

# Module 4: Querying Multiple Tables

## Contents:

### Module Overview

**Lesson 1: Understanding Joins**

**Lesson 2: Querying with Inner Joins**

**Lesson 3: Querying with Outer Joins**

**Lesson 4: Querying with Cross Joins and Self Joins**

**Lab: Querying Multiple Tables**

### Module Review and Takeaways

## Module Overview

In real-world environments, it is likely that the data you need to query is stored in multiple locations. You have already learned how to write basic single-table queries. In this module, you will learn how to write queries that combine data from multiple sources in Microsoft® SQL Server®. You will write queries containing joins, which allow you to retrieve data from two (or more) tables, based on data relationships between the tables.

In this module, you will learn how to write queries that combine data from multiple sources in Microsoft SQL Server.

## Objectives

After completing this module, you will be able to:

- Describe how multiple tables may be queried in a SELECT statement using joins.
- Write queries that use inner joins.
- Write queries that use outer joins.
- Write queries that use self joins and cross joins.

## Lesson 1: Understanding Joins

In this lesson, you will learn the fundamentals of joins in SQL Server. You will discover how the FROM clause in a T-SQL SELECT statement creates intermediate virtual tables that will be consumed by subsequent phases of the query. You will learn how an unrestricted combination of rows from two tables yields a Cartesian product. This module also covers the common join types in T-SQL multitable queries.

### The FROM Clause and Virtual Tables

- FROM clause determines source tables to be used in SELECT statement
- FROM clause can contain tables and operators
- Result set of FROM clause is virtual table
  - Subsequent logical operations in SELECT statement consume this virtual table
- FROM clause can establish table aliases for use by subsequent phases of query

You have already learned about the logical order of operations performed when SQL Server processes a query. You will recall that the FROM clause of a SELECT statement is the first phase to be processed. This clause determines which table or

tables will be the source of rows for the query. As you will see in this module, this holds true whether you are querying a single table or bringing together multiple tables as the source of your query. To learn about the additional capabilities of the FROM clause, it is useful to think of the clause function as creating and populating a virtual table. This virtual table will hold the output of the FROM clause and be used subsequently by other phases of the SELECT statement, such as the WHERE clause. As you add extra functionality, such as join operators, to a FROM clause, it will be helpful to think of the purpose of the FROM clause elements as either to add rows to, or remove rows from, the virtual table.

**Reader Aid:** The virtual table created by a FROM clause is a logical entity only. In SQL Server, no physical table is created, whether persistent or temporary, to hold the results of the FROM clause, as it is passed to the WHERE clause or other subsequent phases.

## **SELECT Syntax**

```
SELECT ...  
FROM <table> AS <alias>;
```

You have learned that the FROM clause is processed first, and as a result, any table aliases you create there may be referenced in the SELECT clause. You will see numerous examples of table aliases in this module. While these aliases are optional, except in the case of self-join queries, you will quickly see how they can be a convenient tool when writing queries. Compare the following two queries, which have the same output but differ in their use of aliases. (Note that the examples use a JOIN clause, which will be covered later in this module.)

### **Without Table Aliases**

```
USE TSQL ;  
GO  
SELECT sales.Orders.orderid, sales.Orders.orderdate,
```

```
Sales.OrderDetails.productid,Sales.OrderDetails.unitprice,  
    Sales.OrderDetails.qty  
FROM Sales.Orders  
JOIN Sales.OrderDetails ON Sales.Orders.orderid =  
Sales.OrderDetails.orderid ;
```

### With Table Aliases

```
USE TSQL ;  
GO  
SELECT o.orderid, o.orderdate,  
        od.productid, od.unitprice,  
        od.qty  
FROM Sales.Orders AS o  
JOIN Sales.OrderDetails AS od ON o.orderid = od.orderid ;
```

As you can see, the use of table aliases improves the readability of the query, without affecting the performance. It is strongly recommended that you use table aliases in your multitable queries.

**Reader Aid:** Once a table has been designated with an alias in the FROM clause, it is best practice to use the alias when referring to columns from that table in other clauses.

## Join Terminology: Cartesian Product

- Characteristics of a Cartesian product
  - Output or intermediate result of FROM clause
  - Combine all possible combinations of two sets
  - In T-SQL queries, usually not desired
    - Special case: table of numbers

Name	Product		Name	Product
Davis	Alice Mutton		Davis	Alice Mutton
Funk	Crab Meat		Davis	Crab Meat
King	Ipoh Coffee		Davis	Ipoh Coffee
			Funk	Alice Mutton
			Funk	Crab Meat
			Funk	Ipoh Coffee
			King	Alice Mutton
			King	Crab Meat
			King	Ipoh Coffee

When learning about writing multitable queries in T-SQL, it is important to understand the concept of Cartesian products. In mathematics, this is the product of two sets. The product of a set of two items and a set of six is a set of 12 items—or 6 x 2. In databases, a Cartesian product is the result of joining every row of one input table to all rows of another input table. The product of a table with 10 rows and a table with 100 rows is a result set with 1,000 rows. For most T-SQL queries, a Cartesian product is not the desired outcome. Typically, a Cartesian product occurs when two input tables are joined without considering any logical relationships between them. With no information about relationships, the SQL Server query processor will output all possible combinations of rows. While this can have some practical applications, such as creating a table of numbers or generating test data, it is not typically useful and can have severe performance effects. You will learn a useful application of Cartesian joins later in this module.

**Reader Aid:** In the next topic, you will compare two different methods for specifying the syntax of a join. You will see that one method may lead you toward writing accidental Cartesian product queries.

## Overview of Join Types

- Join types in FROM clauses specify the operations performed on the virtual table:

Join Type	Description
Cross	Combines all rows in both tables (creates Cartesian product)
Inner	Starts with Cartesian product; applies filter to match rows between tables based on predicate
Outer	Starts with Cartesian product; all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders

To populate the virtual table produced by the FROM clause in a SELECT statement, SQL Server uses join operators. These add or remove rows from the virtual table, before it is handed off to subsequent logical phases of the SELECT statement:

- A cross join operator (**CROSS JOIN**) adds all possible combinations of the two input tables' rows to the virtual table. Any filtering of the rows will happen in a WHERE clause. For most querying purposes, this operator is to be avoided.
- An inner join operator (**INNER JOIN**, or just **JOIN**) first creates a Cartesian product, and then filters the results using the predicate supplied in the ON clause, removing any rows from the virtual table that do not satisfy the predicate. The inner join is a very common type of join for retrieving rows with attributes that match across tables, such as matching Customers to Orders by a common custid.
- An outer join operator (**LEFT OUTER JOIN**, **RIGHT OUTER JOIN**, **FULL OUTER JOIN**) first creates a Cartesian product, and like an inner join, filters the results to find rows that match in each table. However, all rows from one table are preserved, and added back to the virtual table after the initial filter is applied. NULLs are placed on attributes where no matching values are found.

**Reader Aid:** Unless otherwise qualified with CROSS or OUTER, the JOIN operator defaults to an INNER join.

## T-SQL Syntax Choices

- ANSI SQL-92

- Tables joined by JOIN operator in FROM Clause

```
SELECT ...  
FROM Table1 JOIN Table2  
ON <on_predicate>
```

- ANSI SQL-89

- Tables joined by commas in FROM Clause
  - Not recommended: accidental Cartesian products!

```
SELECT ...  
FROM Table1, Table2  
WHERE <where_predicate>
```

Throughout the history of SQL Server, the product has changed to keep pace with variations in the American National Standards Institute (ANSI) standards for the SQL language. One of the most notable places where these changes are visible is in the syntax for the join operator in a FROM clause. In ANSI SQL-89, no ON operator was defined. Joins were represented in a comma-separated list of tables, and any filtering, such as for an inner join, was performed in the WHERE clause. This syntax is still supported by SQL Server, but due to the complexity of representing the filters for an outer join in the WHERE clause, in addition to any other filtering, it is not recommended here. Additionally, if a WHERE clause is accidentally omitted, ANSI SQL-89-style joins can easily become Cartesian products and cause performance problems.

### ***Cartesian Product***

```
USE TSQL;
```

```
GO
```

```
/* This is ANSI SQL-89 syntax for an inner join, with the filtering
performed in the WHERE clause. */
SELECT c.companyname, o.orderdate
FROM Sales.Customers AS c, Sales.Orders AS o
WHERE c.custid = o.custid;
....
(830 row(s) affected)
```

```
/*
```

This is ANSI SQL-89 syntax for an inner join, omitting the WHERE clause and causing an inadvertent Cartesian join.

```
*/
```

```
SELECT c.companyname, o.orderdate
FROM Sales.Customers AS c, Sales.Orders AS o;
...
(75530 row(s) affected)
```

With the advent of the ANSI SQL-92 standard, support for the ON clause was added. T-SQL also supports this syntax. Joins are represented in the FROM clause by using the appropriate JOIN operator. The logical relationship between the tables, which becomes a filter predicate, is represented with the ON clause.

## JOIN Clause

```
SELECT c.companyname, o.orderdate
FROM Sales.Customers AS c JOIN Sales.Orders AS o
ON c.custid = o.custid;
```

**Reader Aid:** The ANSI SQL-92 syntax makes it more difficult to create accidental Cartesian joins. Once the keyword JOIN has been added, a syntax error will be raised if an ON clause is missing.



# Demonstration: Understanding Joins

In this demonstration, you will see how to use joins.

## Demonstration Steps

### Use Joins

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod04\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, and then press Enter. When the script has completed, press any key.
5. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
6. Open the **Demo.ssmssIn** solution in the **D:\Demofiles\Mod04\Demo** folder.
7. In Solution Explorer, expand **Queries**, and then double-click the **11 - Demonstration A.sql** script file.
8. Select the code under the comment **Step 1**, and then click **Execute**.
9. Select the code under the comment **Step 2**, and then click **Execute**.
10. Select the code under the comment **Step 3**, and then click **Execute**.
11. Select the code under the comment **Step 4**, and then click **Execute**.
12. Select the code under the comment **Step 5**, and then click **Execute**. Note the error message.
13. Keep SQL Server Management Studio open for the next demonstration.

## Check Your Knowledge

## Select the best answer

You have the following T-SQL query:

```
SELECT o.ID AS OrderID, o.CustomerName, p.ProductName, p.ModelNumber,  
FROM Sales.Orders AS o  
JOIN Sales.Products AS p  
ON o.ProductID = p.ID;
```

Which of the following types of join will the query perform?

A cross join

An inner join

An outer left join

An outer right join

[Check answer](#)

[Show solution](#)

[Reset](#)

## Lesson 2: Querying with Inner Joins

In this lesson, you will learn how to write inner join queries, the most common type of multitable query in a business environment. By expressing a logical relationship between the tables, you will retrieve only those rows with matching attributes present in both.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe inner joins.
- Write queries using inner joins.
- Describe the syntax of an inner join.

### Understanding Inner Joins

- Returns only rows where a match is found in both input tables
- Matches rows based on attributes supplied in predicate
  - ON clause in SQL-92 syntax (preferred)
  - WHERE clause in SQL-89 syntax
- Why filter in ON clause?
  - Logical separation between filtering for purposes of join and filtering results in WHERE
  - Typically no difference to query optimizer
- If join predicate operator is =, also known as equi-join

T-SQL queries that use inner joins are the most common types to solve many business problems, especially in highly normalized database environments. To retrieve data that has been stored across multiple tables, you will often need to reassemble it via inner join queries. As you have learned, an inner join begins its logical processing phase as a Cartesian product, which is then filtered to remove any rows that don't match the predicate.

### **SQL-89 and SQL-92 Join Syntax Compared**

```
--ANSI SQL-89 syntax
SELECT c.companyname, o.orderdate
FROM Sales.Customers AS c, Sales.Orders AS o
WHERE c.custid = o.custid;

--ANSI SQL-92 syntax
SELECT c.companyname, o.orderdate
FROM Sales.Customers AS c JOIN Sales.Orders AS o
ON c.custid = o.custid;
```

From a performance standpoint, you will find that the query optimizer in SQL Server does not favor one syntax over the other. However, as you learn about additional types of joins, especially outer joins, you will likely decide that you prefer to use the SQL-92 syntax and filter in the ON clause. Keeping the join filter logic in the ON clause and leaving other data filtering in the WHERE clause will make your queries easier to read and test.

### **ANSI-92 Join**

```
1) SELECT c.companyname, o.orderdate
2) FROM Sales.Customers AS c
3) JOIN Sales.Orders AS o
4) ON c.custid = o.custid;
```

As you learned earlier, the FROM clause will be processed before the SELECT clause. Let's track the processing, beginning with line 2:

- The FROM clause designates the Sales.Customers table as one of the input tables, giving it the alias of "c".
- The JOIN operator in line 3 reflects the use of an INNER join (the default type in T-SQL) and designates Sales.Orders as the other input table, which has an alias of "o".
- SQL Server will perform a logical Cartesian join on these tables and pass the results to the next phase in the virtual table. (Note that the physical processing of the query may not actually perform the Cartesian product operation, depending on the optimizer's decisions.)
- Using the ON clause, SQL Server will filter the virtual table, retaining only those rows where a custid value from the "c" table (Sales.Customers has been replaced by the alias) matches a custid from the "o" table (Sales.Orders has been replaced by an alias).
- The remaining rows are left in the virtual table and handed off to the next phase in the SELECT statement. In this example, the virtual table is next processed by the SELECT clause, and only two columns are returned to the client application.

- The result? A list of customers who have placed orders. Any customers who have never placed an order have been filtered out by the ON clause, as have any orders that happen to have a customer ID that doesn't correspond to an entry in the customer list.

## Inner Join Syntax

- List tables in FROM Clause separated by JOIN operator
- Table aliases preferred
- Table order does not matter

```
FROM t1 JOIN t2
      ON t1.column = t2.column
```

```
SELECT o.orderid,
       o.orderdate,
       od.productid,
       od.unitprice,
       od.qty
FROM Sales.Orders AS o
     JOIN Sales.OrderDetails AS od
     ON o.orderid = od.orderid;
```

When writing queries using inner joins, consider the following guidelines:

- As you have seen, table aliases are preferred, not only for the SELECT list, but also for expressing the ON clause.
- Inner joins may be performed on a single matching attribute, such as an orderid, or on multiple matching attributes, such as the combination of orderid and productid. Joins that match multiple attributes are called composite joins.
- The order in which tables are listed and joined in the FROM clause does not matter to the SQL Server optimizer. (This will not be the case for OUTER JOIN queries in the next topic.) Conceptually, joins will be evaluated from left to right.
- Use the JOIN keyword once for each two tables in the FROM list. For a two-table

query, specify one join. For a three-table query, you will use JOIN twice—once between the first two tables, and once again between the output of the first two tables and the third table.

## Inner Join Examples

- Join based on single matching attribute

```
SELECT ...
FROM Production.Categories AS C
JOIN Production.Products AS P
ON C.categoryid = P.categoryid;
```

- Join based on multiple matching attributes (composite join)

```
-- List cities and countries where both --
-- customers and employees live
SELECT DISTINCT e.city, e.country
FROM Sales.Customers AS c
JOIN HR.Employees AS e
ON c.city = e.city AND
c.country = e.country;
```

The following are some examples of inner joins:

### Inner Join Example

```
SELECT c.categoryid, c.categoryname, p.productid, p.productname
FROM Production.Categories AS c
JOIN Production.Products AS p
ON c.categoryid = p.categoryid;
```

### Inner Join Example

```
SELECT DISTINCT e.city, e.country
FROM Sales.Customers AS c
```

```
FROM Sales.Customers AS c
JOIN HR.Employees AS e
ON c.city = e.city AND c.country = e.country;
```

**Reader Aid:** The demonstration code for this lesson also uses the `DISTINCT` operator to filter duplicates.

This next example shows how an inner join may be extended to include more than two tables. Note that the `Sales.OrderDetails` table is joined not to the `Sales.Orders` table, but to the output of the `JOIN` between `Sales.Customers` and `Sales.Orders`. Each instance of `JOIN ... ON` performs its own population and filtering of the virtual output table. The SQL Server query optimizer determines the order in which the joins and filtering will be performed.

### Inner Join Example

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate,
od.productid, od.qty
FROM Sales.Customers AS c

JOIN Sales.Orders AS o
ON c.custid = o.custid
JOIN Sales.OrderDetails AS od
ON o.orderid = od.orderid;
```

## Demonstration: Querying with Inner Joins

In this demonstration, you will see how to use inner joins.

### Demonstration Steps

#### Use Inner Joins

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.

2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Select the code under the comment **Step 6**, and then click **Execute**.
8. Keep SQL Server Management Studio open for the next demonstration.

## Check Your Knowledge

### Discovery

You have the following T-SQL query:

```
SELECT HumanResources.Employees.ID, HumanResources.Employers.ID AS  
CompanyID,
```

```
HumanResources.Employees.Name, HumanResources.Employers.Name AS  
CompanyName
```

```
FROM HumanResources.Employees
```

```
JOIN HumanResources.Employers
```

```
ON HumanResources.Employees.EmployerID = HumanResources.Employers.ID;
```

How can you improve the readability of this query?

Show solution

Reset

## Lesson 3: Querying with Outer Joins

In this lesson, you will learn how to write queries that use outer joins. While not as common as inner joins, the use of outer joins in a multitable query can provide an alternative view of your business data. As with inner joins, you will express a logical relationship between the tables. However, you will retrieve not only rows with matching attributes, but also all rows present in one of the tables, whether or not there is a match in the other table.



## Lesson Objectives

After completing this lesson, you will be able to:

- Understand the purpose and function of outer joins.
- Write queries using outer joins.
- Combine an OUTER JOIN operator in a FROM clause with a nullability test in a WHERE clause to reveal nonmatching rows.

## Understanding Outer Joins

- Returns all rows from one table and any matching rows from second table
- One table's rows are "preserved"
  - Designated with LEFT, RIGHT, FULL keyword
  - All rows from preserved table output to result set
- Matches from other table retrieved
- Additional rows added to results for nonmatched rows
  - NULLs added in places where attributes do not match
- Example: return all customers and, for those who have placed orders, return order information; customers without matching orders will display NULL for order details

In the previous lesson, you learned how to use inner joins to match rows in separate tables. As you saw, SQL Server built the results of an inner join query by filtering out rows that failed to meet the conditions expressed in the ON clause predicate. The result is that only rows that matched from both tables were displayed. With an outer join, you may choose to display all the rows from one table, along with those that match from the second table. Let's look at an example, then explore the process.

### Inner Join

```

USE Adventureworks;
GO
SELECT c.CustomerID, soh.SalesOrderID
FROM Sales.Customer AS c JOIN Sales.SalesOrderHeader AS soh
ON c.CustomerID = soh.CustomerID
--(31465 row(s) affected)

```

Note that this example uses the AdventureWorks2016 database for these samples. When written as an inner join, the query returns 31,465 rows. These rows represent a match between customers and orders. Only those CustomerIDs that are in both tables will appear in the results. Only customers who have placed orders will be returned.

### Outer Left Join

```

USE Adventureworks;
GO

SELECT c.CustomerID, soh.SalesOrderID
FROM Sales.Customer AS c LEFT OUTER JOIN Sales.SalesOrderHeader AS
soh
ON c.CustomerID = soh.CustomerID
--(32166 row(s) affected)

```

This example uses a LEFT OUTER JOIN operator which, as you will learn, directs the query processor to preserve all rows from the table on the left (Sales.Customer) and displays the SalesOrderID values for matching rows in Sales.SalesOrderHeader. However, there are more rows returned in this example. All customers are returned, whether or not they have placed an order. As you will see in this lesson, an outer join

will display all the rows from one side of the join or another, whether or not they match.

What does an outer join query display in columns where there was no match? In this example, there are no matching orders for 701 customers. In place of the SalesOrderID column, SQL Server will output NULL where values are otherwise missing.

## Outer Join Syntax

- Return all rows from first table, only matches from second:

```
FROM t1 LEFT OUTER JOIN t2 ON  
t1.col = t2.col
```

- Return all rows from second table, only matches from first:

```
FROM t1 RIGHT OUTER JOIN t2 ON  
t1.col = t2.col
```

- Return only rows from first table, with no match in second:

```
FROM t1 LEFT OUTER JOIN t2 ON  
t1.col = t2.col  
WHERE t2.col IS NULL
```

When writing queries using outer joins, consider the following guidelines:

- As you have seen, table aliases are preferred not only for the SELECT list, but also for expressing the ON clause.
- Outer joins are expressed using the keywords LEFT, RIGHT, or FULL preceding OUTER JOIN. The purpose of the keyword is to indicate which table (on which side of the keyword JOIN) should be preserved and have all its rows displayed, match or no match.
- As with inner joins, outer joins may be performed on a single matching attribute, such as an orderid, or on multiple matching attributes, such as orderid and

productid.

- Unlike inner joins, the order in which tables are listed and joined in the FROM clause does matter, as it will determine whether you choose LEFT or RIGHT for your join.
- Multitable joins are more complex when an OUTER JOIN is present. The presence of NULLs in the results of an outer join may cause issues if the intermediate results are then joined, via an inner join, to a third table. Rows with NULLs may be filtered out by the second join's predicate.
- To display only rows where no match exists, add a test for NULL in a WHERE clause following an OUTER JOIN predicate.

## Outer Join Examples

- All customers with order details if present:

```
SELECT c.custid, c.contactname, o.orderid,
       o.orderdate
FROM Sales.Customers AS C
LEFT OUTER JOIN Sales.Orders AS O
  ON c.custid = o.custid;
```

- Customers who did not place orders:

```
SELECT c.custid, c.contactname, o.orderid,
       o.orderdate
FROM Sales.Customers AS C LEFT OUTER JOIN
Sales.Orders AS O
  ON c.custid = o.custid
WHERE o.orderid IS NULL;
```

The following are some examples of outer joins:

### Outer Join Example

```
USE TSQL;
```

GO

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate  
FROM sales.Customers AS c
```

```
LEFT OUTER JOIN sales.Orders AS o  
ON c.custid =o.custid;
```

### Outer Join Example

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate  
FROM sales.Customers AS c  
LEFT OUTER JOIN sales.Orders AS o  
ON c.custid =o.custid  
WHERE o.orderid IS NULL;
```

## Demonstration: Querying with Outer Joins

In this demonstration, you will see how to use outer joins.

### Demonstration Steps

#### Use Outer Joins

1. In Solution Explorer, open the **31 - Demonstration C.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Select the code under the comment **Step 6**, and then click **Execute**.
8. Select the code under the comment **Step 7**, and then click **Execute**.

9. Keep SQL Server Management Studio open for the next demonstration.

## Check Your Knowledge

### Select the best answer

You have a table named PoolCars and a table named Bookings in your ResourcesScheduling database. You want to return all the pool cars for which there are zero bookings. Which of the following queries should you use?

```
SELECT pc.ID, pc.Make, pc.Model, pc.LicensePlate
FROM ResourcesScheduling.PoolCars AS pc, ResourcesScheduling.Bookings AS b
WHERE pc.ID = b.CarID;
```

```
SELECT pc.ID, pc.Make, pc.Model, pc.LicensePlate
FROM ResourcesScheduling.PoolCars AS pc
RIGHT OUTER JOIN ResourcesScheduling.Bookings AS b
ON pc.ID = b.CarID;
```

```
SELECT pc.ID, pc.Make, pc.Model, pc.LicensePlate
FROM ResourcesScheduling.PoolCars AS pc
JOIN ResourcesScheduling.Bookings AS b
ON pc.ID = b.CarID;
```

```
SELECT pc.ID, pc.Make, pc.Model, pc.LicensePlate
FROM ResourcesScheduling.PoolCars AS pc
LEFT OUTER JOIN ResourcesScheduling.Bookings AS b
ON pc.ID = b.CarID
WHERE b.BookingID IS NULL;
```

Check answer Show solution Reset

## Lesson 4: Querying with Cross Joins and Self Joins

In this lesson, you will learn about additional types of joins, which are useful in some more specialized scenarios.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe a use for a cross join.
- Write queries that use cross joins.
- Describe a use for a self join.
- Write queries that use self joins.

## Understanding Cross Joins

- Combine each row from first table with each row from second table
- All possible combinations output
- Logical foundation for inner and outer joins
  - Inner join starts with Cartesian product, adds filter
  - Outer join takes Cartesian output, filtered, adds back nonmatching rows (with NULL placeholders)
- Due to Cartesian product output, not typically a desired form of join
- Some useful exceptions:
  - Table of numbers, generating data for testing

Cross join queries create a Cartesian product that, as you have learned in this module so far, are to be avoided. Although you have seen a means to create one with ANSI SQL-89 syntax, you haven't seen how or why to do so with ANSI SQL-92. This topic will revisit cross joins and Cartesian products.

To explicitly create a Cartesian product, you would use the CROSS JOIN operator.

### **Cross Join**

```
SELECT ...  
FROM table1 AS t1 CROSS JOIN table2 AS t2;
```

While this is not typically a desired output, there are a few practical applications for writing an explicit cross join:

- Creating a table of numbers, with a row for each possible value in a range.
- Generating large volumes of data for testing. When cross joined to itself, a table with as few as 100 rows can readily generate 10,000 output rows with very little work from you.

## Cross Join Syntax

- No matching performed, no ON clause used
- Return all rows from left table combined with each row from right table (ANSI SQL-92 syntax):

```
SELECT ...  
FROM t1 CROSS JOIN t2
```

- Return all rows from left table combined with each row from right table (ANSI SQL-89 syntax):

```
SELECT ...  
FROM t1, t2
```

When writing queries with **CROSS JOIN**, consider the following:

- There is no matching of rows performed, and therefore no ON clause is required.
- To use ANSI SQL-92 syntax, separate the input table names with the **CROSS JOIN** operator.



## Cross Join Examples

- Create test data by returning all combinations of two inputs:

```
SELECT e1.firstname, e2.lastname  
FROM HR.Employees AS e1  
CROSS JOIN HR.Employees AS e2;
```


The following is an example of using CROSS JOIN to create all combinations of two input sets:

### **Cross Join Example**

```
SELECT e1.firstname, e2.lastname  
FROM HR.Employees e1 CROSS JOIN HR.Employees e2;
```

## Understanding Self Joins

- Why use self joins?
  - Compare rows in same table to each other
- Create two instances of same table in FROM clause
  - At least one alias required
- Example: Return all employees and the name of the employee's manager



Employees (HR)	
empid	
lastname	
firstname	
title	
titleofcourtesy	
birthdate	
hiredate	
address	
city	
region	
postalcode	
country	
phone	
mgrid	

So far, the joins you have learned about have involved separate multiple tables. There may be scenarios in which you need to compare and retrieve data stored in the same table. For example, in a classic human resources application, an Employees table might include information about the supervisor of each employee in the employee's own row. Each supervisor is also listed as an employee. To retrieve the employee information and match it to the related supervisor, you can use the table twice in your query, joining it to itself for the purposes of the query.

There are other scenarios in which you will want to compare rows in a table with one another. As you have seen, it's fairly easy to compare columns in the same row using T-SQL, but how to compare values from different rows (such as a row which stores a starting time with another row in the same table that stores a corresponding stop time) is less obvious. Self joins are a useful technique for these types of queries.

To accomplish tasks like this, you should consider the following guidelines:

- Create two instances of the same table in the FROM clause, and join them as needed, using inner or outer joins.
- Use table aliases to create two separate aliases for the same table. At least one of these must have an alias.

- Use the ON clause to provide a filter using separate columns from the same table.

The following example, which you will examine closely in the next topic, illustrates these guidelines:

### Self Join Example

```
SELECT e.empid ,e.lastname AS empname,e.title,e.mgrid, m.lastname AS
mgrname
FROM HR.Employees AS e
JOIN HR.Employees AS m
ON e.mgrid=m.empid;
```

This yields results like the following:

empid	empname	title	mgrid	mgrname
2	Funk	Vice President, Sales	1	Davis
3	Lew	Sales Manager	2	Funk
4	Peled	Sales Representative	3	Lew
5	Buck	Sales Manager	2	Funk
6	Suurs	Sales Representative	5	Buck
7	King	Sales Representative	5	Buck
8	Cameron	Sales Representative	3	Lew
9	Dolgopyatova	Sales Representative	5	Buck

### Self Join Examples

- Return all employees with ID of employee's manager when a manager exists (inner join):

```
SELECT e.empid, e.lastname,
       e.title, e.mgrid, m.lastname
FROM   HR.Employees AS e
JOIN   HR.Employees AS m
ON     e.mgrid=m.empid;
```

- Return all employees with ID of manager (outer join). This will return NULL for the CEO:

```
SELECT e.empid, e.lastname,
       e.title, m.mgrid
FROM   HR.Employees AS e
LEFT OUTER JOIN HR.Employees AS m
ON     e.mgrid=m.empid;
```

The following are some examples of self joins:

### Self Join Example

```
SELECT e.empid ,e.lastname AS empname,e.title,e.mgrid, m.lastname AS
mgrname
FROM HR.Employees AS e
JOIN HR.Employees AS m
ON e.mgrid=m.empid;
```

### Self Join Example

```
SELECT e.empid ,e.lastname AS empname,e.title,e.mgrid, m.lastname AS
mgrname
FROM HR.Employees AS e
LEFT OUTER JOIN HR.Employees AS m
ON e.mgrid=m.empid;
```

# Demonstration: Querying with Cross Joins and Self Joins

In this demonstration, you will see how to use self joins and cross joins.

## Demonstration Steps

### Use Self Joins and Cross Joins

1. In Solution Explorer, open the **41 - Demonstration D.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Close SQL Server Management Studio without saving any files.

## Check Your Knowledge

### Discovery

You have two tables named **FirstNames** and **LastNames**. You want to generate a set of fictitious full names from this data. There are 150 entries in the **FirstNames** table and 250 entries in the **LastNames** table. You use the following query:

```
SELECT (f.Name + ' ' + l.Name) AS FullName
FROM FirstNames AS f
CROSS JOIN LastNames AS l
```

How many fictitious full names will be returned by this query?

Show solution

Reset

## Lab: Querying Multiple Tables

### Scenario

You are an Adventure Works business analyst who will be writing reports using corporate databases stored in SQL Server. You have been given a set of business

requirements for data and you will write T-SQL queries to retrieve the specified data from the databases. You notice that the data is stored in separate tables, so you will need to write queries using various join operations.

## Objectives

After completing this lab, you will be able to:

- Write queries that use inner joins.
- Write queries that use multiple-table inner joins.
- Write queries that use self joins.
- Write queries that use outer joins
- Write queries that use cross joins.

## Lab Setup

Estimated Time: 50 minutes

Virtual machine: **20761C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

## Exercise 1: Writing Queries That Use Inner Joins

---

### Scenario

You no longer need the supplied mapping information between categoryid and categoryname because you now have the Production.Categories table with the needed mapping rows. Write a SELECT statement using an inner join to retrieve the productname column from the Production.Products table and the categoryname column from the Production.Categories table.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Write a SELECT Statement That Uses an Inner Join

Detailed Steps ▼

Detailed Steps ▼

**Result:** After this exercise, you should know how to use an inner join between two tables.

## Exercise 2: Writing Queries That Use Multiple-Table Inner Joins

### Scenario

The sales department would like a report of all customers who placed at least one order, with detailed information about each one. A developer prepared an initial SELECT statement that retrieves the custid and contactname columns from the Sales.Customers table and the orderid column from the Sales.Orders table. You should observe the supplied statement and add additional information from the Sales.OrderDetails table.

The main tasks for this exercise are as follows:

1. Execute the T-SQL Statement
2. Apply the Needed Changes and Execute the T-SQL Statement
3. Change the Table Aliases
4. Add an Additional Table and Columns

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

**Result:** After this exercise, you should have a better understanding of why aliases are important and how to do a multiple-table join.

## Exercise 3: Writing Queries That Use Self Joins

---

### Scenario

The HR department would like a report showing employees and their managers. They want to see the lastname, firstname, and title columns from the HR.Employees table for each employee, and the same columns for the employee's manager.

The main tasks for this exercise are as follows:

1. Write a Basic SELECT Statement
2. Write a Query That Uses a Self Join

Detailed Steps ▼

Detailed Steps ▼

**Result:** After this exercise, you should have an understanding of how to write T-SQL statements that use self joins.

## Exercise 4: Writing Queries That Use Outer Joins

---

### Scenario

The sales department was satisfied with the report you produced in exercise 2. Now sales staff would like to change the report to show all customers, even if they did not



have any orders, and still include order information for the customers who did. You need to write a SELECT statement to retrieve all rows from Sales.Customers (columns custid and contactname) and the orderid column from the table Sales.Orders.

The main task for this exercise is as follows:

- Write a SELECT Statement That Uses an Outer Join

Detailed Steps ▼

**Result:** After this exercise, you should have a basic understanding of how to write T-SQL statements that use outer joins.

## Exercise 5: Writing Queries That Use Cross Joins

---

### Scenario

The HR department would like to prepare a personalized calendar for each employee. The IT department supplied you with T-SQL code that will generate a table with all dates for the current year. Your job is to write a SELECT statement that would return all rows in this new calendar date table for each row in the HR.Employees table.

The main tasks for this exercise are as follows:

1. Execute the T-SQL Statement
2. Write a SELECT Statement That Uses a Cross Join
3. Drop the HR.Calendar Table

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▾

**Result:** After this exercise, you should have an understanding of how to write T-SQL statements that use cross joins.

## Module Review and Takeaways

In this module, you have learned how to:

- Describe how multiple tables may be queried in a SELECT statement using joins.
- Write queries that use inner joins.
- Write queries that use outer joins.
- Write queries that use self joins and cross joins.

**Best Practice:** Table aliases should always be defined when joining tables. Joins should be expressed using SQL-92 syntax, with JOIN and ON keywords.

### Review Question(s)

#### Check Your Knowledge

##### Discovery

How does an inner join differ from an outer join?

Show solution   Reset

#### Check Your Knowledge

##### Discovery

Which join types include a logical Cartesian product?

Show solution   Reset

# Check Your Knowledge

## Discovery

Can a table be joined to itself?

Show solution

Reset

This document belongs to Sebastian Amihaesie.  
sebastian.amihaesie@gmail.com  
No unauthorized copies allowed!

This document belongs to Sebastian Amihaesie.  
sebastian.amihaesie@gmail.com  
No unauthorized copies allowed!

This document belongs to Sebastian Amihaesie.  
sebastian.amihaesie@gmail.com  
No unauthorized copies allowed!

This document