

Module 3: The JOIN Operators

Introduction to Joining Multiple Tables

The typical relational database spreads related data across multiple tables. Most of the time your queries will need to retrieve columns from several different tables and join these together based on matching row values or another logical comparison.

To introduce joins, we're going to start with a simple, hypothetical database named **JoinDemo**. This database contains three tables:

1. **dbo.MsMertz**, which contains a list of the five students in Ms. Mertz' class;
2. **dbo.TestScores**, which contains a list of test scores of four students, three of whom are in Ms. Mertz' class;
3. **dbo.CurlingTeam**, which contains a list of the four students on the Curling team. Three of these students are in Ms. Mertz' class.

MsMertz	
StudentID	
FirstName	
LastName	

CurlingTeam	
MemberID	
StudentID	
Position	

TestScores	
TestTakerNum	
StudentID	
TestScore	

Creating the JoinDemo tables

Rather than create a new database, for this demonstration we're going to simply insert our three tables into CarDealer and add a few rows of data to each.

Demonstration Steps

1. Open a new query window against the JoinDemo database. Enter and run the following code:

```
( Create table MsMertz
  StudentID int,
```

```

        FirstName varchar(20),
        LastName varchar(20)
    );

insert into MsMertz (StudentID,FirstName,LastName) values
(1, 'Jimmy', 'McGill'), (2, 'Mike', 'Ehrmantraut'),
(3, 'Kim', 'Wexler'), (4, 'Nacho', 'Varga'), (5, 'Howard', 'Hamlin');

Create table TestScores
(
    TestTakerNum int,
    StudentID int,
    TestScore int
);

insert into TestScores (TestTakerNum,StudentID,TestScore) values
(1,1,100), (2,2,90), (3,35,96), (5,5,98);

create table CurlingTeam
(
    MemberID int,
    StudentID int,
    Position varchar(10)
)

insert into CurlingTeam (MemberID,StudentID,Position) values
(1,2, 'Lead'), (2,5, 'Third'), (3,10, 'Skipper')

```

1. The above TSQL code will create our three tables and add data to each. Refresh the Tables list under JoinDemo in ObjectExplorer to see the three new tables.
2. Run a SELECT * query against each of the three tables to see their contents. Note that all three tables share a StudentID field.

CROSS JOIN, and the Predicate Applied to the Full Cartesian Product

To understand how joins works, we're first going to look at the CROSS JOIN operation. CROSS JOIN simply crosses each row in one table with each row in another other. In practice this isn't something we need to do very often, and the use of CROSS JOIN is limited to things like creating sample data and tables of numbers. But as we'll see, CROSS JOIN demonstrates something very important about the way other JOIN operations work – the ***full Cartesian product***.

Consider the following query. Each row in dbo.MsMertz will be crossed with each row in dbo.TestScores. As there are five rows in dbo.MsMertz and four rows in dbo.TestScores, there will be 5 x 4 or 20 rows in the full Cartesian product produced by CROSS JOIN:

```

SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m

```

CROSS JOIN `dbo.TestScores t;`

<i>Table dbo.MsMertz</i>				<i>Table dbo.TestScores</i>			
	StudentID	FirstName	LastName		TestTakerNum	StudentID	TestScore
1	1	Jimmy	McGill	1	1	1	100
2	2	Mike	Ehmantraut	2	2	2	90
3	3	Kim	Wexler	3	3	35	96
4	4	Nacho	Varga	4	5	5	98
5	5	Howard	Hamlin				

<i>Full Cartesian Product</i>						
	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	1	1	100
3	3	Kim	Wexler	1	1	100
4	4	Nacho	Varga	1	1	100
5	5	Howard	Hamlin	1	1	100
6	1	Jimmy	McGill	2	2	90
7	2	Mike	Ehmantraut	2	2	90
8	3	Kim	Wexler	2	2	90
9	4	Nacho	Varga	2	2	90
10	5	Howard	Hamlin	2	2	90
11	1	Jimmy	McGill	35	3	96
12	2	Mike	Ehmantraut	35	3	96
13	3	Kim	Wexler	35	3	96
14	4	Nacho	Varga	35	3	96
15	5	Howard	Hamlin	35	3	96
16	1	Jimmy	McGill	5	5	98
17	2	Mike	Ehmantraut	5	5	98
18	3	Kim	Wexler	5	5	98
19	4	Nacho	Varga	5	5	98
20	5	Howard	Hamlin	5	5	98

As mentioned earlier, we generally aren't interested in the full Cartesian product. It would be more useful to only select the rows from each table that having matching values on a particular column. In this case, we'd like to return only those rows from the full Cartesian product that have the same StudentID value in both tables. In other words, when we look at the full Cartesian product, there are only three rows we want from it:

1. Jimmy McGill
2. Mike Ehrmantraut
3. Howard Hamlin

We want those three rows because they are the only rows that have matching values on the StudentID column in both tables.

We could apply a **predicate** in a WHERE clause to the full Cartesian product to select just those rows, as follows:

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m
CROSS JOIN dbo.TestScores t
where m.StudentID=t.StudentID;
```

Technically, the above query performs an **inner join**. Our results consist only of the records from the full Cartesian product for which the WHERE clause predicate evaluates to true. However, in the next section we'll investigate the JOIN operator, which allows us to perform the JOIN without having to use a WHERE clause.

3.3 INNER JOIN

Now let's take a look at the most frequently used JOIN operation, the INNER JOIN. Most queries written for data analysis and reporting purposes will include one or more inner join operations.

The INNER JOIN is the default join, so the word INNER can be left out. Let's say we want to produce a list of the students in Ms. Mertz' class who took the test. The following query will do that:

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m
JOIN dbo.TestScores t
ON m.StudentID=t.StudentID;
```

Table <i>dbo.MsMertz</i>				Table <i>dbo.TestScores</i>			
	StudentID	FirstName	LastName		TestTakerNum	StudentID	TestScore
1	1	Jimmy	McGill	1	1	1	100
2	2	Mike	Ehmantraut	2	2	2	90
3	3	Kim	Wexler	3	3	35	96
4	4	Nacho	Varga	4	5	5	98
5	5	Howard	Hamlin				

Result of INNER JOIN on *m.StudentID = t.StudentID*

	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	2	2	90
3	5	Howard	Hamlin	5	5	98

As you can see, the result contains 3 rows. Here's how it works.

1. The full Cartesian product is produced by crossing each row of `dbo.MsMertz` with each row of `dbo.TestScores` to produce a full Cartesian product of 20 rows.
2. The **ON** operator is applied. The On operator applies a predicate to each row in the full Cartesian product. A predicate is a logical expression that evaluates to true, false, or null. Only those rows in the full Cartesian product for which the predicate is true are retained in the result set to be returned by the FROM clause. In this case, only those rows that have matching values on the StudentID field from each table are returned.

	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	1	1	100
3	3	Kim	Wexder	1	1	100
4	4	Nacho	Varga	1	1	100
5	5	Howard	Hamlin	1	1	100
6	1	Jimmy	McGill	2	2	90
7	2	Mike	Ehmantraut	2	2	90
8	3	Kim	Wexder	2	2	90
9	4	Nacho	Varga	2	2	90
10	5	Howard	Hamlin	2	2	90
11	1	Jimmy	McGill	35	3	96
12	2	Mike	Ehmantraut	35	3	96
13	3	Kim	Wexder	35	3	96
14	4	Nacho	Varga	35	3	96
15	5	Howard	Hamlin	35	3	96
16	1	Jimmy	McGill	5	5	98
17	2	Mike	Ehmantraut	5	5	98
18	3	Kim	Wexder	5	5	98
19	4	Nacho	Varga	5	5	98
20	5	Howard	Hamlin	5	5	98

The diagram above shows the 20 rows of the full Cartesian product. For three of these rows the predicate specified by ON evaluates to true because there is a match between the StudentID column in dbo.MsMertz and the StudentID column in dbo.TestScores. These rows will be returned in the result set. The rest will not.

Notes about JOINS:

1. The columns in the ON operation are usually the foreign key of one table and the primary key of the other, although this isn't required and will not always be the case.
2. Primary keys and their foreign key dependencies will generally have the same name, therefore the columns in the ON operation will too. Again, this is not required and often won't be the case.
3. Tables joined on foreign key-primary key pairings will almost always be an *equi-join*, meaning the predicate will be a simple column value match. However, any logical expression can be used within ON.

OUTER JOINS

There are three versions of the OUTER JOIN: **RIGHT OUTER JOIN**, **LEFT OUTER JOIN**, and **FULL OUTER JOIN**. As these are all OUTER joins, the word OUTER can be left out, leaving **RIGHT JOIN**, **LEFT JOIN**, and **FULL JOIN**. The outer joins work in a similar way to the inner join, except as follows:

1. **RIGHT JOIN**: Non-matching records from the table to the RIGHT of the JOIN operator are returned;
2. **LEFT JOIN**: Non-matching records from the table to the LEFT of the JOIN operator are returned;
3. **FULL JOIN**: Non-matching records from the table on each side of the JOIN operator are returned.

3.4.1 LEFT OUTER JOIN

Let's say we want to retrieve a list of all the students in Ms. Mertz class, along with their test scores if the score is listed in dbo.TestScores. The following query will do this:

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m
LEFT JOIN dbo.TestScores t
ON m.StudentID=t.StudentID;
```

Results of the above LEFT JOIN

	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	2	2	90
3	3	Kim	Wexler	NULL	NULL	NULL
4	4	Nacho	Varga	NULL	NULL	NULL
5	5	Howard	Hamlin	5	5	98

As you can see in the result set, all five students are returned. Kim and Nacho, however, have NULL in the t.StudentID, t.TestTakeNum, and TestScore columns since they did not take the test.

The LEFT JOIN is the same as the JOIN except that the non-matching records from the table to the LEFT of the JOIN operator are returned. In this case, since there is no StudentID value in dbo.TestScores for the non-matching records, NULL appears in the result set for the columns from dbo.TestScores.

3.4.2 RIGHT OUTER JOIN

If we simply change this to a RIGHT JOIN, the same thing happens except that this time the non-matching records in the table to the RIGHT of the join operator are returned:

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m
RIGHT JOIN dbo.TestScores t
ON m.StudentID=t.StudentID;
```

In this case, there is one record in dbo.TestScores for which the StudentID value does not match any StudentID values in dbo.MsMertz.

Results of the above RIGHT JOIN

	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	2	2	90
3	NULL	NULL	NULL	35	3	96
4	5	Howard	Hamlin	5	5	98

3.4.3 FULL OUTER JOIN

Finally, we have the FULL JOIN. This is just like the others, except that the non-matching rows from both tables are returned. The following query returns six rows, with NULL values from `dbo.MsMertz` for the student with `StudentID` 35 in `dbo.TestScores`, and NULL values from `dbo.TestScores` for Kim Wexler and Nacho Varga, who are in Ms. Mertz class but don't have records in `dbo.TestScores`.

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
FROM dbo.MsMertz m
FULL JOIN dbo.TestScores t
ON m.StudentID=t.StudentID;
```

Result of the above FULL JOIN

	m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore
1	1	Jimmy	McGill	1	1	100
2	2	Mike	Ehmantraut	2	2	90
3	3	Kim	Wexler	NULL	NULL	NULL
4	4	Nacho	Varga	NULL	NULL	NULL
5	5	Howard	Hamlin	5	5	98
6	NULL	NULL	NULL	35	3	96

3.5.1 Joining Three Tables with INNER JOIN

Now let's join a third table. Suppose we want to produce a list of the students in Ms. Mertz' class who took the test and are also on the curling team. The query below returns the two students from Ms. Mertz' class who appear in both the `dbo.TestScores` and `dbo.CurlingTeam` tables.

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
    ,ct.StudentID as [ct.StudentID]
    ,ct.MemberID
    ,ct.Position
FROM dbo.MsMertz m
JOIN dbo.TestScores t
ON m.StudentID=t.StudentID
JOIN dbo.CurlingTeam ct
ON m.StudentID=ct.StudentID;
```


Results of the above query in which three tables are joined

m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore	ct.StudentID	MemberID	Position
2	Mike	Ehmantraut	2	2	90	2	1	Lead
5	Howard	Hamlin	5	5	98	5	3	Third

Here’s how it works:

1. Tables `dbo.MsMertz` and `dbo.TestScores` are joined exactly as before. A full Cartesian product is produced, then the `ON` operation is performed and only the matching records are retained. This produces an intermediate result set.
2. The `JOIN` operation is then performed on results of the first `JOIN` and the `dbo.CurlingTeam` table. Again, since this is an `INNER JOIN`, only matching records from both sides of the `JOIN` operator are retained.

3.5.2 Combining INNER and OUTER JOINS

Combining `INNER` and `OUTER JOINS` follows the same rules. The query below returns the three students in Ms. Mertz’ class who took the test, along with the curling team member ID and position for the students who are in Ms. Mertz’ class `AND` took the test `AND` are on the curling team. Jimmy McGill is in Ms. Mertz class and took the test, but is not on the curling team, so he has `NULL` values for the columns from `dbo.CurlingTeam`.

```
SELECT
    m.StudentID as [m.StudentID]
    ,m.FirstName
    ,m.LastName
    ,t.StudentID as [t.StudentID]
    ,t.TestTakerNum
    ,t.TestScore
    ,ct.StudentID as [ct.StudentID]
    ,ct.MemberID
    ,ct.Position
FROM dbo.MsMertz m
JOIN dbo.TestScores t
ON m.StudentID=t.StudentID
LEFT JOIN dbo.CurlingTeam ct
ON m.StudentID=ct.StudentID;
```

Results of the above INNER JOIN followed by OUTER JOIN

m.StudentID	FirstName	LastName	t.StudentID	TestTakerNum	TestScore	ct.StudentID	MemberID	Position
1	Jimmy	McGill	1	1	100	NULL	NULL	NULL
2	Mike	Ehmantraut	2	2	90	2	1	Lead
5	Howard	Hamlin	5	5	98	5	3	Third

1. The INNER JOIN is performed between dbo.MsMertz and dbo.TestScores, producing an intermediate result with records for the three students (Jimmy McGill, Mike Ehrmantraut, and Howard Hamlin) who have test scores in dbo.TestScores.
2. The LEFT JOIN is performed between this intermediate result and dbo.CurlingTeam. Mike and Howard are both on the curling team, but Jimmy is not, so NULL values appear for Jimmy in the columns from dbo.CurlingTeam.

3.5.3 Joining More Than Three Tables

Joining additional tables happens in exactly the same way. Now that we've learned the basics of JOINS, let's go back to the CarDealer database. Let's say we want to produce a list of all car sales for the year 2011, including the make and model of the car sold, the sales amount, and the name of the customer who purchased the car. As is typical of most SELECT queries you'll write, the JOINS will be between the foreign key in one table and the primary key of another.

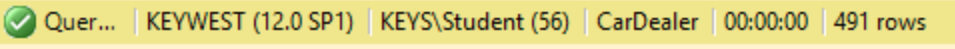
To produce the result set we need, we'll join dbo.Sales to dbo.Customer, then join dbo.Person and dbo.Cars. As these will all be INNER JOINS, the order in which tables are joined won't matter. Finally, we'll need to include a WHERE clause in order to filter out all sales not occurring in 2011.

```
SELECT
    c.Make
    ,c.Model
    ,s.SalesAmount
    ,p.LastName + ', ' + p.FirstName as [Customer]
FROM dbo.Sales s
JOIN dbo.Cars c
ON s.CarID=c.CarID
JOIN dbo.Customers cust
ON s.CustomerID=cust.CustomerID
JOIN dbo.Person p
ON cust.PersonID=p.PersonID
WHERE s.SalesDate >= '20110101' AND s.SalesDate < '20120101'
```

2

<i>Results of the Above Four-Way JOIN</i>
--

	Make	Model	SalesAmount	Customer
1	Mercury	Cyclone	160849.1167	Okey, Twila
2	Ford	Thunderbolt	40499.2094	Thoams, Oren
3	Ford	Mustang Mach 1 Fastb	48398.9409	Ruschmann, Elton
4	Buick	Gran Sport	188271.652	Borgese, Tamika
5	Pontiac	GTO	53987.6192	Flot, Eloise
6	Chevrolet	Corvette T-Top	128949.0123	Spalter, Calvin
7	Plymouth	Duster	191514.7063	Mckeehan, Dorothy
8	Datsun	240Z	142868.8479	Trembley, Gilberto
9	Buick	Wildcat	41973.6864	Siers, Quintin
10	Dodge	Charger Daytona	120873.5446	Deering, Reynaldo
11	Mercury	Comet	144467.351	Hilderbrandt, Deidra
12	Ford	Galaxie	97815.4124	Durrenberger, Brid...
13	Chevrolet	Chevelle SS 396	137583.6029	Linnane, Rosie
14	Chevrolet	Nova SS	61350.3629	Shenker, Lvn


 Quer... | KEYWEST (12.0 SP1) | KEYS\Student (56) | CarDealer | 00:00:00 | 491 rows

3.6 Combining JOIN Operations with WHERE and ORDER BY

Remember the order of operations:

1. FROM
2. WHERE
3. GROUP BY
4. SELECT list
5. ORDER BY

Since JOIN (regardless of the type) is an operation within the FROM clause, all JOINS are processed and a result set produced before the WHERE clause is processed. So the WHERE clause operates on the individual rows in the result set produced by all JOINS in the FROM clause.

Consider the following query:

```

SELECT
    p.LastName + ', ' + p.FirstName as [Sales Person]
    , s.SalesAmount
FROM dbo.Sales s
JOIN dbo.SalesPeople sp
ON s.SalesPersonID=sp.EmployeeID
JOIN dbo.Person p
ON sp.PersonID=p.PersonID
WHERE sp.Active=1 AND s.SalesDate >= '20110101' AND s.SalesDate < '20120101'
ORDER BY s.SalesAmount DESC
  
```

Results of the Above Query Combining a Three-Way Join With WHERE and ORDER BY

	Sales Person	SalesAmount
1	Addiego, Oscar	228470.7932
2	Kisling, Irving	221677.6932
3	Ruddock, Curt	217559.2782
4	Zirkind, Kathy	216935.002
5	Genest, Amber	216158.2015
6	Riesgraf, Vicente	204870.7698
7	Ruddock, Curt	199008.2086
8	Treakle, Hong	197746.5546
9	Thrasher, Elias	191514.7063
10	Geldmacher, Vonda	188271.652
11	Ocus, Lenard	187660.7944
12	Geldmacher, Vonda	185964.2411
13	Treakle, Hong	179307.2147

) | KEYS\Student (56) | CarDealer | 00:00:00 | 170 rows

1. The JOIN operations in the FROM clause are performed, producing a result set containing all sales people who have ever sold a car, along with their names and the sales amount of the car sold.
2. The WHERE clause is processed, and only sales in the year 2011 made by currently active sales people are kept in the result set.
3. The SELECT list is processed and a column alias is assigned to the string concatenation expression producing the full name of the sales person.
4. Finally, the results are sorted by the sales amount column and the result set is returned.

Additional Reading

SQL Joins. A W3 Schools tutorial on SQL Join operations. https://www.w3schools.com/sql/sql_join.asp.

Joins (SQL Server). Microsoft documentation on Joins. <https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins?view=sql-server-ver16>