

Exploring the World of PROC SQL[®] Joins

Kirk Paul Lafler, Software Intelligence Corporation

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 split 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 presentation illustrates a variety of join processes including Cartesian Product joins, equijoins, many table joins, and outer joins.

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

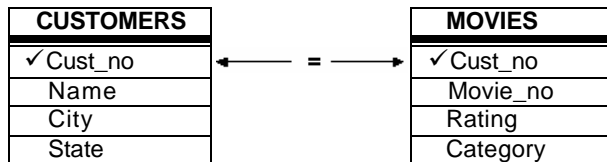
<u>CUST_NO</u>	<u>MOVIE_NO</u>	<u>RATING</u>	<u>CATEGORY</u>
44555	1011	PG-13	Adventure
21713	3090	G	Comedy
44555	2198	G	Comedy
37753	4456	PG	Suspense

ACTORS

MOVIE_NO	LEAD_ACTOR
1011	Mel Gibson
2198	Clint Eastwood
3090	Sylvester Stallone

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.

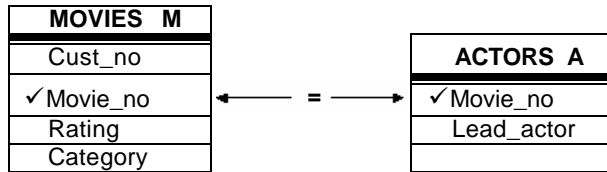


To inspect the results of a Cartesian Product, you could submit the same code as before but without the WHERE clause.

```
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.



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.

```

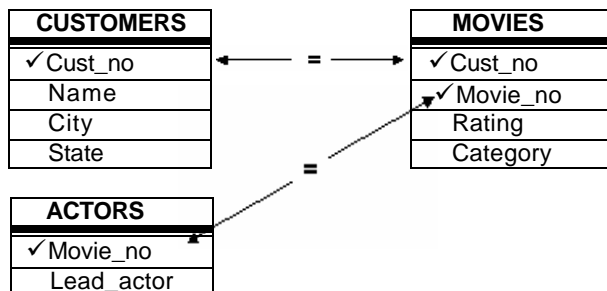
PROC SQL;
  SELECT M.MOVIE_NO,
         M.RATING,
         A.LEADING_ACTOR
  FROM MOVIES M, ACTORS A
  WHERE M.MOVIE_NO =
        A.MOVIE_NO;
QUIT;

```

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;

```

Introduction to Outer Joins

Generally a join is a process of relating rows in one table with rows in another. But occasionally, you may want to include rows from one or both tables that have no related rows. This concept is referred to as row preservation and is a significant feature offered by the outer join construct.

There are operational and syntax differences between inner (natural) and outer joins. First, the maximum number of tables that can be specified in an outer join is two (the maximum number of tables that can be specified in an inner join is 32). Like an inner join, an outer join relates rows in both tables. But this is where the similarities end because the result table also includes rows with no related rows from one or both of the tables. This special handling of “matched” and “unmatched” rows of data is what differentiates an outer join from an inner join.

An outer join can accomplish a variety of tasks that would require a great deal of effort using other methods. This is not to say that a process similar to an outer join can not be programmed – it would probably just require more work. Let’s take a look at a few tasks that are possible with outer joins:

- List all customer accounts with rentals during the month, including customer accounts with no purchase activity.
- Compute the number of rentals placed by each customer, including customers who have not rented.
- Identify movie renters who rented a movie last month, and those who did not.

Another obvious difference between an outer and inner join is the way the syntax is constructed. Outer joins use keywords such as LEFT JOIN, RIGHT JOIN, and FULL JOIN, and has the WHERE clause replaced with an ON clause. These distinctions help identify outer joins from inner joins.

Finally, specifying a left or right outer join is a matter of choice. Simply put, the only difference between a left and right join is the order of the tables they use to relate rows of data. As such, you can use the two types of outer joins interchangeably and is one based on convenience.

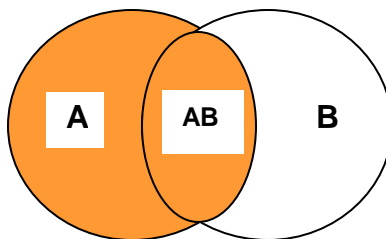
Exploring Outer Joins

Outer joins process data relationships from two tables differently than inner joins. In this section a different type of join, known as an outer join, will be illustrated. The following code example illustrates a left outer join to identify and match movie numbers from the MOVIES and ACTORS tables. The resulting output would contain all rows for which the SQL expression, referenced in the ON clause, matches both tables and retaining all rows from the left table (MOVIES) that did not match any row in the right (ACTORS) table. Essentially the rows from the left table are preserved and captured exactly as they are stored in the table itself, regardless if a match exists.

SQL Code

```
PROC SQL;  
  SELECT movies.movie_no, leading_actor, rating  
  FROM MOVIES  
  LEFT JOIN  
  ACTORS  
  ON movies.movie_no = actors.movie_no;  
QUIT;
```

The result of a Left Outer join is illustrated by the shaded area (A and AB) in the following diagram.

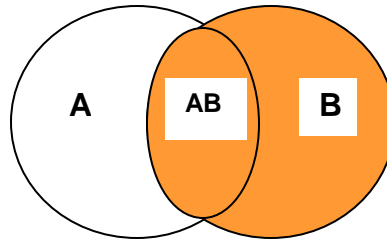


The next example illustrates the result of using a right outer join to identify and match movie titles from the MOVIES and ACTORS tables. The resulting output would contain all rows for which the SQL expression, referenced in the ON clause, matches in both tables (is true) and all rows from the right table (ACTORS) that did not match any row in the left (MOVIES) table.

SQL Code

```
PROC SQL;  
  SELECT movies.movie_no, actor_leading, rating  
  FROM MOVIES  
  RIGHT JOIN  
  ACTORS  
  ON movies.movie_no = actors.movie_no;  
QUIT;
```

The result of a Right Outer join is illustrated by the shaded area (AB and B) in the following diagram.



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 the 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

- Lafler, Kirk Paul (2007), *"Undocumented and Hard-to-find PROC SQL Features,"* Proceedings of the PharmaSUG 2007 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.
- Lafler, Kirk Paul and Ben Cochran (2007), *"A Hands-on Tour Inside the World of PROC SQL Features,"* Proceedings of the SAS Global Forum (SGF) 2007 Conference, Software Intelligence Corporation, Spring Valley, CA, and The Bedford Group, USA.
- Lafler, Kirk Paul (2006), *"A Hands-on Tour Inside the World of PROC SQL,"* Proceedings of the 31st Annual SAS Users Group International Conference, Software Intelligence Corporation, Spring Valley, CA, USA.
- Lafler, Kirk Paul (2005), *"Manipulating Data with PROC SQL,"* Proceedings of the 30th Annual SAS Users Group International Conference, Software Intelligence Corporation, Spring Valley, CA, USA.
- Lafler, Kirk Paul (2004). *PROC SQL: Beyond the Basics Using SAS*, SAS Institute Inc., Cary, NC, USA.
- Lafler, Kirk Paul (2003), *"Undocumented and Hard-to-find PROC SQL Features," Proceedings of the Eleventh Annual Western Users of SAS Software Conference.*
- Lafler, Kirk Paul (1992-2006). *PROC SQL for Beginners*; Software Intelligence Corporation, Spring Valley, CA, USA.
- Lafler, Kirk Paul (1998-2006). *Intermediate PROC SQL*; Software Intelligence Corporation, Spring Valley, CA, USA.
- SAS[®] *Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition (1990)*. SAS Institute, Cary, NC.
- SAS[®] *SQL Procedure User's Guide, Version 8 (2000)*. SAS Institute Inc., Cary, NC, USA.

Trademark Citations

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. and other countries. ® indicates USA registration.

Acknowledgments

I would like to thank Dr. George Fernandez, WUSS 2007 Tutorials Section Chair; Dr. Besa Smith, WUSS 2007 Academic Program Chair; and MaryAnne Hope, WUSS 2007 Operations Chair for accepting my abstract and paper, as well as the WUSS Leadership for their support of a great Conference.

About the Author

Kirk Paul Lafler is consultant and founder of Software Intelligence Corporation and has been programming in SAS since 1979. As a SAS Certified Professional and SAS Institute Alliance Member (1996 – 2002), Kirk provides IT consulting services and training to SAS users around the world. As the author of four books including *PROC SQL: Beyond the Basics Using SAS* (SAS Institute. 2004), he has written more than two hundred peer-reviewed papers and articles that have appeared in professional journals and SAS User Group proceedings. Kirk has also been an Invited speaker at more than two hundred SAS International, regional, local, and special-interest user group conferences and meetings throughout North America. His popular SAS Tips column, "Kirk's Korner of Quick and Simple Tips", appears regularly in several SAS User Group newsletters and Web sites, and his fun-filled SASword Puzzles is featured in SAScommunity.org.

Comments and suggestions can be sent to:

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

