Join or Merge? The Differences Between PROC SQL Join and Data Step Merge and When to Use Them

Ted A. Shelly, Pinnacle Project Group LLC, Stamford, CT

ABSTRACT

A very common data manipulation task is to bring two or more sets of data together based on a common key. In SQL, this is known as a join. The SAS® DATA step has the MERGE statement that permits the same thing. If you know SQL, you might never look at using MERGE. And if you know data step programming very well, you might never bother learning how to do an SQL join. But each method has its own advantages and it pays to know when it is best to use either of the two methods.

INTRODUCTION

Data manipulation and analysis often requires that you bring together data from different sources or from different data sets. Very often, the way you bring this data together is by using a common set of variables that form a key to the data. This has been true going all the way back to a time when data was dealt with on paper. In the early days of computer processing, sequential files were often merged to do such things as update a master file from transactions.

With the development of the relational data model and the structured query language (SQL), a new method of bringing together data from different data sets appeared – the join. Those working with relational databases now had a simple and powerful way to bring this data together.

SAS data step processing is built on the older model of working with sequential files. Using the MERGE statement with the BY option a programmer could easily bring together data sets with matching keys, all without having to hand-code the logic required to do the matching.

When SAS introduced PROC SQL, programmers had a choice of two very different ways to bring their data together. A SAS data set is exactly equivalent to the concept of a table (formally, a "tuple") in the relational model. The programmer could use either data step MERGE or use a join in PROC SQL. Which one to use can be a matter of personal preference. But if you learn each method well, you can choose the best tool for each specific situation.

THE SIMPLEST JOIN

Lets create two very simple datasets:

```
data A;
    input key data1;
datalines;
1 512
2 1024
3 2048
4 4096
5 8192
;
run;
data B;
    input key data2 $;
datalines;
1 AAA
2 BBB
```

```
3 CCC
4 DDD
5 EEE
;
run;
```

To join these data sets we can use one line in PROC SQL:

```
proc sql;
    select * from A natural join B;
quit;
```

The natural join lets the system do the work of choosing the key(s) to use to do the join. It will assume that common variables from the two data sets are the key. In our case, this is valid. The result is a report as follows:

key	data2	data1
1	AAA	512
2	BBB	1024
3	CCC	2048
4	DDD	4096
5	EEE	8192

Note that there are two other ways to code the above join, with syntax that may be more familiar but are exactly equivalent.

```
proc sql;
    select * from A join B on a.key=b.key;
quit;

or

proc sql;
    select * from A, B where A.key=B.key;
quit;
```

Joins, of course, can get much more complex. The key variables may not have the same name in each data set, so a natural join can't be used. It is also quite possible to join more than two tables in the same statement. You can even do transformations and conditional logic in SQL, but that is beyond the scope of this paper.

THE SIMPLEST MERGE

The join process described above can also be done using data step programming, as shown in the example below. (Caution: if the data sets used in a MERGE with the BY option are not already sorted, use a PROC SORT step first.)

```
data D;
  merge A B;
  by key;
proc print data=D;
```

```
run;
```

Which is simpler to code or understand? About equal, I would say.

Note one immediate difference. The SELECT statement creates a printed report by default. Whereas the data step creates a new data set by default and you have to add another statement to print the result. To have SQL create a data set, add one line to the code above:

```
proc sql;
    create table C as select * from A natural join B;
quit;
```

A MORE COMPLEX SITUATION

The two data sets we started with are perhaps unrealistically simple. Each record in the first data set is matched by a record in the second. More commonly, you will not have such a neat situation. Let's create another data set.

```
data B2;
    input key data2 $;
datalines;
1 AAA
2 BBB
3 CCC
4 DDD
6 FFF
;
run;
```

Our simple natural join:

```
proc sql;
   select * from A natural join B2;
quit;
```

Creates this output:

key	data2	data1
1	AAA	512
2	BBB	1024
3	CCC	2048
4	DDD	4096

Since key=5 in A and key=6 in B had no match in the other data set, they don't show up at all in the result. Is that what we expected or wanted? Disappearing data can be a real issue.

In contrast, our data step would produce this output:

key	data1	data2
1	512	AAA
2	1024	BBB
3	2048	CCC
4	4096	DDD
5	8192	
6	•	FFF

Now that's more like it. It is easy to see the missing data. So if you want to see missing values, use data step, right? But we can do the same with SQL:

```
proc sql;
   select * from A full outer join B2 on A.key=B2.key;
quit;
```

This creates essentially the same output as the data step merge.

A MORE REALISTIC MATCHING SCENARIO

More often, you will be interested in matching a master data set against a data set of matching transactions. There might not be a transaction for every master and the master drives the processing. We can create a new data set to represent this situation.

```
data B3;
    input key data2 $;
datalines;
1 AAA
3 CCC
4 DDD
;
run;
```

Then to match the master to the transactions we can use this data step.

```
data F;
  merge A(in=a) B3(in=b);
    by key;
  if a;
run;
```

That's simple enough, though it will seem a bit obscure if you don't know data step programming well. The results are all observations from A and those from B that match A. To accomplish the same thing with SQL, you would use a left outer join.

```
proc sql;
   select * from A left outer join B3 on A.KEY=B3.KEY;
quit;
```

Some would say that this reads a bit more easily.

If you're wondering if there is an "inner join", there is – it is what we referred to above as a simple join.

MERGE OR JOIN WITH NO KEY

Now you've seen the major types of joins and merge types that use a key. Data step merge also allows you to do a merge without a key, which works nicely if you know you have matching sets of records, as in our first two data sets A and B. This is extremely simple code.

```
data G;
  merge A B;
run;
```

The result looks like the following, which is the same as the merge by key for the same data.

key	data1	data2
1	512	AAA
2	1024	BBB
3	2048	CCC
4	4096	DDD
5	8192	EEE

You might think that the following SQL code would be the same as the above data step code.

```
proc sql;
   select * from A join B;
quit;
```

But it gives you an error. A join must have a key. You could code it this way:

```
proc sql;
    select * from A, B;
quit;
```

That looks good, but what a result!

_		_	_
key	data1	key	data2
1	512	1	AAA
2	1024	1	AAA
3	2048	1	AAA
4	4096	1	AAA
5	8192	1	AAA
1	512	2	BBB
2	1024	2	BBB
3	2048	2	BBB
4	4096	2	BBB

5	8192	2	BBB
1	512	3	CCC
2	1024	3	CCC
3	2048	3	CCC
4	4096	3	CCC
5	8192	3	CCC
1	512	4	DDD
2	1024	4	DDD
3	2048	4	DDD
4	4096	4	DDD
5	8192	4	DDD

Every observation of A is matched with every observation in B. this is the dreaded Cartesian join. If you do this with a couple of even moderately-sized datasets, you may notice your query running for a very long time and producing massive output. Use this with great care and only on very small datasets.

THREE-WAY JOIN/MERGE

Suppose you have three data sets you need to join. It might be a customer data set joined to the customer address data set and in turn joined to a data set of demographics where zip code is the key. The SQL could be something like this:

```
select C.*, A.*, D.*
from customer C, address A, demographics D
where C.cust_no = A.cust_no and A.zip = D.zip;
```

That's pretty simple. How would you do this with data step MERGE? You can merge 3 datasets, but they must have the same key (BY variables). You could use two data steps with a sort step in between. But there is a simpler and more efficient solution. I won't give the trick here, but the clue is "hash" ³.

CONCLUSION

It doesn't matter very much whether you use data step merge or and SQL join. Use whichever is more comfortable to you. But depending on the objective of your coding, one may be substantially simpler than the other. The natural join of SQL is particularly appealing because of its simplicity and readability.

Experiment with both and understand them well. You'll have a set of powerful tools in your arsenal.

REFERENCES

- [1] Smith, Khan, Zhang, SUGI 28, PROC SQL vs. Merge. The Miller Lite Question of 2002 and Beyond.
- [2] Kahane, NESUG 2008, Using DATA Step MERGE and PROC SQL JOIN to Combine SAS® Datasets

[3] Secosky, Bloom, SAS Global Forum 2007. Getting Started with the DATA Step Hash Object.

Contact Information

Your comments and questions are welcome: Ted A. Shelly Principal Consultant Pinnacle Project Group LLC Tel: +1 203 388 8053 tshelly@pinnacleprojectgroup.com

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 trademarks of their respective companies.