Module 12: Using Set Operators

Contents:

Module Overview

Microsoft® SQL Server® provides methods for performing operations using the sets that result from two or more different queries. In this module, you will learn how to use the set operators UNION, INTERSECT, and EXCEPT to compare rows between two input sets.

You will also learn how to use forms of the APPLY operator to use the result of one query to collect the output of a second query, returning the output as a single result set.

Objectives

After completing this module, you will be able to:

- Write queries that combine data using the UNION operator.
- Write queries that compare sets using the INTERSECT and EXCEPT operators.
- Write queries that manipulate rows in a table by using APPLY, combining them with the results of a derived table or function.

Lesson 1: Writing Queries with the UNION Operator

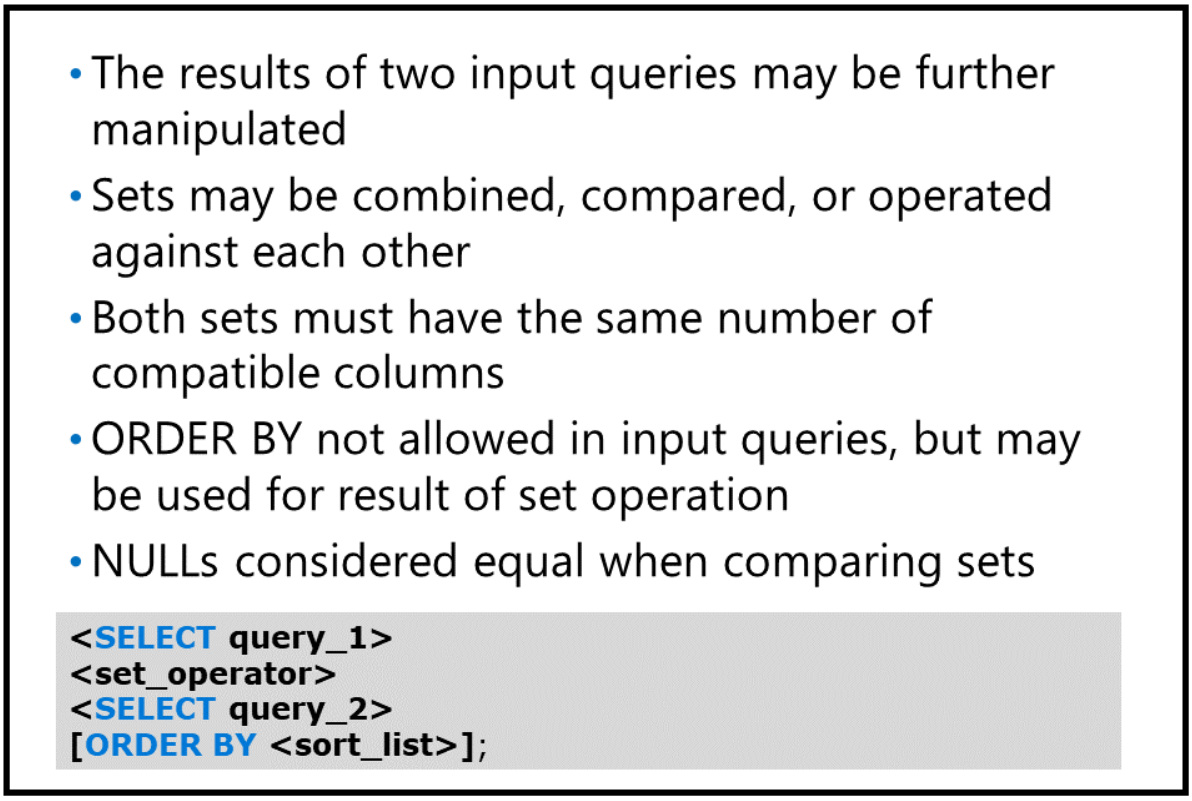
In this lesson, you will learn how to use the UNION operator to combine multiple input sets into a single result. UNION and UNION ALL provide a mechanism to add one set to another; you can then stack result sets from two or more queries into a single output result set. UNION stacks rows, compared to JOIN, which combines columns from different sources.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the conditions necessary to interact between input sets.
- Write queries that use UNION to combine input sets.
- Write queries that use UNION ALL to combine input sets.

Interactions Between Sets



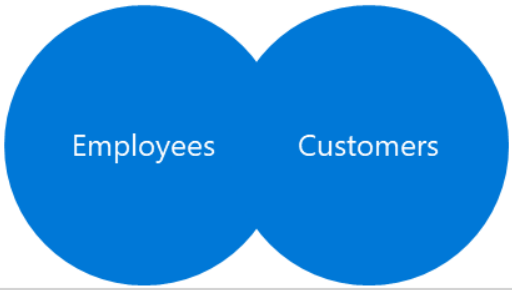
SQL Server provides several operators that act on sets, each of which has a different effect on the input sets. The set operators have a number of common features that you need to understand before starting to use them:

- Each input set is the result of a query, which may include any SELECT statement components you have already learned about, except an ORDER BY clause.
- The input sets must have the same number of columns and the columns must have compatible data types. The column data types, if not initially compatible, must be made compatible through conversion—this may be implicit if the data types support it (using the rules for data type precedence discussed in Module 6 of this course, *Working with SQL Server Data Types*); otherwise an explicit conversion might be required (using CAST or CONVERT).
- A NULL in one set is treated as equal to a NULL in another, despite what you have learned about comparing NULLs earlier in this course.
- Each operator can be thought of as having two forms: DISTINCT and ALL. For example, UNION DISTINCT eliminates duplicate rows while combining two sets; UNION ALL combines all rows, including duplicates. Not all set operators support both forms in SQL Server.

> **Note:** When working with set operators, it is useful to remember that, in set theory, a set does not provide a sort order and includes only distinct rows. If you need the results sorted, you should add an ORDER BY to the final results, as you may not use it inside the input queries.

Using the UNION Operator

- UNION returns a result set of distinct rows combined from both input sets
- Duplicates are removed during query processing (affects performance)




```
-- only distinct rows from both queries are returned
SELECT country, region, city FROM HR.Employees
UNION
SELECT country, region, city FROM Sales.Customers;
```

By using the UNION operator, you can combine rows from one input set with rows from another into a resulting set. If a row appears in either of the input sets, it will be returned in the output. Duplicate rows are eliminated by the UNION operator.

For example, in the TSQL sample database, there are 29 rows in the Production.Suppliers table and 91 rows in the Sales.Customers table.

UNION Example

```
SELECT country, city
FROM Production.Suppliers
UNION
SELECT country, city
FROM Sales.Customers;
```

A partial result:

```
country   city
--------- ---------------
Argentina Buenos Aires
Australia Melbourne
...
USA       Walla Walla
Venezuela Barquisimeto
Venezuela Caracas
Venezuela I. de Margarita
Venezuela San Cristóbal

(93 row(s) affected)
```

> **Note:** As with all T-SQL statements, remember that no sort order is guaranteed by set operators unless one is explicitly specified. Although the results might appear to be sorted, this is a by-product of the filtering performed and is not assured. If you require sorted output, add an ORDER BY clause at the end of the second query.

As previously mentioned, set operators can conceptually be thought of in two forms: DISTINCT and ALL. SQL Server does not implement an explicit UNION DISTINCT, though it does implement UNION ALL. ANSI SQL standards do specify both as explicit forms (UNION DISTINCT and UNION ALL). In T-SQL, the use of DISTINCT is not supported but is the implicit default. UNION combines all rows from each input set, and then filters out duplicates.

From a performance standpoint, the use of UNION will include a filter operation, whether or not there are duplicate rows. If you need to combine sets and know that there are no duplicates, consider using UNION ALL to save the overhead of the distinct filter.

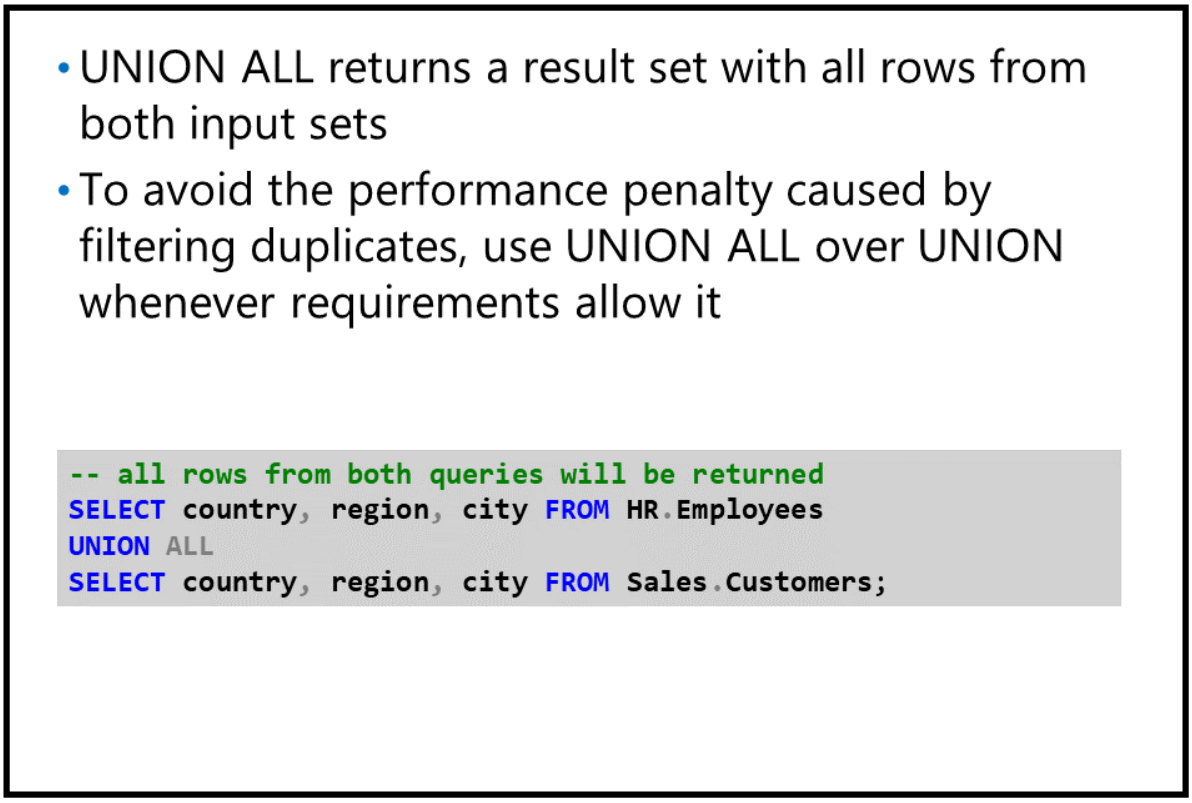
> **Note:** You will learn about UNION ALL in the next lesson.

For more information, see *UNION (Transact-SQL)* in Microsoft Docs:

UNION (Transact-SQL)

http://aka.ms/omv6m7

Using the UNION ALL Operator



The UNION ALL operator works in a similar way to the UNION operator—it combines the two input result sets into one output result set. Unlike UNION, UNION ALL does not filter out duplicate rows.

UNION ALL Example

```
SELECT country, city
FROM Production.Suppliers
UNION ALL
SELECT Country, City
FROM Sales.Customers;
```

Using UNION ALL, 120 rows are returned (29 rows from the Production.Suppliers table and 91 rows from the Sales.Customers table):

```
country city
------- ---------------
UK      London
USA     New Orleans
...
Finland Helsinki
Poland  Warszawa

(120 rows affected)
```

As UNION ALL does not perform any filtering of duplicates, UNION ALL should be used in place of UNION in cases where you know there will be no duplicate input rows (or where duplicates exist and are required).

UNION ALL will often run significantly faster than UNION on the same data set; this performance difference becomes more obvious as the number of rows in the input result sets increases.

Demonstration: Using UNION and UNION ALL

In this demonstration, you will see how to use UNION and UNION ALL.

Demonstration Steps

1. Ensure that the **20761C-MIA-DC**, and **20761C-MIA-SQL** virtual machines are running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod12\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.
5. Wait for the script to finish, and then press any key.
6. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod12\Demo** folder.
8. In Solution Explorer, expand **Queries**, and double-click the **11 - Demonstration A.sql** script file.
9. Select the code under the comment **Step 1**, and then click **Execute**.
10. Select the code under the comment **Step 2**, and then click **Execute**.
11. Select the code under the comment **Step 3**, and then click **Execute**.
12. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

True or False Question

True or false? The results from a UNION query can contain duplicate rows.

True False

Check answer Reset

False. The UNION operator eliminates duplicate rows as part of its operation.

Check Your Knowledge

True or False Question

When combining the output of two sets, UNION and UNION ALL queries cannot include rows with NULL values, because NULL values cannot be compared.

True False

Check answer Reset

False. In a WHERE clause, NULL never equals another value, not even other NULLs. In UNION and UNION ALL queries, NULLs are treated as equal to other NULLs when they are compared between input sets.

Lesson 2: Using EXCEPT and INTERSECT

While UNION and UNION ALL combine all rows from input sets, you might need to return either only those rows in one set but not in the other—or only rows that are present in both sets. For these purposes, the EXCEPT and INTERSECT operators might be useful to your queries. You will learn how to use EXCEPT and INTERSECT in this lesson.

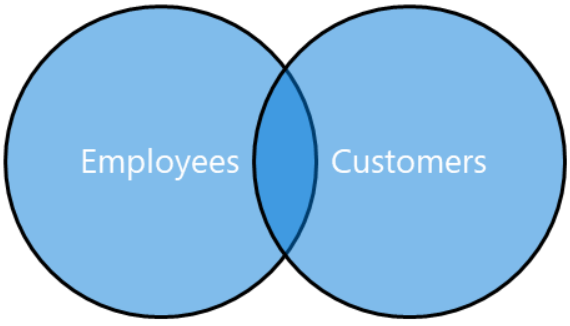
Lesson Objectives

After completing this lesson, you will be able to:

- Write queries that use the EXCEPT operator to return only rows in one set but not another.
- Write queries that use the INTERSECT operator to return only rows that are present in both sets.

Using the INTERSECT Operator

- INTERSECT returns the distinct set of rows that appear in both input result sets




```
-- only rows that exist in both queries will be returned
SELECT country, region, city FROM HR.Employees
INTERSECT
SELECT country, region, city FROM Sales.Customers;
```

The T-SQL INTERSECT operator, added in SQL Server 2005, returns only distinct rows that appear in both input sets.

> **Note:** While UNION supports both the conceptual forms DISTINCT and ALL, INTERSECT currently only provides an implicit DISTINCT option. No duplicate rows will be returned by the operation.

INTERSECT Example


```
SELECT country, city
FROM Production.Suppliers
INTERSECT
SELECT country, city
FROM Sales.Customers;
```

Returns:

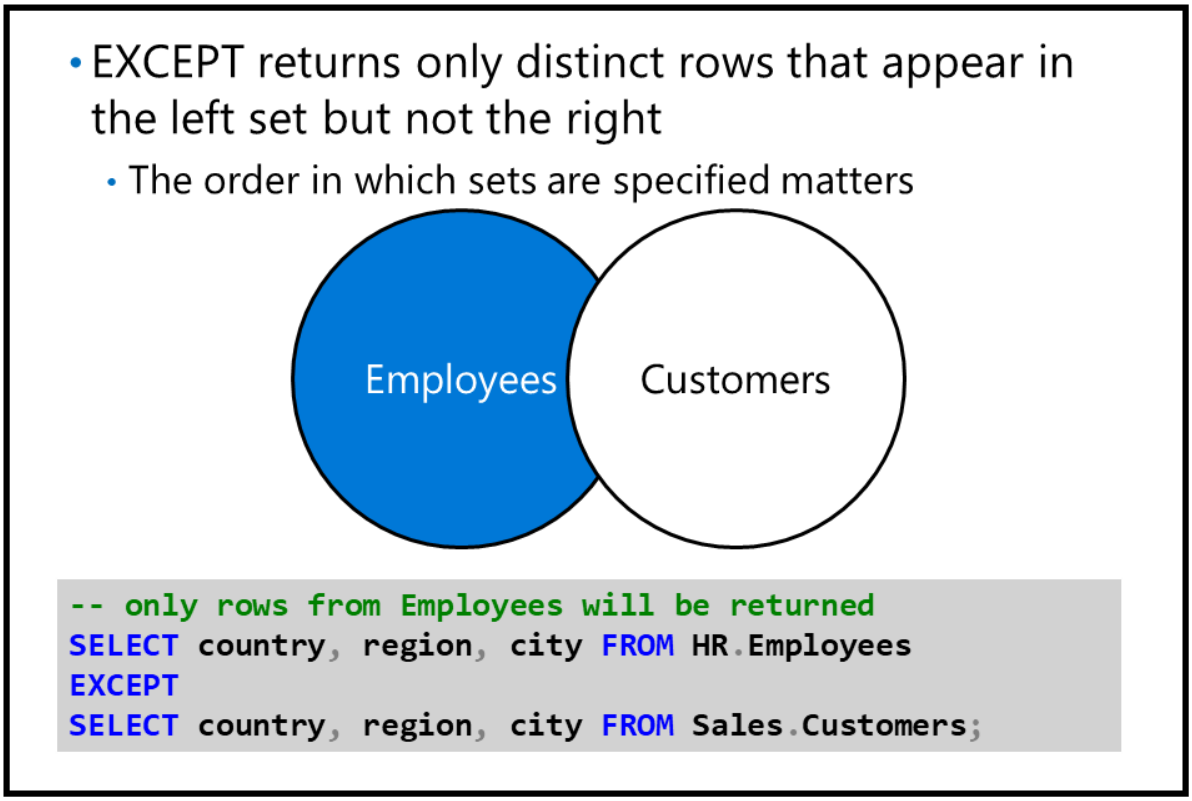
```
country  city
-------- ---------
Germany  Berlin
UK       London
Canada   Montréal
France   Paris
Brazil   Sao Paulo

(5 row(s) affected)
```

For more information, see *Set Operators - EXCEPT and INTERSECT (Transact-SQL)* in Microsoft Docs:

Set Operators - EXCEPT and INTERSECT (Transact-SQL)

http://aka.ms/uo4qu9

Using the EXCEPT Operator



The T-SQL EXCEPT operator, added in SQL Server 2005, returns only distinct rows that appear in one set and not the other. Specifically, EXCEPT returns rows from the input set listed first in the query. As with queries that use a LEFT OUTER JOIN or RIGHT OUTER JOIN, the order in which the inputs are listed is important.

> **Note:** While UNION supports both conceptual forms DISTINCT and ALL, EXCEPT currently only provides an implicit DISTINCT option. No duplicate rows will be returned by the operation.

EXCEPT Example

```
SELECT country, city
FROM Production.Suppliers
EXCEPT
SELECT country, city
FROM Sales.Customers;
```

There are 24 rows returned. Part of the result set is displayed here:

```
country    city
---------- -------------
Australia  Melbourne
Australia  Sydney
Canada     Ste-Hyacinthe
Denmark    Lyngby
Finland    Lappeenranta
```

```
France      Annecy
France      Montceau

(24 row(s) affected)
```

EXCEPT Example - Input Set Order Reversed

```
SELECT country, city
FROM Sales.Customers
EXCEPT
SELECT country, city
FROM Production.Suppliers;
```

This returns 64 rows. When using EXCEPT, plan the order of the input result sets carefully.

Demonstration: Using EXCEPT and INTERSECT

In this demonstration, you will see how to use INTERSECT and EXCEPT.

Demonstration Steps

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

Check Your Knowledge

Select the best answer

You have a table of employees and a table of customers, both of which contain a column holding the name of the country where the customer or employee is located. You want to know which countries have at least one customer and at least one employee. Which set operator should you use?

UNION ALL

UNION

EXCEPT

INTERSECT

None of the above

Check answer Show solution Reset

You should use INTERSECT.
UNION lists all results that appear in either set, which would list all countries with at least one customer or at least one employee. UNION ALL would include duplicates found in the sets.
EXCEPT lists results that appear in only the first set, which would list countries with at least one customer but no employee; or a list of countries with at least one employee but no customer, depending on the order in which the tables appear in your query.
INTERSECT lists results that appear in both sets—which is the list of countries you want to find.

Lesson 3: Using APPLY

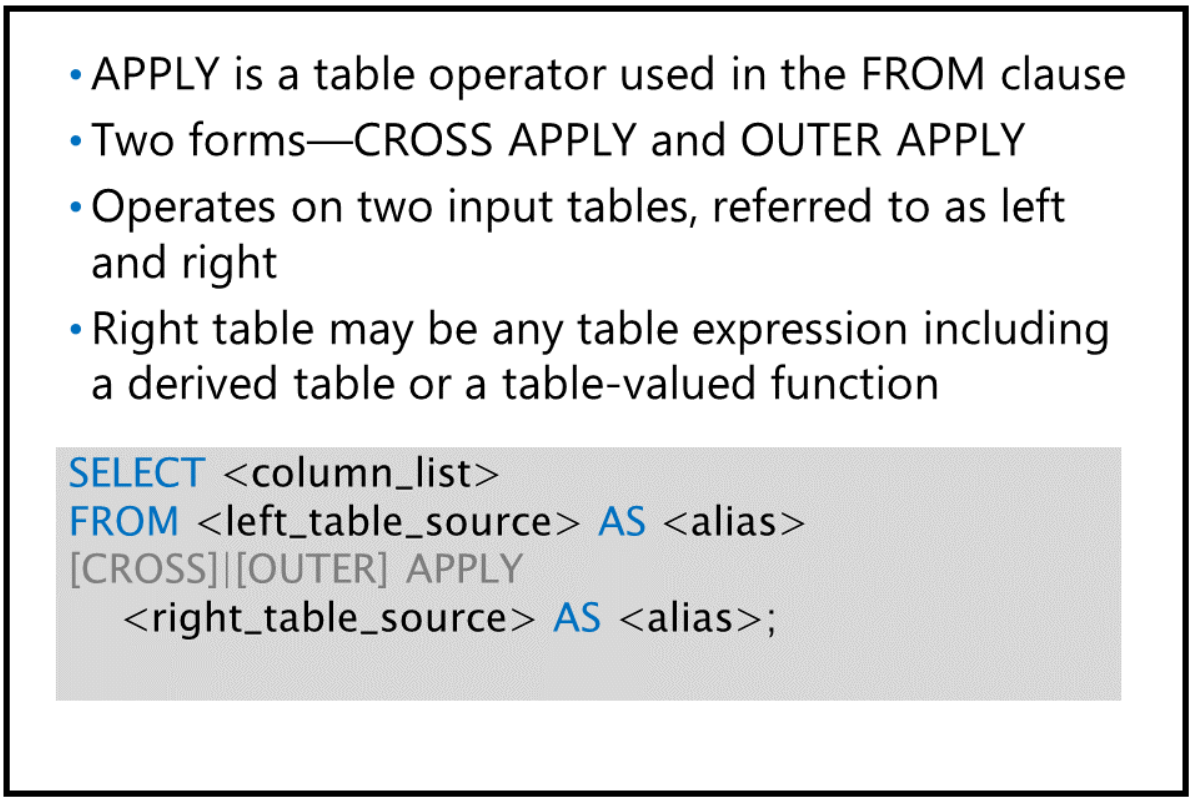
As an alternative to combining or comparing rows from two sets, SQL Server provides a mechanism to apply a table expression from one set on each row in the other set. In this lesson, you will learn how to use the APPLY operator to process rows in one set using rows in another.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the use of the APPLY operator to manipulate sets.
- Write queries using the CROSS APPLY operator.
- Write queries using the OUTER APPLY operator.

Using the APPLY Operator



SQL Server provides the APPLY operator to enable queries that evaluate rows in one input set against the expression that defines the second input set. Strictly speaking, APPLY is a table operator, not a set operator. You will use APPLY in a FROM clause, like a JOIN, rather than as a set operator that operates on two compatible result sets of queries.

Conceptually, the APPLY operator is similar to a correlated subquery in that it applies a correlated table expression to each row from a table. However, APPLY differs from correlated subqueries by returning a table-valued result rather than a scalar or multi-valued result. For example, the table expression could be a TVF; you can pass elements from the left row as input parameters to the TVF.

> **Note:** When describing input tables used with APPLY, the terms “left” and “right” are used in the same way as they are with the JOIN operator, based on the order in which they appear, relative to one another in the FROM clause.

To use APPLY, you will supply two input sets within a single FROM clause. With APPLY, unlike the set operators you have learned about, the second, or right, table source is logically processed once per row found in the first, or left, table source.

APPLY supports two different forms: CROSS APPLY and OUTER APPLY, which you will learn about in this lesson.

APPLY Syntax

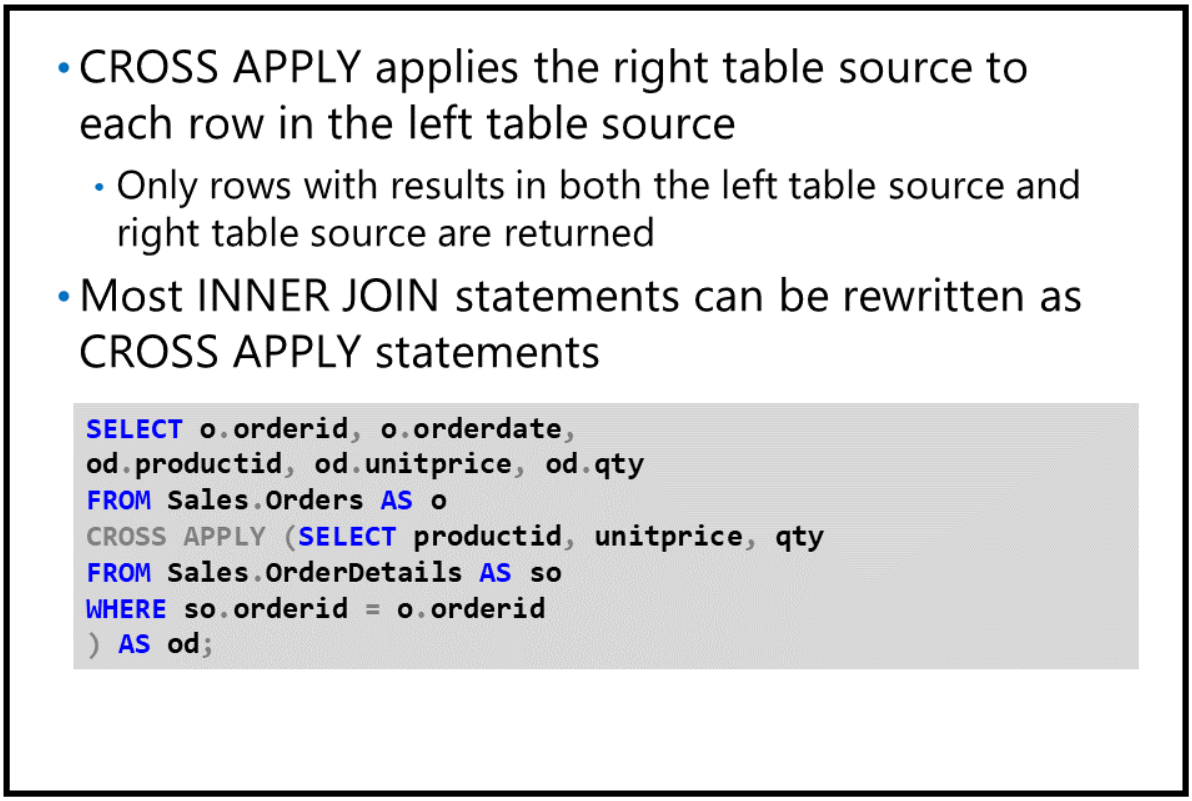
```
SELECT <column_list>
FROM <left_table_source> AS <alias>
[CROSS]|[OUTER] APPLY
    <right_table_source> AS <alias>;
```

See Using APPLY in the “Remarks” section of *FROM (Transact-SQL)* in Microsoft Docs:

FROM (Transact-SQL)

http://aka.ms/r0uc2i

The CROSS APPLY Operator



As you learned in the previous topic, APPLY executes the right table source for each of the rows in the left table source—and returns the results as a single result set.

The CROSS APPLY form of the operator will include in the output result set only those values from the left table source where a value is found in the right table source.

> **Note:** Note that the term CROSS, when used in CROSS APPLY, does not have the same meaning as CROSS when used in CROSS JOIN. Whereas a CROSS JOIN returns all the possible combinations of the left and right table sources, CROSS APPLY returns only the values from the left table source where a value is found in the right table source.

This makes a CROSS APPLY statement very similar to an INNER JOIN—this similarity is such that almost all T-SQL statements that include an INNER JOIN between two tables can be rewritten as a statement using CROSS APPLY.

CROSS APPLY; INNER JOIN Example

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

A partial result from the TSQL sample database:

```
orderid     orderdate               productid   unitprice     qty
----------- ----------------------- ----------- -----------   ------
10248       2006-07-04 00:00:00.000 11          14.00         12
10248       2006-07-04 00:00:00.000 42          9.80          10
10248       2006-07-04 00:00:00.000 72          34.80         5
10249       2006-07-05 00:00:00.000 14          18.60         9
10249       2006-07-05 00:00:00.000 51          42.40         40
10250       2006-07-08 00:00:00.000 41          7.70          10
10250       2006-07-08 00:00:00.000 51          42.40         35
10250       2006-07-08 00:00:00.000 65          16.80         15
...
(2155 row(s) affected)
```

CROSS APPLY; INNER JOIN Rewritten Example

```
SELECT o.orderid, o.orderdate,
od.productid, od.unitprice, od.qty
FROM Sales.Orders AS o
CROSS APPLY (   SELECT productid, unitprice, qty
                FROM Sales.OrderDetails AS so
                WHERE so.orderid = o.orderid
                 ) AS od;
```

Note: Notice that the JOIN predicate

```
Sales.OrderDetails.orderid = Sales.Orders.orderid
```

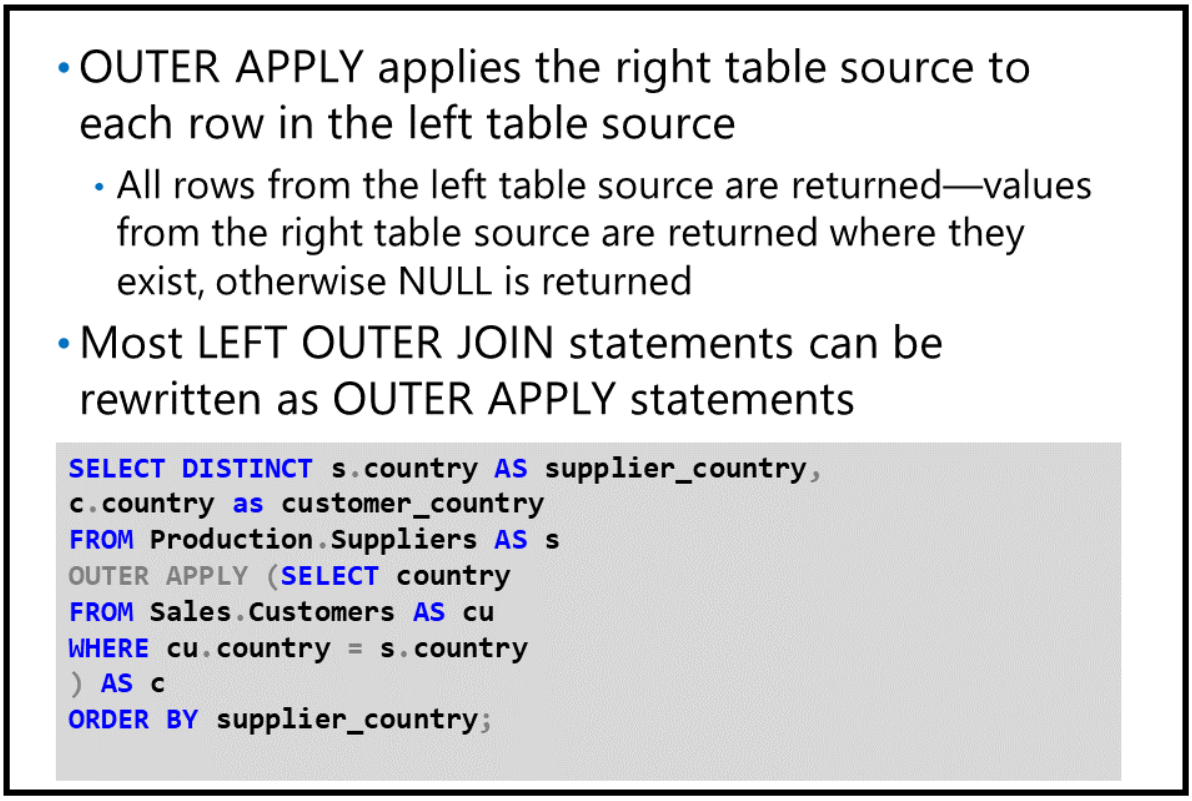
moves from the INNER JOIN clause to the WHERE clause of the right table source when the query is rewritten to use CROSS APPLY.

When executed, this query returns the same result as the version written using INNER JOIN:

```
orderid     orderdate               productid   unitprice     qty
----------- ----------------------- ----------- -----------   ------
10248       2006-07-04 00:00:00.000 11          14.00         12
10248       2006-07-04 00:00:00.000 42          9.80          10
10248       2006-07-04 00:00:00.000 72          34.80         5
10249       2006-07-05 00:00:00.000 14          18.60         9
10249       2006-07-05 00:00:00.000 51          42.40         40
10250       2006-07-08 00:00:00.000 41          7.70          10
10250       2006-07-08 00:00:00.000 51          42.40         35
10250       2006-07-08 00:00:00.000 65          16.80         15
```

```
...
(2155 row(s) affected)
```

The OUTER APPLY Operator



As you learned in an earlier topic, APPLY executes the right table source for each of the rows in the left table source, and returns the results as a single result set.

The OUTER APPLY form of the operator will include all the values from the left table source in the output result set and values from the right table source where they exist. Where the right table source does not contain a value for a left table source value, columns derived from the right table source will have a NULL value.

This makes an OUTER APPLY statement very similar to a LEFT OUTER JOIN—this similarity is such that almost all T-SQL statements that include a LEFT OUTER JOIN between two tables can be rewritten as a statement using OUTER APPLY.

As with LEFT OUTER JOIN, the order in which the table sources appear might influence the result.

OUTER APPLY; LEFT OUTER JOIN Example

```
SELECT DISTINCT s.country AS supplier_country, c.country as customer_country
FROM Production.Suppliers AS s
LEFT OUTER JOIN Sales.Customers AS c
ON c.country = s.country
ORDER BY supplier_country;
```

> **Note:** Notice that the JOIN predicate
>
> ```
> Sales.Customers.Country = Production.Suppliers.Country
> ```
>
> moves from the LEFT OUTER JOIN clause to the WHERE clause of the right table source when the query is rewritten to use OUTER APPLY.

This query returns the same result as the LEFT OUTER JOIN version of the query:

```
supplier_country customer_country
---------------- ----------------
Australia        NULL
Brazil           Brazil
Canada           Canada
Denmark          Denmark
Finland          Finland
France           France
Germany          Germany
Italy            Italy
Japan            NULL
Netherlands      NULL
Norway           Norway
Singapore        NULL
Spain            Spain
Sweden           Sweden
UK               UK
USA              USA

(16 row(s) affected)
```

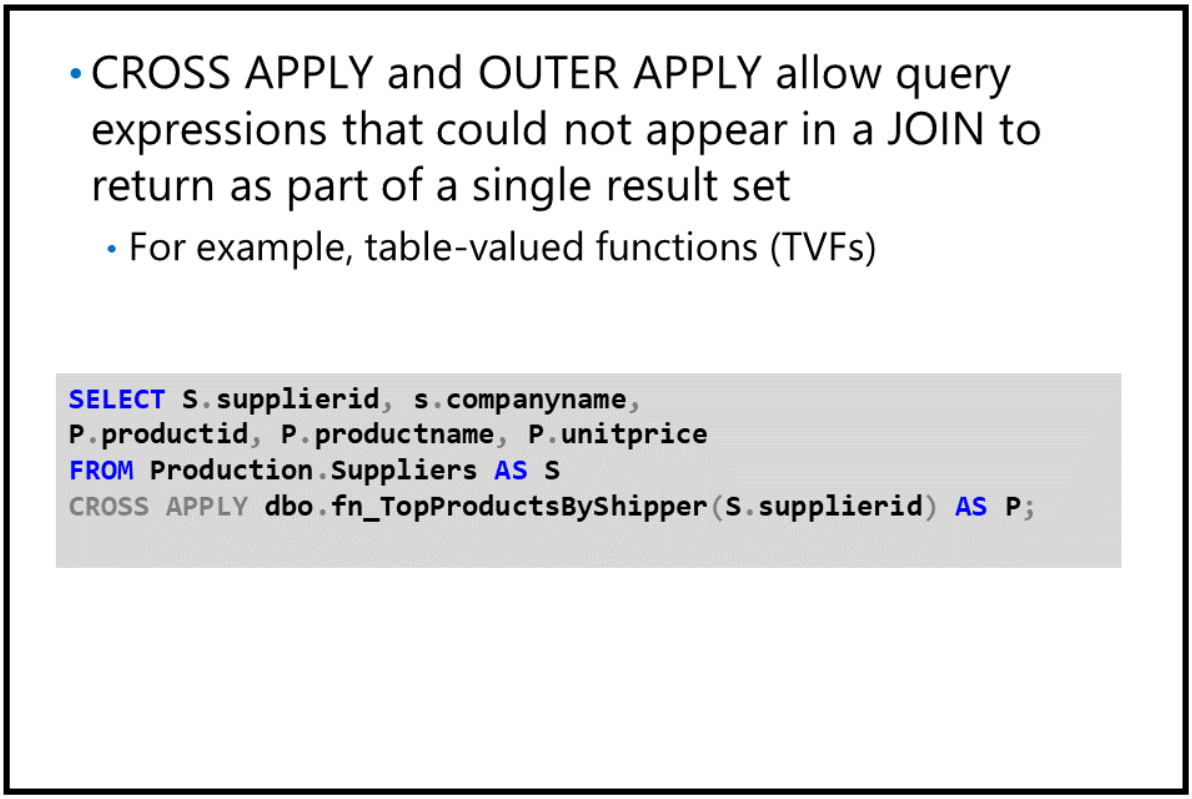
OUTER APPLY; LEFT OUTER JOIN Rewritten Example

```
SELECT DISTINCT s.country AS supplier_country, c.country as customer_country
FROM Production.Suppliers AS s
OUTER APPLY (   SELECT country
                            FROM Sales.Customers AS cu
                            WHERE cu.country = s.country
                        ) AS c
ORDER BY supplier_country;
```

Returns:

```
supplier_country customer_country
---------------- ----------------
Australia        NULL
Brazil           Brazil
Canada           Canada
Denmark          Denmark
Finland          Finland
France           France
Germany          Germany
Italy            Italy
Japan            NULL
Netherlands      NULL
Norway           Norway
Singapore        NULL
Spain            Spain
Sweden           Sweden
UK               UK
USA              USA

(16 row(s) affected)
```

CROSS APPLY and OUTER APPLY Features



As you learned in the previous topics, there are many similarities between CROSS APPLY and INNER JOIN, and OUTER APPLY and LEFT OUTER JOIN.

However, the APPLY operators enable some types of query to be executed which could not be written using JOINs. These queries rely on the left table source being processed before being applied to the right table source. Two examples shown in this topic are using a query returning top results for each input value and a TVF as the right table source.

OUTER APPLY: Three Most Recent Orders Per Customer Example

```
SELECT C.custid, TopOrders.orderid, TopOrders.orderdate
FROM Sales.Customers AS C
OUTER APPLY
        (SELECT TOP (3) orderid, CAST(orderdate AS date) AS orderdate
         FROM Sales.Orders AS O
         WHERE O.custid = C.custid
        ORDER BY orderdate DESC, orderid DESC) AS TopOrders;
```

> **Note:** Note that because OUTER APPLY is used here, customers with no orders are included in the result (with NULL in the orderid and orderdate columns). If CROSS APPLY were used instead of OUTER APPLY, customers with no orders would not appear in the results.

Partial results, including rows with NULLs, appear as follows:

```
custid      orderid     orderdate
----------- ----------- ----------
1           11011       2008-04-09
1           10952       2008-03-16
1           10835       2008-01-15
2           10926       2008-03-04
2           10759       2007-11-28
2           10625       2007-08-08
22          NULL        NULL
57          NULL        NULL
58          11073       2008-05-05
58          10995       2008-04-02
58          10502       2007-04-10
(265 row(s) affected)
```

A TVF might be used as the right table source for an instance of the APPLY operator.

CROSS APPLY: Calling a Table-Valued Function Example

```
SELECT S.supplierid, s.companyname, P.productid, P.productname, P.unitprice
FROM Production.Suppliers AS S
CROSS APPLY dbo.fn_TopProductsByShipper(S.supplierid) AS P;
```

> **Note:** Note that because CROSS APPLY is used here, suppliers with no products are excluded from the result.

Partial results appear as follows:

```
supplierid  companyname    productid   productname   unitprice
----------- -------------- ----------- ------------- ---------
1           Supplier SWRXU 2           Product RECZE 19.00
1           Supplier SWRXU 1           Product HHYDP 18.00
1           Supplier SWRXU 3           Product IMEHJ 10.00
2           Supplier VHQZD 4           Product KSBRM 22.00
2           Supplier VHQZD 5           Product EPEIM 21.35
2           Supplier VHQZD 65          Product XYWBZ 21.05
3           Supplier STUAZ 8           Product WVJFP 40.00
3           Supplier STUAZ 7           Product HMLNI 30.00
3           Supplier STUAZ 6           Product VAIIV 25.00
```

Demonstration: Using CROSS APPLY and OUTER APPLY

In this demonstration, you will see how to use forms of the APPLY Operator.

Demonstration Steps

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 **Test with CROSS APPLY**, and then click **Execute**.
6. Select the code under the comment **Step 4**, and then click **Execute**.
7. Select the code under the comment **Step 5**, and then click **Execute**.
8. Select the code under the comment **Use OUTER APPLY to include customers with no orders**, and then click **Execute**.
9. Close SQL Server Management Studio, without saving any changes.

Check Your Knowledge

Discovery

What is the difference between CROSS APPLY and CROSS JOIN?

Show solution Reset

CROSS JOIN returns all the possible combinations of the left and right table sources; CROSS APPLY returns only the values from the left table source where a value is found in the right table source.

Lab: Using Set Operators

Scenario

As a business analyst for Adventure Works, you will be writing reports using corporate databases stored in SQL Server. 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. Because of the complex business requirements, you will need to prepare combined results from multiple queries using set operators.

Objectives

After completing this lab, you will be able to:

- Write queries that use the UNION and UNION ALL operators.
- Write queries that use the CROSS APPLY and OUTER APPLY operators.
- Write queries that use the EXCEPT and INTERSECT operators.

Lab Setup

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Queries That Use UNION Set Operators and UNION ALL Multi-Set Operators

Scenario

The marketing department needs some additional information regarding segmentation of products and customers. It would like to have a report, based on multiple queries, which is presented as one result. You will use the UNION operator to write different SELECT statements, and then merge them together into one result.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Write a SELECT Statement to Retrieve Specific Products
3. Write a SELECT Statement to Retrieve All Products with a Total Sales Amount of More Than $50,000
4. Merge the Results from Task 1 and Task 2
5. Write a SELECT Statement to Retrieve the Top 10 Customers by Sales Amount for January 2008 and February 2008

Detailed Steps ▲

Task 1: Prepare the Lab Environment

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 **Setup.cmd** in the **D:\Labfiles\Lab12\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Write a SELECT Statement to Retrieve Specific Products

1. In SQL Server Management Studio, open the project file **D:\Labfiles\Lab12\Starter\Project\Project.ssmssln** and the T-SQL **script 51 - Lab Exercise 1.sql**. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement to return the **productid** and **productname** columns from the **Production.Products** table. Filter the results to include only products that have a categoryid value 4.
3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\52 - Lab Exercise 1 - Task 1 Result.txt. Remember the number of rows in the results.

Detailed Steps ▲

Task 3: Write a SELECT Statement to Retrieve All Products with a Total Sales Amount of More Than $50,000

1. Write a SELECT statement to return the **productid** and **productname** columns from the **Production.Products** table. Filter the results to include only products that have a total sales amount of more than $50,000. For the total sales amount, you will need to query the **Sales.OrderDetails** table and aggregate all order line values (qty * unitprice) for each product.

2. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\53 - Lab Exercise 1 - Task 2 Result.txt. Remember the number of rows in the results.

Detailed Steps ▲

Task 4: Merge the Results from Task 1 and Task 2

1. Write a SELECT statement that uses the UNION operator to retrieve the **productid** and **productname** columns from the T-SQL statements in task 1 and task 2.
2. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\54 - Lab Exercise 1 - Task 3_1 Result.txt.
3. What is the total number of rows in the results? If you compare this number with an aggregate value of the number of rows from tasks 1 and 2, is there any difference?
4. Copy the T-SQL statement and modify it to use the UNION ALL operator.
5. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\55 - Lab Exercise 1 - Task 3_2 Result.txt.
6. What is the total number of rows in the result? What is the difference between the UNION and UNION ALL operators?

Detailed Steps ▲

Task 5: Write a SELECT Statement to Retrieve the Top 10 Customers by Sales Amount for January 2008 and February 2008

1. Write a SELECT statement to retrieve the **custid** and **contactname** columns from the **Sales.Customers** table. Display the top 10 customers by sales amount for January 2008 and display the top 10 customers by sales amount for February 2008. (Hint: write two SELECT statements, each joining Sales.Customers and Sales.OrderValues, and use the appropriate set operator.)
2. Execute the T-SQL code and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\56 - Lab Exercise 1 - Task 4 Result.txt.

Result: After this exercise, you should know how to use the UNION and UNION ALL set operators in T-SQL statements.

Exercise 2: Writing Queries That Use the CROSS APPLY and OUTER APPLY Operators

Scenario

The sales department needs a more advanced analysis of buying behavior. Staff want to find out the top three products, based on sales revenue, for each customer. Use the APPLY operator to achieve this result.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Last Two Orders for Each Product
2. Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Top Three Products, Based on Sales Revenue, for Each Customer
3. Use the OUTER APPLY Operator
4. Analyze the OUTER APPLY Operator
5. Remove the TVF Created for This Lab

Detailed Steps ▲

Task 1: Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Last Two Orders for Each Product

1. Open the T-SQL script **61 - Lab Exercise 2.sql**. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement to retrieve the **productid** and **productname** columns from the **Production.Products** table. In addition, for each product, retrieve the last two rows from the **Sales.OrderDetails** table based on orderid number.
3. Use the CROSS APPLY operator and a correlated subquery. Order the result by the column **productid**.
4. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab12\Solution\62 - Lab Exercise 2 - Task 1 Result.txt.

Detailed Steps ▲

Task 2: Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Top Three Products, Based on Sales Revenue, for Each Customer

1. Execute the provided T-SQL code to create the inline TVF fnGetTop3ProductsForCustomer:

```
DROP FUNCTION IF EXISTS dbo.fnGetTop3ProductsForCustomer;

GO

CREATE FUNCTION dbo.fnGetTop3ProductsForCustomer
```

```
(@custid AS INT) RETURNS TABLE
AS
RETURN
SELECT TOP(3)
d.productid,
p.productname,
SUM(d.qty * d.unitprice) AS totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
INNER JOIN Production.Products AS p ON p.productid = d.productid
WHERE custid = @custid
GROUP BY d.productid, p.productname
ORDER BY totalsalesamount DESC;
```

2. Write a SELECT statement to retrieve the **custid** and **contactname** columns from the **Sales.Customers** table. Use the CROSS APPLY operator with the dbo.fnGetTop3ProductsForCustomer function to retrieve **productid**, **productname**, and **totalsalesamount** columns for each customer.

3. Execute the written statement and compare the results that you achieved with the recommended result shown in the file D:\Labfiles\Lab12\Solution\63 - Lab Exercise 2 - Task 2 Result.txt. Remember the number of rows in the results.

Detailed Steps ▲

Task 3: Use the OUTER APPLY Operator

1. Copy the T-SQL statement from the previous task and modify it by replacing the CROSS APPLY operator with the OUTER APPLY operator.

2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab12\Solution\64 - Lab Exercise 2 - Task 3 Result.txt. Notice that more rows are returned than in the previous task.

Detailed Steps ▲

Task 4: Analyze the OUTER APPLY Operator

1. Copy the T-SQL statement from the previous task and modify it by filtering the results to show only customers without products. (Hint: in a WHERE clause, look for any column returned by the inline TVF that is NULL.)

2. Execute the written statement and compare the results that you achieved with the recommended results

shown in the file D:\Labfiles\Lab12\Solution\65 - Lab Exercise 2 - Task 4 Result.txt.

3. What is the difference between the CROSS APPLY and OUTER APPLY operators?

Detailed Steps ▲

Task 5: Remove the TVF Created for This Lab

1. Remove the created inline TVF by executing the provided T-SQL code:

```
DROP FUNCTION IF EXISTS dbo.fnGetTop3ProductsForCustomer;
```

2. Execute this code exactly as written inside a query window.

Result: After this exercise, you should be able to use the CROSS APPLY and OUTER APPLY operators in your T-SQL statements.

Exercise 3: Writing Queries That Use the EXCEPT and INTERSECT Operators

Scenario

The marketing department was satisfied with the results from exercise 1, but the staff now need to see specific rows from one result set that are not present in the other result set. You will have to write different queries using the EXCEPT and INTERSECT operators.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Return All Customers Who Bought More Than 20 Distinct Products
2. Write a SELECT Statement to Retrieve All Customers from the USA, Except Those Who Bought More Than 20 Distinct Products
3. Write a SELECT Statement to Retrieve Customers Who Spent More Than $10,000
4. Write a SELECT Statement That Uses the EXCEPT and INTERSECT Operators
5. Change the Operator Precedence

Detailed Steps ▲

Task 1: Write a SELECT Statement to Return All Customers Who Bought More Than 20 Distinct Products

1. Open the T-SQL script **71 - Lab Exercise 3.sql**. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement to retrieve the **custid** column from the **Sales.Orders** table. Filter the results to include only customers who bought more than 20 different products (based on the **productid** column from the **Sales.OrderDetails** table).
3. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab12\Solution\72 - Lab Exercise 3 - Task 1 Result.txt.

Detailed Steps ▲

Task 2: Write a SELECT Statement to Retrieve All Customers from the USA, Except Those Who Bought More Than 20 Distinct Products

1. Write a SELECT statement to retrieve the **custid** column from the **Sales.Orders** table. Filter the results to include only customers from the country USA and exclude all customers from the previous (task 1) result. (Hint: use the EXCEPT operator and the previous query.)
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab12\Solution\73 - Lab Exercise 3 - Task 2 Result.txt.

Detailed Steps ▲

Task 3: Write a SELECT Statement to Retrieve Customers Who Spent More Than $10,000

1. Write a SELECT statement to retrieve the **custid** column from the **Sales.Orders** table. Filter only customers who have a total sales value greater than $10,000. Calculate the sales value using the **qty** and **unitprice** columns from the **Sales.OrderDetails** table.
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab12\Solution\74 - Lab Exercise 3 - Task 3 Result.txt.

Detailed Steps ▲

Task 4: Write a SELECT Statement That Uses the EXCEPT and INTERSECT Operators

1. Copy the T-SQL statement from task 2. Add the INTERSECT operator at the end of the statement. After the INTERSECT operator, add the T-SQL statement from task 3.
2. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab12\Solution\75 - Lab Exercise 3 - Task 4 Result.txt. Notice the total number of rows in the results.

3. In business terms, can you explain which customers are part of the result?

Detailed Steps ▲

Task 5: Change the Operator Precedence

1. Copy the T-SQL statement from the previous task and add parentheses around the first two SELECT statements (from the beginning until the INTERSECT operator).
2. Execute the T-SQL statement and compare the results that you achieved with the recommended result shown in the file D:\Labfiles\Lab12\Solution\76 - Lab Exercise 3 - Task 5 Result.txt. Notice the total number of rows in the results.
3. Are the results different to the results from task 4? Please explain why.
4. What is the precedence among the set operators?
5. Close SQL Server Management Studio, without saving any changes.

Result: After this exercise, you should have an understanding of how to use the EXCEPT and INTERSECT operators in T-SQL statements.

Module Review and Takeaways

In this module, you have learned about set operators and the APPLY operator.

Review Question(s)

Check Your Knowledge

Discovery

Which set operator would you use to combine sets if you knew there were no duplicates and wanted the best possible performance?

Show solution Reset

UNION ALL. UNION ALL does not carry out any deduplication of the output result set.

Check Your Knowledge

Discovery

Which form of the APPLY operator will not return rows from the left table if the result of the right table expression is empty?

Show solution Reset

CROSS APPLY

Check Your Knowledge

Discovery

Which form of the APPLY operator can be used to rewrite LEFT OUTER JOIN queries?

Show solution Reset

OUTER APPLY