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

Zizhong Fan MPH
Uniformed Services University of the Health Sciences

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.

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 Data Set
One classical 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 offender 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.

<table>
<thead>
<tr>
<th>CASE</th>
<th>O_SSN</th>
<th>V_SSN</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000-00-0001</td>
<td>000-00-0002</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>2</td>
<td>000-00-0002</td>
<td>000-00-0001</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>3</td>
<td>000-00-0003</td>
<td>000-00-0004</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>4</td>
<td>000-00-0005</td>
<td>000-00-0006</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>5</td>
<td>000-00-0007</td>
<td>000-00-0008</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>6</td>
<td>000-00-0008</td>
<td>000-00-0007</td>
<td>01/01/2000</td>
</tr>
</tbody>
</table>

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 result output contains only couples that both husband and wife are involved in an incident on the same day.
the single table joins itself by matching O_SSN with V_SSN, and DATE with DATE. It is a process where two identical data sets matching each other by O_SSN with V_SSN, and DATE with DATE.

**Self-join processing**

<table>
<thead>
<tr>
<th>CASE</th>
<th>O_SSN</th>
<th>V_SSN</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000-00-0001</td>
<td>000-00-0002</td>
<td>01/01/02</td>
</tr>
<tr>
<td>2</td>
<td>000-00-0002</td>
<td>000-00-0001</td>
<td>01/01/02</td>
</tr>
</tbody>
</table>

**Data Step**

```sas
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 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;
```

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.

<table>
<thead>
<tr>
<th>CASE</th>
<th>V_SSN</th>
<th>O_SSN</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>000-00-0002</td>
<td>000-00-0001</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>1</td>
<td>000-00-0001</td>
<td>000-00-0002</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>3</td>
<td>000-00-0003</td>
<td>000-00-0004</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>4</td>
<td>000-00-0005</td>
<td>000-00-0006</td>
<td>01/01/2000</td>
</tr>
<tr>
<td>6</td>
<td>000-00-0008</td>
<td>000-00-0007</td>
<td>01/01/2000</td>
</tr>
</tbody>
</table>

Then, result data set MUTUAL is created by merging data set A and B. Listing output is created by PRINT procedure.

**Discussion**

Both SQL procedure and data step accomplished the task: subsetting data set based on the relations between variables in different observations. SQL procedure is more efficient because: 1. No new transitional data sets (in this example: data set A and B) need to be created. Thus, data sets do not need to be sorted, which save resources. 2. Program is shorter. In the example in this paper, two SORT procedures, one data merge 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.

However, DATA step is preferred when processing large data sets because DATA step has a better performance than SQL procedure.

**Reference**

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

**Contact Information**

Zizhong Fan
Uniformed Services University of the Health Sciences, PSY
4301 Jones Bridge Road
Bethesda, MD 20814

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.