Paper S110-2011

Fuzzy Merges - A Guide to Joining Data sets with Non-Exact Keys Using the SAS SQL Procedure

Robert W. Graebner, Quintiles, Overland Park, KS, USA

Abstract

Most of us are familiar with merging SAS data sets based on one or more common variables that have exact matching values in each of the files. There are situations however, when it is necessary to match records based on criteria other than simple equality. One example is merging data regarding an adverse event in a clinical trial with data regarding the most recent treatment received prior to the onset of the event. Because this type of merge involves non-exact matches, they are sometimes referred to as fuzzy merges. This paper provides a step by step guide to how SAS SQL Procedure performs a merge and how the features of SQL can be used to perform fuzzy merges.

Introduction

The SAS SQL procedure provides many different methods of combining SAS data sets. This flexibility and power is a feature of the basic process SQL uses when performing a select query with two or more data sets. While care must be taken to avoid inefficient processing, this process allows you to combine data sets in ways that would be difficult or even impossible using other methods.

The focus of this paper is on methods of merging data sets that can be used when one or more of the key variables do not have exact matching values. The examples presented are based on the task of assigning the most recent treatment received to a treatment-emergent adverse event that occurs during a crossover study where each subject receives multiple treatments. Adverse events are said to be treatment emergent if they start on or after the first dose or if they were present prior to dosing, but increased in severity after dosing. An end-of-study cutoff period is usually specified, typically two weeks, after which adverse events are no longer considered to be treatment emergent. Once an adverse event has been determined to be treatment emergent, you must also determine which treatment to assign it to and PROC SQL provides an easy solution. Before attempting to program an advanced join using PROC SQL, it is helpful to go through the basic process PROC SQL uses to combine data sets using a simple example based on exact matches.

Combining Data Sets Using PROC SQL - Behind the Scenes

A simple example involving exact matches is merging demographic data with adverse events. The demographic data set has one record per subject and the adverse events data set has one or more records per subject. In this case, a one-to-many join can be performed matching rows based on the unique subject identifier. While SAS will attempt to optimize this process for the sake of efficiency, the following example will go through the full process to illustrate the basic concepts. As an example, assume we need to merge sex and age from the demographic data set into adverse events in order to generate summary tables that are subset by sex and age group. This task could be accomplished using the following code:

```
PROC SQL NOPRINT;
  CREATE TABLE AE AS
    SELECT DEMOG.SUBID, DEMOG.SEX, DEMOG.AGE, AE.*
    FROM ANALYSIS.DEMOG AS DEMOG,
         ANALYSIS.AE AS AE
    WHERE DEMOG.SUBID = AE.SUBID;
QUIT;
```

PROC SQL begins the process by creating a Cartesian product of the two data sets. This is done by pairing each row in the demographic data set with each row of the adverse event data set without regard to the data in each row. If there were 20 subjects and a total of 50 adverse events, the Cartesian product would have 1,000 rows. In this case, the vast majority of the pairings are meaningless. Each adverse event would be paired with the demographics data for each subject. Only the pairings in which the subject ID is the same in both data sets are useful. This is where the WHERE statement comes into play. The condition expressed in the WHERE statement is applied to all pairings of rows and only those where the condition is true are kept in the data set AE being created. In this case, only five percent of the Cartesian product is used for the new data set. When working with large data sets, the Cartesian product can become very large. This explains some of the potential inefficiency of using SQL, however this "brute force" method also provides a great deal of flexibility in combining data sets.

To illustrate the process of an exact match, we will look at a simple example with only a few rows and columns. Suppose we have the following demographics and adverse event data sets:

Demog

	SubID	Sex	Age
1	1001	M	29
2	1002	F	31

AE

	SubID	AE	StartDate
1	1001	Headache	16OCT2004
2	1001	Chills	23OCT2004
3	1002	Nausea	16OCT2004

The Cartesian product of these two data sets is shown below.

	Demog_SubID	AE_SubID	SubID	Sex	Age	AE	StartDate
1	1001	1001	1001	M	29	Headache	16OCT2004
2	1001	1001	1001	M	29	Chills	23OCT2004
3	1001	1002	1001	M	29	Nausea	16OCT2004
4	1002	1001	1002	F	31	Headache	16OCT2004
5	1002	1001	1002	F	31	Chills	23OCT2004
6	1002	1002	1002	F	31	Nausea	16OCT2004

The first two columns are only included to illustrate the subject ID coming from each source data set. As mentioned above, only those rows where subject ID matches are of any value and once the WHERE statement is applied, we end up with the resulting data set shown below.

	Demog_SubID	AE_SubID	SubID	Sex	Age	AE	StartDate
1	1001	1001	1001	M	29	Headache	16OCT2004
2	1001	1001	1001	M	29	Chills	23OCT2004
3	1002	1002	1002	F	31	Nausea	16OCT2004

As a side note, the Cartesian product is sometimes useful on its own, as when creating zero-fill data sets that span all combinations of preferred term and system organ class that are present in your data set. A Cartesian product can be produced by simply using a SQL SELECT statement with two or more data sets, but without using a WHERE statement.

Let's Get Fuzzy

The merge performed above is a simple example where a key variable present in both data sets has values that match exactly. Such a merge could also be performed using a DATA step with MERGE and BY statements. Now let's take a look at our example of merging the most recent treatment with each adverse event. We can see that while the subject IDs will match exactly, we also need to match each adverse event with the treatment record having the most recent dosing date on or before the start date of the adverse event. While this may seem to be a daunting task at first, it is easy to accomplish with PROC SQL. We simply need to add an additional condition to the WHERE statement and then add GROUP BY and HAVING statements as shown below. The WHERE statement now restricts the selection of rows in the Cartesian product to those that have both the same subject ID in each data set and that have a dose date on or before the AE start date. The remaining rows are then formed into groups based on the subject ID and AE start date, and the single row for each AE for each subject with the maximum (most recent) dosing date within that group.

```
PROC SQL NOPRINT;
  CREATE TABLE TEAE AS
    SELECT TREAT.SUBID, TREAT.DOSEDATE, AE.*
    FROM ANALYSIS.TREAT AS TREAT,
         ANALYSIS.AE AS AE
    WHERE TREAT.SUBID = AE.SUBID AND
          TREAT.DOSEDATE <= AE.STARTDATE
    GROUP BY AE.SUBID, AE.STARTDATE
    HAVING TREAT.DOSEDATE = MAX(TREAT.DOSEDATE);
QUIT;
```

The MAX() function used above is a SQL aggregate function as opposed to the SAS DATA step function of the same name. They are referred to as aggregate functions because they are calculated using the specified column over all rows that meet the criteria in the WHERE statement. If a GROUP BY statement is used, a separate result will be computed for each group of rows. SQL aggregate functions cannot be used in the WHERE statement because they are processed after the WHERE selection is completed, however they can be used in a HAVING statement. SAS DATA step functions can also be used with PROC SQL. The main difference is that they compute a result using one or more columns, but only for a single row at a time. Because they only work on one row at a time, regular SAS functions can be used in a WHERE statement. SAS functions can also be used to derive new variables in the SELECT statement. If you use a derived variable in the WHERE statement, it must be proceeded with the modifier CALCULATED.

To illustrate method used to process the SQL statements above, assume that we have the following treatment data set:

	SubID	Treat	DoseDate
1	1001	A	15OCT2004
2	1001	B	22OCT2004
3	1002	B	15OCT2004
4	1002	A	22OCT2004

In this case, the study has two treatments and each subject is randomized to a treatment sequence. In this example a dose date is used for simplicity. In practice, it is better to use SAS datetime variables for dosing and adverse events to ensure proper sequencing of events occurring within the same day. The Cartesian product of the adverse event and treatment data sets is shown below.

	Treat_SubID	AE_SubID	SubID	Treat	DoseDate	AE	StartDate
1	1001	1001	1001	A	15OCT2004	Headache	16OCT2004
2	1001	1001	1001	A	15OCT2004	Chills	23OCT2004
3	1001	1002	1001	A	15OCT2004	Nausea	16OCT2004
4	1001	1001	1001	B	22OCT2004	Headache	16OCT2004
5	1001	1001	1001	B	22OCT2004	Chills	23OCT2004
6	1001	1002	1001	B	22OCT2004	Nausea	16OCT2004
7	1002	1001	1002	B	15OCT2004	Headache	16OCT2004
8	1002	1001	1002	B	15OCT2004	Chills	23OCT2004
9	1002	1002	1002	B	15OCT2004	Nausea	16OCT2004
10	1002	1001	1002	A	22OCT2004	Headache	16OCT2004
11	1002	1001	1002	A	22OCT2004	Chills	23OCT2004
12	1002	1002	1002	A	22OCT2004	Nausea	16OCT2004

The WHERE statement is then processed, resulting in only the rows where the subject ID is the same and the dose date is on or before the adverse event start date.

	Treat_SubID	AE_SubID	SubID	Treat	DoseDate	AE	StartDate
1	1001	1001	1001	A	15OCT2004	Headache	16OCT2004
2	1001	1001	1001	A	15OCT2004	Chills	23OCT2004
3	1001	1001	1001	B	22OCT2004	Chills	23OCT2004
4	1002	1002	1002	B	15OCT2004	Nausea	16OCT2004

Grouping by subject ID and start date will result in three groups in this case. The first will contain row one, the second rows two and three, and the third row four. The HAVING statement will keep only those rows in each group where the dose date is the most recent, resulting in the final data set shown below.

	Treat_SubID	AE_SubID	SubID	Treat	DoseDate	AE	StartDate
1	1001	1001	1001	A	15OCT2004	Headache	16OCT2004
2	1001	1001	1001	B	22OCT2004	Chills	23OCT2004
3	1002	1002	1002	B	15OCT2004	Nausea	16OCT2004

This basic method can be used for a wide variety on non-exact matches. An example of a similar fuzzy merge is the closest value merge. Suppose we need to merge the subject's weight from the vital sign data set, using the value recorded closest to (before or after) the adverse event. The SQL source code for such a merge is shown below.

```
PROC SQL NOPRINT;
  CREATE TABLE TEAE AS
    SELECT VITAL.SUBID, VITAL.VISITDATE, VITAL.WEIGHT,
           AE.*, ABS(VITAL.VISITDATE - AE.STARTDATE) AS DATEDIFF
    FROM ANALYSIS.VITAL AS VITAL,
         ANALYSIS.AE AS AE
    WHERE VITAL.SUBID = AE.SUBID
    GROUP BY AE.SUBID, AE.STARTDATE
    HAVING DATEDIFF = MIN(DATEDIFF);
QUIT;
```

To accomplish this, a new variable is derived in the select statement as the absolute value of the difference between the two dates. We then group the rows based on subject ID and AE start date, and keep only the pairings where this difference is the minimum within the group.

Conclusion

By using the power of SAS PROC SQL it is possible to perform complex, inexact merges efficiently with very little programming. By understanding the basic process used by SQL to combine data sets you can better exploit the power that PROC SQL has to offer.

Contact Information

Robert Graebner
16107 Russell Rd.
Stilwell, KS 66085

Email: bobgraebner@kc.rr.com