

## Paper TU06

# A Visual Introduction to PROC SQL Joins

Kirk Paul Lafler, Software Intelligence Corporation  
Charles Edwin Shipp, Shipp Consulting

### Abstract

Real systems rarely store all their data in one large table. To do so would require maintaining several duplicate copies of the same values and could threaten the integrity of the data. Instead, IT departments everywhere almost always divide their data among several different tables. Because of this, a method is needed to simultaneously access two or more tables to help answer the interesting questions about our data. This paper visually illustrates how a join process works and then shows how a PROC SQL query is constructed so some or all of the specified tables contents can be brought together.

### Introduction

Joining two or more tables of data is a powerful feature found in the relational model and the SQL procedure. Information in a database system is rarely stored in a single table because it would result in the duplication of data values. A duplicated data value is not only inefficient, but also makes for more complex queries and updates. As a result, data is split between two or more tables.

The SQL procedure is a simple and flexible tool for joining tables of data together. This paper presents the importance of joins, how joins are performed without a WHERE clause, with a WHERE clause, using table aliases, and with three tables of data. Certainly many of these techniques can be accomplished using other methods, but the simplicity and flexibility found in the SQL procedure makes it especially interesting, if not indispensable, as a tool for the information practitioner.

### Why join Anyway?

As relational database systems continue to grow in popularity, the need to access normalized data that has been stored in separate tables becomes increasingly important. By relating matching values in key columns in one table with key columns in two or more tables, information can be retrieved as if the data were stored in one huge file. Consequently, the process of joining data from two or more tables can provide new and exciting insights between data relationships.

### SQL Joins

A join of two or more tables provides a means of gathering and manipulating data in a single SELECT statement. A "JOIN" statement does not exist in the SQL language. The way two or more tables are joined is to specify the tables names in a WHERE clause of a SELECT statement. A comma separates each table specified in an inner join.

Joins are specified on a minimum of two tables at a time, where a column from each table is used for the purpose of connecting the two tables. Connecting columns should have "*like*" values and the same datatype attributes since the join's success is dependent on these values.

### Example Tables

A relational database is simply a collection of tables. Each table contains one or more columns and one or more rows of data. The examples presented in this paper apply an example database consisting of three tables: CUSTOMERS, MOVIES, and ACTORS. Each table appears below.

CUSTOMERS			
<u>CUST_NO</u>	<u>NAME</u>	<u>CITY</u>	<u>STATE</u>
11321	John Smith	Miami	FL
44555	Alice Jones	Baltimore	MD
21713	Ryan Adams	Atlanta	GA

MOVIES			
<u>CUST_NO</u>	<u>MOVIE_NO</u>	<u>RATING</u>	<u>CATEGORY</u>
44555	1011	PG-13	Adventure
21713	3090	G	Comedy
44555	2198	G	Comedy
37753	4456	PG	Suspense

ACTORS	
<u>MOVIE_NO</u>	<u>LEAD_ACTOR</u>
1011	Mel Gibson
2198	Clint Eastwood
3090	Sylvester Stallone

### DATA Step Merge versus a Join

The difference between a DATA Step merge and a Join are subtle, but differences do exist. The merge process overlays the duplicate by-column, where the join process does not overlay the duplicate matching column.

#### Merge Features

1. Data must be sorted by the by-value.
2. More steps are often needed than with PROC SQL.
3. Requires common variable names.
4. Results are not automatically printed.

#### Join Features

1. Duplicate matching column is not automatically overlaid as it is with merge.
2. Results are automatically printed unless NOPRINT option is specified.

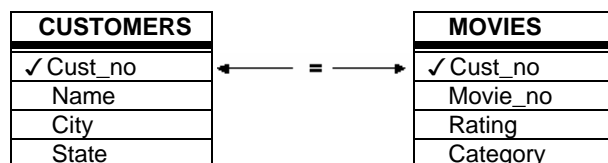
### What Happens During a Join?

Joins are processed in two distinct phases. The first phase determines whether a FROM clause references two or more tables. When it does, a *virtual* table, known as the Cartesian product, is created resulting in each row in the first table being combined with each row in the second table, and so forth. The Cartesian product (internal virtual table) can be extremely large since it represents all possible combinations of rows and columns in the joined tables. As a result, the Cartesian product can be, and often is, extremely large.

The second phase of every join processes, if present, is specified in the WHERE clause. When a WHERE clause is specified, the number of rows in the Cartesian product is reduced to satisfy this expression. This data subsetting process generally results in a more manageable end product containing meaningful data.

### Joining Two Tables with a Where Clause

Joining two tables together is a relatively easy process in SQL. To illustrate how a join works, a two-table join is linked in the following diagram.



The following SQL code references a join on two tables with CUST\_NO specified as the connecting column.

```
PROC SQL;
  SELECT *
    FROM CUSTOMERS, MOVIES
     WHERE CUSTOMERS.CUST_NO =
           MOVIES.CUST_NO;
QUIT;
```

In this example, tables CUSTOMERS and MOVIES are used. Each table has a common column, CUST\_NO which is used to connect rows together from each when the value of CUST\_NO is equal, as specified in the WHERE clause. A WHERE clause restricts what rows of data will be included in the resulting join.

### Creating a Cartesian Product

When a WHERE clause is omitted, all possible combinations of rows from each table is produced. This form of join is known as the **Cartesian Product**. Say for example you join two tables with the first table consisting of 10 rows and the second table with 5 rows. The result of these two tables would consist of 50 rows. Very rarely is there a need to perform a join operation in SQL where a WHERE clause is not specified. The primary importance of being aware of this form of join is to illustrate a base for all joins. Visually, the two tables would be combined without a corresponding WHERE clause as illustrated in the following diagram. Consequently, no connection between common columns exists.

CUSTOMERS	MOVIES
Cust_no	Cust_no
Name	Movie_no
City	Rating
State	Category

To inspect the results of a Cartesian Product, you could submit the same code as before but without the WHERE clause.

```
PROC SQL;
  SELECT *
    FROM CUSTOMERS, MOVIES;
QUIT;
```

### Table Aliases

Table aliases provide a "short-cut" way to reference one or more tables within a join operation. One or more aliases are specified so columns can be selected with a minimal number of keystrokes. To illustrate how table aliases in a join works, a two-table join is linked in the following diagram.

MOVIES M	ACTORS A
Cust_no	Movie_no
✓ Movie_no	✓ Movie_no
Rating	Lead_actor
Category	

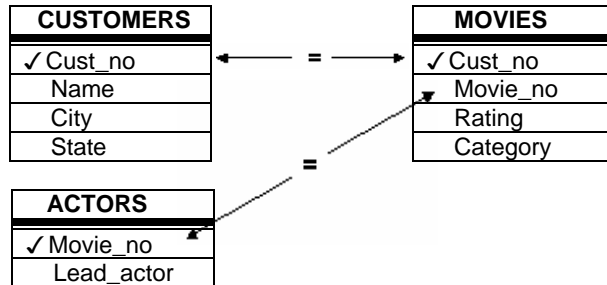
The following SQL code illustrates a join on two tables with MOVIE\_NO specified as the connecting column. The table aliases are specified in the SELECT statement as qualified names, the FROM clause, and the WHERE clause.

```
PROC SQL;
  SELECT M.MOVIE_NO,
         M.RATING,
         A.LEADING_ACTOR
    FROM MOVIES M, ACTORS A
     WHERE M.MOVIE_NO =
           A.MOVIE_NO;
QUIT;
```

## Joining Three Tables

In an earlier example, you saw where customer information was combined with the movies they rented. You may also want to display the leading actor of each movie along with the other information. To do this, you will need to extract information from three different tables: CUSTOMERS, MOVIES, and ACTORS.

A join with three tables follows the same rules as in a two-table join. Each table will need to be listed in the FROM clause with appropriate restrictions specified in the WHERE clause. To illustrate how a three table join works, the following diagram should be visualized.



The following SQL code references a join on three tables with CUST\_NO specified as the connecting column for the CUSTOMERS and MOVIES tables, and MOVIE\_NO as the connecting column for the MOVIES and ACTORS tables.

```
PROC SQL;  
  SELECT C.CUST_NO,  
         M.MOVIE_NO,  
         M.RATING,  
         M.CATEGORY,  
         A.LEADING_ACTOR  
  FROM CUSTOMERS C,  
       MOVIES M,  
       ACTORS A  
  WHERE C.CUST_NO = M.CUST_NO AND  
        M.MOVIE_NO = A.MOVIE_NO;  
QUIT;
```

## Conclusion

The SQL procedure provides a powerful way to join two or more tables of data. It's easy to learn and use. More importantly, since the SQL procedure follows ANSI (American National Standards Institute) guidelines, your knowledge is portable to other platforms and vendor implementations. The simplicity and flexibility of performing joins with the SQL procedure makes it an especially interesting, if not indispensable, tool for the information practitioner.

## References

- Laffer, Kirk, PROC SQL for Beginners; Software Intelligence Corporation, Spring Valley, CA, USA; 1992-2003.
- Laffer, Kirk, Intermediate PROC SQL; Software Intelligence Corporation, Spring Valley, CA, USA; 1998-2003.
- Laffer, Kirk, Advanced PROC SQL; Software Intelligence Corporation, Spring Valley, CA, USA; 2001-2003.
- Laffer, Kirk, PROC SQL Programming Tips; Software Intelligence Corporation, Spring Valley, CA, USA; 2002.
- SAS® Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition; SAS Institute, Cary, NC, USA; 1990.
- SAS® SQL Procedure User's Guide, Version 8; SAS Institute Inc., Cary, NC, USA; 2000.

## Acknowledgments

The authors would like to thank Kent Letourneau and Alfredo Rojas (Tutorials Co-Chairs) for accepting our paper, as well as the PharmaSUG Leadership for their support of a great Conference.

## **Trademark Citations**

SAS, SAS Alliance Partner, and SAS Certified Professional are registered trademarks of SAS Institute Inc. in the USA and other countries. The ® symbol indicates USA registration.

## **About the Authors**

Kirk Paul Lafler, a SAS Certified Professional® and SAS Alliance Partner® (1996 – 2002) with more than 25 years of SAS software experience, provides SAS consulting services and hands-on training around the world. Kirk has written four books including Power SAS by Apress and more than one hundred articles for professional journals and SAS User Group proceedings. His popular SAS Tips column appears regularly in the BASAS, HASUG, SANDS, and SESUG Newsletters.

Charles Edwin Shipp is a programmer and consultant with 25 years experience working with the SAS System. He has authored numerous articles and co-authored the popular Books by Users (BBU) book, Quick Results with SAS/GRAPH Software. He has also been involved with web design, content creation, and development. Currently, Charlie is working in JMP instruction/web.

Comments and suggestions can be sent to:

Kirk Paul Lafler  
Software Intelligence Corporation  
P.O. Box 1390  
Spring Valley, California 91979-1390  
E-mail: KirkLafler@cs.com  
Voice: 619.660.2400