

## Paper 065-29

### Subsetting SAS® Data Set by Using PROC SQL Self-join with Compound Key

Zizhong Fan, Westat, Rockville, MD

#### Abstract

PROC SQL is a powerful procedure that has many advantages over the DATA step. It can be used for data retrieval, manipulation and report writing. One of the advantages of using PROC SQL for data manipulation is how the data sets are merged together (joining tables). Most of the literature has been focusing on how to join multiple data sets. This paper will focus on one of the other important functions of PROC SQL: the Self-join. The PROC SQL self-join is more efficient than DATA step processing in the area of subsetting data set when the subsetting needs to be based on the relationship between variables in different observations within the same data set. By utilizing multiple common columns (compound key) within the same data set, the self-join technique efficiently simplifies programming. This paper will show the strategy and programming of the PROC SQL self-join. Examples will be shown to compare SQL procedure with the DATA step. SAS product: Base SAS. Skill Level: All skill levels.

#### Introduction

The Structured Query Language (SQL) is a standardized, widely used language that retrieves and updates data in relational databases. One of the most significant advantages of SAS® SQL procedure is joining tables. There are different types of joins: inner join, outer join and self-join. The reason for joining tables is that data are always stored in different tables (data sets). This improves performance of processing. This is also one of the database principles, which means that only limited number of columns (variables) are stored in one table (data set). Meanwhile, there is always a need for putting data from different tables (data sets) together to get more information, which requires table joining by various means.

Joining multiple tables by using SAS SQL procedure has been extensively discussed by our colleagues. The focus of this paper will be another important function of SQL procedure: Self-join.

Self-join is also called reflexive join. It is a process in which a single table is joined with itself to produce more information. This is necessary in situations where subsetting needs to be based on the relations between variables (columns) in different observations within the same data set (table).

#### Example 1

One classic example of using PROC SQL self-join procedure is identifying mutual spousal abuse in domestic violence data.

The original data set ABUSE contains spousal abuse information about the offenders and victims. The relevant variables for the purpose of this paper include O\_SSN (Offender Social Security Number), V\_SSN (Victim Social Security Number) and DATE. The objective is to identify the families that have mutual cases. In other words, husband beats wife and wife fights back, and vice versa.

#### Data Set: ABUSE

| CASE | O_SSN       | V_SSN       | Date       |
|------|-------------|-------------|------------|
| 1    | 000-00-0001 | 000-00-0002 | 01/01/2000 |
| 2    | 000-00-0002 | 000-00-0001 | 01/01/2000 |
| 3    | 000-00-0003 | 000-00-0004 | 01/01/2000 |
| 4    | 000-00-0005 | 000-00-0006 | 01/01/2000 |
| 5    | 000-00-0007 | 000-00-0008 | 01/01/2000 |
| 6    | 000-00-0008 | 000-00-0007 | 01/01/2000 |

The result output contains only couples that both husband and wife are involved in an incident on the same day.

#### Result Output

| CASE | O_SSN       | V_SSN       | Date       |
|------|-------------|-------------|------------|
| 1    | 000-00-0001 | 000-00-0002 | 01/01/2000 |
| 2    | 000-00-0002 | 000-00-0001 | 01/01/2000 |
| 5    | 000-00-0007 | 000-00-0008 | 01/01/2000 |
| 6    | 000-00-0008 | 000-00-0007 | 01/01/2000 |

This becomes a task of subsetting data set. The criteria are when O\_SSN and V\_SSN from different observations match, and DATEs are the same. There are two ways to accomplish this task. One is PROC SQL self-join procedure. The other one is the traditional data step.

### SQL procedure

```
proc sql;
  select a.*,
  from abuse a, abuse b
  where a.o_ssn=b.v_ssn and
        a.v_ssn=b.o_ssn and
        a.date=b.date;
quit;
```

In the self-join process, the same table is listed twice in the FROM clause. Both tables *MUST* be given an alias. Otherwise, we are not able to distinguish between references to columns in both tables. What happens in this process is that the single table joins itself by matching O\_SSN with V\_SSN, and DATE with DATE.

### Self-join processing

| CASE | O_SSN       | V_SSN       | DATE       |
|------|-------------|-------------|------------|
| 1    | 000-00-0001 | 000-00-0002 | 01/01/2000 |
| 2    | 000-00-0002 | 000-00-0001 | 01/01/2000 |
| :    | :           | :           | :          |
| :    | :           | :           | :          |

### Data Step

First of all, two data sets containing the same data are created by using two SORT procedures. Data set A and data set B are exactly same except that in data set B, O\_SSN is renamed as V\_SSN and V\_SSN is renamed as O\_SSN. This is because SAS merge requires that common columns have to have the same variable names. Then, result data set MUTUAL is created by merging data set A and B. Listing output is created by PRINT procedure.

#### Data Set A

| CASE | O_SSN       | V_SSN       | Date       |
|------|-------------|-------------|------------|
| 1    | 000-00-0001 | 000-00-0002 | 01/01/2000 |
| 2    | 000-00-0002 | 000-00-0001 | 01/01/2000 |
| 3    | 000-00-0003 | 000-00-0004 | 01/01/2000 |
| 4    | 000-00-0005 | 000-00-0006 | 01/01/2000 |
| 5    | 000-00-0007 | 000-00-0008 | 01/01/2000 |
| 6    | 000-00-0008 | 000-00-0007 | 01/01/2000 |

```
proc sort data=abuse out=a;
  by o_ssn v_ssn date;
run;

proc sort data=abuse out=b
  (rename=(o_ssn=v_ssn
           v_ssn=o_ssn));
  by v_ssn o_ssn date;
run;
```

#### Data Set B

| CASE | V_SSN       | O_SSN       | Date       |
|------|-------------|-------------|------------|
| 2    | 000-00-0002 | 000-00-0001 | 01/01/2000 |
| 1    | 000-00-0001 | 000-00-0002 | 01/01/2000 |
| 3    | 000-00-0003 | 000-00-0004 | 01/01/2000 |
| 4    | 000-00-0005 | 000-00-0006 | 01/01/2000 |
| 6    | 000-00-0008 | 000-00-0007 | 01/01/2000 |
| 5    | 000-00-0007 | 000-00-0008 | 01/01/2000 |

```
data mutual;
  merge a (in=a) b (in=b);
  by o_ssn v_ssn date;
  if a and b;
run;

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

## Example 2

The above example is based on an equation ( $V\_SSN=O\_SSN \dots$ ). Now let's look at an example where the self-join is used for inequality. The data set Trip is a simulation from National Household Travel Survey data set. There are four variables: person ID (PersonID), trip ID (TripID), trip start time (StrtTime), and trip end time (EndTime). You may have noticed that person 001 and person 003 are legitimate trips. But person 0002's trip 1 and 2 are overlapped. Trip 1 is from 7:00 am to 8:00 am and trip 2 is from 7:30 am to 8:30 am. Our task here is to identify the overlapped trips for the same persons.

### Data Set: Trip

| PersonID | TripID | StrtTime | EndTime |
|----------|--------|----------|---------|
| 0001     | 1      | 0830     | 0900    |
| 0001     | 2      | 0915     | 1015    |
| 0001     | 3      | 1700     | 1730    |
| 0002     | 1      | 0700     | 0800    |
| 0002     | 2      | 0730     | 0830    |
| 0002     | 3      | 1630     | 1730    |
| 0003     | 1      | 0730     | 0830    |
| 0003     | 2      | 1730     | 1830    |

Again, one simple straight forward SQL procedure can accomplish this edit check task.

```
proc sql;
  create table OverLap as
  select distinct a.*
  from Trip a, Trip b
  where a.PersonID = b.PersonID and
        a.TripID ^= b.TripID and
        ((b.StrtTime<a.StrtTime<b.EndTime
          or
          b.StrtTime<a.Endtime<b.EndTime)
          or
          (a.StrtTime<b.StrtTime<a.EndTime
          or
          a.StrtTime<b.Endtime<a.EndTime));
quit;
```

Now we have our result data set Overlap.

### Data Set: Overlap

| PersonID | TripID | StrtTime | EndTime |
|----------|--------|----------|---------|
| 0002     | 1      | 0700     | 0800    |
| 0002     | 2      | 0730     | 0830    |

To identify the overlapped trips by using DATA step, you may use this logic below.

```
proc sort data=trip out=trip_s;
  by PersonID TripID StrtTime;
run;

data overlap;
  set trip_s;
  by PersonID TripID StrtTime;

  retain fst lst;

  if first.Trip then
  do;
    fst = StrTime ;
    lst = EndTime ;
  end;
  else
  do;
    if StrtTime le lst then err = 1;
  else
  do;
    fst = StrtTime;
    lst = EndTime;
  end;
  end;
run;
```

### Discussion

Both SQL procedure and DATA step accomplished the tasks: subsetting data set based on the relations between variables in *different* observations. SQL procedure is more efficient because: 1. No new transitional data set(s) need to be created. Thus, data sets do not need to be sorted, which saves resources. 2. Program is shorter. In the first example, two SORT procedures, one DATA step and one PRINT procedure are used compared to only one SQL procedure. This is because SQL procedure does not require the names of the keys (common variables) to be the same. Additionally, sorting the data sets by the common variables is not necessary.

### Reference

SAS Guide to the SQL procedure: Usage and Reference, Version 6, First Edition; SAS Institute, Cary, NC, USA

2001 National Household Travel Survey (NHTS); US Department of Transportation

### Contact Information

Zizhong Fan  
Westat  
1650 Research Boulevard  
Rockville, MD 20850  
(240) 314 -2486  
E-mail: [JamesFan@westat.com](mailto:JamesFan@westat.com)

### Disclaimer

The contents of this paper are the work of the author(s) and do not necessarily represent the options, recommendations, or practice of Westat.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.