How to use Proc SQL

Tasha Chapman, Oregon Department of Consumer and Business Services

SQL[1] is most commonly known as an acronym for "Structured Query Language." SQL is primarily used for modifying and querying databases. Of course, so is the SAS® programming language, so many Data Steppers ask, "Why should I learn SQL?" My response is, "Why shouldn't you?"

SQL a fairly ubiquitous programming language used in a number of different platforms, including Oracle, MS Access, SPSS, and others. Knowing this standard language, on top of your SAS skills, will only make you more attractive to prospective employers. In addition, SQL can do many of the things that Data Steps and Procedure steps can do, but sometimes SQL can do it better, easier, or more efficiently.

When PROC SQL first became available in SAS version 6.0, it allowed SAS Users to incorporate SQL syntax in their programs. The purpose of this paper is to acquaint the reader with the basics of Proc SQL syntax, including an introduction to table joins.

The following examples were generated using SAS version 9.2.

Getting started – the basics of Proc SQL

SQL has many uses, but these uses can be divided into two basic categories: a) summarizing and reporting data; and b) creating a data table to be used for future analysis. Let's start with the basic building blocks of the SAS SQL procedure.

The most important thing you must know about SQL is this: Every SQL query must have a SELECT clause and FROM clause.

The **SELECT** clause is used to select the variables from the source table. It is analogous to a KEEP statement in a DATA step, or a VAR statement in a SAS procedure. You can select a single variable, or multiple variables in a SELECT clause. In our example, we'll select the variables "Title," "Author," and "ISBN." Multiple variables must be separated by commas.

The **FROM** clause tells SAS which data table(s) the pull the data from. It is analogous to a SET statement in a DATA step, or a DATA= option in a SAS procedure. You can pull data from one source table or multiple source tables. In our example, we'll select data from a single source dataset titled "Books."

In SAS, SQL statements are sandwiched between a Proc SQL statement and a QUIT statement. Unlike most procedures, the SQL procedure does not require a RUN statement. In addition, multiple SQL statements can be used within a single SQL procedure.

The syntax to select the three variables from the dataset Books using Proc SQL is:

[1] The acronym is sometimes pronounced "S-Q-L," "Sequel," or even (as the author has heard on one bizarre occasion) "Squirrel."

```
proc sql;
select Title, Author, ISBN
from Books;
quit;
```

The above procedure will print the data for Title, Author, and ISBN, from the dataset Books, just as if we had used the Print Procedure. A sample of the output is below. We could have also sent this data to a new table, which we'll cover later.

Example 1. Basic SQL statement

Title	Author	ISBN
The Little SAS Book	Delwiche	1590473337
SAS Survival Handbook	Wiseman	0060578793
SAS for Dummies	McDaniel	0471788325
Learning SAS by Example	Cody	1599941651
Output Delivery System	Haworth	158025859X
SAS Functions by Example	Cody	1590473787
Annotate: Simply the Basics	Carpenter	1580255787
SAS Programming Shortcuts	Aster	1891957112
Survival Analysis Using SAS	Allison	155544279X
Longitudinal Data and SAS	Cody	1580259243
SAS Macro Programming	Burlew	1590478827

There are a few things that should be noted about the Proc SQL syntax. First, the SQL clauses are ended by a single semi-colon (as opposed to other typical SAS procedures in which a semi-colon ends each individual statement). Second, as mentioned earlier, there is no RUN statement, only a QUIT statement at the end of the procedure. It is possible to put more than one SQL statement within a single SQL procedure. Each statement is ended by a semi-colon, and the full SQL procedure is ended with a QUIT. For example:

```
proc sql;

select Title, Author, ISBN
from Books;

select ISBN, Order_date, Customer
from Orders;

quit;
```

The Veruca Salt predicament: What to do when you want everything…

In the above examples, we specifically listed which variables we wanted using the SELECT clause. However, it is possible to select every variable in the source dataset, without having to list each variable by name in the SELECT clause. This is done using an asterisk:

```
proc sql;
select *
from Books;
quit;
```

The above syntax will bring in all the variables from the Books table. It is analogous to using KEEP _ALL_ in a DATA step.

Selecting rows using the WHERE clause

As written above, the SQL procedure will select all of the rows from the source table. However, perhaps you only wanted to bring in certain rows that met certain criteria. This is most commonly controlled using the WHERE clause.

The **WHERE** clause is used to implement criteria for selecting rows from the source table. It is the same as the WHERE statement used in DATA steps and SAS Procedures. Using a WHERE clause is optional, but it can be useful in getting correct results and also is important for efficient programming. When using Proc SQL to create data tables, the WHERE clause can help reduce the demand on computing resources by only bringing in the data that is actually needed.[2]

Let's say that instead of bringing in all the observations from the Books table we only want to bring in the information for those books written by Cody. To do this, we use the WHERE clause to restrict the incoming data:

```
proc sql;
select Title, Author, ISBN
from Books
where Author = 'Cody';
quit;
```

Example 2. Where Author = "Cody"

Title	Author	ISBN
Learning SAS by Example	Cody	1599941651
SAS Functions by Example	Cody	1590473787
Longitudinal Data and SAS	Cody	1580259243

It should be noted that the WHERE clause **is** case sensitive. As such, `where Author = 'CODY'` or `where Author = 'cody'` would not have resulted in a match and no rows would have been selected.

Sorting rows using the ORDER BY clause

The output data from the SQL procedure can be sorted using the optional clause ORDER BY.

[2] For more information on the importance of the WHERE clause in efficient programming, see ***Using SAS with Oracle: Writing efficient and accurate SQL***, by Tasha Chapman and Lori Carleton.

The **ORDER BY** clause sorts the data by one or more variables. The data can be sorted in either ascending or descending order (the default is ascending). This is analogous to Proc Sort in SAS.

To sort the data alphabetically by title, the syntax would be:

```
proc sql;
select Title, Author, ISBN
from Books
where Author = 'Cody'
order by Title;
quit;
```

Example 3. Order by Title

Title	Author	ISBN
Learning SAS by Example	Cody	1599941651
Longitudinal Data and SAS	Cody	1580259243
SAS Functions by Example	Cody	1590473787

To sort the data in descending order, use the DESCENDING or DESC option after the variable name. To sort by descending title, the syntax would be:

```
proc sql;
select Title, Author, ISBN
from Books
where Author = 'Cody'
order by Title desc;
quit;
```

You can also sort by more than one variable. The multiple variables in the ORDER BY clause are separated by commas. To sort the data first by descending title, then ascending ISBN code:

```
proc sql;
select Title, Author, ISBN
from Books
where Author = 'Cody'
order by Title desc, ISBN;
quit;
```

Renaming, Relabeling, and More

You can apply some basic renaming, labeling, functions, calculations, and reformatting in the SELECT clause of an SQL statement[3]. For example, if you wanted to change the name of the ISBN variable to "Book_ID" you would use the AS keyword to do so. For example:

[3] If you are working with a non-SAS database and an older version of SAS (prior to 9.2), using some of these commands in your SQL may create inefficiencies in your code. For more information, see ***Using SAS with Oracle: Writing efficient and accurate SQL***, by Tasha Chapman and Lori Carleton.

```
proc sql;
select Customer,
       Order_Date,
       ISBN as Book_ID
from Orders;
quit;
```

Or if you wanted to change the label for Customer to read as "Book ordered by:" in the resulting output, or display an Order Date using the mmddyy10 format, you could do so by using the LABEL= and FORMAT= options respectively, like so:

```
proc sql;
select Customer label='Book ordered by:',
       Order_Date format=mmddyy10.,
       ISBN as Book_ID
from Orders;
quit;
```

You get the drift, so let's go crazy. Let's say that in addition to everything above, we also wanted to label Order_Date to display as "Order date:" and label ISBN to display as "Book ID:". Then we also decided we wanted to create a new variable called Pred_Ship_Date (the predicted shipping date) that was calculated to be two days after the Order Date. Finally we want to format this using the worddate format and label it "Predicted shipping date:"… Think we can do it?[4]

```
proc sql;
select Customer label='Book ordered by:',
       Order_Date format=mmddyy10. label='Order date:',
       ISBN as Book_ID label='Book ID:',
       (Order_Date + 2) as Pred_Ship_Date format=worddate.
            label='Predicted shipping date:'
from Orders;
quit;
```

In the end our output might look like so:

Book ordered by:	Order date:	Book ID:	Predicted shipping date:
Jan Thomas	01/23/2009	1599941651	January 25 , 2009
Kelly Sims	07/07/2007	158025859X	July 9, 2007
Joe Young	05/09/2008	1590473787	May 11, 2008
Brian Smith	12/24/2008	1891957112	December 26, 2008

Creating datasets using Proc SQL

In the above examples, the SQL procedure is performing similarly to Proc Print. It's taken the few variables we've selected from a specified table and printed the data to the output window. However, the SQL procedure is more commonly used to create data tables, similar to a DATA step. To create a data table in SQL we simply add a CREATE TABLE clause.

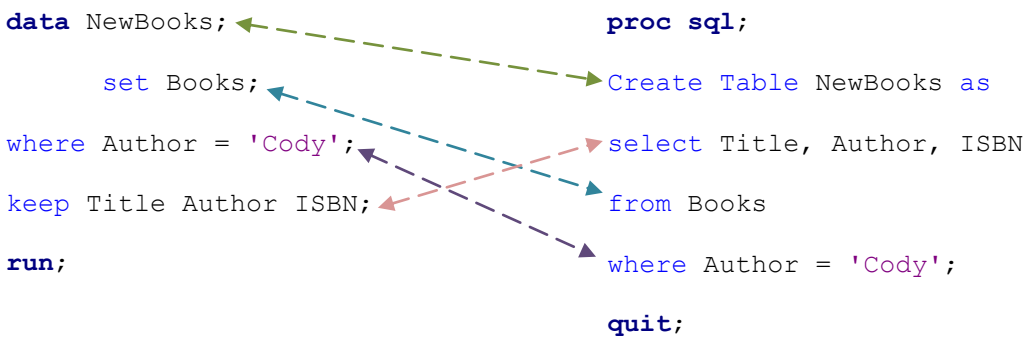
[4] To the person in the back row who said "No"… I heard that, and I'm highly disappointed in your lack of faith in our abilities.

CREATE TABLE will tell Proc SQL to create a dataset with the specified data. It is analogous to the DATA statement in a DATA step, or the OUTPUT statement in a SAS procedure. In the CREATE TABLE clause, you first list the name of the new dataset, followed by the keyword AS. The CREATE TABLE clause is then followed by the applicable SQL statements that will be used to create the dataset (SELECT, FROM, etc.).

In Example 2 (above), we printed the Author, Title and ISBN for those books written by Cody. Instead of printing this output, we can send it to a dataset called "NewBooks" using the CREATE TABLE clause:

```
proc sql;
create table NewBooks as
select Title, Author, ISBN
from Books
where Author = 'Cody';
quit;
```

For comparison, if we were to write the above procedure as a DATA step, it might be:



Referencing Libraries

You can reference SAS libraries in the SQL procedure in the same way that libraries are referenced in the DATA step. For example, if you wanted to use datasets from a library on the C:\ drive, you would use a LIBNAME statement to create the library reference, then use the library reference in your SQL:

```
Libname in 'C:\SAS\Chapmantl\';

proc sql;
Create Table NewBooks as
select Title, Author, ISBN
from in.Books
where Author = 'Cody';
quit;
```

If you are using SQL to read data from a non-SAS database management system (DBMS), your LIBNAME statement may be considerably longer. The LIBNAME statement can be used to pass important information to the DBMS, such as user names, passwords, and other engine/host

options. While the particulars of the LIBNAME statement are beyond the scope of the current paper, below is an example of something you might see:

```
Libname oralib
      oracle
      user = sas_user
      password = my_password
      path = "dev.cdb.or.us"
      connection = unique;
```

Joining tables

One of the most useful functions of the SQL procedure is the ability to join two or more related tables. Table joins can also be done using the MERGE statement in a DATA step, but they can often be done with fewer keystrokes in Proc SQL.

There are many different types of table joins: inner joins, left joins, right joins, full joins, etc. Before we talk about the many types of joins, let's work on the basic syntax of a join. To illustrate how a join works, let's start with our original Books dataset:

Books

Title	Author	ISBN
The Little SAS Book	Delwiche	1590473337
SAS Survival Handbook	Wiseman	0060578793
SAS for Dummies	McDaniel	0471788325
Learning SAS by Example	Cody	1599941651
Output Delivery System	Haworth	158025859X
SAS Functions by Example	Cody	1590473787
Annotate: Simply the Basics	Carpenter	1580255787
SAS Programming Shortcuts	Aster	1891957112
Survival Analysis Using SAS	Allison	155544279X
Longitudinal Data and SAS	Cody	1580259243
SAS Macro Programming	Burlew	1590478827

In addition to our Books dataset, we also have a dataset "Orders," which lists the ISBN and Order Date for every order that was received:

Orders

BookNum	Order_Date
1599941651	01/23/2009
158025859X	07/07/2007
1590473787	05/09/2008
1891957112	12/24/2008
155544279X	11/05/2007
1580259243	08/30/2008
0471788325	03/25/2009
1563893428	05/29/2009
0393327345	08/03/2008

We can use SQL to join these two tables together using the common variable[5] ISBN. This will allow us to see the Author and Title of the books that were ordered, along with the date they were ordered on.

Table joins are usually performed in the FROM clause. In the FROM clause, you need to list the tables that will be joined together, the type of join you will use (more on this later), and the variable(s) that the tables have in common:

```
from First Table <Type of Join> Second Table
on Matching Variables
```

If we want to join Books to Orders using ISBN as a common variable, the syntax for the FROM clause would be:

```
from Books join Orders
on Books.ISBN=Orders.ISBN
```

Now, I know what you're saying… What's this "Books.ISBN" and "Orders.ISBN" stuff? Now that we're using more than one source dataset, SAS needs to know which dataset to use to find each variable. This is sometimes referred to as a **column qualifier**, as it indicates which table (or dataset) contains the column (or variable) being referenced. "Books.ISBN" is short for "The ISBN variable in the dataset Books," and "Orders.ISBN" is short for "The ISBN variable in the dataset Orders."

The dataset name must also be referred to when calling variables in the SELECT clause[6] when joining two or more tables. As such, our new query looks like so:

[5] A common variable is also referred to as a "key".
[6] Sometimes, if the variable is only found on one table, SAS will still be able to find it without the dataset name. However, it is usually good practice to add the dataset name to the variable when joining two or more tables, just in case.

```
proc sql;
Create Table NewBooks as
select  Books.Title,
        Books.Author,
        Books.ISBN,
        Orders.Order_Date
from Books join Orders
on Books.ISBN=Orders.ISBN;
quit;
```

Using the above syntax, Proc SQL will perform an inner join, including only those records that appear on both the Books table and the Orders table:

Example 4. Inner joins

Title	Author	ISBN	Order_Date
SAS for Dummies	McDaniel	0471788325	03/25/2009
Learning SAS by Example	Cody	1599941651	01/23/2009
Output Delivery System	Haworth	158025859X	07/07/2007
SAS Functions by Example	Cody	1590473787	05/09/2008
SAS Programming Shortcuts	Aster	1891957112	12/24/2008
Survival Analysis Using SAS	Allison	155544279X	11/05/2007
Longitudinal Data and SAS	Cody	1580259243	08/30/2008

Table aliases

We'll discuss more about the different types of joins a little later in the paper. But first let's talk about table aliases.

If you are only selecting a few variables, writing "Books." or "Orders." before each variable might not be so bad. But what if you were selecting a lot of variables? Or what if your table names were particularly long? After a while, all those keystrokes would get tiring.

Instead of writing the full table name every time, SQL allows you to use table aliases instead. A **table alias** is a temporary, alternate name for a table. It only applies during a single SQL statement and can be a handy shortcut for referring to tables. A table alias can be as short or as long as you like, but since single letter table aliases are the shortest, they are the most commonly used.

Table aliases are assigned in the FROM clause, using the AS keyword. For example:

```
from Books as B
```

Now, when using the column qualifier for the table books, you can use the "B." instead of "Books." For example:

```
proc sql;
Create Table NewBooks as
select  B.Title,
        B.Author,
        B.ISBN,
        O.Order_Date
from Books as B join Orders as O
on B.ISBN=O.ISBN;
quit;
```

Once a table alias is assigned, it must be used throughout the rest of the SQL statement. You can't mix and match between "Books." and "B." once the table alias for Books has been assigned to B. However, once you've created such a handy alias, why would you ever want to use the full table name again?

SQL joins versus DATA step MERGE

When it comes to keystrokes, Proc SQL has a huge advantage over the DATA step MERGE. You can join tables using many fewer lines of code in SQL.

Let's say, for example, that our Orders table had a different name for the variable ISBN. Instead, the Orders dataset called this variable "BookNum." The Books dataset, on the other hand, still called this variable "ISBN."

Orders

BookNum	Order_Date
1599941651	01/23/2009
158025859X	07/07/2007
1590473787	05/09/2008
1891957112	12/24/2008
155544279X	11/05/2007
1580259243	08/30/2008
0471788325	03/25/2009
1563893428	05/29/2009
0393327345	08/03/2008

If we were to use a DATA step MERGE, this would be a big problem, because common variables must have identical field names. In addition, the two datasets must be sorted by the common variable when using the DATA step MERGE, something that the SQL procedure doesn't require. In other words, using the DATA step MERGE requires four steps:

1. Change the variables name BookNum in the Orders dataset to "ISBN"
2. Sort the Orders dataset by the new ISBN variable
3. Sort the Books dataset by ISBN

4. Merge

```
Proc Datasets library = work;
modify Orders;
rename BookNum = ISBN;
run;
quit;

Proc Sort data = Orders; by ISBN; run;

Proc Sort data = Books; by ISBN; run;

Data NewBooks;
Merge Books (in=books) Orders (in=orders);
by ISBN;
if books=1 and orders=1;
run;
```

In Proc SQL, all this work can be done in one easy step:

```
proc sql;
Create Table NewBooks as
select  B.Title,
        B.Author,
        B.ISBN,
        O.Order_Date
from Books as B join Orders as O
on B.ISBN=O.BookNum;
quit;
```

The SQL procedure doesn't care how the incoming data tables are sorted, nor does it care what the variable names are for the common variable. This means less lines of code are needed, which makes life much easier for the happy programmer.

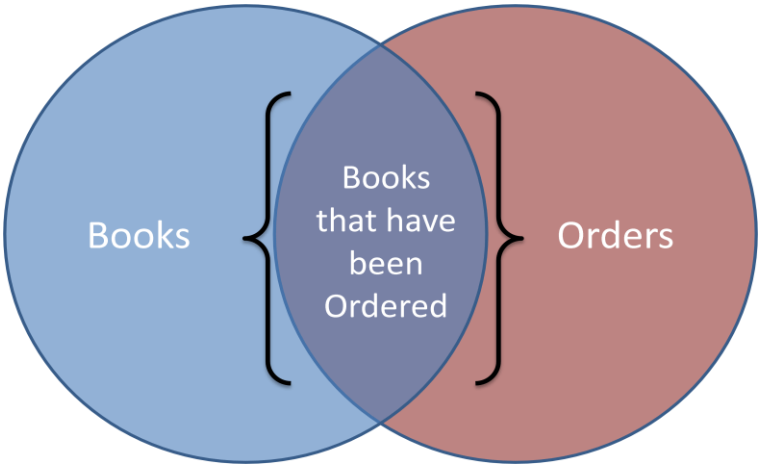
It should be noted that in some instances Proc SQL may require more CPU processing time than a DATA step MERGE, depending on the type of join you are using and the type of data that you have. In addition Proc SQL may return different results than the DATA step MERGE when performing a many-to-many join. While these topics are beyond the scope of this paper, there are other resources available that address these issues if they are of concern to you.[7]

Inner joins

So far, all of the examples above have used **inner joins**. Inner joins are table joins where the resulting output only includes rows for which there was data in each of the source tables. Joins can be most easily visualized as a Venn diagram. Inner joins will only return records that appear in both tables:

[7] See ***Dear Ms. SAS Answers: A guide to efficient Proc SQL coding***, by Jane Stroupe and Linda Jolley

Figure 1. Inner joins



Another way to conceptualize an inner join is to see what the syntax looks like when using a DATA step MERGE:

```
Data NewBooks; Merge Books (in=books) Orders (in=orders);
by ISBN;
if books=1 and orders=1;
run;
```

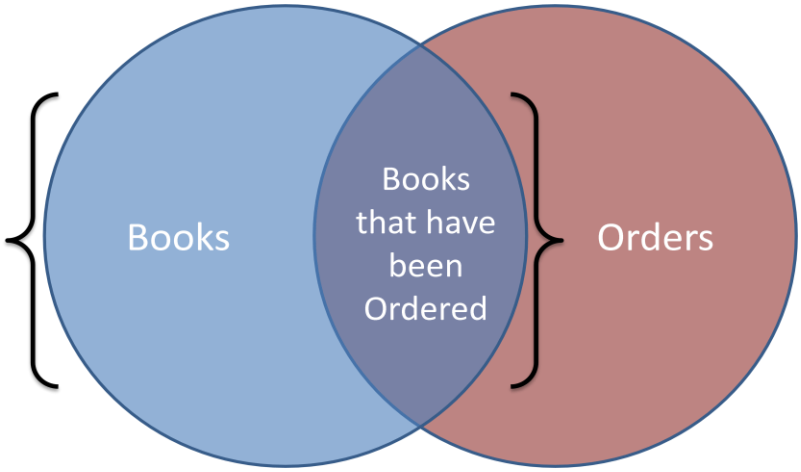
When using an inner join, if there are any books that do not appears on the Orders table, or any orders that do not appear on the Books table, they will be excluded from the resulting output. What you are left with, essentially, are books that have been ordered. If you look at Example 4, you can see that some of the records on the Books table, such as "The Little SAS Book" and "SAS Survival Handbook," do not appear in the resulting output, because they are missing a matching row in the Orders table.

Outer joins: Left joins

Inner joins are only one type of join available in SQL. There are also a variety of outer joins available, specifically left joins, right joins, and full joins.

Let's say we wanted to see which books had been ordered, but you also still wanted to know about books that hadn't been ordered. How do you keep information from the Books table when there are no corresponding orders? You use a **left join**.

Figure 2. Left joins



Another way to conceptualize a left join is to see what the syntax looks like when using a DATA step MERGE:

```
Data NewBooks; Merge Books (in=books) Orders (in=orders);
by ISBN;
if books=1;
run;
```

The word "left" in "left join" refers to the table on the left side of the join condition. In our example, the Books table is to the left of the join keyword. When using a left join, if there are any orders from the Orders table that do not appear on the Books table, they will be excluded from the resulting output. However, any books from the Books table that do not appears on the Orders table will be retained. What you are left with are books from the Books table, and, if applicable, order information about those books from the Orders table. If a book doesn't have an Order Date on the Orders table, that field is left blank.

```
proc sql;
Create Table NewBooks as
select  B.Title,
        B.Author,
        B.ISBN,
        O.Order_Date
from Books as B left join Orders as O
on B.ISBN=O.ISBN;
quit;
```

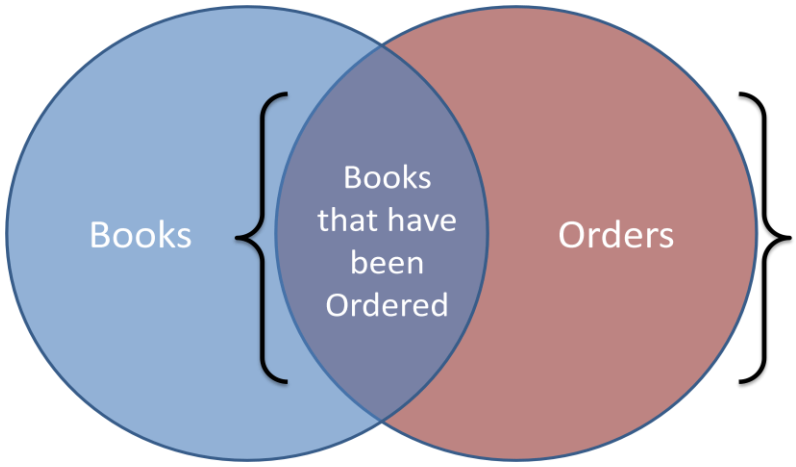
Example 5. Left joins

Title	Author	ISBN	Order_date
The Little SAS Book	Delwiche	1590473337	
SAS Survival Handbook	Wiseman	0060578793	
SAS for Dummies	McDaniel	0471788325	03/25/2009
Learning SAS by Example	Cody	1599941651	01/23/2009
Output Delivery System	Haworth	158025859X	07/07/2007
SAS Functions by Example	Cody	1590473787	05/09/2008
Annotate: Simply the Basics	Carpenter	1580255787	
SAS Programming Shortcuts	Aster	1891957112	12/24/2008
Survival Analysis Using SAS	Allison	155544279X	11/05/2007
Longitudinal Data and SAS	Cody	1580259243	08/30/2008
SAS Macro Programming	Burlew	1590478827	

Outer joins: Right joins

By contrast, we could just as easily perform a **right join**. A right join is the same as a left join, except it will take into consideration the table on the right side of the join condition.

Figure 3. Right joins



Another way to conceptualize a left join is to see what the syntax looks like when using a DATA step MERGE:

```
Data NewBooks; Merge Books (in=books) Orders (in=orders);
by ISBN;
if orders=1;
run;
```

Just like with the left join, the word right in "right join" refers to the table on the right side of the join condition, which is the Orders table. When using a right join, if there are any books from the Books table that do not appear on the Orders table, they will be excluded from the resulting output. However, any orders from the Orders table that do not appears on the Books table will be retained. What you are left with are orders from the Orders table, and, if applicable, title and author information about those orders from the Books table. If an order doesn't have a matching record in the Books table, those fields are left blank.

```
proc sql;
Create Table NewBooks as
select  B.Title,
        B.Author,
        O.ISBN,
        O.Order_Date
from Books as B right join Orders as O
on B.ISBN=O.ISBN;
quit;
```

If you look at the SELECT clause in the SQL syntax above you'll notice that the ISBN field is now being populated from the Orders table rather than the Books table. If we had used the Books table to populate our common key, the ISBN would be missing for those Orders that did not appear on the Books table. Instead we chose to select ISBN from the table that we knew would be fully populated for the records we were interested in.

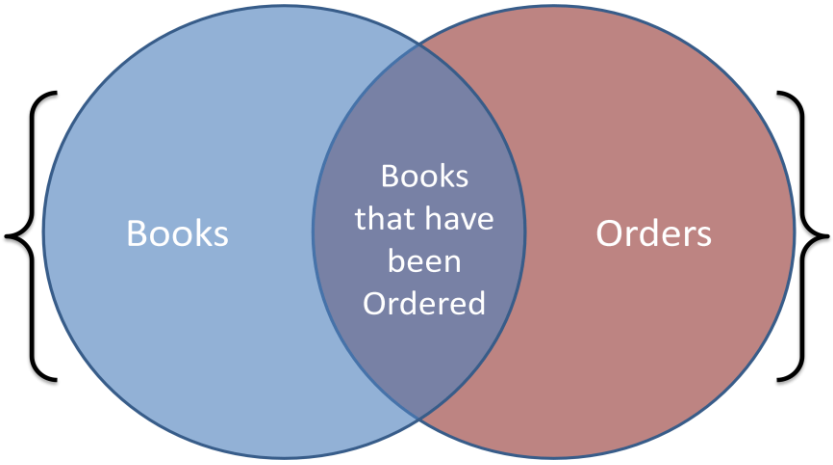
Example 6. Right joins

Title	Author	ISBN	Order_date
SAS for Dummies	McDaniel	0471788325	03/25/2009
Learning SAS by Example	Cody	1599941651	01/23/2009
Output Delivery System	Haworth	158025859X	07/07/2007
SAS Functions by Example	Cody	1590473787	05/09/2008
SAS Programming Shortcuts	Aster	1891957112	12/24/2008
Survival Analysis Using SAS	Allison	155544279X	11/05/2007
Longitudinal Data and SAS	Cody	1580259243	08/30/2008
		1563893428	05/29/2009
		0393327345	08/03/2008

Outer joins: Full joins

The last type of outer join is the **full join**. A full join will select all records from both tables, regardless of whether there is a matching record in the other table or not.

Figure 4. Full joins



When using a full join, no records are excluded from the resulting output. In addition to books that have been ordered, any orders from the Orders table that do not appear on the Books table will be retained, and any books from the Books table that do not appear on the Orders table will be retained.

In some cases our common variable, ISBN, will be only available in the Books table, and in other cases ISBN will only be available in the Orders table. To ensure that ISBN is populated in every record in the output, we can use the coalesce function[8] in our SELECT clause. The coalesce function will select the first non-missing argument. For example:

```
coalescec(B.ISBN, O.ISBN) as ISBN
```

This tells SAS to pick the first non-missing ISBN variable that is found. If the ISBN is found in the Books table (B.ISBN), then it will select that variable to populate ISBN in the resulting output. However, if the ISBN is missing in the Books table it will go to the Orders table (O.ISBN) to populate ISBN in the resulting output.

```
proc sql;
Create Table NewBooks as
select  B.Title,
        B.Author,
        coalescec(B.ISBN, O.ISBN) as ISBN,
        O.Order_Date
from Books as B right join Orders as O
on B.ISBN=O.ISBN;
quit;
```

[8] COALESCE is used for numeric arguments. COALESCEC is used for character arguments. Since ISBN is a character variable, we are using the latter function.

Since we're using the coalesce function, we're essentially created a new variable. As such, we needed to use the AS keyword to rename the new variable (as shown above), otherwise SAS would have given it some default and completely meaningless name like "TEMA001".

As you can see from the example below, the resulting output includes all records from the Books and Orders table. When a match is found in both tables, it joins the records based on ISBN. When a match is not found, the variables from the non-matching table are left blank. And, thanks to our use of the coalesce function, the ISBN is fully populated, using the ISBN from the Orders table only when the ISBN from the Books table is missing.

Example 7. Full joins

Title	Author	ISBN	Order_date
The Little SAS Book	Delwiche	1590473337	
SAS Survival Handbook	Wiseman	0060578793	
SAS for Dummies	McDaniel	0471788325	03/25/2009
Learning SAS by Example	Cody	1599941651	01/23/2009
Output Delivery System	Haworth	158025859X	07/07/2007
SAS Functions by Example	Cody	1590473787	05/09/2008
Annotate: Simply the Basics	Carpenter	1580255787	
SAS Programming Shortcuts	Aster	1891957112	12/24/2008
Survival Analysis Using SAS	Allison	155544279X	11/05/2007
Longitudinal Data and SAS	Cody	1580259243	08/30/2008
SAS Macro Programming	Burlew	1590478827	
		1563893428	05/29/2009
		0393327345	08/03/2008

Final thoughts

Hopefully you've found these tips useful. But this is only the beginning. For additional resources, check out SAS Help and Documentation and sasCommunity.org (www.sascommunity.org/wiki).

Your comments and questions are always valued and encouraged. If you have additional questions, feel free to contact the author at:

Tasha Chapman
Oregon Department of Consumer and Business Services, Salem, Oregon
Tasha.L.Chapman@state.or.us

Many thanks to all the people who first taught me how to use SQL and helped me along the way.