		3	0
2		0 (first event cannot be sequential)	2 (if third source S3) 1 (if third source S1)
3		2 (Third event is back at source S1 and hence is neither concurrent nor sequential)	0
4		0 (all events occurring at same source)	0
5		2	1
6		3 (although S3 is sequential with S2, it is also concurrent with S1 and hence S3 is a concurrent event)	0
7		4 (Although third event is back at source S1 it is concurrent with S3 and hence needs to be accounted for)	0
8		- (scenario not possible)	-

PROGRAMMING STEPS

After clearly defining the entities, events, sources, and time periods perform the following steps. Again, this is an attempt to present the chronology of the steps involved and not necessarily to optimize the associated SAS code; there may be more efficient ways of coding using other SAS features (such as Hash objects).

STEP 1: Sort the SAS dataset (say Test1) in the order: Entity #, Begin date of event, End date. The output data may look like this, as an example:

VIEWTABLE: Work.Test1

	Entity	Beg_Dt	End_Dt	Source
1	1	25AUG2010	15DEC2010	S11
2	1	01SEP2010	14NOV2010	S12
3	1	05OCT2010	18DEC2010	S13
4	2	20AUG2010	10OCT2010	S21
5	2	15OCT2010	20NOV2010	S22
6	2	22NOV2010	15DEC2010	S21
7	3	15AUG2010	20NOV2010	S31
8	3	27SEP2010	01NOV2010	S32
9	3	17NOV2010	17DEC2010	S33

STEP 2: Identify the entities which have multiple event histories. In the above example, all three entities have multiple events. One can use a RETAIN statement to create a flag for entities with multiple events in a new dataset (say Test1a) for every unique entity (below):

VIEWTABLE: Work.Test1a

	Entity	MulEvnt_Ind
1	1	Y
2	2	Y
3	3	Y

STEP 3: This step is perhaps the most important programming step. Since, for every entity, we want to compare each event with other events, we need to create a dataset that contains all the combinations of time periods. And we know from simple probability theory that if there are n cases and we need to select r and the order doesn't matter (since comparing event 1 to event 2 is the same as comparing event 2 to event 1!), the number of combinations is given by :

$$\frac{n!}{r!(n-r)!}$$

From a programming perspective we need to create many-to-many joins on event date ranges for every entity and this can be achieved thru PROC SQL (**SAS® CODE 1**). Hence, to obtain all combinations of every begin date (and the source as well) we could do a self-join of Test1 to itself and also to Test1a (to obtain the multiple events indicator). Create new variables for begin and end date comparisons and source comparisons.

The self-join creates duplicate rows and we can use the multiple event indicator to remove the duplicates while at the same time not deleting entities which have only one event in their event history.

Again, if there are n events (with unique time periods) for an entity we will have to delete n rows from the combined dataset.

Figures 2A and 2B show the event time period combinations for entity 1 and how the combinations are processed (datasets Test2 and Test3).

SAS® CODE 1

```
PROC SQL;
CREATE TABLE TEST2 AS
SELECT A.*, B.BEG_DT AS BEG_DT2,
       B.END_DT AS END_DT2, B.SOURCE AS SOURCE2, C.MULEVNT_IND
FROM TEST1 A, TEST1 B, TEST1A C
WHERE A.ENTITY = B.ENTITY
  AND A.ENTITY = C.ENTITY
ORDER BY A.ENTITY, A.BEG_DT, B.BEG_DT, A.END_DT, B.END_DT,
         A.SOURCE, B.SOURCE;
QUIT;

/* For every entity delete rows that have same dates from both tables
   (same dates due to self-join) */
/* The MulEvnt_Ind helps to remove only repetitions in multiple event
   entities, otherwise single event entities will be deleted */
DATA TEST3;
SET TEST2;
IF BEG_DT = BEG_DT2 AND END_DT = END_DT2 AND SOURCE = SOURCE2
   AND MULEVNT_IND = 'Y' THEN DELETE;
RUN;
```

Figure 2A. All Event Begin Date Combinations for Entity 1

VIEWTABLE: Work.Test2

	Entity	Beg_Dt	End_Dt	Source	Beg_Dt2	End_Dt2	Source2	MulEvnt_Ind
1	1	25AUG2010	15DEC2010	S11	25AUG2010	15DEC2010	S11	Y
2	1	25AUG2010	15DEC2010	S11	01SEP2010	14NOV2010	S12	Y
3	1	25AUG2010	15DEC2010	S11	05OCT2010	18DEC2010	S13	Y
4	1	01SEP2010	14NOV2010	S12	25AUG2010	15DEC2010	S11	Y
5	1	01SEP2010	14NOV2010	S12	01SEP2010	14NOV2010	S12	Y
6	1	01SEP2010	14NOV2010	S12	05OCT2010	18DEC2010	S13	Y
7	1	05OCT2010	18DEC2010	S13	25AUG2010	15DEC2010	S11	Y
8	1	05OCT2010	18DEC2010	S13	01SEP2010	14NOV2010	S12	Y
9	1	05OCT2010	18DEC2010	S13	05OCT2010	18DEC2010	S13	Y

Figure 2B. Corrected Event Begin Date Combinations for Entity 1

VIEWTABLE: Work.Test3

	Entity	Beg_Dt	End_Dt	Source	Beg_Dt2	End_Dt2	Source2	MulEvnt_Ind
1	1	25AUG2010	15DEC2010	S11	01SEP2010	14NOV2010	S12	Y
2	1	25AUG2010	15DEC2010	S11	05OCT2010	18DEC2010	S13	Y
3	1	01SEP2010	14NOV2010	S12	25AUG2010	15DEC2010	S11	Y
4	1	01SEP2010	14NOV2010	S12	05OCT2010	18DEC2010	S13	Y
5	1	05OCT2010	18DEC2010	S13	25AUG2010	15DEC2010	S11	Y
6	1	05OCT2010	18DEC2010	S13	01SEP2010	14NOV2010	S12	Y

STEP 4: Create **event level variables** while identifying the concurrent and sequential events by comparing the two sets of dates for every unique begin date of an event, for an entity. Use of binary Y/N variables as concurrent and sequential indicators is useful for subsequent analysis. Temporary variables to assist with this process need to be created to store intermediate values and to adhere to the assumptions and conditions discussed earlier. Coding involves using the RETAIN statement and a series of DO loops to compare the time periods and sources (**SAS® CODE 2** shown in the Appendix).

For entity 1 with 3 event sequences at 3 different sources, the Conc_Ind, Seq_Ind , and Conc_Days variables are calculated for each event time period. As seen below, begin date of '25AUG2010' is compared with '01SEP2010' and '05OCT2010' (along with the corresponding end dates) to conclude that this event at source S11 has the longest overlap with (and hence concurrent with) the event at source S12. The variable Conc_Source lists the 2 concurrent sources selected finally for the first event. Similar comparisons are performed for the other two events for entity 1. Entity 1 is similar to scenario 1 (in Figure 1), entity 2 is similar to scenario 2 (3rd event at S1), and entity 3 is similar to scenario 6.

VIEWTABLE: Work.Test4

	Entity	Beg_Dt	End_Dt	Source	MulEvnt_Ind	Conc_Source	Conc_Ind	Seq_Ind	Conc_Days
1	1	25AUG2010	15DEC2010	S11	Y	S11S12	Y	N	75
2	1	01SEP2010	14NOV2010	S12	Y	S12S11	Y	N	75
3	1	05OCT2010	18DEC2010	S13	Y	S13S11	Y	N	72
4	2	20AUG2010	10OCT2010	S21	Y		N	N	0
5	2	15OCT2010	20NOV2010	S22	Y		N	Y	0
6	2	22NOV2010	15DEC2010	S21	Y		N	N	0
7	3	15AUG2010	20NOV2010	S31	Y	S31S32	Y	N	36
8	3	27SEP2010	01NOV2010	S32	Y	S32S31	Y	N	36
9	3	17NOV2010	17DEC2010	S33	Y	S33S31	Y	N	4

STEP 5: After processing step 4 for all the entities create **entity level variables**; these will be used for analysis. For every entity now flag whether or not it had at least one concurrent or sequential event. Also, track all the concurrent sources and calculate the maximum # of days of overlap over the entire event history. Note, that this will be 0 in single event cases or if there are no overlaps at all (entity 2 below). These evaluations are shown below for the first three entities. Use the RETAIN statement and DO loops again, this time at the entity level, to evaluate the entity level variables.

VIEWTABLE: Work.Test5

	Entity	MulEvnt_Ind	List_Conc_Source	Conc_Ever	Seq_Ever	Tot_Conc_Source	Tot_Seq_Sourc	Max_Conc_Day	Tot_Conc_Event
1	1	Y	S11S12S13	Y	N	3	0	75	3
2	2	Y		N	Y	0	1	0	0
3	3	Y	S31S32S33	Y	N	3	0	36	3

A useful function to list all the unique concurrent sources is the FIND function. To see if the source in the current record is already part of the cumulative list (and hence need not be included) one can use:

```
Find(List_Conc_Source, Source,'t') = 0
```

List_Conc_Source is a retained variable and gets updated with new sources if the source doesn't already exist in the list.

ANALYZING EVENTS

To get insights in to the patterns of events associated with entities and to make practical conclusions about them the following sample statistics may be obtained:

- Frequency and % of entities having multiple events in their event history.
- Frequency and % of entities that have one or more concurrent/sequential events.
- Distribution of # of concurrent sources.
- List of most common concurrent sources.
- Average # of days of concurrency (overlap) per entity.

The types of outcome statistics to analyze really depend on the application area and the list above is only a sample. As an example, if there is a high % of entities with concurrent events at different sources, this may be an indication of either insufficient services provided at the first source or insufficient resources available at the first source or simply that those two events are overlapping purely by chance. Another example is that if there is a small % of the entities with events at a large # of sources, these entities may comprise a special group and should be controlled for in subsequent analysis.

EXAMPLE OF AN APPLICATION AREA

A recent area of research interest in the field of post-secondary education is the effects of concurrent enrollment at and transfers (sequential) to post-secondary institutions on student outcomes and college policies. The set of assumptions outlined earlier were developed to reflect the policies related to the process of student enrollments at institutions. The programming code in **SAS® CODE 2** is also developed based on the same assumptions and applicable to tracking students across enrollment terms.

Students are entities, the event is enrollment at a post-secondary educational institution, institutions are sources, and the academic term (or enrollment date) ranges are the time periods to be evaluated.
Tracking enrollments of such students at various institutions becomes a daunting task for educators and researchers. As an example, students placed in developmental courses (no credits earned for these courses) in 2 year colleges or students required to take certain pre-requisite courses may transfer to another institution to by-pass these policies. Or, students may be concurrently enrolled at two institutions due to scheduling conflicts or to satisfy graduation requirements (certain courses required for graduation). Tracking such students at various institutions may help administrators revisit college policies and procedures or provide details of effects on student performance, retention, or graduation. Since concurrent enrollments and transfers are two entirely different concepts, the ability to correctly track these events become even more crucial in educational research.

CONCLUSION

Tracking and analyzing contemporaneous events and their effects on other related factors have always posed challenges. But with proper definitions and outlining the rules one can conduct a fair analysis of such events. This paper attempted to steer readers towards the various approaches that may be taken and how to appropriately track such events using SAS. The intent is to keep the topic open for newer ideas, better approaches, and also more efficient coding techniques.

ACKNOWLEDGEMENTS

I would like to thank Dr. Doug Shapiro, Senior Director of the Research Center at NSC, for encouraging and supporting my paper contribution to the SESUG 2011 conference.

CONTACT INFORMATION

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

Vijayalakshmi Sampath
National Student Clearinghouse
2300 Dulles Station Boulevard, Suite 300
Herndon, VA 20171
E-mail: sampath@studentclearinghouse.org or
vibha_atm75@yahoo.com

TRADEMARKS

APPENDIX

SAS® CODE 2

```
* FLAG CONCURRENT AND SEQUENTIAL & EVALUATE LONGEST OVERLAP FOR EACH EVENT *;

DATA TEST4 (DROP = BEG_DT2 END_DT2 SOURCE2 TEMP_DAYS SAME_SOURCE );
 SET TEST3;
BY ENTITY BEG_DT BEG_DT2 END_DT END_DT2 SOURCE;
FORMAT CONC_SOURCE SAME_SOURCE $10. CONC_IND SEQ_IND $1. CONC_DAYS  TEMP_DAYS 8.;
RETAIN CONC_SOURCE SAME_SOURCE CONC_IND SEQ_IND CONC_DAYS  ;
 IF FIRST.BEG_DT THEN DO;     ** (1) **;
   CONC_IND = 'N'; SEQ_IND = 'N'; CONC_DAYS = 0; TEMP_DAYS = 0; CONC_SOURCE = "";
SAME_SOURCE = "";
    IF (BEG_DT2 > END_DT OR END_DT2 < BEG_DT) THEN DO;  /* NOT CONCURRENT, CHECK IF
SEQUENTIAL**/  ** (11) **;
     IF END_DT2 < BEG_DT THEN DO; /* THE FIRST EVENT IN THE ENTIRE SEQUENCE CANNOT
BE A SEQUENTIAL EVENT, CHECK FOR THIS (111) **/
        IF SOURCE NE SOURCE2 THEN DO; /** MAY BE SEQUENTIAL **/   ** (1111) ** ;
             IF CONC_IND = 'N' THEN SEQ_IND = 'Y';
          ELSE SEQ_IND = 'N';
           END;  ** (1111) **;
           ELSE SAME_SOURCE = SOURCE;
        END;  ** (111) ***;
      END; ** (11) **;
    ELSE DO; /* BEG_DT2 <= END_DT OR END_DT2 > BEG_DT I.E MAY BE CONCURRENT **/  **
(12) **;
       IF SOURCE NE SOURCE2 THEN DO; /** CONCURRENT **/ **  (121)  **;
       CONC_SOURCE = LEFT(TRIM(SOURCE)||TRIM(SOURCE2));
       CONC_IND = 'Y';
          SEQ_IND = 'N'; /** IF AN EVENT IS CONCURRENT IT CANNOT BE SEQUENTIAL**/
         IF BEG_DT2 >= BEG_DT THEN DO;   ***  (1211)  **;
           IF END_DT2 <= END_DT THEN TEMP_DAYS = (END_DT2 - BEG_DT2) + 1;
           ELSE TEMP_DAYS = (END_DT - BEG_DT2) + 1 ;
              END;  *** (1211) ***;
         ELSE DO;  /** BEG_DT2 < BEG_DT **/   ***  (1212) ***;
           IF END_DT2 >= END_DT THEN DO;
            TEMP_DAYS = (END_DT - BEG_DT) + 1;
                END;
                ELSE DO ;  /** END_DT2 < END_DT **/  *** (12121) ***;
             IF END_DT2 >= BEG_DT THEN TEMP_DAYS = (END_DT2 - BEG_DT) + 1;
                    END; **  (12121) ***;
              END;  ** (1212) **;
         IF TEMP_DAYS > CONC_DAYS THEN DO;
         CONC_DAYS = TEMP_DAYS;
              CONC_SOURCE = LEFT(TRIM(SOURCE)||TRIM(SOURCE2));
          END;
      END;  **  (121) **;
         ELSE SAME_SOURCE= SOURCE;
    END;  ** (12) **;
 END; ** (1) **;
```

CONTINUED ON NEXT PAGE…….

SAS® CODE 2 CONTINUED……

```
ELSE DO; /** NOT FIRST.BEG_DT **/   ** (2) **;
   IF (BEG_DT2 > END_DT OR END_DT2 < BEG_DT) THEN DO;  /* NOT CONCURRENT, CHECK IF
SEQUENTIAL**/  ** (21) **;
     IF END_DT2 < BEG_DT THEN DO; /* THE FIRST EVENT IN THE ENTIRE SEQUENCE CANNOT
BE A SEQUENTIAL EVENT, CHECK FOR THIS (211) **/
       IF SOURCE NE SOURCE2 THEN DO; /** MAY BE SEQUENTIAL **/   ** (2111) ** ;
            IF (CONC_IND = 'N' AND SOURCE NE SAME_SOURCE) THEN SEQ_IND = 'Y';
         ELSE SEQ_IND = 'N';
          END;  ** (2111) **;
          ELSE SAME_SOURCE = SOURCE;
       END; ** (211) **;
   END; ** (21) **;
   ELSE DO; /* BEG_DT2 <= END_DT OR END_DT2 > BEG_DT I.E MAY BE CONCURRENT **/  **
(22) **;
     IF SOURCE NE SOURCE2 THEN DO; /** CONCURRENT **/ **  (221)  **;
       CONC_IND = 'Y';
        SEQ_IND = 'N'; /* IF AN EVENT IS CONCURRENT IT CANNOT BE SEQUENTIAL**/
         IF BEG_DT2 >= BEG_DT THEN DO;   ***  (2211)  **;
           IF END_DT2 <= END_DT THEN TEMP_DAYS = (END_DT2 - BEG_DT2) + 1;
           ELSE TEMP_DAYS = (END_DT - BEG_DT2) + 1 ;
              END;  *** (2211) ***;
         ELSE DO;  /** BEG_DT2 < BEG_DT **/   ***  (2212) ***;
           IF END_DT2 >= END_DT THEN TEMP_DAYS = (END_DT - BEG_DT) + 1;
                ELSE DO ;  /** END_DT2 < END_DT **/  *** (22121) ***;
                IF END_DT2 >= BEG_DT THEN TEMP_DAYS = (END_DT2 - BEG_DT) + 1;
                END; **  (22121) ***;
            END;  ** (2212) **;
               IF TEMP_DAYS > CONC_DAYS THEN DO;
                CONC_DAYS = TEMP_DAYS;
                CONC_SOURCE = LEFT(TRIM(SOURCE)||TRIM(SOURCE2));
               END;
      END;  **  (221) **;
        ELSE SAME_SOURCE = SOURCE;
    END;  ** (22) **;
 END;  ** (2) **;

 IF LAST.BEG_DT THEN OUTPUT;
RUN;
```