Demystifying JOINs

Russell Sinclair

95 97 2000 2002 DOWNLOAD

In this month's installment of Working SQL, Russell Sinclair explains the mysteries of the JOIN statement in SQL and how to use it with both Jet and SQL Server.

IF you use Access enough that you're reading this publication, the chances are that you've spent a good deal of time designing queries. You've used the Access or SQL designers to create queries that link tables together and retrieve data from multiple tables. Chances are that you've also spent at least some time writing SQL statements yourself to execute from code or to use in your own queries. But if you want to get the real power out of any database system, you need to understand SQL. And the first step in understanding how to write SQL is understanding how to create JOINs.

JOINs seem to be a hurdle that users have to jump when writing SQL statements. The biggest problem with them is that the documentation on how to write a JOIN statement in Access is very limited. Also, the JOIN syntax in Access is totally different from the JOIN syntax in SQL Server (although you can use Access syntax in SQL Server—you just don't want to).

So that everyone is starting from the same page, I'll first explain what the different types of JOINs are. If you already know them (including the one Access doesn't support), feel free to skip ahead.

Types of JOINs

There are three basic types of JOIN: INNER, OUTER, and CROSS. In order to illustrate the effects of the different types of JOINs, I'll use the data from the tables shown in **Figure 1**, **Figure 2**, and **Figure 3**.

INNER JOINs are the most commonly used type of JOIN. An INNER JOIN combines the data in two or more tables and returns the data from both tables where the linked fields match. Any records in either of the tables that don't have a corresponding value in the other table won't be returned. Using my example data, an INNER JOIN of tlkpCompanyType with tblCustomer would yield the results shown in **Figure 4**.

As you can see, the customer name "Friendly Building Management" isn't returned in the results. Neither are the "GIS" or "Network Support" company types. This is because INNER JOINs only return data where the joined fields match. All other data is

Figure 1. Company Types table.

tlkpCompanyType : Table

CompanyTypeID	CTDescription
1	Software Development
2	Manufacturing
3	GIS
4	Network Support
5	Retail
6	Consulting
(AutoNumber)	

Record: 1 of 6

tblCustomer : Table

CustomerID	CustomerName	CompanyTypeID
1	ABC Inc.	2
2	Synthesystems	1
3	PH&V	6
4	Acme Roadrunner Products	5
5	Friendly Building Management	
AutoNumber)		

Record: 6 of 6

Figure 2. Customers table.

tblOrder : Table

OrderID	CustomerID	OrderDate	OrderTotal
1	1	01/07/1997	$4,887.00
2	2	05/04/1998	$595.00
3	4	05/11/1999	$166,745.00
4	1	14/06/2002	$1,489,975.00
5	2	13/06/2002	$5,998.00
6	2	14/06/2002	$166,647.00
Number)		14/06/2002	$0.00

Record: 1 of 6

Figure 3. Orders table.

CompanyTypeID	CTDescription	CustomerID	CustomerName
1	Software Development	2	Synthesystems
2	Manufacturing	1	ABC Inc.
5	Retail	4	Acme Roadrunner Products
6	Consulting	3	PH&V
(AutoNumber)		AutoNumber)	

Record: 5 of 5

Figure 4. INNER JOIN of Company Types and Customers.

excluded from the results.

OUTER JOINs differ from INNER JOINs in that the data returned contains all of the data from the INNER JOIN plus all the rows from one of the tables even if that row doesn't have matching data in the other table (based on the joined fields). Any row that doesn't have a matching row will be matched by NULL values, which represent that no data is coming from the other table. There are three different types of OUTER JOIN: LEFT, RIGHT, and FULL.

With LEFT and RIGHT OUTER JOINs, the words "left" and "right" refer to the table for which all rows will be returned. The left or rightness of a table has nothing to do with the relationships defined between the tables. The location of the table (left or right) is based on the order of the tables in the SQL statement. The table on the left side of the JOIN statement is the left table, and the table on the right side of the JOIN statement is the right table. A LEFT OUTER JOIN between tlkpCompanyType and tblCustomer where tlkpCompanyType is on the left side of the JOIN statement yields the results shown in **Figure 5**.

With this LEFT OUTER JOIN, the two unused Company Types are listed, but NULL values appear in the fields from the Customers table. I would have gotten the same results if I'd used a RIGHT OUTER JOIN and also switched the order of the tables in the JOIN.

qselLEFTJOIN : Select Query

CompanyTypeID	CTDescription	CustomerID	CustomerName
1	Software Development	2	Synthesystems
2	Manufacturing	1	ABC Inc.
3	GIS		
4	Network Support		
5	Retail	4	Acme Roadrunner Products
6	Consulting	3	PH&V
(AutoNumber)		AutoNumber)	

Record: 6 of 6

Figure 5. LEFT OUTER JOIN from Company Types to Customers.

qselRIGHTJOIN : Select Query

CompanyTypeID	CTDescription	CustomerID	CustomerName
		5	Friendly Building Management
1	Software Development	2	Synthesystems
2	Manufacturing	1	ABC Inc.
5	Retail	4	Acme Roadrunner Products
6	Consulting	3	PH&V
(AutoNumber)		AutoNumber)	

Record: 1 of 5

Figure 6. RIGHT OUTER JOIN from Company Types to Customers.

qunFULLJOIN : Union Query

CompanyTypeID	CTDescription	CustomerID	CustomerName
		5	Friendly Building Management
1	Software Development	2	Synthesystems
2	Manufacturing	1	ABC Inc.
3	GIS		
4	Network Support		
5	Retail	4	Acme Roadrunner Products
6	Consulting	3	PH&V

Record: 1 of 7

Figure 7. FULL OUTER JOIN between Company Types and Customers.

Changing the JOIN type to a RIGHT OUTER JOIN and not switching the tables would return the data shown in **Figure 6**. All the records are returned from the Customers table, and any Customers without associated Company Types have NULL values to represent that no matches exist in the Company Types table.

FULL OUTER JOINs take the concepts of LEFT and RIGHT OUTER JOINs and combine them. All data from all tables is returned, but unmatched data on either side of the JOIN statement is returned with NULL values representing unmatched data in the other table. A FULL OUTER JOIN of the same data would return the results shown in **Figure 7**.

The final type of JOIN, a CROSS JOIN, returns all possible combinations of all rows in one table with all of the rows in another table (called the Cartesian product of the two tables). **Figure 8** shows some of the data from a CROSS JOIN between tlkpCompanyType and tblCustomer.

Although you can only see the first 10 records (you can view the rest by running qselCROSSJOIN in the sample database in this month's Download file, available at www.smartaccessnewsletter.com), you can see that each Company Type is shown matched with all five Companies. The number of records returned by a CROSS JOIN can be determined by multiplying the

number of records in each table together (in this case, five companies * six company types = 30 rows).

Now that you know what the different types of JOINs are, you need to know how to write them.

JOIN syntax

The basic syntax for a JOIN statement in a FROM clause is:

```
FROM Table1
     [INNER|[LEFT|RIGHT|FULL] [OUTER]|CROSS] JOIN Table2
     [ON MatchExpression]
```

Table1 and Table2 refer to the left and right tables, respectively. MatchExpression refers to the comparison made between the two tables. When creating a CROSS JOIN, the ON portion of the statement should be omitted. In almost all RDBMS systems, the keyword OUTER is optional, and this is true for Access and SQL Server. Just using LEFT, RIGHT, or FULL is enough to signal that you want to use an OUTER JOIN. It's unlikely that you'll see many places where the word "OUTER" is actually used by a programmer. In fact, Access will quietly remove the OUTER keyword if you type it into the Query Designer's SQL view just as soon as you save and reopen your query.

In order to understand the syntax for creating a JOIN, look at the statements used for some of the previous examples. This is the FROM portion of the SQL statement for the results shown in Figure 4—qselINNERJOIN:

```
FROM tlkpCompanyType
    INNER JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
```

In this example, tlkpCompanyType is the left table, and tblCustomer is the right table. The expression used to join the tables links them only where the CompanyTypeID in one table is exactly equal to the CompanyTypeID in the other table.

The LEFT JOIN used to return the data in Figure 5 is:

```
FROM tlkpCompanyType LEFT JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
```

The tables are in the same order here, and the match expression hasn't changed. The only difference is that the INNER has changed to a LEFT. All data is returned from tlkpCompanyType, but only the matching records are returned from tblCustomer.

The standard CROSS JOIN syntax used to return the data in Figure 8 would be:

```
FROM tlkpCompanyType CROSS JOIN tblCustomer
```

In this case, you don't enter any JOIN criteria.

This statement can be written another way, without using the JOIN keyword—by simply separating the table names by commas, as in:

```
FROM tlkpCompanyType, tblCustomer
```

When creating queries in Access, you must use the comma syntax because Access doesn't understand the CROSS JOIN statement.

I've left off FULL OUTER JOINs until this point, as there's one caveat with them. Microsoft Access doesn't support this type of JOIN. You can easily create a FULL OUTER JOIN in SQL Server using the statement:

```
FROM tlkpCompanyType FULL JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
```

However, you must use a UNION query to simulate a FULL JOIN in Access. You do this by unioning a LEFT JOIN with a RIGHT JOIN. Since a standard UNION query eliminates duplicate records,

qselCROSSJOIN : Select Query

CompanyTypeID	CTDescription	CustomerID	CustomerName
1	Software Development	1	ABC Inc.
1	Software Development	2	Synthesystems
1	Software Development	3	PH&V
1	Software Development	4	Acme Roadrunner Products
1	Software Development	5	Friendly Building Management
2	Manufacturing	1	ABC Inc.
2	Manufacturing	2	Synthesystems
2	Manufacturing	3	PH&V
2	Manufacturing	4	Acme Roadrunner Products
2	Manufacturing	5	Friendly Building Management

Record: 1 of 30

Figure 8. CROSS JOIN between Company Types and Customers.

only those records that aren't returned in the first query will be returned in the second. The full statement used in Access to return the data shown in Figure 7 is:

```
SELECT tlkpCompanyType.CompanyTypeID,
    tlkpCompanyType.CTDescription,
    tblCustomer.CustomerID,
    tblCustomer.CustomerName
FROM tlkpCompanyType LEFT JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
UNION
SELECT tlkpCompanyType.CompanyTypeID,
    tlkpCompanyType.CTDescription,
    tblCustomer.CustomerID,
    tblCustomer.CustomerName
FROM tlkpCompanyType RIGHT JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
```

Don't use the optional ALL keyword with the UNION keyword because UNION ALL specifies that all records should be returned, without filtering out duplicate records. If there's a possibility that duplicate records could be omitted when they're wanted, use the ALL keyword but add a WHERE clause to the second query to remove those rows that are returned by the first query.

Nested JOINs

A common area for people to run into problems in Access when creating queries is in nested JOIN statements. These are used in queries that retrieve data from multiple tables and require that each table be joined to other tables in the FROM statement.

Creating nested JOINs in SQL Server is easy. All you have to do is add multiple JOIN statements with an ON section (if required). **Figure 9** shows the data returned by joining the Company Type, Customer, and Order tables. This query uses a nested JOIN.

The FROM statement in this query, if written for SQL Server, would be:

```
FROM tlkpCompanyType INNER JOIN tblCustomer
    ON tlkpCompanyType.CompanyTypeID
       = tblCustomer.CompanyTypeID
    INNER JOIN tblOrder
    ON tblCustomer.CustomerID
       = tblOrder.CustomerID
```

Each JOIN can be written in succession without regard for the order of the tables in the FROM statement as long as your ON statement links the necessary fields together. Effectively, you can think of the two tables that have been joined together as a single new table that you can join to a new table.

Access requires that you nest JOINs by bracketing JOIN statements when combining them. A more appropriate definition for the syntax of creating an INNER or OUTER JOIN in Access would be:

```
FROM TableOrJOINExpression1
    [INNER|[LEFT|RIGHT] [OUTER]] JOIN
    TableOrJOINExpression2 ON MatchExpression
```

The best way to think of this is to break down each JOIN that must take place into a single unit. Suppose I wanted to join four tables together based on ID values in each table. I'll use the examples A.aID, B.bID, C.cID, and D.dID to make things easier and only use INNER JOINs. In my example, table B is related to table A on a field called B.aID; C is related to B on a field called C.bID; and (just to make things interesting), C is also related to D on a field called C.dID. In order to break this down for use in Access, I need to perform each JOIN separately.

The first JOIN I want to make is between C and D. This can be done by using the simplest syntax linking the two tables.

```
C INNER JOIN D ON C.dID = D.dID
```

This statement can be thought of as a single JOIN expression. In order to use it in later JOINs, I simply need to wrap it in brackets. Now, I also want to relate C to B. Normally, I'd do this using the same syntax.

```
B INNER JOIN C ON C.bID = B.bID
```

In this case, however, I need to combine this statement with the previous JOIN statement. I need to substitute the "C" in "JOIN C ON" with the expression previously created and wrap the replacement in brackets.

```
B INNER JOIN (C INNER JOIN D ON C.dID = D.dID)
    ON C.bID = B.bID
```

I'm one step closer to my final JOIN statement.

qselMultipleJOINs : Select Query

CompanyTypeID	CTDescription	CustomerID	CustomerName	OrderID	OrderDate	OrderTotal
2	Manufacturing	1	ABC Inc.	1	01/07/1997	$4,887.00
2	Manufacturing	1	ABC Inc.	4	14/06/2002	$1,489,975.00
1	Software Development	2	Synthesystems	2	05/04/1998	$595.00
1	Software Development	2	Synthesystems	5	13/06/2002	$5,998.00
1	Software Development	2	Synthesystems	6	14/06/2002	$166,647.00
5	Retail	4	Acme Roadrunner Products	3	05/11/1999	$166,745.00
(AutoNumber)		AutoNumber)		Number)		

Record: 1 of 6

Figure 9. Multiple joined tables.

Now, I need to join A and B. This would normally be done using the statement:

```
A INNER JOIN B ON A.aID = B.aID
```

Knowing this, I simply substitute my last JOIN statement for "B" and *voilà!*—I have a nested JOIN.

```
A INNER JOIN
    (B INNER JOIN
      (C INNER JOIN D ON C.dID = D.dID)
     ON C.bID = B.bID
     ) ON A.aID = B.aID
```

JOIN Properties Dialogs

Depending on whether you're working with a Jet or SQL database, Access will show you one of two dialogs for JOIN properties if you right-click a JOIN line and choose Properties from the context menu. In Access, the dialog gives you a numbered option at the bottom of the screen. In this case, option 1 will create an INNER JOIN, 2 a LEFT JOIN, and 3 a RIGHT JOIN.

In the SQL Server designers, you have two check boxes for the Include Rows option. Leaving both check boxes blank will set the JOIN to an INNER JOIN; selecting the upper check box will specify a LEFT JOIN; selecting the lower box will specify a RIGHT JOIN; and selecting them both will result in a FULL JOIN.

Extending JOINs

Now that you know the syntax for creating JOINs, you can put this syntax to use. However, don't feel that you should be limited to JOINs where one field in the match criteria is always exactly equal to the other field. The June 1997 installment of Working SQL (available online to paid subscribers in the "View Past Issues" area of the *Smart Access* Web site) contains some very interesting JOIN ideas. Peter Vogel takes you through unequal JOINs using greater-than comparisons. This is just one way you can put JOINs to work.

Happy JOINing. ▲

ALLJOIN.ZIP at www.smartaccessnewsletter.com

Russell Sinclair is an MSCD and is the owner of Synthesystems, a technology consulting firm specializing in Visual Basic, SQL Server, and Microsoft Access development. He's the author of *From Access to SQL Server*, an Access developer's guide to migrating to SQL Server; the senior programmer with Questica, Inc., a company specializing in software for custom-design manufacturers; and a *Smart Access* Contributing Editor. russell@synthesystems.com.