Paper 062-2009

A Propaedeutics for PROC SQL Joins

Lisa Mendez , SRA International Inc. , San Antonio , Tx.
Toby Dunn , AMEDDC&S (CASS) , San Antonio , Tx.

ABSTRACT

Would you like to be able to join information from separate tables quickly and easily? Have you ever had two or more tables that you want to join but just don't know where to begin? If the answer is "yes" then "Join" us for a crash course over Proc SQL joins. This paper will help get you started by introducing you to the four basic types of SQL joins: inner, outer, natural, and Cartesian. Through the use of carefully crafted examples and detailed explanations we will take you on a guided tour of the SQL joins in SAS®.

INTRODUCTION

The power of SQL joins can be seen when a user must join datasets from a relational database, when joining large datasets, or when two or more datasets must be joined with some complex set of logic. One major advantage of using Proc SQL comes from the fact that datasets do not need to be explicitly presorted as Proc SQL will implicitly do the sorting for you.

The world has become a huge data hog, collecting, organizing, and storing huge amounts of data. Businesses aren't collecting all of this data just for the fun of it; they are at an ever increasing rate turning to this data to drive their business decisions. One of the side effects of collecting all this data is the need to some how efficiently store and organize it. Computer science came up with relational data bases. One of the central tenants of data base design is normalizing the tables of a data base. This normalization insures efficient data storage and accuracy by minimizing the storage of redundant data and ensuring data integrity. Explaining normalization is a subject worthy of its own paper or book, suffice for our purposes is to know only that normalization improves overall efficiency and creates a bunch of tables.

When normalization is achieved redundant data is minimized, storage is maximized and the need to access this normalized data that has been stored in separate tables becomes important. By relating key variables between tables, information can then be retrieved as if the data were stored in one large table. Relating these tables together is where Proc SQL joins comes into play.

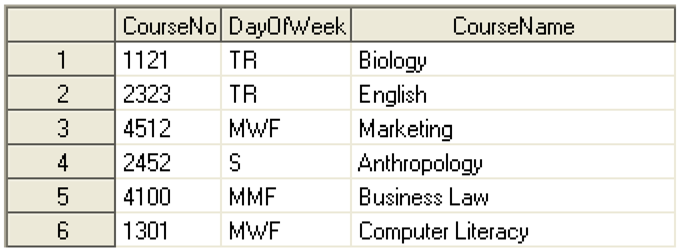
A join in its essence will return, delete, or update data from more than one data source as one set of data. Whether the join is simple, extremely complex or even mimics a data step merge, all of them rely on a few basic types of joins. This paper introduces you to the four basic types of SQL joins through examples and detailed explanations.

CARTESIAN OR CROSS JOIN

The Cartesian or Cross join is the simplest of all the possible joins to write and understand. It is simply a combination of two tables with out a qualifying Where clause. The result of a Cartesian join is every row of the first table joined to every row of the second table. Mathematically one can calculate the number of rows in the resulting table by multiplying the number of rows in the first table by the number of rows in the second table. Let's say you have two tables with 10 rows each, the final table produced would be (10 * 10 = 100) rows.

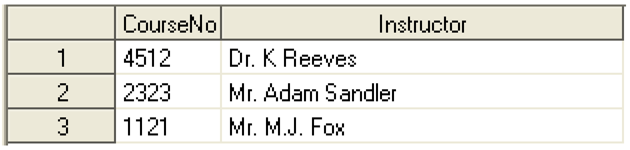
To illustrate a Cartesian product, let us look at two example tables. One is named Course Table. It has six records and lists CourseNo, CourseName, and DayOfWeek. The other is named Instructor Table. It has three records and lists CourseNo and InstructorName. We can see that CourseNo is a common variable or in data base terms a primary key between the two tables.

Course Table



	CourseNo	DayOfWeek	CourseName
1	1121	TR	Biology
2	2323	TR	English
3	4512	MWF	Marketing
4	2452	S	Anthropology
5	4100	MMF	Business Law
6	1301	MWF	Computer Literacy

Instructor Table



	CourseNo	Instructor
1	4512	Dr. K Reeves
2	2323	Mr. Adam Sandler
3	1121	Mr. M.J. Fox

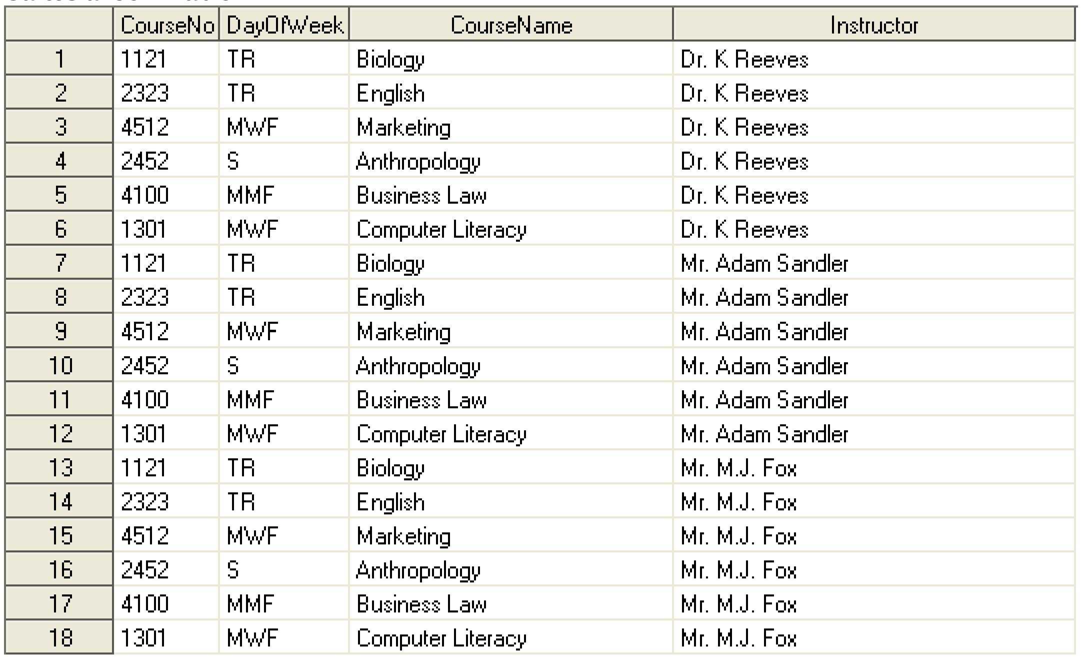
Using the sample code below, we can create a Cartesian product as a new table.

```
Proc SQL;
 Create Table CartesianJoin AS
  Select A.* , B.*
    From CourseTable     As A ,
         InstructorTable As B ;

Quit;
```

The result of the sample code produces a table where each record in the Course Table is joined with each record in the Instructor Table.

CartesianJoin Table



	CourseNo	DayOfWeek	CourseName	Instructor
1	1121	TR	Biology	Dr. K Reeves
2	2323	TR	English	Dr. K Reeves
3	4512	MWF	Marketing	Dr. K Reeves
4	2452	S	Anthropology	Dr. K Reeves
5	4100	MMF	Business Law	Dr. K Reeves
6	1301	MWF	Computer Literacy	Dr. K Reeves
7	1121	TR	Biology	Mr. Adam Sandler
8	2323	TR	English	Mr. Adam Sandler
9	4512	MWF	Marketing	Mr. Adam Sandler
10	2452	S	Anthropology	Mr. Adam Sandler
11	4100	MMF	Business Law	Mr. Adam Sandler
12	1301	MWF	Computer Literacy	Mr. Adam Sandler
13	1121	TR	Biology	Mr. M.J. Fox
14	2323	TR	English	Mr. M.J. Fox
15	4512	MWF	Marketing	Mr. M.J. Fox
16	2452	S	Anthropology	Mr. M.J. Fox
17	4100	MMF	Business Law	Mr. M.J. Fox
18	1301	MWF	Computer Literacy	Mr. M.J. Fox

There are six records in the Course Table and three records in the Instructor Table; therefore, there are 18 records in the new CartesianJoin table (6 X 3 = 18). Cartesian Joins usually generate large unwieldy data set, require a lot of resources to build compared to other types of joins and usually are not intended as a final table. However, the syntax is easy to write and remember which makes them popular, albeit they are usually accompanied by a Where clause. As we will see next in the Inner Joins section, using a Where clause helps manage the data.

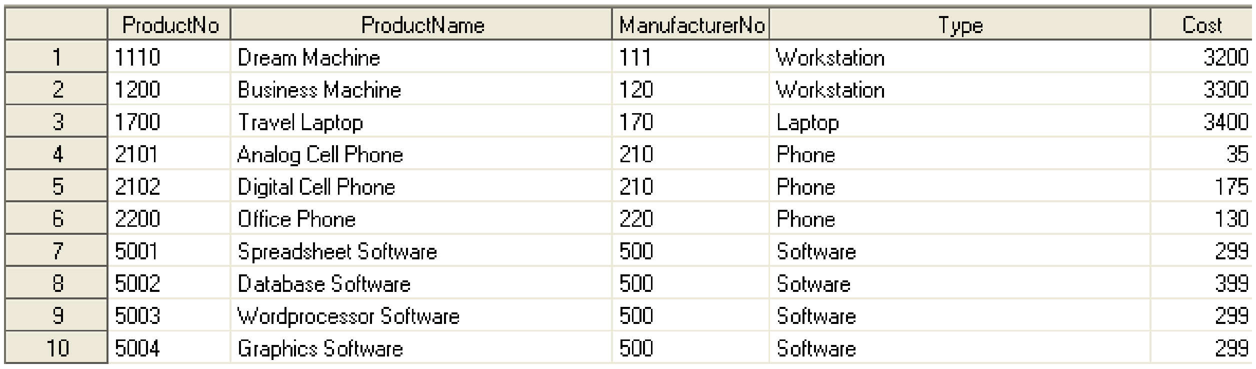
INNER JOINS

In the previous section we looked at Cartesian joins or the Cartesian/Cross product of joining two tables. While a Cartesian product rarely, if ever, gives you the results you want it is a helpful stepping stone to other types of joins. Inner joins are nothing more than a Cartesian product with the addition of a Where clause to specify what data is to be kept in the final data set. We have already mentioned that a Cartesian product is a type of Inner Join however, there are three other types of inner joins: Equijoin, Non-Equijoin, and Reflexive. Inner joins are the most common type of join and unless some other type of join is specified it is the default and as such one does not have to explicitly specify an Inner Join in the From Clause. Inner joins require at least two data sets and can join up too 32 data sets.

EQUIJOIN

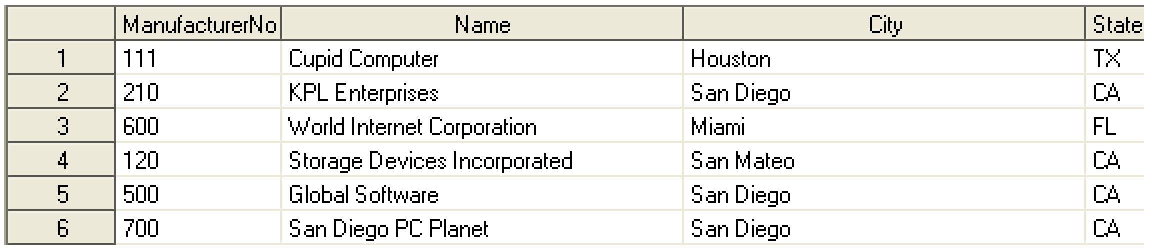
An Equijoin is the most common type of inner join and can be distinguished by the equal sign predicate between two linking variables in the Where clause. For example, let us say you want to find out who manufactures which products within your company. Two example tables we will use are Product Table and Manufacturer Table. The common column in each table is the ManufacturerNo.

Product Table



	ProductNo	ProductName	ManufacturerNo	Type	Cost
1	1110	Dream Machine	111	Workstation	3200
2	1200	Business Machine	120	Workstation	3300
3	1700	Travel Laptop	170	Laptop	3400
4	2101	Analog Cell Phone	210	Phone	35
5	2102	Digital Cell Phone	210	Phone	175
6	2200	Office Phone	220	Phone	130
7	5001	Spreadsheet Software	500	Software	299
8	5002	Database Software	500	Sotware	399
9	5003	Wordprocessor Software	500	Software	299
10	5004	Graphics Software	500	Software	299

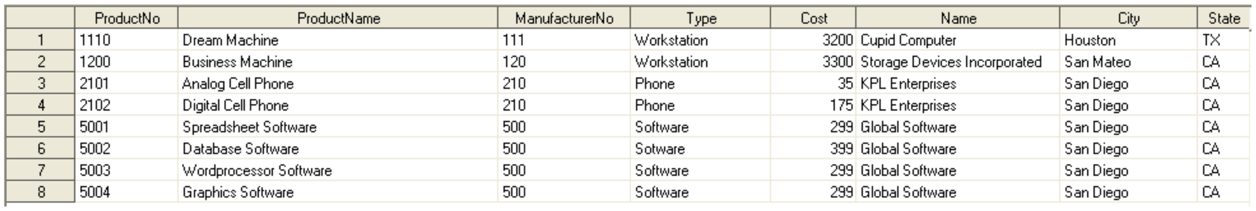
Manufacturer Table



	ManufacturerNo	Name	City	State
1	111	Cupid Computer	Houston	TX
2	210	KPL Enterprises	San Diego	CA
3	600	World Internet Corporation	Miami	FL
4	120	Storage Devices Incorporated	San Mateo	CA
5	500	Global Software	San Diego	CA
6	700	San Diego PC Planet	San Diego	CA

```
Proc SQL ;
Create Table InnerJoin AS
  Select *
    From ProductTable , ManufacturerTable
    Where ProductTable.ManufacturerNo =ManufacturerTable.ManufacturerNo ;
Quit ;
```

InnerJoin Table



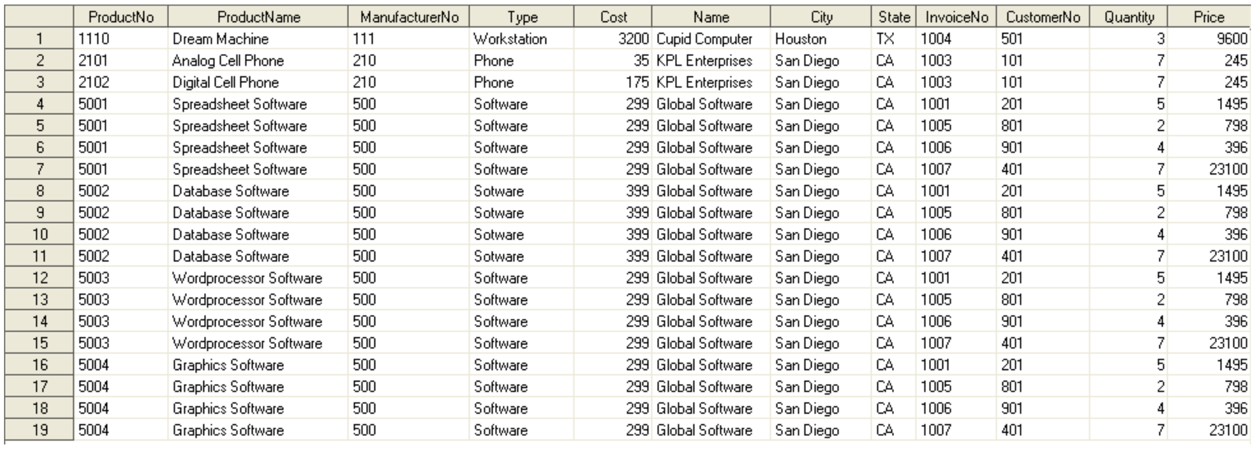
	ProductNo	ProductName	ManufacturerNo	Type	Cost	Name	City	State
1	1110	Dream Machine	111	Workstation	3200	Cupid Computer	Houston	TX
2	1200	Business Machine	120	Workstation	3300	Storage Devices Incorporated	San Mateo	CA
3	2101	Analog Cell Phone	210	Phone	35	KPL Enterprises	San Diego	CA
4	2102	Digital Cell Phone	210	Phone	175	KPL Enterprises	San Diego	CA
5	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA
6	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA
7	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA
8	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA

The resulting InnerJoin Table contains data that has only data where the values of ManufacturerNo match in both tables. An Equi-Join will first build the Cartesian product table as a intermediate table and then discard any rows that do not meet the equality condition(s) set in the Where clause. Note you can use as many equality conditions as you desire in the Where clause statement.

Previously we mention that you can have up to 32 tables specified in an inner join. The difference to the code would be to add the third table to the From clause and specify the variables to match upon in the Where clause. To illustrate how we join three tables with an inner equijoin, let us suppose we want to find out the products, along with their manufacturer information, that have been invoiced.

```
Proc SQL;
Create Table InnerJoinThree AS
 Select *
  From ProductTable, ManufacturerTable, InvoiceTable
  Where ProductTable.ManufacturerNo = ManufacturerTable.ManufacturerNo And
        ManufacturerTable.ManufacturerNo = InvoiceTable.ManufacturerNo ;
Quit ;
```

InnerJoinThree Table



	ProductNo	ProductName	ManufacturerNo	Type	Cost	Name	City	State	InvoiceNo	CustomerNo	Quantity	Price
1	1110	Dream Machine	111	Workstation	3200	Cupid Computer	Houston	TX	1004	501	3	9600
2	2101	Analog Cell Phone	210	Phone	35	KPL Enterprises	San Diego	CA	1003	101	7	245
3	2102	Digital Cell Phone	210	Phone	175	KPL Enterprises	San Diego	CA	1003	101	7	245
4	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA	1001	201	5	1495
5	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA	1005	801	2	798
6	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA	1006	901	4	396
7	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA	1007	401	7	23100
8	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA	1001	201	5	1495
9	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA	1005	801	2	798
10	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA	1006	901	4	396
11	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA	1007	401	7	23100
12	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA	1001	201	5	1495
13	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA	1005	801	2	798
14	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA	1006	901	4	396
15	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA	1007	401	7	23100
16	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA	1001	201	5	1495
17	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA	1005	801	2	798
18	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA	1006	901	4	396
19	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA	1007	401	7	23100

As we can see from the table above we have all the information from the first join and the data from the Invoice Table. It is important to note that if you have multiple rows in one table that can suffice the matching condition(s) the resulting table will have all matching data and the table with one matching row will be duplicated for all matching data from the second table. An example of such a case in this example is ProductNo 5001. While the match between the ProductTable and ManufacturerTable result in one row the InvoiceTable has multiple matching rows and thus the data from the ProductTable and ManufacturerTable is duplicated.

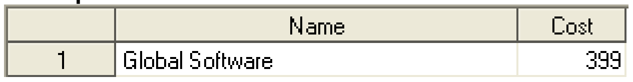
NON-EQUIJOIN

If Equi-Joins have only '=' or EQ as predicates in the Where clause what about the case where at least one of the predicates are not = or EQ? Well, these are called Non-EquiJoins. Non-EquiJoins have to have at least one predicate that would create a condition that isn't an equality. As with the other Inner joins, an intermediate Cartesian product is created and the conditions specified in the Where clause weed out the unwanted observations from the final table.

Let us continue to use the Product and Manufacturer tables. Suppose we want to create a table to display products manufactured by Global Software that cost more than $300.

```
Proc SQL ;
Create Table NonEquiJoin As
 Select B.Name , A.Cost
  From ProductTable          As A ,
           ManufacturerTable As B
  Where A.ManufacturerNo = B.ManufacturerNo  And
        UpCase( B.Name ) = 'GLOBAL SOFTWARE' And
        A.Cost > 300 ;
Quit ;
```

NonEquiJoin Table



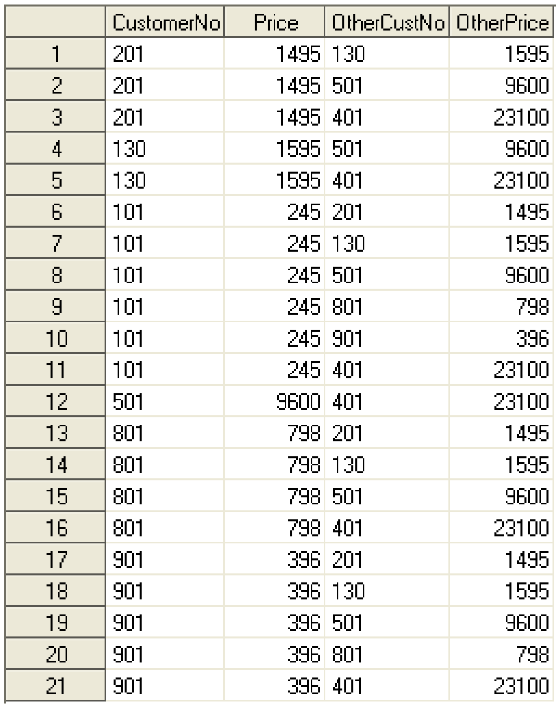
	Name	Cost
1	Global Software	399

REFLEXIVE JOIN

A Reflexive Join, or Self Join as it is more commonly called, is an Inner join which references the same table. Remember back at the begging we said that an Inner join must have at least two data sets but no more than 32. There is nothing that says those two data sets can not be the same one. While this may seem a bit odd at first it is quit handy. SQL simply creates two copies of the table in memory and treats them as two separate data sets. This functionality is very useful, as seen in or example, where we want to see how the price for each customer compares to every other costumer that has a larger dollar amount on their invoices.

```
Proc SQL ;
Create Table ReflexiveJoin As
 Select A.CustomerNo , A.Price ,
        B.CustomerNo As OtherCustNo , B.Price As OtherPrice
  From InvoiceTable As A ,
       InvoiceTable As B
  Where A.Price < B.Price ;
Quit ;
```

ReflexiveJoin Table



	CustomerNo	Price	OtherCustNo	OtherPrice
1	201	1495	130	1595
2	201	1495	501	9600
3	201	1495	401	23100
4	130	1595	501	9600
5	130	1595	401	23100
6	101	245	201	1495
7	101	245	130	1595
8	101	245	501	9600
9	101	245	801	798
10	101	245	901	396
11	101	245	401	23100
12	501	9600	401	23100
13	801	798	201	1495
14	801	798	130	1595
15	801	798	501	9600
16	801	798	401	23100
17	901	396	201	1495
18	901	396	130	1595
19	901	396	501	9600
20	901	396	801	798
21	901	396	401	23100

OUTER JOINS

As we have seen, Inner joins only keep rows that meet the condition(s) specified in the Where clause. While this is very useful, there are many cases where you will want to keep all rows that match and rows from one or both tables that do not meet the matching criteria. In cases such as these you need to switch join types and use what is known as an Outer join. Outer joins are only able to process two tables at a time. Keywords are added to the syntax of the From Clause between the dataset/table names: LEFT JOIN, RIGHT JOIN, FULL JOIN. As you may have guessed by the keywords, there are three main types of outer joins: Left, Right, and Full. Since an Outer join can only join two tables at a time think about the first table as the Left hand side table and the second table as the Right Hand side table, thus giving us the Left Outer Join and the Right Outer Join. Think of it this way, the left table is always the table that precedes the keyword Join and the right table is always the table name that follows the keyword Join. We will discuss these further as we discuss each specific type of Outer Join. Outer joins can only process two datasets/tables at a time regardless of how many Outer joins you specify in one SQL statement. Lastly the matching criteria does not use a Where clause but rather the On Clause.

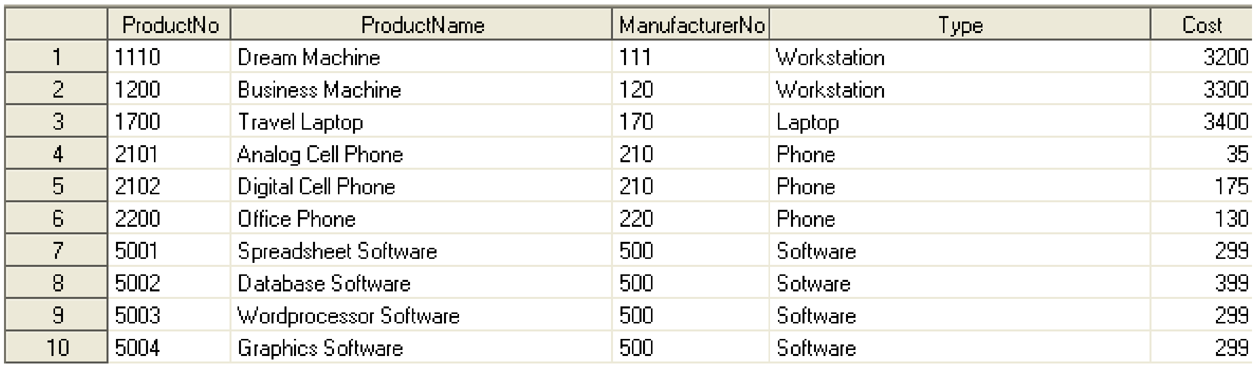
LEFT OUTER JOIN

Since outer joins are designed to keep rows that match the matching conditions as well as those that do not match from one or both tables it stands to reason that one or both datasets needs to be considered a Master data set. In the case of a Left Join the table that precedes the key word “Left Join” in the From clause is considered the Master data set. That means all rows regardless of whether they have a matching row on the Right hand side table will be kept in the final data set. In the case where there are multiple rows in the Right hand side table, the Left hand side table’s data will be duplicated.

Suppose you want to see a list of all products and their corresponding manufacturer information. You will want to also see all those products that do not have a manufacturer. In this example we will use the Product table and the Manufacturer table.

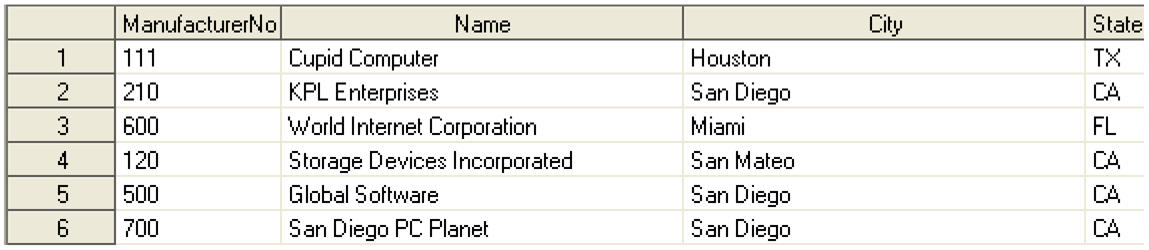
Let us look at two of our example tables again.

Product Table



	ProductNo	ProductName	ManufacturerNo	Type	Cost
1	1110	Dream Machine	111	Workstation	3200
2	1200	Business Machine	120	Workstation	3300
3	1700	Travel Laptop	170	Laptop	3400
4	2101	Analog Cell Phone	210	Phone	35
5	2102	Digital Cell Phone	210	Phone	175
6	2200	Office Phone	220	Phone	130
7	5001	Spreadsheet Software	500	Software	299
8	5002	Database Software	500	Sotware	399
9	5003	Wordprocessor Software	500	Software	299
10	5004	Graphics Software	500	Software	299

Manufacturer Table

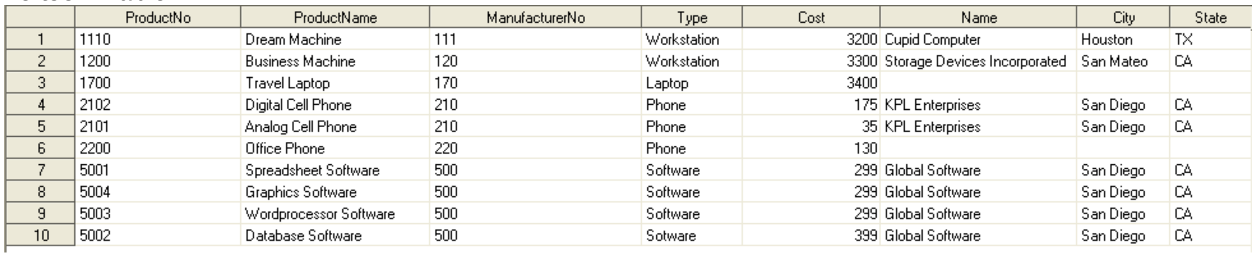


	ManufacturerNo	Name	City	State
1	111	Cupid Computer	Houston	TX
2	210	KPL Enterprises	San Diego	CA
3	600	World Internet Corporation	Miami	FL
4	120	Storage Devices Incorporated	San Mateo	CA
5	500	Global Software	San Diego	CA
6	700	San Diego PC Planet	San Diego	CA

The Product table will act as the Left (master) table and the Manufacturer table will act as the subordinate table. Since we have such a small number of records, we can predict our results. Our new table should not have manufacturer records for product numbers 1700 or 2200.

```
Proc SQL ;
Create Table LEFTJoin As
 Select *
  From ProductTable LEFT JOIN ManufacturerTable
  ON ProductTable.ManufacturerNo = ManufacturerTable.ManufacturerNo ;
Quit;
```

LeftJoin Table



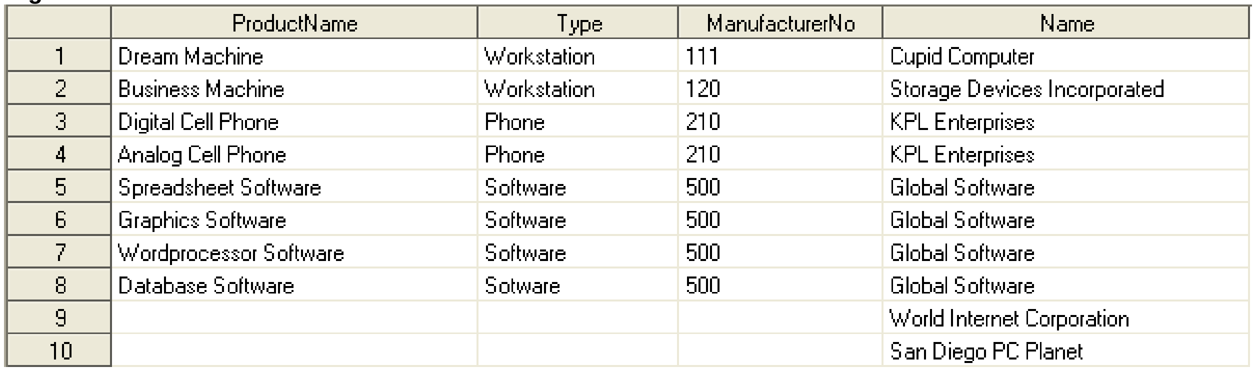
	ProductNo	ProductName	ManufacturerNo	Type	Cost	Name	City	State
1	1110	Dream Machine	111	Workstation	3200	Cupid Computer	Houston	TX
2	1200	Business Machine	120	Workstation	3300	Storage Devices Incorporated	San Mateo	CA
3	1700	Travel Laptop	170	Laptop	3400			
4	2102	Digital Cell Phone	210	Phone	175	KPL Enterprises	San Diego	CA
5	2101	Analog Cell Phone	210	Phone	35	KPL Enterprises	San Diego	CA
6	2200	Office Phone	220	Phone	130			
7	5001	Spreadsheet Software	500	Software	299	Global Software	San Diego	CA
8	5004	Graphics Software	500	Software	299	Global Software	San Diego	CA
9	5003	Wordprocessor Software	500	Software	299	Global Software	San Diego	CA
10	5002	Database Software	500	Sotware	399	Global Software	San Diego	CA

RIGHT OUTER JOIN

Right Outer Joins, as you have probably guessed by now, use the Right hand side table as the master data set and the left hand side as the look up table. Suppose we want to create a table to show all products along with the product type, manufacturer number, and manufacturer name.

```
Proc SQL ;
Create Table RIGHTJoin As
 Select ProductName , Type , ProductTable.ManufacturerNo , Name
  From ProductTable RIGHT JOIN ManufacturerTable
  ON ProductTable.ManufacturerNo = ManufacturerTable.ManufacturerNo ;
Quit ;
```

RightJoin Table



	ProductName	Type	ManufacturerNo	Name
1	Dream Machine	Workstation	111	Cupid Computer
2	Business Machine	Workstation	120	Storage Devices Incorporated
3	Digital Cell Phone	Phone	210	KPL Enterprises
4	Analog Cell Phone	Phone	210	KPL Enterprises
5	Spreadsheet Software	Software	500	Global Software
6	Graphics Software	Software	500	Global Software
7	Wordprocessor Software	Software	500	Global Software
8	Database Software	Sotware	500	Global Software
9				World Internet Corporation
10				San Diego PC Planet

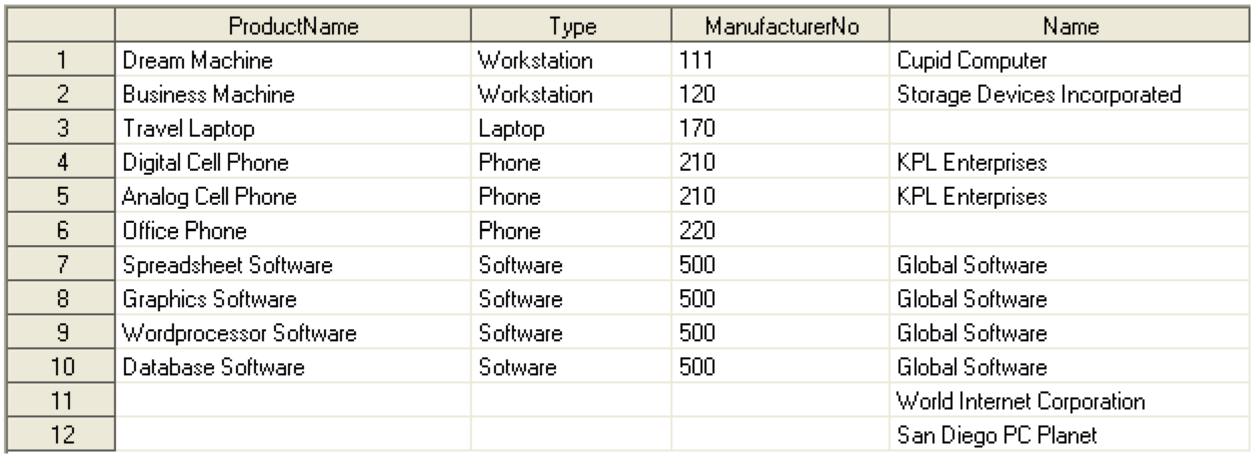
FULL OUTER JOIN

The last Outer Join is the Full Outer Join or more commonly called Full Join. A Full join is the combination of the Left and Right Joins. That is to say it will keep matching and non-matching rows from both tables. While this type of join is less commonly used than Inner joins and the Left and Right Outer Joins there are real world applications where this type of join is very useful.

Using the same Product and Manufacturer tables, we will use the FULL JOIN to create a table that will contain manufacturers with no products and products without any known manufacturers. We already have this information from the LEFT and RIGHT join examples, but this time we want to create a table with only those records. We will use the same sample code as the Right Outer Join, but replace RIGHT keyword with the FULL keyword.

```
Proc SQL ;
Create Table FULLJoin As
  Select ProductName , Type , ProductTable.ManufacturerNo , Name
   From ProductTable FULL JOIN ManufacturerTable
   ON ProductTable.ManufacturerNo = ManufacturerTable.ManufacturerNo ;
Quit ;
```

FullJoin Table



	ProductName	Type	ManufacturerNo	Name
1	Dream Machine	Workstation	111	Cupid Computer
2	Business Machine	Workstation	120	Storage Devices Incorporated
3	Travel Laptop	Laptop	170	
4	Digital Cell Phone	Phone	210	KPL Enterprises
5	Analog Cell Phone	Phone	210	KPL Enterprises
6	Office Phone	Phone	220	
7	Spreadsheet Software	Software	500	Global Software
8	Graphics Software	Software	500	Global Software
9	Wordprocessor Software	Software	500	Global Software
10	Database Software	Sotware	500	Global Software
11				World Internet Corporation
12				San Diego PC Planet

By using the FULL JOIN we preserve all the rows from both the left and right tables even when there are no matching rows.

To help you determine when to use a specific inner or outer join, I have augmented a table from Stuelpner and Bahler's poster paper.

Type of Join	Informational Requirements
Inner Join	Only those data values from all data sets that contain the same data values within the joining variables
Left or Right Outer Join	All data values from a single data set (master) and all data values from the other data set(s) that match the data values of the joining variables within the master.
Full Outer Join	All data values from all data sets

NATURAL JOINS

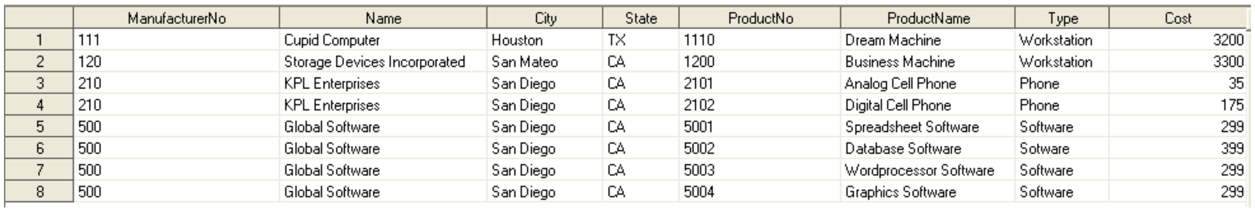
Natural joins are a special type of join that uses all variables that have matching name and types as the primary keys to join the tables. Natural joins are by default a special type of Inner join but can also be specified as an Outer join. If there are no variables that have a matching name and type a Cartesian product is performed. When performing a Natural Left or Right Join do not use the On clause as it will cause the join to fail. Since there is no implementation as of version 9.1.3 to specify which variables are to be considered for the match all variables matching name and type are used; this severely limits the Natural joins usability.

Since the common column names are implied, we can take the code we used for the Inner Join and modify it to illustrate the Natural Join. We modify the code by changing the name of the new table, insert the NATURAL JOIN keywords between the two tables (do not forget to remove the comma), and remove the entire Where clause.

```
Proc SQL ;
Create Table NaturalJoin As
 Select *
  From ProductTable NATURAL JOIN ManufacturerTable ;
Quit ;
```

After running the code, our new table has the same variables and records as the table created using the Inner join code. The only difference is the order of the variables.

NaturalJoin Table



	ManufacturerNo	Name	City	State	ProductNo	ProductName	Type	Cost
1	111	Cupid Computer	Houston	TX	1110	Dream Machine	Workstation	3200
2	120	Storage Devices Incorporated	San Mateo	CA	1200	Business Machine	Workstation	3300
3	210	KPL Enterprises	San Diego	CA	2101	Analog Cell Phone	Phone	35
4	210	KPL Enterprises	San Diego	CA	2102	Digital Cell Phone	Phone	175
5	500	Global Software	San Diego	CA	5001	Spreadsheet Software	Software	299
6	500	Global Software	San Diego	CA	5002	Database Software	Sotware	399
7	500	Global Software	San Diego	CA	5003	Wordprocessor Software	Software	299
8	500	Global Software	San Diego	CA	5004	Graphics Software	Software	299

A few things to think about regarding Natural joins:

- A natural join assumes that you want to base the join on equal values of all pairs of common columns, not just those on which we want to base the join; therefore, you will need to drop any common columns that are not part of the join *before* doing the join (Fleming, 2007).
- You cannot base a join on columns with different names.
- When you specify a natural join on tables that do not have at least one column in common, the result is a Cartesian product. Use a Where clause to limit the output (Fleming, 2007).

For this purpose of this paper a simple example was used. To read more about Natural Joins and its additional advantages, refer to John Fleming's presentation Natural Joins in PROC SQL (2007).

CONCLUSION

Using the Proc SQL procedure provides you with many ways to join two or more tables. It is easy to learn the basics and easy to begin to add flexibility. The information provided in this paper should give you a great starting point to help you write the code you need, and to help you understand when you should use specific types of joins. Remember that each type of join has caveats, and you should always clearly know what you want to do, choose the join which meets your needs, and finally always QC your results.

REFERENCES

Borowiak, Kenneth W. 2006. "Using Data Set Options in PROC SQL", Proceedings of the 31st Annual SAS Users Group International, San Francisco, CA, 131-31.

Fleming, John. 2007. "Natural Joins in PROC SQL", Proceedings of the Edmonton SAS User Group, Edmonton, AB.

Lafler, Kirk Paul. 2004. PROC SQL: Beyond the Basics Using SAS®. Cary, NC: SAS Institute Inc.

Lafler, Kirk Paul and Shipp, Charles Edwin. 2002. "A Visual Introduction to PROC SQL Joins", Proceedings of the 27th Annual SAS Users Group International, Orlando, FL, TU06.

Matthews, JoAnn. 2006. "Proc SQL versus The Data Step", Hands on Workshop of the Northeast SAS User Group, Philadelphia, PA.

Stueopner, Janet and Bahler, Caroline. 2002. "Putting It Together – The Poster A Data Set Joining Primer", Poster of the Southeast SAS User Group, PS04.

ACKNOWLEDGEMENTS

We would like to offer our sincere thanks to Kirk Lafler for the use of some of the datasets used in this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Lisa Mendez
SRA International
1777 NE Loop 410, Ste 510
San Antonio, TX 78217
Phone: 210-832-5213
Fax: 210-824-9578
E-mail: lisa_mendez@sra.com

Toby Dunn
AMEDDC&S
Fort Sam Houston. TX
E-mail: Toby.Dunn@amedd.army.mil