Paper 234-2008

# Save time! Merge SAS® Files to Themselves

Ann Malby & Sally Williams, (Financial Services) UK

## ABSTRACT:

This is a useful piece of code demonstrating the use of merging a SAS file to itself using the WHERE statement and RENAME in order to re-order data for analysis.

Why would you want to merge a file to itself? Because when pulling back data from Oracle which is not in the format you want, running this simple merge allows you to order the data into your desired format without use of the more time-consuming **PROC TRANSPOSE**. The examples included in this paper demonstrates a significant time saving.

The example given is based on dates, however this is a useful piece of code which will have a number of applications and is not restricted by industry.

The examples used in this document were done using PC SAS version 8.2. The intended audience for this paper is all SAS programmers from beginner to advanced who extract data from a database.

For the purposes of this document the method of merging a dataset to itself will be called 'self-merge'.

## KEYWORDS:

Merge, Oracle data step, WHERE, RENAME, PROC TRANSPOSE, formatting

## INTRODUCTION / PROBLEM:

Working within the financial services industry, there is a requirement to regularly pull back snapshots of account level data for multiple months. This data is extracted from an Oracle server using proc SQL, SAS (PC SAS v8.2) is then used to analyse the data which has been extracted.

However, before being able start analysing the data in SAS the format of the dataset needs to be changed as Oracle outputs the data in a 'stacked' or 'long' format whereas to analyse it in SAS the format needs to be 'wide'.

The two examples below demonstrate this using a simplified view of the balance and credit limit of three theoretic accounts between two months.

Oracle pulls the data back in a 'long' format - eg:

| **Account_no** | **MnthEnd_bal** | **YYYY_MM** | **cred_lim** |
|---|---|---|---|
| 11111111 | 100.00 | 200505 | 800.00 |
| 11111111 | 200.00 | 200506 | 800.00 |
| 22222222 | 150.00 | 200505 | 4200.00 |
| 22222222 | 120.00 | 200506 | 5000.00 |
| 33333333 | 400.00 | 200505 | 2500.00 |
| 33333333 | 600.00 | 200506 | 2500.00 |

In order to carry out analysis using SAS, the data would need to be in a 'wide' format - eg:

| **Account_no** | **MnthEnd_bal _200605** | **MnthEnd_bal _200606** | **Credit_limit_ 200605** | **Credit_limit_ 200606** |
|---|---|---|---|---|
| 11111111 | 100.00 | 200.00 | 800.00 | 800.00 |
| 22222222 | 150.00 | 120.00 | 4200.00 | 5000.00 |
| 33333333 | 400.00 | 600.00 | 2500.00 | 2500.00 |

The examples above show the account balance (at month end) for three accounts during May 2006 and June 2007. In order to analyse the data at account level, in the first example it would not be easy to do this until the structure of the data had been changed.

The second example shows how the data would ideally need to be organised. This leads on to the next logical question of what is the best way to re-organise the data?

The popular way of doing this has historically been to use PROC TRANSPOSE. The diagram below gives a high level view of the differences between using PROC TRANSPOSE and using 'self-merge'.

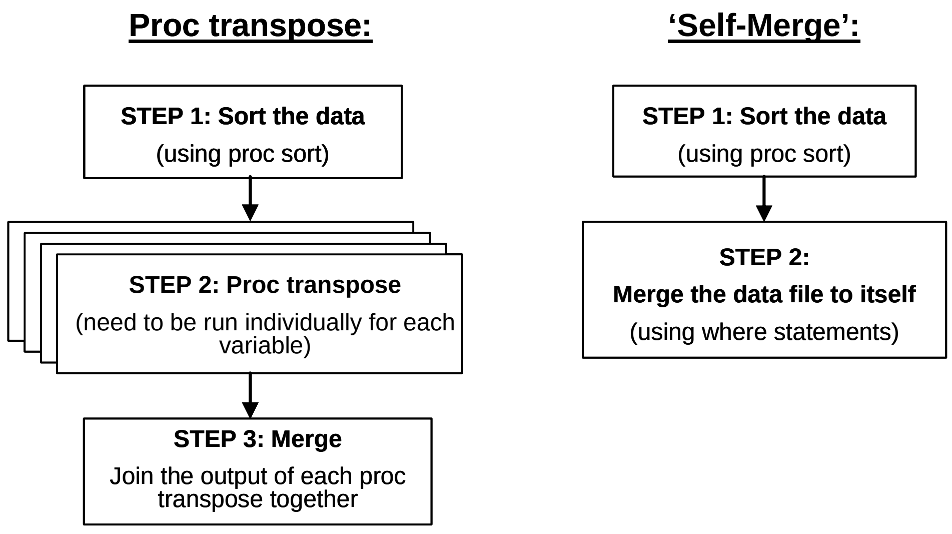

***Diagram 1: Comparison between methods***

In order to demonstrate the differences between using PROC TRANSPOSE and 'self-merge' we will first look at the code for a PROC TRANSPOSE, followed by the code for a 'self-merge' on the same data.

Finally we will look at the subsequent benefits of 'self-merge' in terms of time savings both for real time and CPU time.

## THE DRAWBACKS OF PROC TRANSPOSE:

### *Why not use PROC TRANSPOSE?*

When reshaping the data using **PROC TRANSPOSE**, you would have to run the transpose command for each variable and then merge the files back together.

If you have many variables within the dataset you would therefore need to do the PROC TRANSPOSE multiple times which will subsequently lead to longer processing times. In short, the more variables you have, the longer this would take.

***The following three steps explain how this PROC TRANSPOSE would be done, using the 'month end balance by account' example above.***

Start with the following dataset:

**Ann.test_total**

A snapshot of data for multiple accounts for May and June 2006:

| | |
|---|---|
| • Accno | (Account number) |
| • Month_end_bal | (The account balance at month end) |
| • Cred_lim | (The credit limit for this account) |

We assume the dataset is not sorted.

The aim of the following steps is to re-structure the data from the 'long' dataset to a dataset which is 'wide'.

## STEP 1 – Sort the data:

```
proc sort data=ann.test_total;
      by accno;
run;
```

## STEP 2 – Run the PROC TRANSPOSE:

```
proc transpose data=ann.test_total out=ann.test_merge_tran_var1
prefix=BCM_TOT_NEW_BAL;
      by accno;
      id snap;
      var BCM_TOT_NEW_BAL;
run;


proc transpose data=ann.test_total out=ann.test_merge_tran_var2
prefix=bcm_credit_lmt;
      by accno;
      id snap;
      var bcm_credit_lmt;
run;
```

**** Note: there are 2 PROC TRANSPOSE steps because there are 2 variables ****

## STEP 3 – Merge the output of the PROC TRANSPOSE steps:

```
data ann.test_merge_tran;
      merge ann.test_merge_tran_var1(drop=_name_ _label_)
              ann.test_merge_tran_var2(drop=_name_ _label_);
      by accno;
run;
```

HOWEVER, THERE IS A MUCH EASIER WAY! .

## ‘SELF-MERGE’:

The solution defined in this paper is that if you merge the ‘long’ format dataset with itself, by first using PROC SORT, and then ‘clever’ use of the where statement, it is possible to order the data into the desired format by a means which is both straightforward and less time-consuming.

***The following two steps explain how this ‘self-merge’ would be done, using the same ‘month end balance by account’ example above.***

Start with the following dataset:

**Ann.test_total**

A snapshot of data for multiple accounts for May and June 2006:

|  |  |
|---|---|
| • Accno | (Account number) |
| • Month_end_bal | (The account balance at month end) |
| • Cred_lim | (The credit limit for this account) |

We again assume the dataset is not sorted.

The aim of the following steps is to re-structure the data from the ‘long’ dataset to a dataset which is ‘wide’.

### STEP 1 – Sort the existing dataset:

Sort the existing dataset. In this case it is being sorted by account number:

```
proc sort data=ann.test_total;
      by accno;
run;
```

### STEP 2 – Merge the dataset to itself:

Create a new dataset by merging the existing dataset (the combined one) against itself.

```
data ann.test_merge;
      merge ann.test_total (where=(snap='200605')
rename=(month_end_bal=month_end_bal_200605 cred_lim=cred_lim_200605))
              ann.test_total (where=(snap='200606')
rename=(month_end_bal=month_end_bal_200606 cred_lim=cred_lim_200606))
              ;
      by accno;
run;
```

**End result:**

The end result is that the data is now structured in a format suitable to carry out analysis at account level using SAS, as defined in the introduction:

| **Account_no** | **MnthEnd_bal _200605** | **Credit_limit_ 200605** | **MnthEnd_bal _200606** | **Credit_limit_ 200606** |
|---|---|---|---|---|
| 11111111 | 100.00 | 800.00 | 200.00 | 800.00 |
| 22222222 | 150.00 | 4200.00 | 120.00 | 5000.00 |
| 33333333 | 400.00 | 2500.00 | 600.00 | 2500.00 |

**BENEFITS:**

In order to show the benefits of the code, the above method was applied to a larger dataset of theoretic data intended to test the benefits of the 'self-merge' over PROC TRANSPOSE.

The dataset created for this test is larger both in terms of the number of rows created (6,248,851) and the amount of variables used (14).

Further to this the data is split over six segments as opposed to the two 'month' segments given in the 'credit card account' example earlier in this document.

The benefits are defined by comparing the complexity and running times of this method against use of PROC TRANSPOSE to achieve the same results using the same data. The results of this comparison are highlighted in the table below:

| | **Using PROC TRANSPOSE** | **Using 'self-merge'** |
|---|---|---|
| Size of dataset | 6,248,851 rows | 6,248,851 rows |
| Number of variables | 14 | 14 |
| **Real time:** | | |
| PROC SORT | 29:55.11 | 29:55.11 |
| PROC TRANSPOSE | 46:05.00 | (Not applicable) |
| Proc merge | 3:43.00 | 3:03.03 |
| **CPU time:** | | |
| PROC SORT | 0:39.23 | 0:39.23 |
| PROC TRANSPOSE | 18:04.56 | (Not applicable) |
| Proc merge | 1:26.78 | 0:53.04 |
| ***TOTAL REAL TIME:*** | **01:19:43.11** | **00:32:58.14** |
| ***TOTAL CPU:*** | **00:20:10.57** | **00:01:32.27** |

***Table 1: Benefits comparison between methods***

**CONCLUSIONS:**

1. Merging a SAS file to itself in order to re-order data for analysis (rather than using PROC TRANSPOSE) gives significant benefits both in terms of real time and CPU time.

2. The examples given relate to the context of financial services, however this code is generic and is therefore not restricted to use by SAS users in any one industry.

3. The code is simple enough to be used by SAS users of all levels.

4. Those users who are likely to use this code, would be likely to re-use it a number of times, gaining time saving benefits each time it is used.