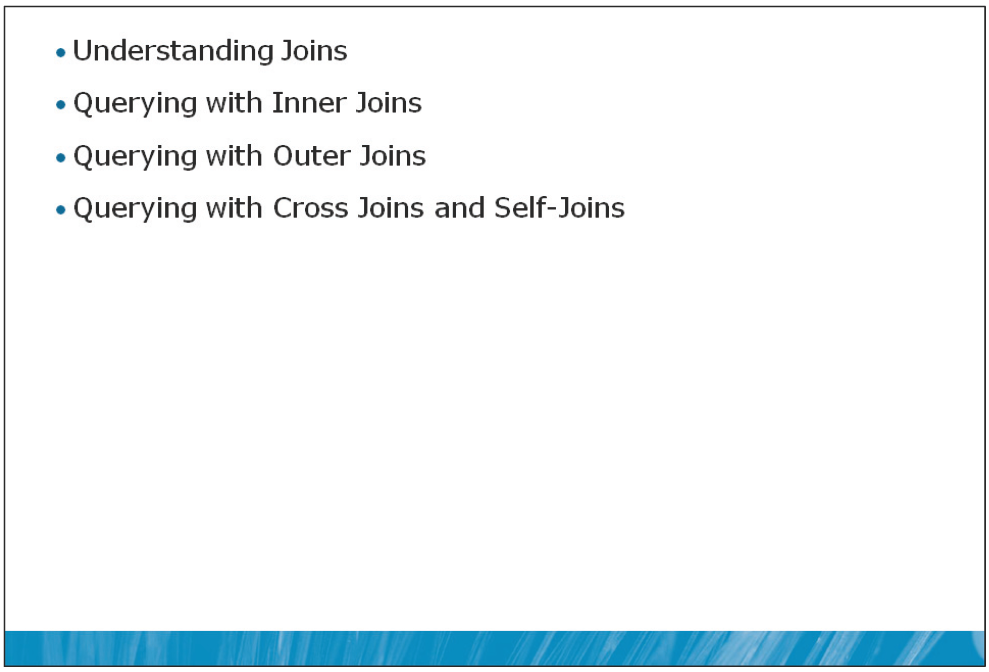
Module 5: Querying Multiple Tables

Contents:

Module Overview



In real-world environments, it is likely that the data you need to query is stored in multiple locations. Earlier, you learned how to write basic single-table queries. In this module, you will learn how to write queries that combine data from multiple sources

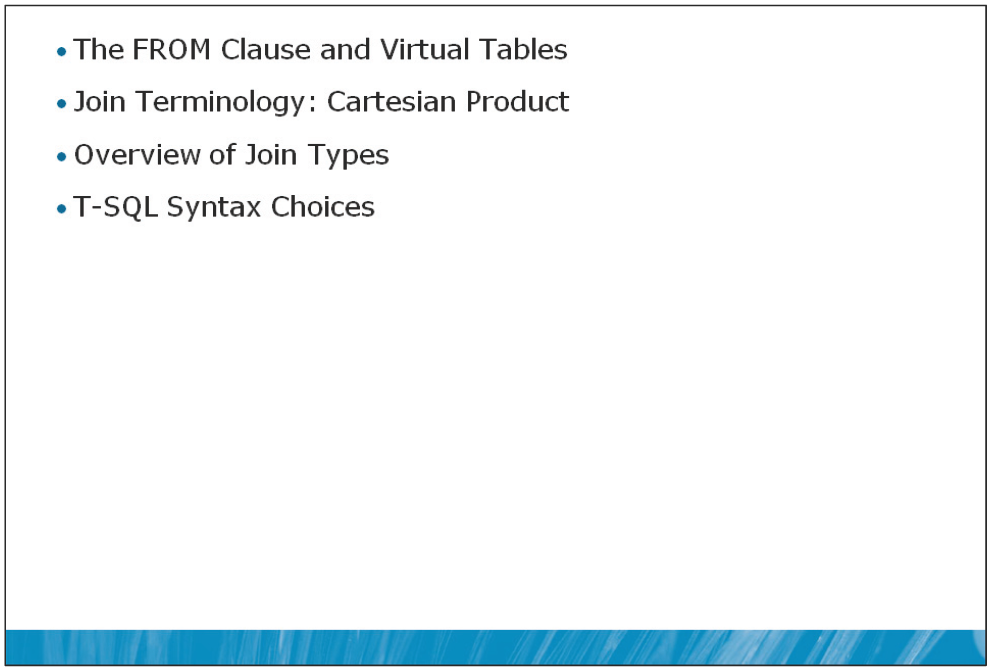
in Microsoft® SQL Server® 2012. You will do so by writing queries containing joins, which allow you to retrieve data from two (or more) tables based on data relationships between the tables.

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 2012. You will learn 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

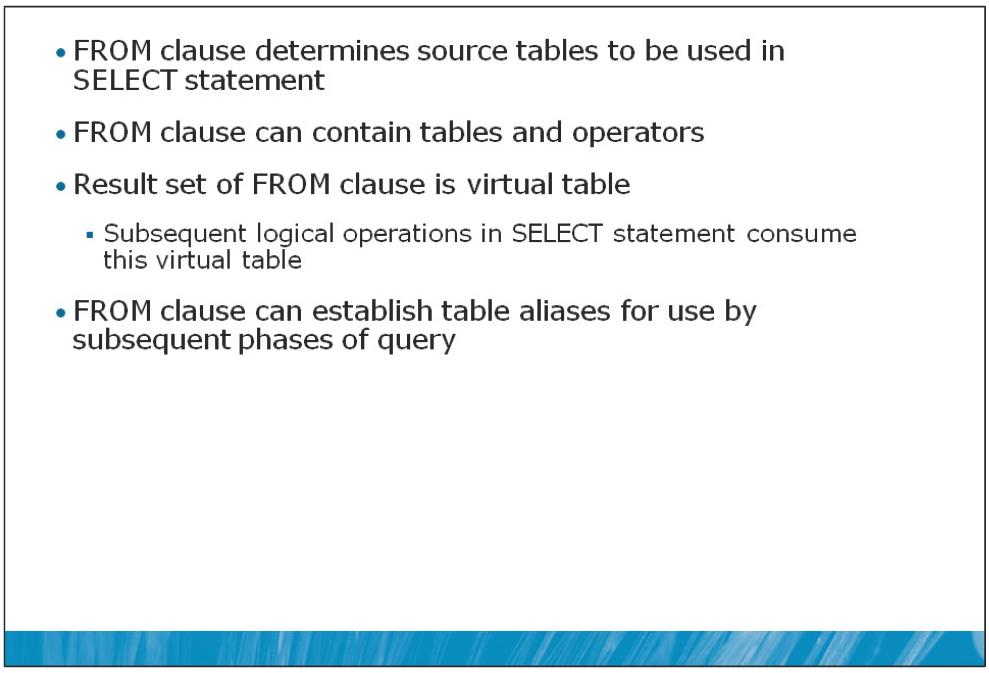
an unrestricted combination of rows from two tables yields a Cartesian product. You will also learn about the common join types in T-SQL multi-table queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the relationship between the FROM Clause and virtual tables in a SELECT statement.
- Describe a Cartesian product and how it may be created by a join.
- Describe the common join types in T-SQL queries.
- Understand the difference between ANSI SQL-89 and SQL-92 join syntax.

The FROM Clause and Virtual Tables



Earlier, you learned about the logical order of operations performed when SQL Server 2012 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 will hold true

whether you are querying a single table or bringing together multiple tables as the source of your query.

In order to learn about the additional capabilities of the FROM clause, it will be useful to think of the function of the clause as creating and populating a virtual table. This virtual table will hold the output of the FROM clause and will be used subsequently by other phases of the SELECT statement, such as the WHERE clause. As you add additional 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.

> **Note** The virtual table created by a FROM clause is a logical entity only. In SQL Server 2012, 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.

The syntax for the SELECT statement that you have used in earlier queries in this course has appeared as follows:

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

You learned earlier 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 they are optional, except in the case of self-join queries, you will quickly see how they can be a convenient tool when writing your queries. Compare the following two queries, which have the same output, but which differ in their use of aliases. (Note that the examples use a JOIN clause, which will be covered later in this module.) The first query uses no table aliases:

```
USE TSQL2012 ; 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 ;
```

The second example retrieves the same data but uses table aliases:

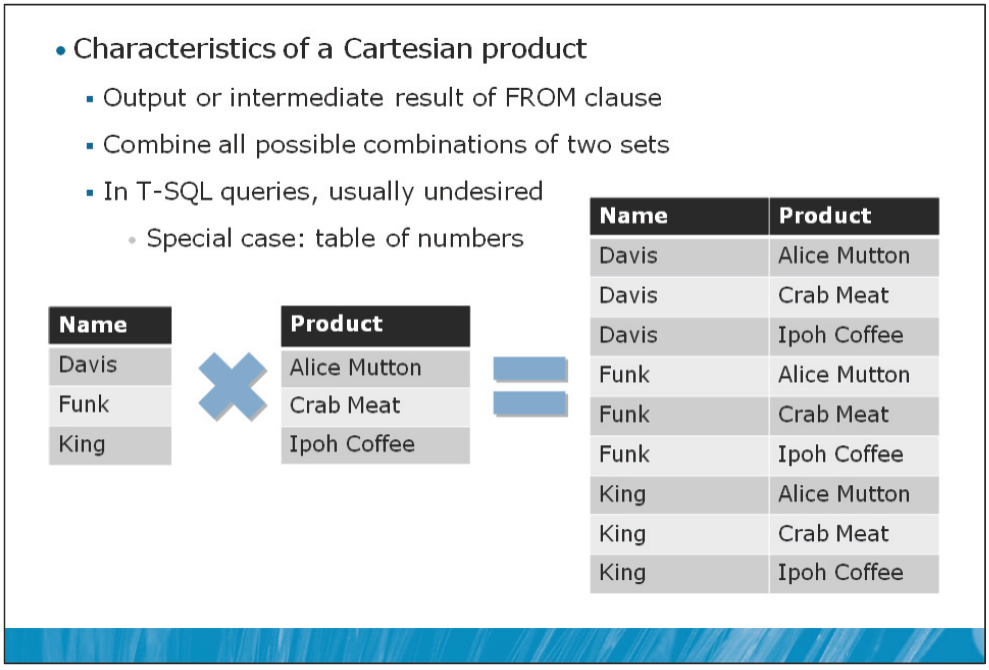
```
USE TSQL2012 ; GO SELECT o.orderid, o.orderdate, od.productid,
od.unitprice, od.qty FROM Sales.Orders AS o JOIN Sales.OrderDetails
AS od ON o.orderid = od.orderid ;
```

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 multi-table queries.

> **Note** Once a table as been designated with an alias in the FROM clause, it is a best

practice to use the alias when referring to columns from that table in other clauses.

Join Terminology: Cartesian Product



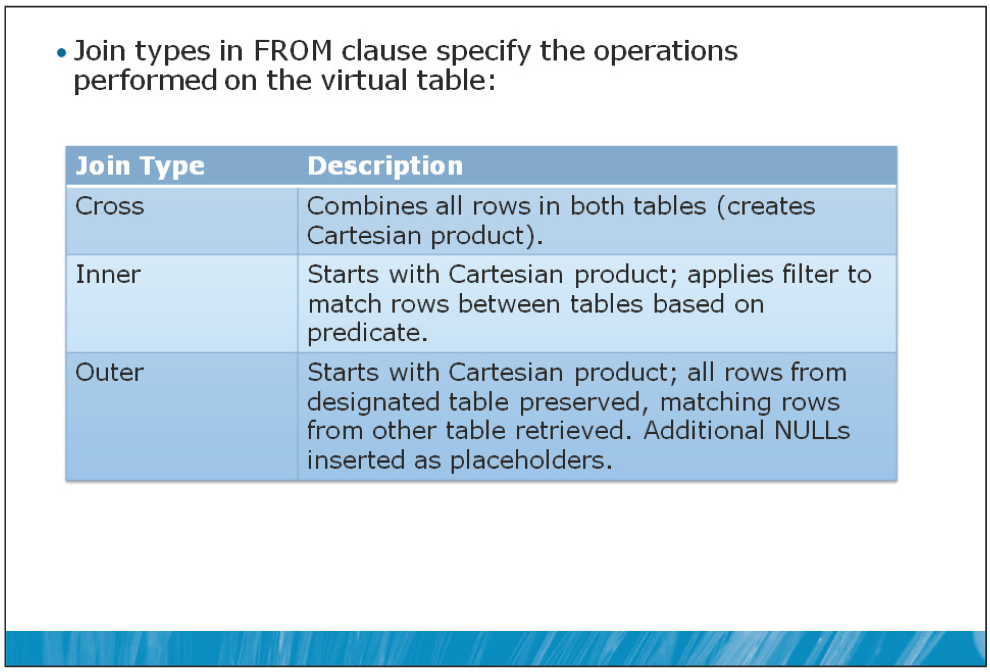
When learning about writing multi-table queries in T-SQL, it is important to understand the concept of Cartesian products. In mathematics, a Cartesian product is the product of two sets. The product of a set of 2 items and a set of 6 items is a set of 12 items, or 6 x 2. In databases, a Cartesian product is the result of joining every row of one input table to every row 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. In the absence of any 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.

> **Note** 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

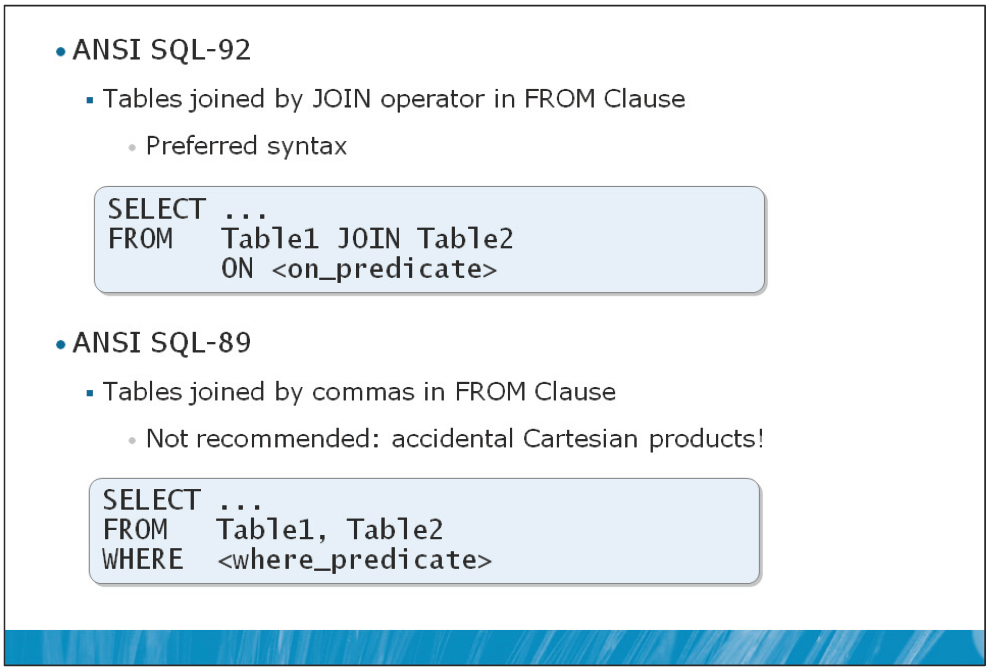


To populate the virtual table produced by the FROM clause in a SELECT statement, SQL Server uses join operators. These operators add or remove rows from the virtual table, before it is handed off to subsequent logical phases of the SELECT statement:

- 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 type 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, added back to the virtual table after the initial filter is applied. NULLs are placed on attributes where no matching values are found.

> **Note** Unless otherwise qualified with CROSS or OUTER, the JOIN operator defaults to an INNER join.

T-SQL Syntax Choices



Through the history of versions of SQL Server, the product has changed to keep pace with changes in the 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 2012, but due to the complexity of representing the filters for an outer join in the WHERE clause, as well as any other filtering, it is not recommended to use this. Additionally, if a WHERE clause is accidentally omitted, ANSI SQL-89-style joins can easily become Cartesian products and cause performance problems. The following queries illustrate this syntax and this potential problem:

```
USE TSQL2012; 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. The following example restates the previous query with the newer syntax:

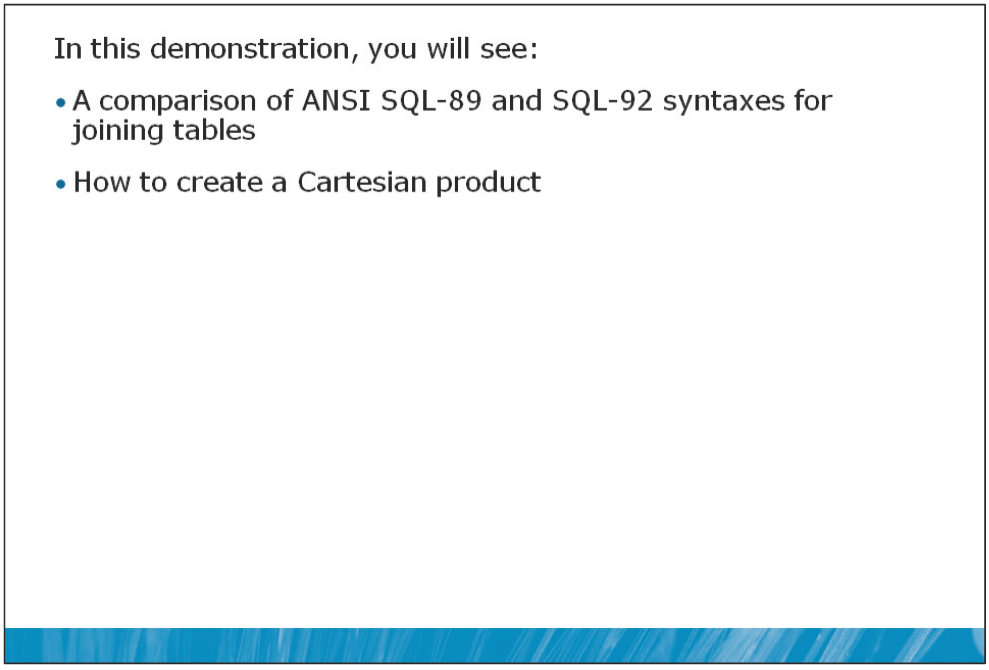
```
SELECT c.companyname, o.orderdate FROM Sales.Customers AS c JOIN
Sales.Orders AS o ON c.custid = o.custid;
```

> **Note** 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



Demonstration Steps

1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**.

   Navigate to F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 11 – Demonstration A.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lesson 2: Querying with Inner Joins

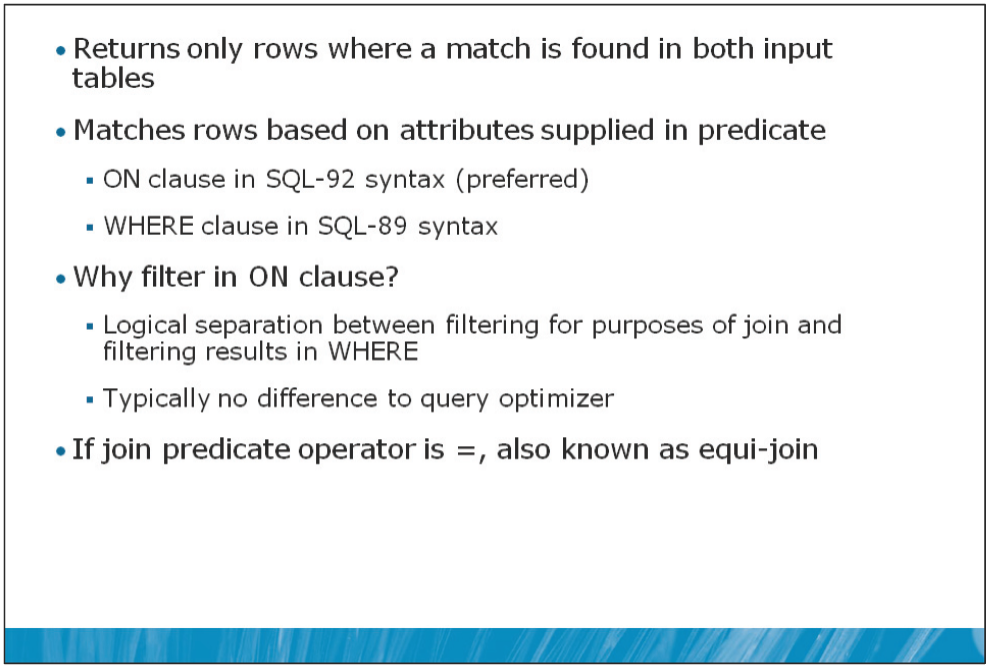
In this lesson, you will learn how to write inner join queries, the most common type of multi-table query in a business environment. By expressing a logical relationship between the tables, you will retrieve only those rows with matching attributes present in both tables.

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



T-SQL queries that use inner joins are the most common types of queries 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 previously 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. In SQL-89 syntax, that predicate is in the WHERE clause. In SQL-92 syntax, that predicate is within the FROM clause in the ON clause:

```
--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 2012 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 easier to test.

Using the ANSI SQL-92 syntax, let's examine the steps by which SQL Server 2012 will logically process this query. (Line numbers are added for clarity and are not submitted to the query engine for execution.)

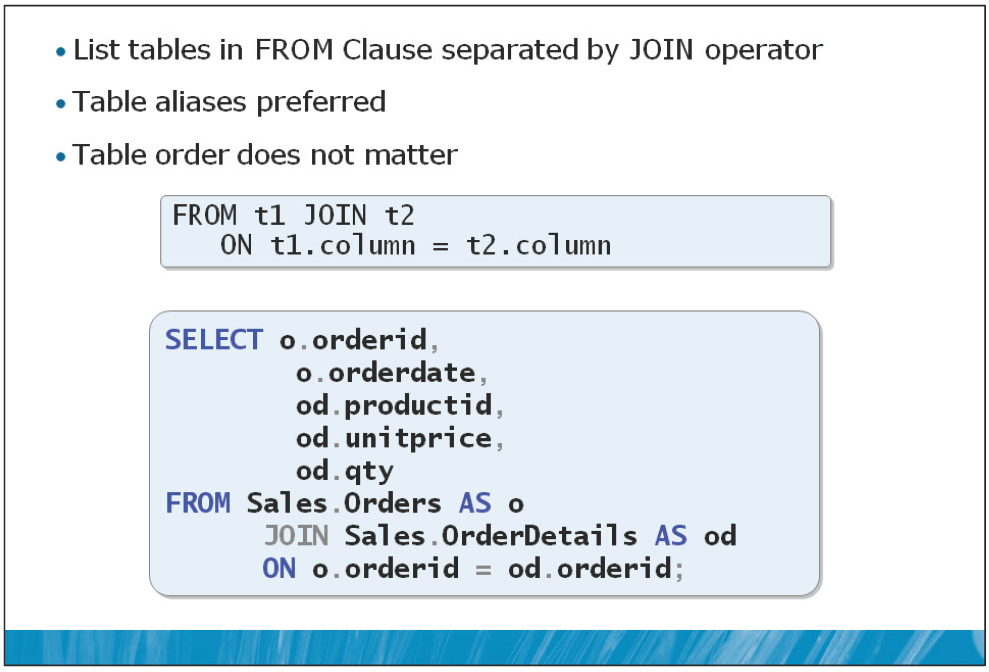
```
1) SELECT c.companyname, o.orderdate 2) FROM Sales.Customers AS c 3)
JOIN Sales.Orders AS o 4) ON c.custid = o.custid;
```

As you learned earlier, the FROM clause will be processed before the SELECT clause. Therefore, 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 the Sales.Orders table 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 p 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 that have placed orders. Any customers that 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

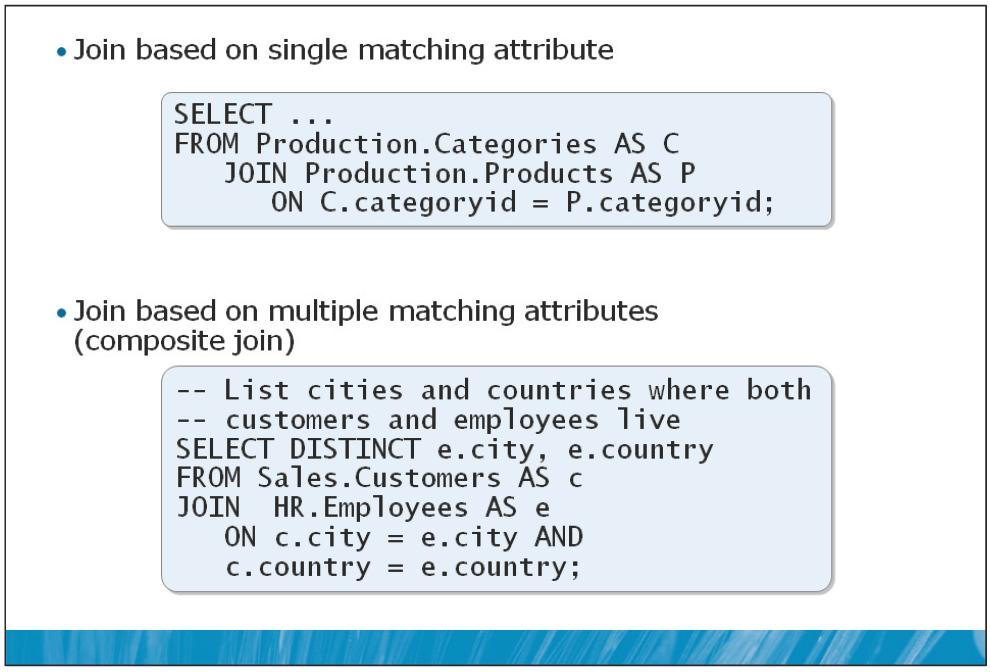


When writing queries using inner joins, consider the following guidelines:

- 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 they may be performed 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 and the third table.

Inner Join Examples



The following are some examples of inner joins:

This query performs a join on a single matching attribute, relating the categoryid from the Production.Categories table to the categoryid from the Production.Products table:

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

This query performs a composite join on two matching attributes, relating city and country attributes from Sales.Customers to HR.Employees. Note the use of the DISTINCT operator to filter out duplicate occurrences of city, country:

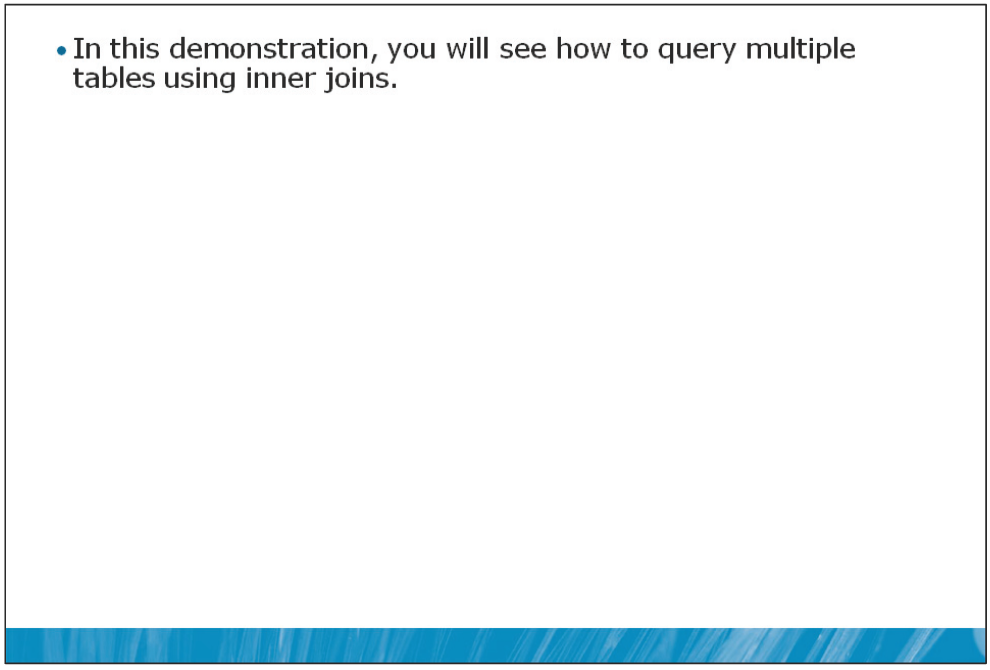
```
SELECT DISTINCT e.city, e.country FROM Sales.Customers AS c JOIN
HR.Employees AS e ON c.city = e.city AND c.country = e.country;
```

> **Note** 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. It is up to the SQL Server query optimizer to decide in which order the joins and filtering will be performed.

```
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



Demonstration Steps

1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**.

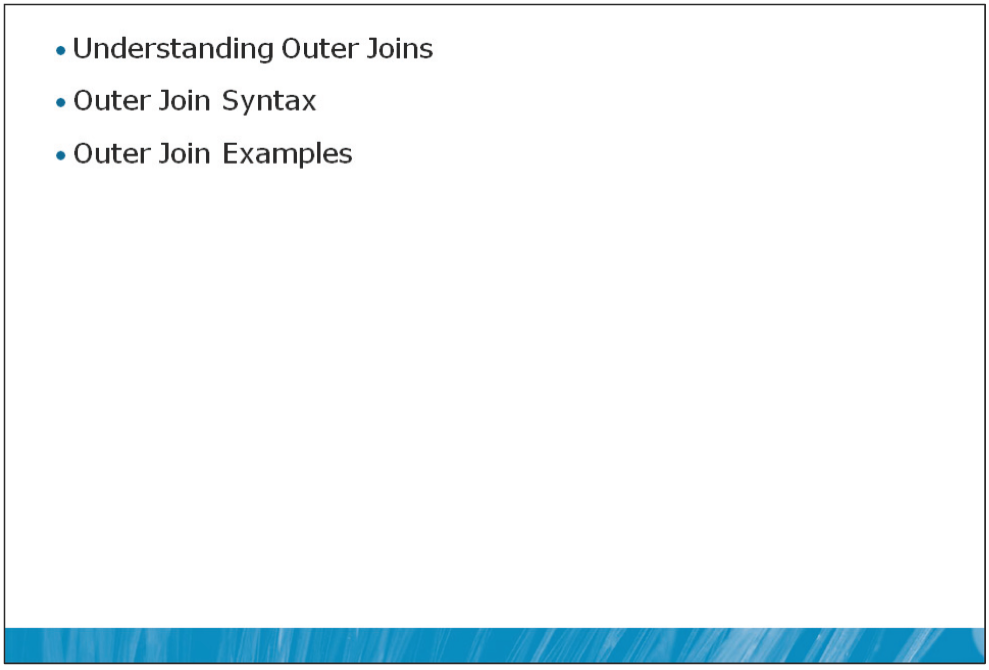
   Navigate to F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 21 – Demonstration B.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lesson 3: Querying with Outer Joins



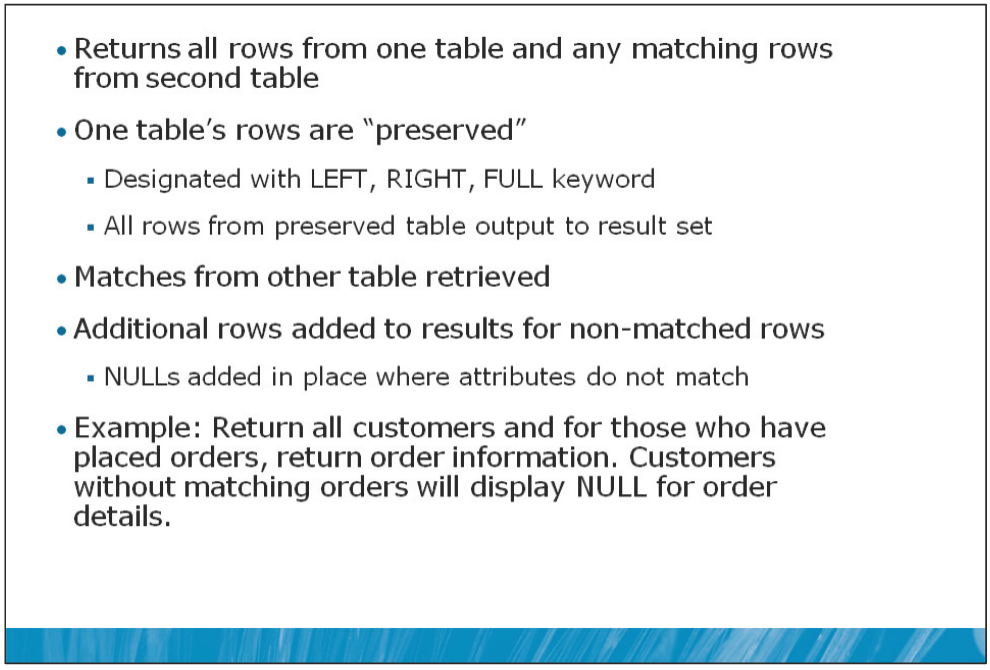
In this lesson, you will learn how to write queries that use outer joins. While not as common as inner joins, the use of outer joins in a multi-table query can provide you with an alternative view of your business data. As with inner joins, you will express a logical relationship between the tables. However, you will retrieve not only rows with matching attributes, but 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.
- Be able to write queries using outer joins.
- Be able to combine an OUTER JOIN operator in a FROM clause with a nullability test in a WHERE clause in order to reveal non-matching rows.

Understanding Outer Joins



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 as well as those that match from the second table. Let's look at an example, then explore the process.

First, let's examine the following query, written as an inner join:

```
USE AdventureWorks2008R2; 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 AdventureWorks2008R2 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 appear in both tables will appear in the results. Only customers that have placed orders will be returned.

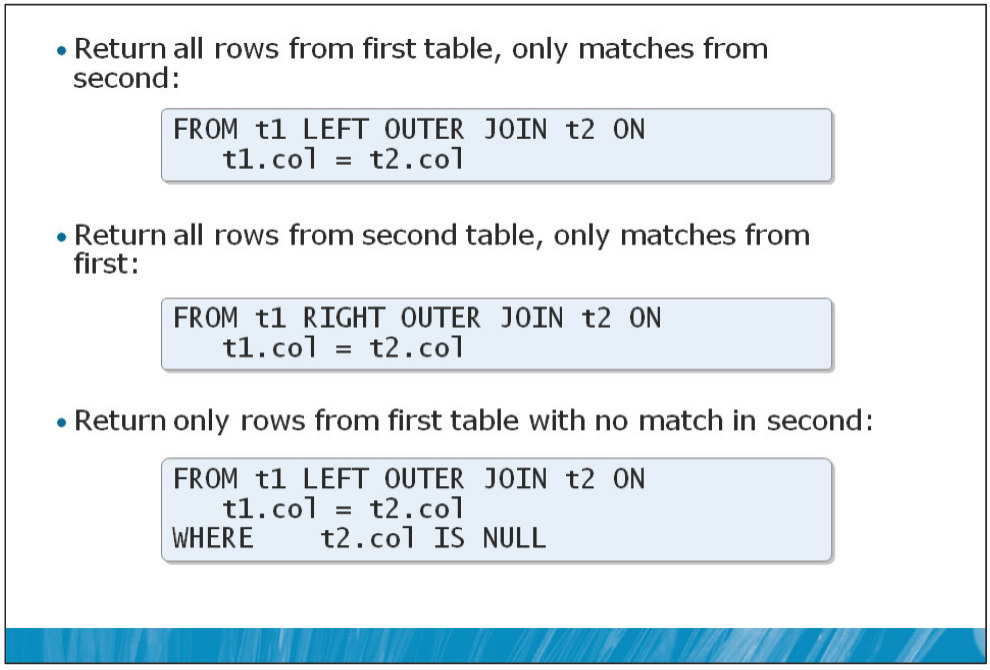
Now, let's examine the following query, written as an outer left join:

```
USE AdventureWorks2008R2; 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 display 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 they match or not.

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 the place of the SalesOrderID column, SQL Server will output NULL where values are otherwise missing.

Outer Join Syntax

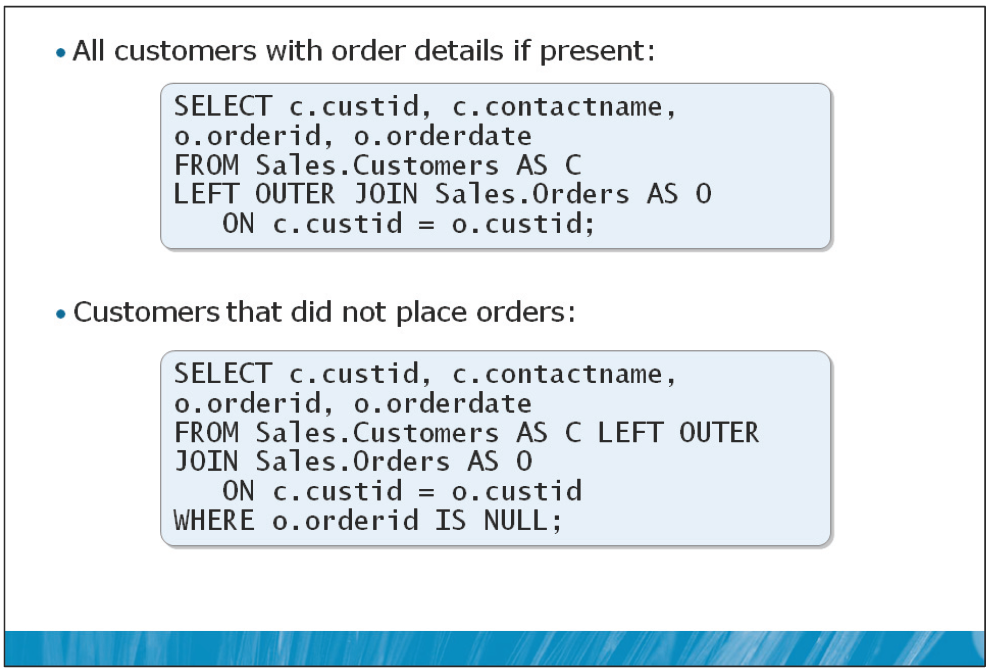


When writing queries using outer joins, consider the following guidelines:

- 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 they may be performed 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.
- Multi-table 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



The following are some examples of outer joins:

This query displays all customers and provides information about each customer's orders if any exist:

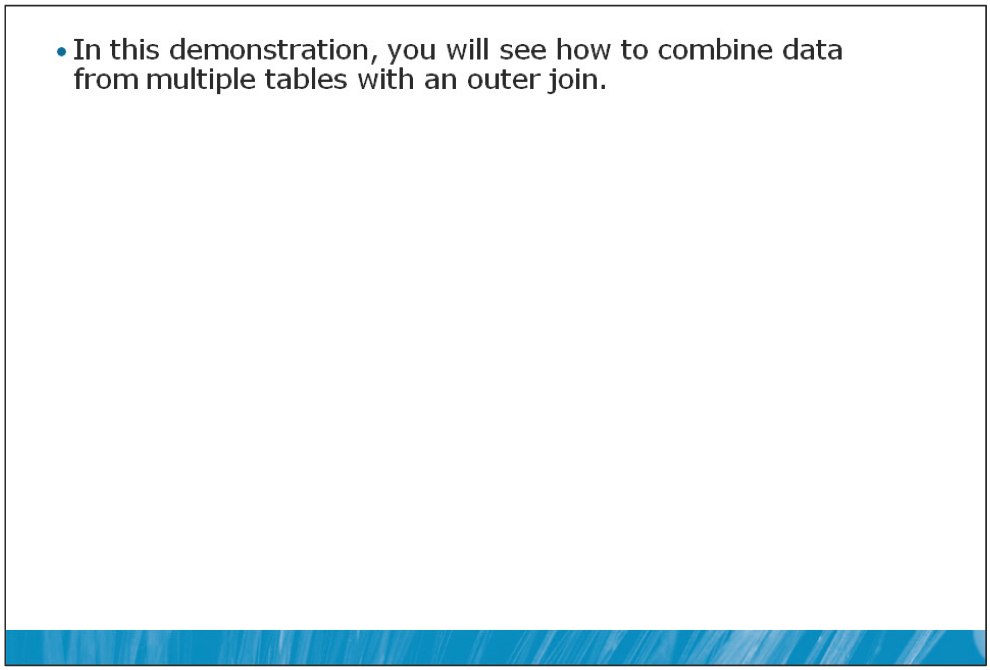
```
USE TSQL2012; GO SELECT c.custid, c.companyname, o.orderid,
o.orderdate FROM Sales.Customers AS c LEFT OUTER JOIN Sales.Orders AS
o ON c.custid =o.custid;
```

This query displays only customers that have never placed an order:

```
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



Demonstration Steps

1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**.

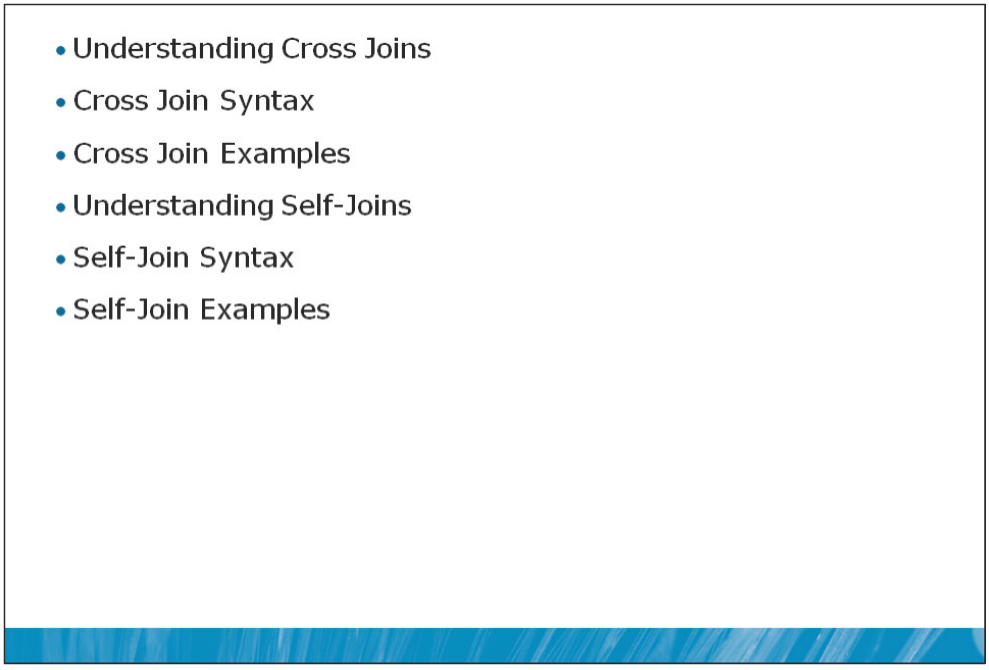
   Navigate to F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 31 – Demonstration C.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lesson 4: Querying with Cross Joins and Self-Joins



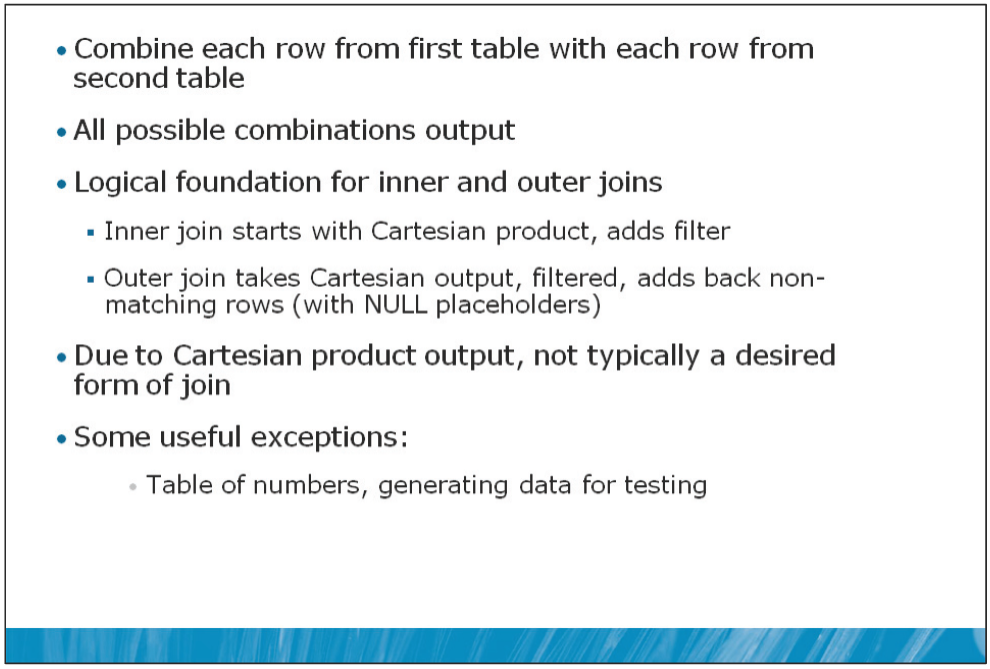
In this lesson, you will learn some additional types of joins, which are useful in some more specialized scenarios.

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



Cross join queries create a Cartesian product. So far in this module, you have learned that Cartesian products 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 create one with ANSI SQL-92. This topic will revisit cross joins and Cartesian products.

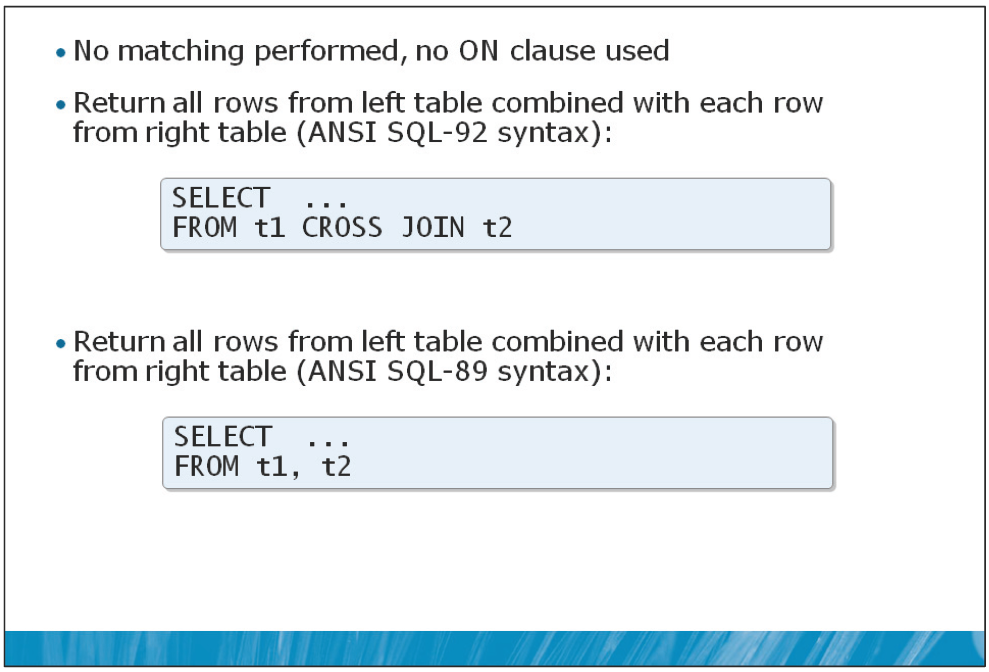
To explicitly create a Cartesian product, you would use the CROSS JOIN operator. This will create a result set with all possible combinations of input rows:

```
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 on your part.

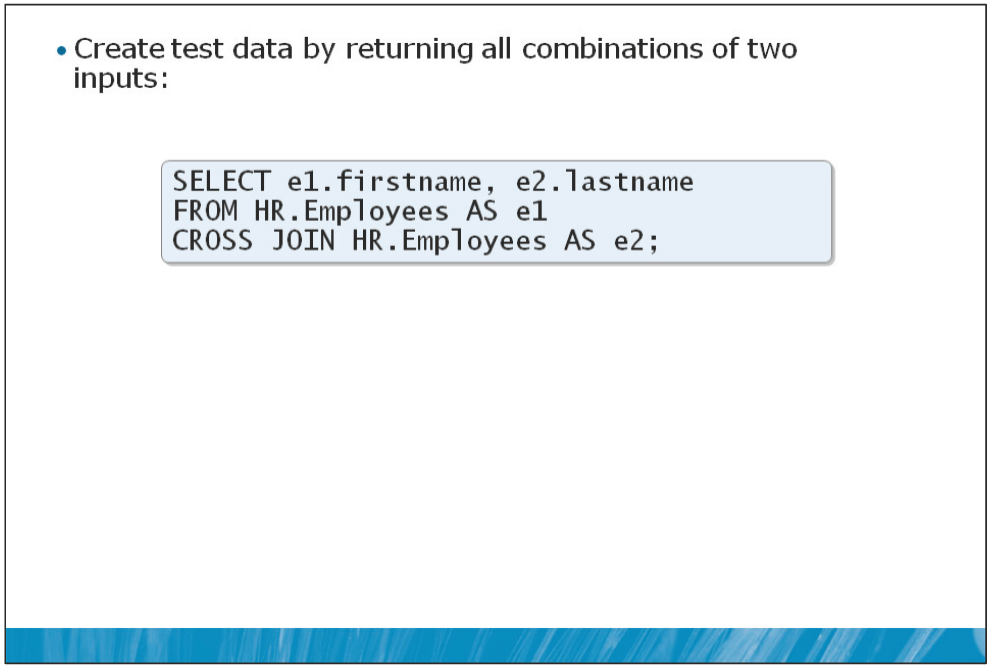
Cross Join Syntax



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.

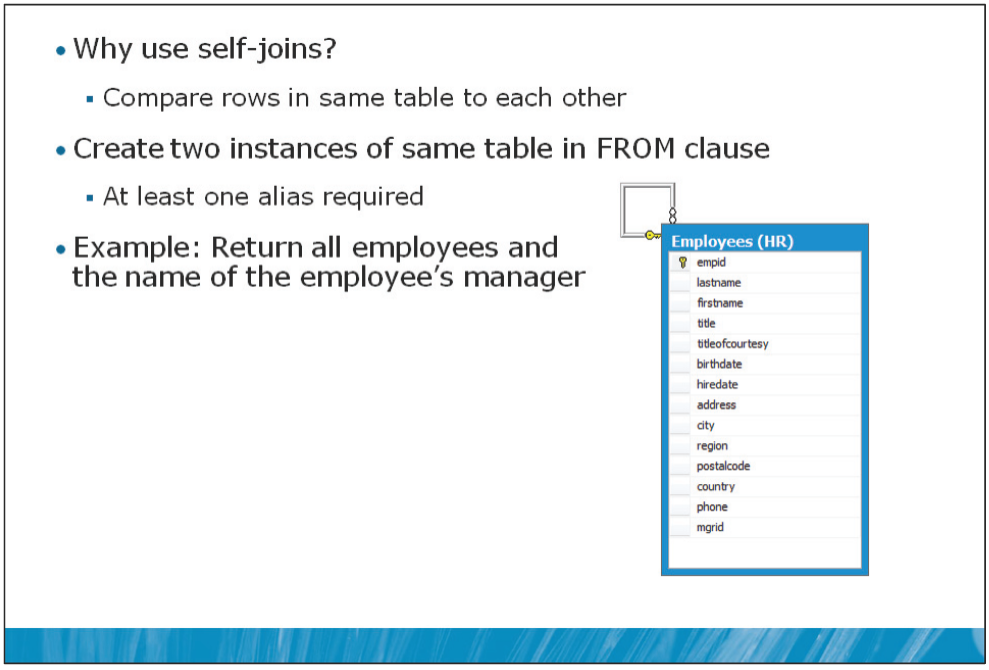
Cross Join Examples



The following is an example of using CROSS JOIN to create all combinations of two input sets. Using the TSQL2012 sample, this will take 9 employee first names and 9 employee last names to generate 81 combinations:

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

Understanding Self-Joins



The joins you have learned about to this point 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 up 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 within a table with one another. As you have seen, it's fairly easy to compare columns within the same row using T-SQL, but it is less obvious 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). Self-joins are a useful technique for these types of queries.

In order to accomplish tasks like this, you will want to 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 table must have an alias.
- Use the ON clause to provide a filter using separate columns from the same table.

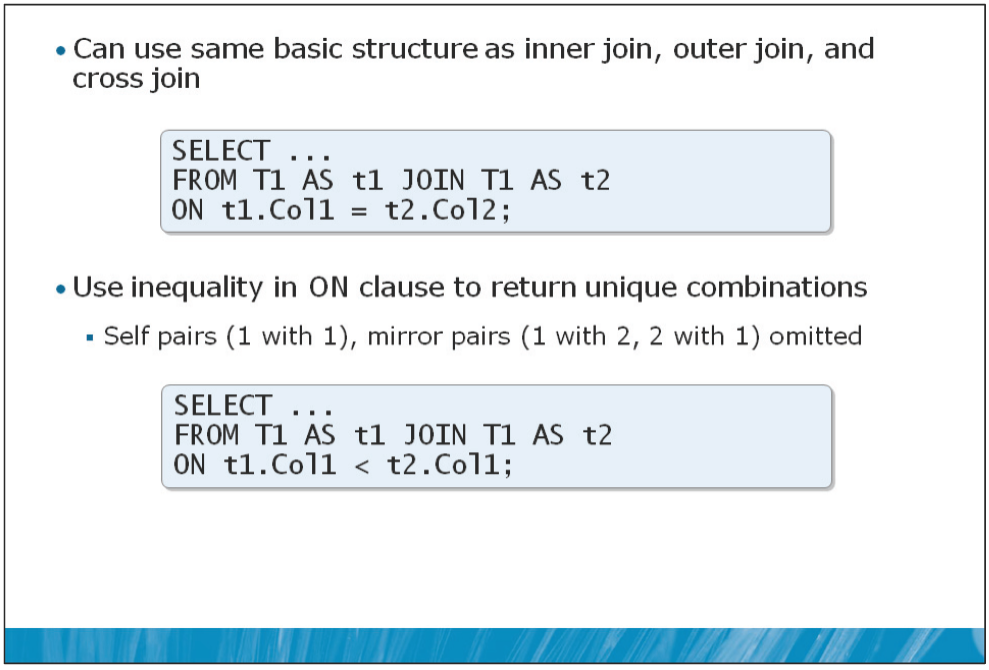
The following example, which you will examine closely in the next topic, illustrates these guidelines. This query retrieves employees and their matching manager information from the Employees table joined to itself:

```
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 Syntax



You write self-join queries much like you write inner or outer joins. The primary difference is that the tables on both sides of the JOIN operator will actually be references to the same table. Pay particular attention to your aliasing and which columns you use in the ON clause in order to ensure that you are joining the table to itself properly. Try using the terms in the business problem (such as employee and manager) to help keep it clear which table you need to refer to.

All of the queries you have seen in this module so far have been written using an equality operator to match rows. Joins that use equality operators are referred to as equijoins. However, there are numerous scenarios in which the use of an inequality operator in a self-join can open up opportunities for interesting queries.

When a join condition uses other operators, such as <, >, <>, the join is called a non-equijoin. When joining a table to itself, using an inequality operator (<, >, <>) will filter out those rows where the columns being compared in the ON clause are equal to each other. This will create unique pairs of values in the column being compared.

For example, consider a table of numbers with only one column, n, and only three rows:

```
SELECT n FROM T;
```

The results:

```
n -- 1 2 3
```

A cross-join query that created a Cartesian product would produce all combinations of the values:

```
SELECT T1.n, T2.n FROM T AS T1 CROSS JOIN T AS T2;
```

The results:

```
n n - - 1 1 2 1 3 1 1 2 2 2 3 2 1 3 2 3 3 3 (9 row(s) affected)
```

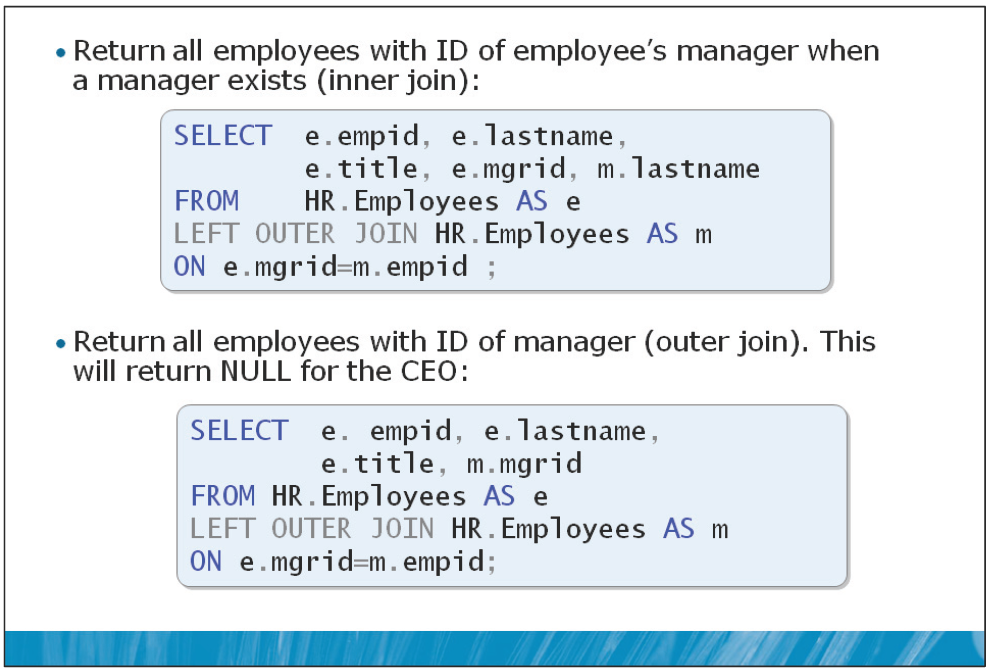
To create only unique combinations of the pairs, create a predicate T1.n < T2.n in the JOIN clause to produce three rows:

```
SELECT T1.n, T2.n FROM T AS T1 JOIN T AS T2 ON T1.n < T2.n;
```

The results:

```
n n - - 1 2 1 3 2 3 (3 row(s) affected)
```

Self-Join Examples



The following are some examples of self-joins:

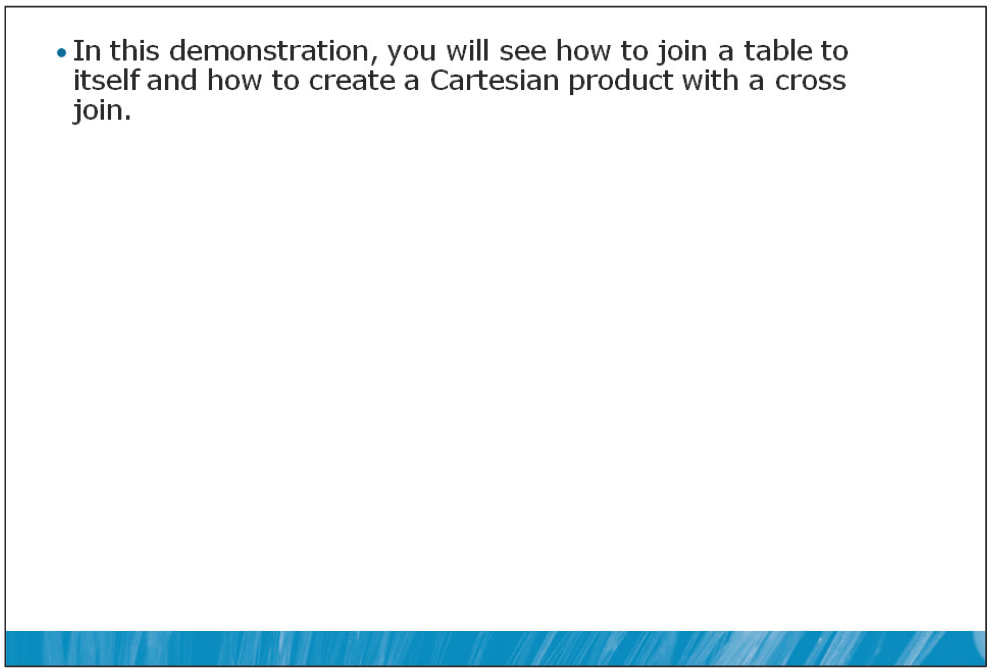
This query returns all employees along with the name of each employee's manager when a manager exists (inner join). Note that a manager appears who is not also listed an employee:

```
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 query returns all employees with the name of each manager (outer join). This restores the missing employee, who turns out to be a CEO with no manager:

```
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



Demonstration Steps

1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**.

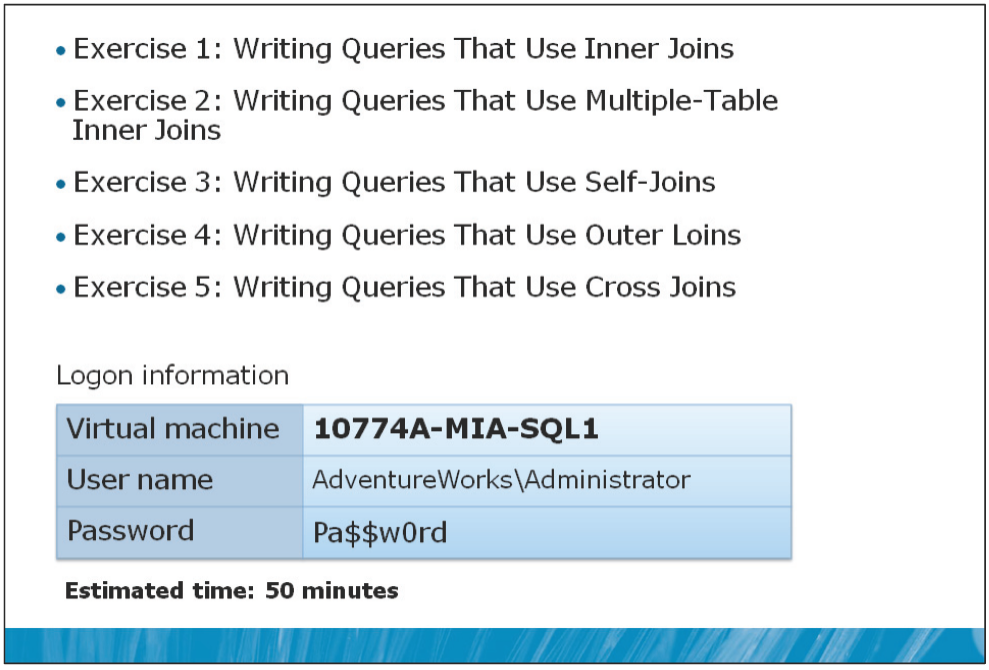
   Navigate to F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 41 – Demonstration D.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lab: Querying Multiple Tables



Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must complete the following steps:

1. On the host computer, click **Start**, point to **Administrative Tools**, and click **Hyper-V Manager**.
2. Maximize the **Hyper-V Manager** window.
3. If the virtual machine **10774A-MIA-DC1** is not started:
   - In the **Virtual Machines** list, right-click **10774A-MIA-DC1** and click **Start**.
   - Right-click **10774A-MIA-DC1** and click **Connect**.
   - In the **Virtual Machine Connection** window, wait until the "Press CTRL+ALT+DELETE to log on" message appears, and then close the **Virtual Machine Connection** window.
4. If the virtual machine **10774A-MIA-SQL1** is not started:
   - In the **Virtual Machines** list, right-click **10774A-MIA-SQL1** and click **Start**.

- Right-click **10774A-MIA-SQL1** and click **Connect**.
- In the **Virtual Machine Connection** window, wait until the "Press CTRL+ALT+DELETE to log on" message appears, and then close the **Virtual Machine Connection** window.

5. In the **Virtual Machine Connection** window, click the **Revert** toolbar icon.
6. If you are prompted to confirm that you want to revert, click **Revert**. Wait for the revert action to complete.
7. If you are not already logged on:
   - In the **Virtual Machine Connection** window, click the **Ctrl-Alt-Delete** menu item on the **Action** menu.
   - Click **Switch User**, and then click **Other User**.
   - Log on using the following credentials:
     - User name: **AdventureWorks\Administrator**
     - Password: **Pa$$w0rd**
8. In the **Virtual Machine Connection** window, click **Full Screen Mode** on the **View** menu.
9. If the **Server Manager** window appears, check the **Do not show me this console at logon** check box and close the **Server Manager** window.
10. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**.
11. In the **Connect to Server** window, type **Proseware** in the **Server name** text box.
12. Click the **Options** button. Under **Connection Properties**, select **<Browse server>** in the **Connect to database** list. Choose **Yes** when prompted for the connection to the database. Under **User Databases**, select the **TSQL2012** database.

13. Choose the authentication type, depending on the type of deployment (ask your instructor for a current list of Microsoft SQL Azure™ enabled labs):

    - For an on-premises Microsoft SQL Server instance, click the **Login** tab, select **Windows Authentication** in the **Authentication** list, and click **Connect**.
    - For Microsoft Windows Azure™, click the **Login** tab, select **SQL Server Authentication** in the **Authentication** list, type your login name in the **Login** text box and the password in the **Password** text box, and click **Connect**.

Lab Scenario

You are a business analyst for Adventure Works who will be writing reports using corporate databases stored in SQL Server 2012. You have been provided with 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.

Exercise 1: Writing Queries That Use Inner Joins

Scenario

You no longer need the mapping information between categoryid and categoryname that was supplied in module 4 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. Write a SELECT statement that uses an inner join.
2. Answer questions.

Task 1: Write a SELECT statement that uses an inner join

- Open the project file F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and the T-SQL script 51 - Lab Exercise 1.sql. Ensure that you are connected to the TSQL2012 database.
- Write a SELECT statement that will return the productname column from the Production.Products table (use table alias "p") and the categoryname column from the Production.Categories table (use table alias "c") using an inner join.
- Execute the written statement and compare the results that you got with the desired results shown in the file 52 - Lab Exercise 1 - Task 1 Result.txt.
- Which column did you specify as a predicate in the ON clause of the join? Why?
- Let us say that there is a new row in the Production.Categories table and this new product category does not have any products associated with it in the Production.Products table. Would this row be included in the result of the SELECT statement written in task 1? Please explain.

Results: 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 that placed at least one order, with detailed information about each order. 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. Analyze and correct the query.
2. Add the productid, qty, and unitprice columns from the Sales.OrderDetails table.

Task 1: Execute the T-SQL statement

- Open the project file F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and the T-SQL script 61 - Lab Exercise 2.sql. Ensure that you are connected to the TSQL2012 database.
- The developer has written this query:

```
SELECT custid, contactname, orderid FROM Sales.Customers INNER join
Sales.Orders ON Customers.custid = Orders.custid;
```

- Execute the query exactly as written inside a query window and observe the result.
- You get an error. What is the error message? Why do you think you got this error?

Task 2: Apply the needed changes and execute the T-SQL statement

- Notice that there are full source table names written as table aliases.
- Apply the needed changes to the SELECT statement so that it will run without an error. Test the changes by executing the T-SQL statement.
- Observe and compare the results that you got with the recommended results shown in the file 62 - Lab Exercise 2 - Task 2 Result.txt.

Task 3: Change the table aliases

- Copy the T-SQL statement from task 2 and modify it to use the table aliases "C" for the Sales.Custumers table and "O" for the Sales.Orders table.
- Execute the written statement and compare the results with the results in task 2.
- Change the prefix of the columns in the SELECT statement with full source table names and execute the statement.
- You get an error. Why?
- Change the SELECT statement to use the table aliases written at the beginning of the task.

Task 4: Add an additional table and columns

- Copy the T-SQL statement from task 3 and modify it to include three additional columns from the Sales.OrderDetails table: productid, qty, and unitprice.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 63 - Lab Exercise 2 - Task 4 Result.txt.

Results: 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 would like 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 SELECT statement using a self-join to retrieve the needed columns.
2. Answer the questions.

Task 1: Write a basic SELECT statement

- Open the project file F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and the T-SQL script 71 - Lab Exercise 3.sql. Ensure that you are connected to the TSQL2012 database.
- In order to better understand the needed tasks, you will first write a SELECT statement against the HR.Employees table showing the empid, lastname, firstname, title, and mgrid columns.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 72 - Lab Exercise 3 - Task 1 Result.txt. Notice the values in the mgrid column. The mgrid column is in a relationship with empid column. This is called a self-referencing relationship.

Task 2: Write a query that uses a self-join

- Copy the SELECT statement from task 1 and modify it to include additional columns for the manager information (lastname, firstname) using a self-join. Assign the aliases mgrlastname and mgrfirstname, respectively, to distinguish the manager names from the employee names.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 73 - Lab Exercise 3 - Task 2 Result.txt. Notice

the number of rows returned.

- Is it mandatory to use table aliases when writing a statement with a self-join? Can you use a full source table name as alias? Please explain.
- Why did you get fewer rows in the T-SQL statement under task 2 compared to task 1?

Results: 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 the sales staff would like to change the report to show all customers, even if they did not have any orders, and still include the information about the orders for the customers that did place orders. 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 using an outer join to retrieve the needed columns.

Task 1: Write a SELECT statement that uses an outer join

- Open the project file F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and the T-SQL script 81 - Lab Exercise 4.sql. Ensure that you are connected to the

TSQL2012 database.

- Write a SELECT statement to retrieve the custid and contactname columns from the Sales.Customers table and the orderid column from the Sales.Orders table. The statement should retrieve all rows from the Sales.Customers table.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 82 - Lab Exercise 4 - Task 1 Result.txt.
- Notice the values in the column orderid. Are there any missing values (marked as NULL)? Why?

Results: 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.

1. Execute the provided T-SQL statement to generate the HR.Calendar table, which includes the calendardate column, and to populate the table with date information.
2. Write a SELECT statement that uses a cross join to retrieve the needed columns.

Task 1: Execute the T-SQL statement

- Open the project file F:\10774A_Labs\10774A_05_PRJ\10774A_05_PRJ.ssmssln and the T-SQL script 91 - Lab Exercise 5.sql. Ensure that you are connected to the TSQL2012 database.
- Execute the T-SQL code under task 1. Do not worry if you do not understand the provided T-SQL code, as it is used here to provide a more realistic example for a cross join in the next task.

Task 2: Write a SELECT statement that uses a cross join

- Write a SELECT statement to retrieve the empid, firstname, and lastname columns from the HR.Employees table and the calendardate column from the HR.Calendar table.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 92 - Lab Exercise 5 - Task 2 Result.txt.
- What is the number of rows returned by the query? There are nine rows in the HR.Employees table. Try to calculate the total number of rows in the HR.Calendar table.

Task 3: Drop the HR.Calendar table

- Execute the provided T-SQL statement to remove the HR.Calendar table.

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

Module Review

Review Questions

1. How does an inner join differ from an outer join?
2. Which join types include a logical Cartesian product?
3. Can a table be joined to itself?

Best Practices

1. Table aliases should always be defined when joining tables.
2. Joins should be expressed using SQL-92 syntax, with JOIN and ON keywords.