Paper 071-2007

PROC SQL - The Dark Side of SAS®?

Kirsty Lauderdale, PRA International, Victoria, BC

ABSTRACT

PROC SQL - two dreaded words for me up until about 6 months ago. As a die-hard DATA step programmer, I shunned SQL for the ease of DATA step programming without having to succumb to dealing with things like "RIGHT OUTER UNION JOIN." Unfortunately, the time comes to us all when we have to deal with stuff we try to avoid. For me, that time came when I inherited some uncommented SQL code. Needless to say, I'm not a full convert - I still love DATA step, but I do see the efficiencies available in SQL code, and think everyone should at least understand the basics.

Intended audience – beginner to intermediate, not limited to any particular operating system, based on SAS 8.2.

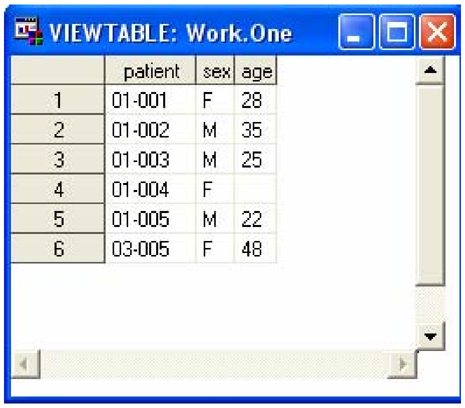
INTRODUCTION

To be honest, DATA step and SQL aren't really all that different. Sure, they use different syntax, but when it comes down to it they are both pretty easy to use, as long as you remember their differences.

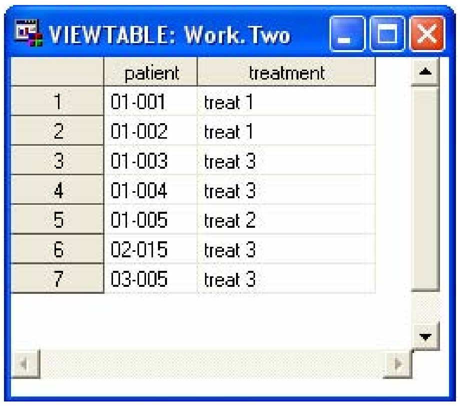
The greatest challenge for me to overcome was setting and merging, I've been using DATA step for over 6 years… why should I change now? In DATA step it's easy, MERGE or SET and use IF A/B/C etc. statements. Why did SQL have to be so complicated? I mean it has INNER JOIN, OUTER JOIN, RIGHT JOIN, and LEFT JOIN.

PROC SQL's version of dataset concatenation actually only uses one statement - JOIN. The statement in front of JOIN defines the type of concatenation performed.

To demonstrate the differences between DATA step and PROC SQL, I created two very simple dummy datasets. These datasets will be used as the basis for the examples that follow.



	patient	sex	age
1	01-001	F	28
2	01-002	M	35
3	01-003	M	25
4	01-004	F	
5	01-005	M	22
6	03-005	F	48



	patient	treatment
1	01-001	treat 1
2	01-002	treat 1
3	01-003	treat 3
4	01-004	treat 3
5	01-005	treat 2
6	02-015	treat 3
7	03-005	treat 3

SET

The SET statement is probably the simplest data manipulation to perform. It is used to read observations from more than one dataset and stack the datasets one on top of the other. I don't use SET very often, but I include the example below to demonstrate that "anything DATA step can do, SQL can do."

DATA STEP SYNTAX	PROC SQL SYNTAX

`DATA set;`
`   SET one two;`
`RUN;`
	`PROC SQL;`
	`   CREATE TABLE set AS`
	`       SELECT * FROM one`
	`       OUTER UNION`
	`       SELECT * FROM two;`
	`QUIT;`

You can also use BY statements in DATA step or WHERE statements in SQL in order to sort the data, while setting it together.

MERGE

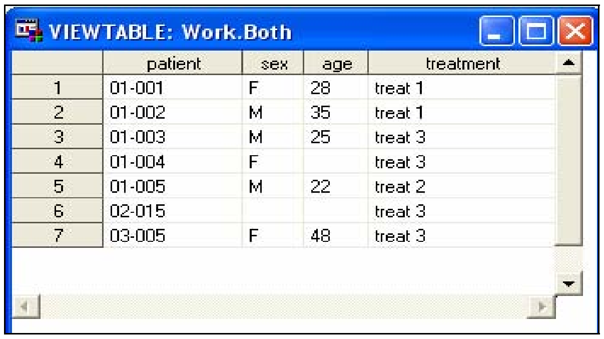
The MERGE statement is probably one of the most commonly used pieces of syntax in the SAS® programming language. MERGE is used to join observations from multiple data sources into a single observation. SAS defines any MERGE that uses a BY statement as a "MANY-TO-MANY" merge.

In DATA step this requires a BY statement and an IF A OR B statement, in SQL it's called a FULL JOIN using ON.

DATA STEP SYNTAX	PROC SQL SYNTAX

`PROC SORT DATA=one;`
`   BY patient;`
`RUN;`
 	
`PROC SORT DATA=two;`
`   BY patient;`
`RUN;`
 	
`DATA both;`
`   MERGE one (IN=A) two (IN=B);`
`   BY patient;`
`   IF A OR B;`
`RUN;`
	`PROC SQL;`
	`   CREATE TABLE Both AS`
	`       SELECT a.*, b.*`
	`       FROM one AS a FULL JOIN`
	`             two AS b`
	`            ON a.patient=b.patient;`
	`QUIT;`

This results in the following dataset:



VIEWTABLE: Work.Both

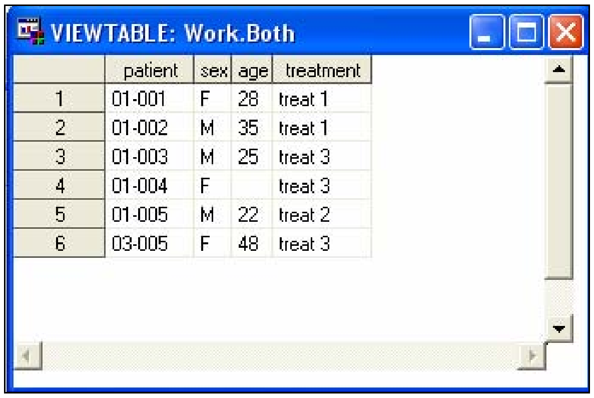
	patient	sex	age	treatment
1	01-001	F	28	treat 1
2	01-002	M	35	treat 1
3	01-003	M	25	treat 3
4	01-004	F		treat 3
5	01-005	M	22	treat 2
6	02-015			treat 3
7	03-005	F	48	treat 3

A more common--and more useful--merge is one that keeps observations that are present in both datasets. In DATA step, use IF A AND B and in SQL use an INNER JOIN. The subsetting IF statement causes the DATA step to process only the observations from the datasets that meet the condition of the expression. The ON statement in SQL follows the exact same rules.

DATA STEP SYNTAX	PROC SQL SYNTAX

`PROC SORT DATA=one;`
`   BY patient;`
`RUN;`
 	
`PROC SORT DATA=two;`
`   BY patient;`
`RUN;`
 	
`DATA both;`
`   MERGE one (IN=A) two (IN=B);`
`   BY patient;`
`   IF A AND B;`
`RUN;`
	`PROC SQL;`
	`   CREATE TABLE Both AS`
	`       SELECT a.*, b.*`
	`       FROM one AS a INNER JOIN`
	`             two AS b`
	`            ON a.patient=b.patient;`
	`QUIT;`

The resulting dataset:


VIEWTABLE: Work.Both

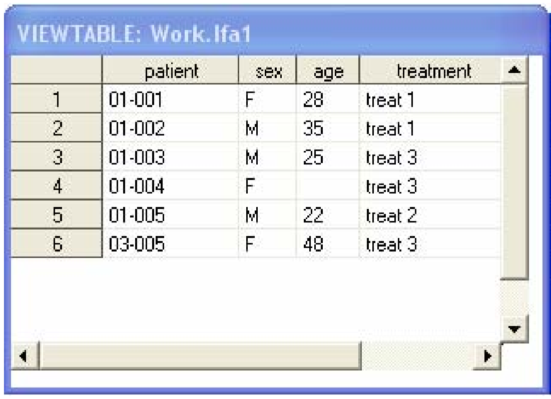
	patient	sex	age	treatment
1	01-001	F	28	treat 1
2	01-002	M	35	treat 1
3	01-003	M	25	treat 3
4	01-004	F		treat 3
5	01-005	M	22	treat 2
6	03-005	F	48	treat 3

We often have cases where we want to keep observations that are in one dataset, but also use some variable from the other. Under DATA step we would use IF A or IF B, using SQL we use RIGHT JOIN or LEFT JOIN.

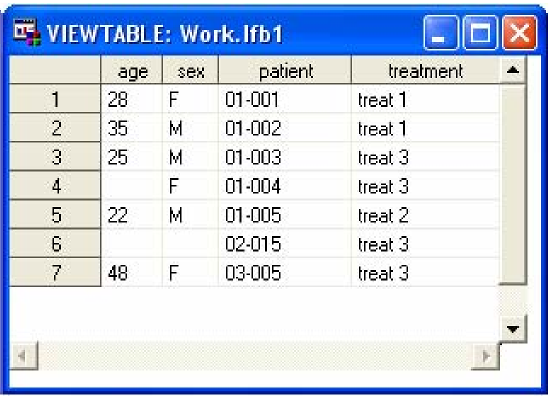
DATA STEP SYNTAX	PROC SQL SYNTAX

`* Sort statements here *;`
 	
`DATA ifa1;`
`   MERGE one (IN=A) two (IN=B);`
`   BY patient;`
`   IF A ;`
`RUN;`
 	
DATA STEP SYNTAX
 	
`* Sort statements here *;`
 	
`DATA ifb1;`
`   MERGE one (IN=A) two (IN=B);`
`   BY patient;`
`   IF B;`
`RUN;`
	`PROC SQL;`
	`   CREATE TABLE ifa1 AS`
	`       SELECT a.*,`
	`              b.treatment`
	`       FROM one AS a LEFT JOIN`
	`            two AS b`
	`            ON a.patient=b.patient;`
	`QUIT;`
	
	PROC SQL SYNTAX
	
	`PROC SQL;`
	`   CREATE TABLE ifb1 AS`
	`       SELECT a.age,`
	`              a.sex,`
	`              b.*`
	`       FROM one AS a RIGHT JOIN`
	`             two AS b`
	`            ON a.patient=b.patient;`
	`QUIT;`

The results:


VIEWTABLE: Work.Ifa1

	patient	sex	age	treatment
1	01-001	F	28	treat 1
2	01-002	M	35	treat 1
3	01-003	M	25	treat 3
4	01-004	F		treat 3
5	01-005	M	22	treat 2
6	03-005	F	48	treat 3


VIEWTABLE: Work.Ifb1

	age	sex	patient	treatment
1	28	F	01-001	treat 1
2	35	M	01-002	treat 1
3	25	M	01-003	treat 3
4		F	01-004	treat 3
5	22	M	01-005	treat 2
6			02-015	treat 3
7	48	F	03-005	treat 3

So, there you have it: Simplified setting and merging demonstrated using both DATA step and SQL code.

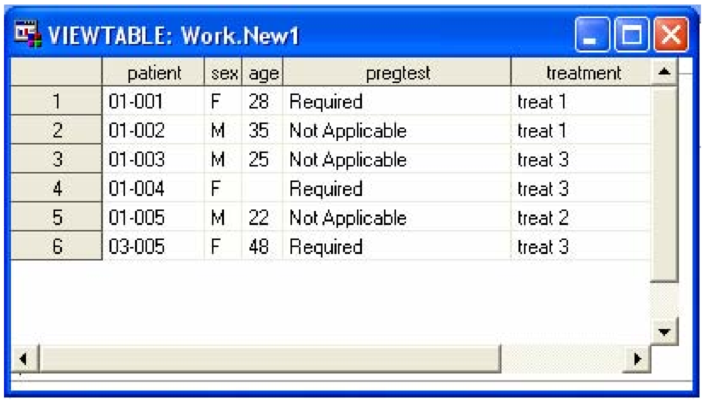
SQL does have some significant advantages. There is no need for pre-sorting variables and there are no issues with discrepancies between variable types (e.g., if in one dataset you have a numeric variable and in the other a character). Also, PROC SQL appears to run faster than DATA step code on smaller tables.

ADDITIONAL FEATURES

Aside from the differences in concatenating datasets, the SQL procedure also has some other nifty little uses, like the CASE statement. The CASE statement can be used to create variables out of existing data, while performing a merge, as follows:

```
PROC SQL;
   CREATE TABLE new1 AS
      SELECT a.*,
         CASE sex
            WHEN 'F' THEN 'Required'
            WHEN 'M' THEN 'Not Applicable'
            ELSE 'Please Advise'
         END AS pregtest,
             b.treatment
      FROM  one AS a INNER JOIN
            two AS b
            ON a.patient=b.patient;
QUIT;
```

This step creates a new variable named "pregtest" with character values based on data from the sex variable, as follows:



	patient	sex	age	pregtest	treatment
1	01-001	F	28	Required	treat 1
2	01-002	M	35	Not Applicable	treat 1
3	01-003	M	25	Not Applicable	treat 3
4	01-004	F		Required	treat 3
5	01-005	M	22	Not Applicable	treat 2
6	03-005	F	48	Required	treat 3

MACRO VARIABLES

In my opinion, by far the most powerful SQL use is for counts and creating macro variables. The syntax is similar for both so I'll show the macro variable code creation. This small snippet of code has proven invaluable to me and I'll never go back to using DATA step for this.

```
PROC SQL NOPRINT;
   SELECT TRIM(LEFT(PUT(COUNT(DISTINCT patient),8.))) INTO : trt1 - : trt3
   FROM two
   GROUP BY treatment;
QUIT;

%PUT trt1=&trt1. trt2=&trt2. trt3=&trt3.;
```

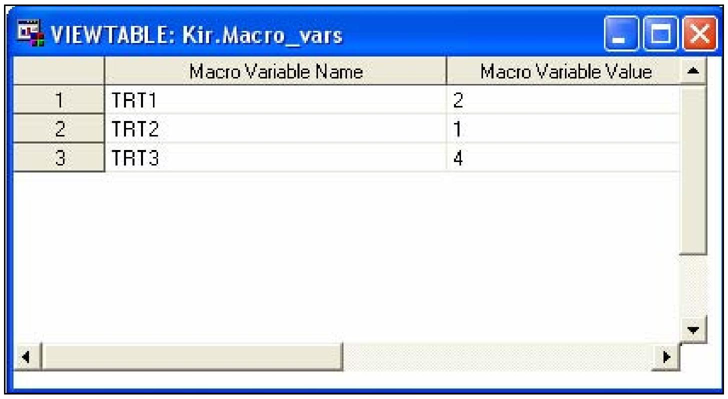
Log Output

```
trt1=2 trt2=1 trt3=4
```

Another favourite tip when using SQL has been to produce the macro variables in a dataset table, and store the view table in a permanent directory, so I can view the numbers whenever needed. To do this, use the following code:

```
PROC SQL NOPRINT;
   CREATE TABLE kir.macro_vars AS
   SELECT name, value
   FROM sashelp.vmacro
   WHERE scope='GLOBAL' AND name IN ('TRT1' 'TRT2' 'TRT3')
   ORDER BY 1;
QUIT;
```

The resulting dataset should look something like this:



	Macro Variable Name	Macro Variable Value
1	TRT1	2
2	TRT2	1
3	TRT3	4

CONCLUSION

In this document I've demonstrated just a few of the most common and basic uses of PROC SQL. Hopefully, this will encourage you to lose the fear of using SQL and to further your use by examining the additional functionality of this flexible procedure.

And, as a side note for programmers of all levels - please always comment your code!!

REFERENCES

SAS Institute Inc. 1999. SAS OnlineDoc® documentation, Version 8.

CONTACT INFORMATION

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

Kirsty Lauderdale
PRA International
300-730 View St,
Victoria, BC, V8W 3Y7
Tel: 250-483-4477
lauderdalekirsty@praintl.com