
A New Approach to Outer Joins with More than Two Tables

Howard Levine, DynaMark, A Fair, Isaac Company

Abstract

When using SQL to join tables together when all of the rows from all of the tables do not match up to at least one row in every other table, you must use outer joins. This situation is rather common: an insurance policyholder with no claims records, a potential customer who has not yet placed an order, and an employee who has not yet had an evaluation are all examples of outer join situations.

Currently, when doing outer joins of more than two tables, it is necessary to first join two tables together and then join the result with another table. This process continues until all tables are joined. In comparison to the code for an inner join using the same tables, the code for an outer join is extremely complicated. Also, changing the order of the joining process can affect the rows of an outer join, but will not affect the rows of an inner join.

The algorithm that will be presented avoids the problems of standard outer joins. It allows outer joins to be requested in a simple manner that also makes the order of the tables irrelevant when determining which rows get generated. It allows outer joins to be as consistent as inner joins and almost as simple.

To understand this paper, you will need to understand what inner and outer joins are. This information is available in *SAS Guide to the SQL Procedure*.

Outline

This paper covers the following topics:

Review of Relational Data Base Terminology

Review of Standard SQL Joins

Limitations of Standard SQL Joins

The Need for Flexible Joining of More Than Two Tables

The N-Table Join

Conclusion

Review of Relational Data Base Terminology

Terms

Here is a list of terms that are used when discussing Relational Data Bases.

ROW

This is the equivalent of a record in a file or an observation in a SAS data set.

COLUMN

This is the equivalent of a field in a sequential file or a variable in a SAS data set.

TABLE

This is the equivalent of a file or a SAS data set.

JOIN

This is similar to a SAS merge. It is a way to combine columns from more than one table to obtain desired information.

Relational in a Nutshell

The essence of relational processing is to use more than one file to store your information in an efficient, easily maintained way. Figure 1 shows how a name file and Zip Code file can be related to show which city each person lives in. The name of the city is not on the file with the person's name. Instead, Zip Code is used to associate a name with a city. There are two advantages to this method: (1) the data can be stored in fewer bytes in most cases and (2) the files are easier to maintain. If the name of a city associated with a Zip Code changes, then only the entry on the Zip Code file will have to be changed. It will not be necessary to change a city field on every individual's record.

Review of Standard SQL Joins

There are a variety of ways of constructing joins. Those relevant for this presentation are old style joins and condition joins which consist of inner, left, right, and full. Old style joins are very similar to inner joins in terms of functionality.

Old Style Joins

These are the standard joins that most people are familiar with. They consist of having more than one table on the FROM clause and a WHERE clause which determines how to join the tables. This always results in an inner join. This example shows how the population of a city can be compared to the population of the entire state:

```
SELECT a.city, b.state, a.pop city_pop, b.pop
statepop
FROM cities a, states b
WHERE a.state = b.state;
```

Condition Joins

An example of a comparable inner join to the example above is:

```
SELECT a.city, b.state, a.pop city_pop, b.pop
statepop
FROM cities a INNER JOIN states b
ON a.state = b.state;
```

This gives the same result using different syntax. Note that with condition joins, it is possible to join only two tables at a time while old style joins allow as many tables as desired to be joined. However, old style joins are always INNER joins.

One advantage of condition joins over old style joins is that OUTER joins are possible. By replacing INNER with either LEFT, RIGHT, or FULL, an appropriate outer join will be performed.

Limitations of Standard SQL Joins

Some of the limitations of SQL joins are described in order to show why a search for an alternative joining technique was begun.

SQL2 Supports Outer Joins Between Exactly 2 Tables

Using OUTER JOINS, it is possible to join only two tables at a time. In order to join three tables, it is

necessary to first select two tables to join and then join the third table to the result of the first join. With more tables, the joining operations can be done in any order, but each joining operation can involve only two tables at a time. An OUTER JOIN in SQL involves multiple join operations between two tables at a time. It is not an integrated operation involving all of the tables simultaneously.

Outer joins involving more than two tables lose the Associative Property

The results of OUTER JOINS are sensitive to the order in which operations are performed. For example, $(36+6)+12 = 36+(6+12) = 54$ is true because addition is associative. The order of the addition operations does not change the result.

With division, things are a bit different. $(36/6)/12 = (6)/12 = 1/2$. However, $36/(6/12) = 36/(1/2) = 72$. The order of the division operations changes the result. Therefore, division does not have the associative property.

OUTER JOINS do not have the associative property as illustrated in figure 2.

Left and Right Outer Joins do not have the Transitive Property

This transitive property means that the position of two items does not affect the result. Addition is transitive as illustrated by $4+8 = 8+4$. Division is not transitive because $4/8$ is not equal to $8/4$.

OUTER JOINS are not transitive. Figure 3 demonstrates this. Actually, this is necessary for them to be useful. Otherwise it would not be possible to generate the desired rows. However, it has the potential to be confusing. A joining operation that permits desired rows to be generated while retaining the transitive property could be easier to use.

The Need for Flexible Joining of More Than Two Tables

Some positive reasons for a new outer joining technique are discussed.

Make Outer Joins as Easy as Inner Joins

Old style joins make inner joins involving more than two tables easy. You simply provide a list of tables on the FROM clause and joining criteria on the WHERE clause. It is fairly simple and straight-

forward. Having the same simplicity for outer joins would make them easier to use.

Some Databases do NOT have Referential Integrity

In situations where data is assembled from a variety of sources, referential integrity will not always exist. Common examples of this are marketing databases and the first cut at assembling data from different operational systems. Since rows in some tables will not match to rows in other tables, this is clearly an OUTER JOIN situation. If the relationships between tables are complex, it will be helpful to have a way of conveniently joining the tables in spite of the lack of referential integrity.

NULL Relationships Often Occur

Many databases have situations where a row in one table has no row in a corresponding table as a normal part of the file structure even when referential integrity is maintained. Examples of this are promotion tables having some rows that do not match to the order table. It is desirable, but highly unlikely, that everyone a company promotes to will respond by ordering something. Another example is a table of employees containing rows that do not match to a table of annual performance reviews. New employees will not have had an annual performance review.

These situations show simple examples of how outer joins can be part of even a well maintained, properly structured database. More complex database structures with many more tables become more complex to query using standard SQL.

Match Information the "Best" Way Possible

The N Table join supports flexible outer joins involving more than two files. In situations with incomplete matches, it does the best job it can to match records. This is especially useful for marketing databases and other databases that might have poor data integrity and null relationships.

For example, with three tables: customer, orders, and promotions; it might be desirable to get the most information you can when you have incomplete data. For example, if you have information from any two files, you might want to match that information together even though it does not match to the third file. To a large extent, this can be handled by using FULL JOINS with three tables, but, with more than three tables, FULL JOINS do not have all of the

flexibility that could be required in an easy to understand manner.

The N-Table Join

The N Table Join is designed to overcome some of the limitations of standard SQL outer joins. By making it possible to join an unlimited number of tables simultaneously, it provides the properties of associativity and transitivity which make it easier to understand what the results will be.

The N Table Join consists of two components: Joining Rules and Joining Conditions. Joining rules describe inner and outer joining conditions between tables - analogous to INNER, RIGHT, LEFT, and FULL JOIN in a FROM clause. Joining conditions describe how tables relate to one another - like the ON clause in SQL.

Joining Rules

The joining conditions determines which tables are joined together and how they are joined together. One method of specifying this is to use two new clauses: (1) with join inner and (2) with join outer. Table names or their aliases could appear in one or both clauses.

A table on the WITH JOIN INNER clause must be included in any join between two or more tables. Tables on the WITH JOIN INNER clause are essentially "driver" tables because any join without them is deemed irrelevant and is not performed.

A table in the WITH JOIN OUTER clause is guaranteed of having all of its rows somehow represented in the result of the join. If a row of a table in the WITH JOIN OUTER clause does not join with a row from another table, then the row is put into the join result with no matching information from other tables.

Alternatively, a WITH JOIN MASK clause could be used to specify which types of joins to make. This would have two items on it: a list of tables (or aliases) and a list of valid join combinations. For example, with tables A, B, and C, it could look something like this:

WITH JOIN MASK (A,B,C), (XXX,X##,##X)

The X and # symbols determine how the joins are set up. The XXX means that rows with information contributed from all three tables, A,B, and C will be used. X## means that rows receiving information only from table A will be used.. ##X means that rows receiving information only from table C will be used.

Notice that rows of table B that do not join to both table A and table C will not be part of the result of this join.

The join could have also been represented as

WITH JOIN INNER A,B,C

WITH JOIN OUTER A,C

Joining Conditions

Joining conditions are like the ON clause and could have a JOIN ON clause associated with them. The clause would consist of a list of joining conditions in parenthesis. An example looks like this:

```
JOIN ON (A.ACCOUNT = B.ACCOUNT),  
(A.ACCOUNT = C.ACCOUNT),  
(B.ACCOUNT = C.ACCOUNT and  
B.PROMOTION = C.PROMOTION)
```

There are three conditions. These conditions determine how tables are joined together. Also, they can determine which tables are joined together. Although the joining rules determine which joins will be attempted, the joining conditions further subset this list. Joins between tables that cannot be related to one another through joining conditions will not be performed unless the CARTESION key word appears immediately following the JOIN ON key words. Thus, unless CARTESION is specified, no joins will be performed on tables that cannot be related to one another either directly or indirectly. For example, the first condition connects table A to table B. The second condition connects table A to table C. Because of these two relationships, table B is connected to table C through table A. The third condition also connects tables B and C and provides additional restrictions on how joining is to be performed between those two tables. The third condition also permits tables B and C to be connected independently of table A which means they could be joined without using rows from table A if that should prove desirable.

Algorithm of the N Table Join

The N Table Join uses joining rules and joining conditions to determine how to join an arbitrary number of tables - say N tables - in a consistent way that permits the user to specify tables in any order without changing the rows that get generated. Of course, changing the joining rules or joining conditions will, in general, change the result rows.

Here is how the joining process works:

Given N tables and joining rules and joining conditions, perform the following steps.

Start by attempting to join all N tables together.

Set M to N. (Note that N must be at least 2.)

Iteration1: Start joining at the M level.

Iteration2: Given that M tables will be joined together at this iteration, find all possible combinations of M tables out of the N tables being joined.

Iteration3: Starting with the first set of M table combinations, repeat until the list of M table combinations is exhausted.

See if the tables to be joined are a valid combination of tables according to the joining rules. If they are, then perform the next step. Otherwise, go to *EndIteration3*.

Determine if all of the tables to be joined can be connected to each other using the joining conditions. If so, then perform the next step. Otherwise, go to *EndIteration3*.

Using any row that was not joined to another table during the previous levels of joining, attempt to join that row to other rows in accordance with the joining conditions.

EndIteration3: Goto *Iteration3* to process the next set of M tables. If there are no more sets of M tables to be processed then continue to the next step.

EndIteration2: Mark all rows in all tables that have been used in a join. This will be needed for the next level of joining.

Decrease M by 1.

If $M > 1$ then goto *Iteration1*.

EndIteration1: If $M = 1$ then generate outer join rows that are requested by the joining rules when rows in the appropriate tables have not been previously used in a join. The N table join is finished at this point.

This algorithm can make it possible to join large numbers of tables in complicated ways while still keeping the process manageable.

Example

Combine Account, Promotion,
and Order Data for a Customer

See figure 4 for a diagram of a sample database. This shows records for one customer. In this database, all records are related within a customer only.

Compare to SQL2 Outer Join

See Figure 5.

Notice that the WITH JOIN INNER and WITH JOIN OUTER clauses are used to control whether the join is an INNER, LEFT, RIGHT, or FULL join. The WITH JOIN INNER clause has no effect on a two table join. The WITH JOIN INNER clause has meaning only when at least three tables are being joined.

Example with 3 Files

Figure 6 shows the results of doing the "fullest" join possible on the data depicted in Figure 4. The code for producing this is shown below.

Select *

From Account as A, Promotion as P, Order
as O

join outer A, P, Y

join on

(A.Customer=P.Customer) ,
(A.Customer=O.Customer) ,
(P.Customer=O.Customer) ,
(A.Account=P.Account) ,
(A.Account=O.Account) and
(P.Promotion=O.Promotion);

Example with 3 Files

Order Oriented View of Data

Get Orders and information
applying to them

Figure 7 shows a different view of the data than Figure 6. Notice that different items were joined based only on changing the JOIN INNER and JOIN OUTER clauses.

Select *

From Account as A, Promotion as P, Order as O

join inner O

join outer O

join on

(A.Customer=P.Customer) ,
(A.Customer=O.Customer) ,
(P.Customer=O.Customer) ,

(A.Account=P.Account) ,
(A.Account=O.Account) ,
(P.Promotion=O.Promotion);

Fundamental Rules and Concepts

Joining Conditions - "Connected" Tables

Joining Conditions determine how the tables used in the join are connected to each other. Unless the CARTESION operator is used, all joining takes place only between tables that are related to each other directly or indirectly through other tables. Therefore, the joining conditions can affect the joining by not just how they relate the tables, but also by whether they relate the tables.

Inner and Outer Joining

It is possible to easily control how joining is performed by controlling the JOIN INNER and the JOIN OUTER clauses. Alternatively, a greater level of control is available by using the JOIN MASK clause.

Conclusion

An alternative to conventional joining is available that makes outer joins almost as easy to manage as inner joins. Since outer joins are a necessity for managing many types of data, the N Table joining technique can be of great value in many applications.

If you think this is a useful technique, feel free to forward the ideas to either ANSI, ISO, or both for possible inclusion as a standard in future releases of SQL.

For more information, feel free to contact the author

Howard Levine

DynaMark

4295 Lexington Ave N

St. Paul, MN 55126

612-486-1793

fax 612-481-8077

Internet: HowardLevine@FairIsaac.com

The author wishes to acknowledge the valuable assistance of David Sommer and others at DynaMark with clarifying the concepts of the N table join.

SAS, SAS/AF, SAS/FSP, and SAS/STAT are registered trademarks of SAS Institute, Cary, NC

Figure 1

Name File

Name	ZipCode
Bill	01249
Glenn	19395
Harriet	39499
Harry	39282
Jane	01837
Mary	39499
Melissa	19395
Mike	39204
Steve	39204

ZipCode File

ZipCode	City	State
01249	New Hope	NH
01837	Little Hope	MA
19395	Friendly	PA
39204	Showme	MO
39282	Blue Grass	KY
39499	Coal Dust	WV
42822	Motown	MI

- Zip Code relates a name to a City and State

Figure 2

Demonstration that Associative Property Does Not Apply to Outer Joins

```

1  DATA NAME;
2  LENGTH NNAME $ 10 NZIP $ 5;
3  INPUT NNAME NZIP;
4  CARDS;
NOTE: The data set WORK.NAME has 7 observations and 2
variables.
14
15  PROC SORT;
16  BY NNAME;
17  RUN;
NOTE: SAS sort was used.
NOTE: The data set WORK.NAME has 7 observations and 2
variables.
19  DATA ZIP;
20  LENGTH ZSTATE $ 2 ZZIP $ 5;
21  INPUT ZSTATE ZZIP;
22  CARDS;
NOTE: The data set WORK.ZIP has 7 observations and 2
variables.
32
33  PROC SORT;
34  BY ZZIP;
35  RUN;
NOTE: SAS sort was used.
NOTE: The data set WORK.ZIP has 7 observations and 2
variables.
38  DATA PROMOTE;
39  LENGTH PNAME $ 10 PPROMO $ 6;
40  INPUT PNAME PPROMO;

```

NOTE: The data set WORK.PROMOTE has 7 observations and 2 variables.

```

52  PROC SORT;
53  BY PNAME;
54  RUN;
NOTE: SAS sort was used.
NOTE: The data set WORK.PROMOTE has 7 observations and 2
variables.

```

NOTE: The data set WORK.PROMOTE has 7 observations and 2 variables.

```

55  PROC SQL;
56  CREATE TABLE JOIN1 AS
57  SELECT *
58  FROM ZIP LEFT JOIN NAME
59  ON NZIP = ZZIP;
NOTE: Table WORK.JOIN1 created, with 7 rows and 4 columns.

```

```

61  CREATE TABLE JOIN2 AS
62  SELECT *
63  FROM NAME RIGHT JOIN PROMOTE
64  ON NNAME = PNAME;
NOTE: Table WORK.JOIN2 created, with 7 rows and 4 columns.

```

```

66  CREATE TABLE RESULT1 AS
67  SELECT *
68  FROM JOIN1 RIGHT JOIN PROMOTE
69  ON NNAME = PNAME
70  ORDER BY NNAME.NZIP.ZSTATE;
NOTE: Table WORK.RESULT1 created, with 7 rows and 6
columns.

```

```

72  CREATE TABLE RESULT2 AS
73  SELECT *
74  FROM ZIP LEFT JOIN JOIN2
75  ON ZZIP = NZIP

```

76 ORDER BY NNAME,NZIP,ZSTATE;
 NOTE: Table WORK.RESULT2 created, with 7 rows and 6 columns.

OBS	DATA TEMP;	ZIP	2
78	IN1 = 0; IN2 = 0;		
79	MERGE RESULT1(IN=IN1)		
80	RESULT2(IN=IN2)		
81	;		
82	BY NNAME NZIP ZSTATE;		
83	RESULT1 = IN1;		
84	RESULT2 = IN2;		
85	IF NOT (IN1 AND IN2);		
86	RUN;		
87			

NOTE: The data set WORK.TEMP has 6 observations and 8 variables.

88 TITLE 'NAME';
 89 PROC PRINT DATA=NAME ; RUN;
 NOTE: The PROCEDURE PRINT printed page 1.

OBS	TITLE 'ZIP';	PROMOTE	3
90	PROC PRINT DATA=ZIP ; RUN;		
91			

NOTE: The PROCEDURE PRINT printed page 2.

92 TITLE 'PROMOTE';
 93 PROC PRINT DATA=PROMOTE; RUN;
 NOTE: The PROCEDURE PRINT printed page 3.

OBS	TITLE 'JOIN1';	JOIN1	4
94	PROC PRINT DATA=JOIN2 ; RUN;		
95			

NOTE: The PROCEDURE PRINT printed page 4.

96 TITLE 'JOIN2';
 97 PROC PRINT DATA=JOIN1 ; RUN;
 NOTE: The PROCEDURE PRINT printed page 5.

98 TITLE 'RESULT1';
 99 PROC PRINT DATA=RESULT1; RUN;
 NOTE: The PROCEDURE PRINT printed page 6.

100 TITLE 'RESULT2';
 101 PROC PRINT DATA=RESULT2; RUN;
 NOTE: The PROCEDURE PRINT printed page 7.

102 TITLE 'DIFFERENCES';
 103 PROC PRINT DATA=TEMP; RUN;
 NOTE: The PROCEDURE PRINT printed page 8.

OBS	NNAME	NZIP	PNAME	PPROMO
1	ARNOLD	30294	BERNIE	WIN95
2	DICK	69332	DICK	SUM95
3	HARRY	20404	HARRY	FALL93
4	JERRY	92918	JERRY	MAR94
5	MAX	30495	MAX	JAN95
6	SAM	30569	SAM	FEB93
7	TOM	55122	TOM	SPR94

OBS	NAME	1	OBS	ZSTATE	JOIN2	5
1	ARNOLD	30294	1	AK	20404	HARRY
2	DICK	69332	2	NY	30294	ARNOLD
3	HARRY	20404	3	RI	30569	SAM
4	JERRY	92918	4	MN	55122	TOM
5	MAX	30495	5	MA	64257	
6	SAM	30569	6	WI	67234	
7	TOM	55122	7	AZ	69332	DICK

OBS	RESULT1	6	OBS	ZSTATE	ZZIP	NNAME	NZIP	PNAME	PPROMO
1			1			BERNIE		WIN95	
2			2			MAX		JAN95	
3			3			JERRY		MAR94	
4	AZ	69332	4	AZ	69332	DICK	69332	DICK	SUM95
5	AK	20404	5	AK	20404	HARRY	20404	HARRY	FALL93
6	RI	30569	6	RI	30569	SAM	30569	SAM	FEB93
7	MN	55122	7	MN	55122	TOM	55122	TOM	SPR94

RESULT2 7						
OBS	ZSTATE	ZZIP	NNAME	NZIP	PNAME	PPROMO
1	MA	64257				
2	NY	30294				
3	WI	67234				
4	AZ	69332	DICK	69332	DICK	SUM95
5	AK	20404	HARRY	20404	HARRY	FALL93
6	RJ	30569	SAM	30569	SAM	FEB93
7	MN	55122	TOM	55122	TOM	SPR94

DIFFERENCES 8								
OBS	ZSTATE	ZZIP	NNAME	NZIP	PNAME	PPROMO	RESULT1	RESULT2
1					BERNIE	WIN95	1	0
2					MAX	JAN95	1	0
3					JERRY	MAR94	1	0
4	MA	64257					0	1
5	NY	30294					0	1
6	WI	67234					0	1

Figure 3
Outer Joins are NOT Transitive

<pre> 1 DATA NAME; 2 LENGTH NNAME \$ 10 NZIP \$ 5; 3 INPUT NNAME NZIP; 4 CARDS; NOTE: The data set WORK.NAME has 7 observations and 2 variables. 15 PROC SORT; 16 BY NNAME; 17 RUN; NOTE: SAS sort was used. NOTE: The data set WORK.NAME has 7 observations and 2 variables. 19 DATA ZIP; 20 LENGTH ZSTATE \$ 2 ZZIP \$ 5; 21 INPUT ZSTATE ZZIP; 22 CARDS; NOTE: The data set WORK.ZIP has 7 observations and 2 variables. 31 RUN; 33 PROC SORT; 34 BY ZZIP; 35 RUN; NOTE: SAS sort was used. NOTE: The data set WORK.ZIP has 7 observations and 2 variables. 38 DATA PROMOTE; 39 LENGTH PNAME \$ 10 PPROMO \$ 6; 40 INPUT PNAME PPROMO; 41 CARDS; NOTE: The data set WORK.PROMOTE has 7 observations and 2 variables. 52 PROC SORT; 53 BY PNAME; 54 RUN; NOTE: SAS sort was used. NOTE: The data set WORK.PROMOTE has 7 observations and 2 variables. 55 PROC SQL; 56 CREATE TABLE JOIN1 AS 57 SELECT * 58 FROM ZIP LEFT JOIN NAME 59 ON NZIP = ZZIP; NOTE: Table WORK.JOIN1 created, with 7 rows and 4 columns. 61 CREATE TABLE JOIN2 AS 62 SELECT * 63 FROM NAME LEFT JOIN ZIP 64 ON NZIP = ZZIP; NOTE: Table WORK.JOIN2 created, with 7 rows and 4 columns. 66 CREATE TABLE RESULT1 AS 67 SELECT * 68 FROM JOIN1 LEFT JOIN PROMOTE 69 ON NNAME = PNAME </pre>	<pre> 70 ORDER BY NNAME,NZIP,ZSTATE; NOTE: Table WORK.RESULT1 created, with 7 rows and 6 columns. 72 CREATE TABLE RESULT2 AS 73 SELECT * 74 FROM PROMOTE LEFT JOIN JOIN2 75 ON NNAME = PNAME 76 ORDER BY NNAME,NZIP,ZSTATE; NOTE: Table WORK.RESULT2 created, with 7 rows and 6 columns. 78 DATA TEMP; 79 IN1 = 0; IN2 = 0; 80 MERGE RESULT1(IN=IN1) 81 RESULT2(IN=IN2) 82 ; 83 BY NNAME NZIP ZSTATE; 84 RESULT1 = IN1; 85 RESULT2 = IN2; 86 IF NOT (IN1 AND IN2); 87 RUN; NOTE: The data set WORK.TEMP has 6 observations and 8 variables. 88 TITLE 'NAME'; 89 PROC PRINT DATA=NAME ; RUN; NOTE: The PROCEDURE PRINT printed page 1. 90 TITLE 'ZIP'; 91 PROC PRINT DATA=ZIP ; RUN; NOTE: The PROCEDURE PRINT printed page 2. 92 TITLE 'PROMOTE'; 93 PROC PRINT DATA=PROMOTE; RUN; NOTE: The PROCEDURE PRINT printed page 3. 94 TITLE 'JOIN1'; 95 PROC PRINT DATA=JOIN2 ; RUN; NOTE: The PROCEDURE PRINT printed page 4. 96 TITLE 'JOIN2'; 97 PROC PRINT DATA=JOIN1 ; RUN; NOTE: The PROCEDURE PRINT printed page 5. 98 TITLE 'RESULT1'; 99 PROC PRINT DATA=RESULT1; RUN; NOTE: The PROCEDURE PRINT printed page 6. 100 TITLE 'RESULT2'; 101 PROC PRINT DATA=RESULT2; RUN; NOTE: The PROCEDURE PRINT printed page 7. 102 TITLE 'DIFFERENCES'; 103 PROC PRINT DATA=TEMP; RUN; NOTE: The PROCEDURE PRINT printed page 8. </pre>
--	---

NAME 1			3			HARRY FALL93		
OBS	NNAME	NZIP	4			JERRY MAR94		
1	ARNOLD	30294	5			MAX JAN95		
2	DICK	69332	6			SAM FEB93		
3	HARRY	20404	7			TOM SPR94		
4	JERRY	92918						
5	MAX	30495						
6	SAM	30569						
7	TOM	55122						
			JOIN1			4		
OBS	NNAME	NZIP	ZSTATE	ZZIP				
1	HARRY	20404	AK	20404				
2	ARNOLD	30294	NY	30294				
3	MAX	30495						
4	SAM	30569	RI	30569				
5	TOM	55122	MN	55122				
6	DICK	69332	AZ	69332				
7	JERRY	92918						
			JOIN2			5		
OBS	ZSTATE	ZZIP	NNAME	NZIP				
1	AK	20404	HARRY	20404				
2	NY	30294	ARNOLD	30294				
3	RI	30569	SAM	30569				
4	MN	55122	TOM	55122				
5	MA	64257						
6	WI	67234						
7	AZ	69332						
PROMOTE			3					
OBS	PNAME	PPROMO						
1	BERNIE	WIN95						
2	DICK	SUM95						
			RESULT			6		
OBS	ZSTATE	ZZIP	NNAME	NZIP	PNAME	PPROMO		
1	MA	64257						
2	WI	67234						
3	NY	30294	ARNOLD	30294				
4	AZ	69332	DICK	69332	DICK	SUM95		
5	AK	20404	HARRY	20404	HARRY	FALL93		
6	RI	30569	SAM	30569	SAM	FEB93		
7	MN	55122	TOM	55122	TOM	SPR94		
			RESULT2			7		
OBS	PNAME	PPROMO	NNAME	NZIP	ZSTATE	ZZIP		
1	BERNIE	WIN95						
2	DICK	SUM95	DICK	69332	AZ	69332		
3	HARRY	FALL93	HARRY	20404	AK	20404		
4	JERRY	MAR94	JERRY	92918				
5	MAX	JAN95	MAX	30495				
6	SAM	FEB93	SAM	30569	RI	30569		
7	TOM	SPR94	TOM	55122	MN	55122		
			DIFFERENCES			8		
OBS	ZSTATE	ZZIP	NNAME	NZIP	PNAME	PPROMO	RESULT1	RESULT2
1					BERNIE	WIN95	0	1
2	MA	64257					1	0
3	WI	67234					1	0
4	NY	30294	ARNOLD	30294			1	0
5			JERRY	92918	JERRY	MAR94	0	1
6			MAX	30495	MAX	JAN95	0	1

Figure 4

- Combine Account, Promotion, and Order Data for a Customer

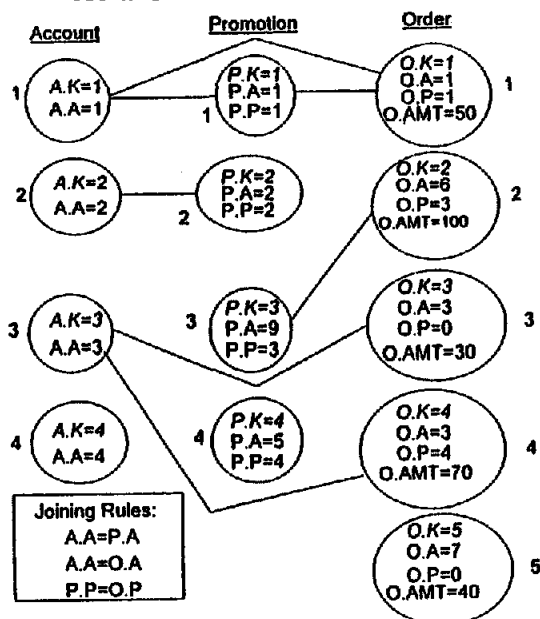


Figure 5-Compare to SQL2 Outer Join

- Simple Example

Names

Name	EmpNum
Bill	1
Bob	2
Babette	3

proc sql;

select *

from Names N full join JobTitle J

on N.EmpNum =

J.EmpNum;

JobTitle

EmpNum	JobTitle
1	Manager
3	Applications Prog.
4	Systems Prog.

Select *

from Names N, JobTitle J)

with join inner N,J

with join outer N,J

join on (N.EmpNum = J.EmpNum);

Figure 6 - Result

Joining Step	Files	A.K Account Key	P.K Promotion Key	O.K Order Key
1	A, P, O	1	1	1
2	A, P	2	2	-
	A, O	3	-	3
		3	-	4
	P, O	-	3	2
3	A	4	-	-
	P	-	4	-
	O	-	-	5

Figure 7
Result - Order
Oriented View of
Data

Joining Step	Files	A.K Account Key	P.K Promotion Key	O.K Order Key
1	A, P, O	1	1	1
2	A, O	3	-	3
		3	-	4
	P, O	-	3	2
3	O	-	-	5