SESUG Paper 068-2021

Using PROC SQL to restructure data for common healthcare applications

Tamar Roomian, MS MPH, Department of Orthopaedic Surgery, Atrium Health Musculoskeletal Institute

ABSTRACT

Electronic medical record (EMR) data is commonly stored in “long” format (multiple rows per an id, where one column stores many pieces of information on the same id). However, for many applications, data must be transposed to “wide” format (one row per id with multiple columns representing each piece of information needed). This paper will present how to use PROC SQL to transpose data from “long” to “wide” using a common example from healthcare utilizing electronic medical record data. This paper will discuss the advantages and disadvantages to using PROC SQL over proc transpose.

In addition, it is commonly required to join two sources of data based on logic that depends on dates. Example questions could include: Which patients received a depression diagnosis 4 months prior to their emergency visit? Which patients had an encounter for a fall within 2 years from their encounter for a fracture? The DATA step MERGE statement requires equal conditions using the by statement. PROC SQL does not require equal conditions and therefore can be used to join data with unequal date conditions. Unequal join conditions using dates will be shown continuing the first healthcare example. This paper will present how to use PROC SQL to determine whether an encounter occurred within a specified time frame from an index date. The INTNX function will also be briefly introduced.

This presentation is aimed at beginner to intermediate SAS programmers who already have a basic understanding of PROC SQL and are looking to expand their SQL abilities.

INTRODUCTION

Typically, diagnosis data per encounter is stored in an Electronic Medical Record (EMR) database in “long” format – i.e. there are multiple rows per encounter and each row represents a diagnosis that occurred at that encounter. In this hypothetical example, a table from a relational database was queried to include encounters where a fracture diagnostic code was recorded. You are performing an analysis where you need one row per encounter, and you want to create indicator variables for the presence or absence of a list of specific fractures categories (for example, upper extremity, lower extremity, pelvis).

The following code introduces the starting data:

```
data fracture_encounters ;
infile datalines missover dsd;
input encounter_id $ patient_id $ encounter_date mmddyy10.
diagnosis_code $ desired_category $;
format encounter_date mmddyy10.;
datalines ;
"1", "A", 1/01/2018 "S82.4", "Lower"
"1", "A", 1/01/2018 "S52.3", "Upper"
"1", "A", 1/01/2018 "E11.8",
"2", "B", 1/07/2018 "S32.5", "Pelvis"
"2", "B", 1/07/2018 "J45.20",
"2", "B", 1/07/2018 "I11.9",
"3", "C", 1/15/2018 "K45.8",
"3", "C", 1/15/2018 "S32.4", "Pelvis"
"3", "C", 1/15/2018 "S82.4", "Lower"
"3", "C", 1/15/2018 "S22.4",
"4", "D", 1/21/2018 "S02.0",
"4", "D", 1/21/2018 "I43",
;
```

```
run;

proc print data=fracture_encounters noobs;
run;
```

The results of the DATA step are presented in Output 1:

encounter_id	patient_id	encounter_date	diagnosis_code	desired_category
1	A	01/01/2018	S82.4	Lower
1	A	01/01/2018	S52.3	Upper
1	A	01/01/2018	E11.8	
2	B	01/07/2018	S32.5	Pelvis
2	B	01/07/2018	J45.20	
2	B	01/07/2018	I11.9	
3	C	01/15/2018	K45.8	
3	C	01/15/2018	S32.4	Pelvis
3	C	01/15/2018	S82.4	Lower
3	C	01/15/2018	S22.4	
4	D	01/21/2018	S02.0	
4	D	01/21/2018	I43	

Output 1. fracture_encounters dataset created by DATA step

Table 1 presents the desired result:

encounter_id	patient_id	encounter_date	upper_extremity_fx	lower_extremity_fx	pelvis_fx
1	A	01/01/2018	1	1	0
2	B	01/07/2018	0	0	1
3	C	01/15/2018	0	1	1
4	D	01/21/2018	0	0	0

Table 1. EMR query converted to "wide" format with indicator variables for each fracture type

In addition, when conducting healthcare research, it is commonly required to join two sources of data based on logic that depends on dates. To continue the example, which patients had an encounter for a fracture within 2 years from their encounter for a fall?

The following DATA step creates a dataset with fall encounters for each patient_id:

```
data fall_encounters;
infile datalines missover dsd;
input patient_id $ fall_date mmddyy10.;
```

```
format fall_date mmddyy10.;
datalines ;
"A", 2/05/2019
"A", 7/03/2021
"B", 6/03/2020
"B", 9/02/2020
"C", 3/18/2020
"C", 5/4/2020
"D", 1/31/2020
"D", 12/4/2019
run;

proc print data=fall_encounters noobs;
run;
```

The results of the DATA step are presented in Output 2:

patient_id	fall_date
A	02/05/2019
A	07/03/2021
B	06/03/2020
B	09/02/2020
C	03/18/2020
C	05/04/2020
D	01/31/2020
D	12/04/2019

Output 2. Fall_encounters dataset created by the DATA step

Table 2 presents the desired data using join logic that depends on date:

Encounter_ID	Patient ID	Encounter Date	Upper_Extremity_fx	Lower_Extremity_fx	pelvix_fx	Future_fall
1	A	1/1/2018	1	1	0	1
2	B	1/7/2018	0	0	1	0
3	C	1/15/2018	0	1	1	0
4	D	1/21/2018	0	0	0	1

Table 2. Encounters with a fall indicator variable 2 years from the encounter date

TRANSPOSE DIAGNOSIS CODE FROM "LONG" (MULTIPLE ROWS PER ENCOUNTER") TO "WIDE" (ONE ROW PER ENCOUNTER)

STEP 1. ASSIGN CATEGORIES

Assign categories. This can be done in a previous data step using if statements, or within the SQL procedure. For the sake of simplicity, it is assumed that you already assigned the diagnosis categories needed. As a disclaimer, be sure to work with a clinical expert (such as a physician) when assigning categories to diagnosis codes.

This method takes advantage of the group by statement in PROC SQL and aggregate functions. A group by statement is used to specify what each row represents. The same variable in the group by statement is used in the select statement without any aggregate functions. Then, case statements are first used to assign 1 if the desired category exists and 0 if not. Finally, the max function is used to indicate the presence or absence of each condition of interest.

```
proc sql;
create table fracture_encounters_wide as
select
      encounter_id
      ,max (patient_id) as patient_id
      ,max (encounter_date) as encounter_date format=mmddyy10.

      ,max (case when desired_category="Lower" then 1 else 0 end)
      as lower_extremity_fx

      ,max (case when desired_category="Upper" then 1 else 0 end)
      as upper_extremity_fx

      ,max (case when desired_category="Pelvis" then 1 else 0
      end) as pelvis_fx

from fracture_encounters
group by encounter_id
;
quit;
```

The sum function can be used instead of the max function when the total number is needed rather than the presence or absence.

WHY USE PROC SQL OVER THE TRANSPOSE PROCEDURE?

Of course, the above could be achieved with PROC TRANSPOSE. PROC TRANSPOSE would require 5 steps that PROC SQL achieved in 1:

1. Create a new column with a value of 1. Assign a value to missing:

```
data fracture_encounters2;
set fracture_encounters;
var=1;
if desired_category=" " then desired_category="NULL";
run;
```

2. Ensure that there is only one row per encounter id per category.

```
proc sort data= fracture_encounters2 out= fracture_encounters3
nodupkey;
by encounter_id desired_category;
run;
```

3. PROC TRANSPOSE will transpose every existing value that exists in the id column.

```
proc transpose data=fracture_encounters3 out=fracture_encounters4
(drop=_name_ NULL);
by encounter_id;
var var;
id desired_category;
run;
```

4. Assign missing values a 0.

```
proc stdize data= fracture_encounters4 out=fracture_encounters5
reponly missing=0;
Run;
```

If you need every value in the data to be transposed, PROC TRANSPOSE is the more efficient option. However, if you have a few categories to transpose, PROC SQL can achieve the same thing in fewer steps.

JOIN USING UNEQUAL DATE CONDITIONS

The fall date must be after the fracture date but before 2 years from the fracture date. The INTNX function can be used to calculate dates using different time intervals. In the join statement, a second condition is used in addition to equal patient_ids. More information on the INTNX function is available using the online SAS documentation.

Here, the INTNX function calculates the date that is 2 years from the fracture date. The optional “same” argument specifies that the interval starts from the fall date and not the default, which is the first of the year. Then, a condition can be added to specify that the fall date must be greater than the fracture date, but within 2 years after fracture date:

```
proc sql;
create table final as
select
 A.encounter_id
, A.patient_id
, A.encounter_date
, A.lower_extremity_fx
, A.upper_extremity_fx
, A.pelvis_fx
, B.fall_date
, case when fall_date is not null then 1 else 0 end as future_fall
from fracture_encounters_wide  as A
left join fall_encounters as B on A.patient_id=B.patient_id
AND encounter_date<fall_date<intnx('year', encounter_date, 2, 'same')
;
quit;
```

```
proc print data=final noobs;
run;
```

Results from the PRINT procedure are presented in Output 3:

encounter_id	patient_id	encounter_date	lower_extremity_fx	upper_extremity_fx	pelvis_fx	fall_date	future_fall
1	A	01/01/2018	1	1	0	02/05/2019	1
2	B	01/07/2018	0	0	1	.	0
3	C	01/15/2018	1	0	1	.	0
4	D	01/21/2018	0	0	0	12/04/2019	1

Output 3. Falls 2 years from the encounter date are joined

After the join, it's likely that multiple encounters could meet the conditions when you only need a binary variable and one row per encounter. An additional step would be needed to ensure 1 row per encounter. This could be achieved using the DATA STEP, PROC SQL, or PROC SORT.

CONCLUSION

Electronic medical record data is commonly stored in "long" format where one column stores multiple pieces of information for the same id. The GROUP BY statement with aggregate functions within PROC SQL can be used to transpose data from "long" to "wide" to achieve one row per id and multiple columns representing the presence or absence of each condition of interest. In addition, health care data analysis frequently requires joining two sources of data on conditions that require dates. While the DATA step MERGE statement requires equal conditions, PROC SQL in conjunction with the INTNX function can be used to join data with unequal date conditions. In combination, these two techniques can be used to clean and prepare EMR data for analysis.

RECOMMENDED READING

- SAS Institute Inc. "SAS Help Center: INTNX Function." August 10, 2021. Available at https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lefunctionsref/p10v3sa3i4kfxfn1sovhi5xzxh8n.htm.

CONTACT INFORMATION

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

Tamar Roomian
tamar.roomian@atriumhealth.org