

Module 12: Using Set Operators

Contents:

Module Overview

Lesson 1: Writing Queries with the UNION Operator

Lesson 2: Using EXCEPT and INTERSECT

Lesson 3: Using APPLY

Lab: Using Set Operators

Module Review and Takeaways

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

- The results of two input queries may be further manipulated
- Sets may be combined, compared, or operated against each other
- Both sets must have the same number of compatible columns
- ORDER BY not allowed in input queries, but may be used for result of set operation
- NULLs considered equal when comparing sets

```
<SELECT query_1>  
<set_operator>  
<SELECT query_2>  
[ORDER BY <sort_list>];
```

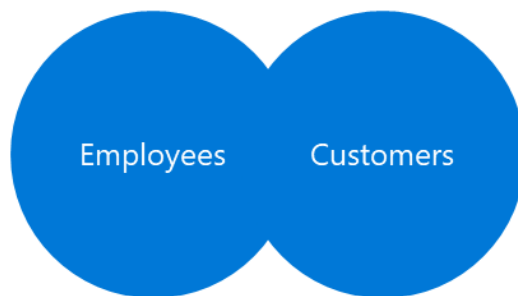
SQL Server provides several operators that act on sets, each of which has a different effect on the input sets. The set operators have a number of common features that you need to understand before starting to use them:

- 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

- UNION ALL returns a result set with all rows from both input sets
- To avoid the performance penalty caused by filtering duplicates, use UNION ALL over UNION whenever requirements allow it

```
-- all rows from both queries will be returned
SELECT country, region, city FROM HR.Employees
UNION ALL
SELECT country, region, city FROM Sales.Customers;
```

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;
```

```
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):

```

This
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.ssmssIn** 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

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

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.

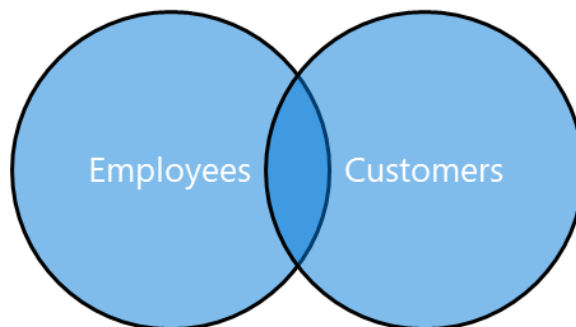
Lesson Objectives

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)

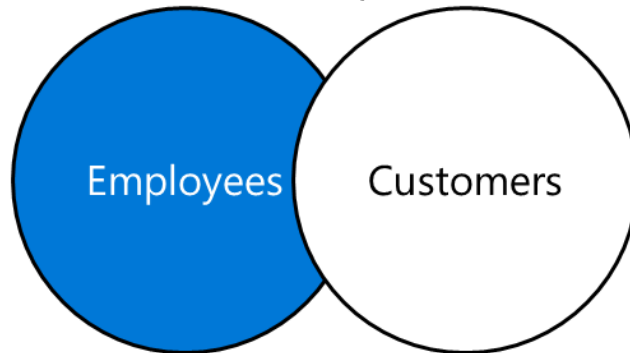
For more information, see *Set Operators - EXCEPT and INTERSECT (Transact-SQL)* in Microsoft Docs:

Set Operators - EXCEPT and INTERSECT (Transact-SQL)

<http://aka.ms/uo4qu9>

Using the EXCEPT Operator

- EXCEPT returns only distinct rows that appear in the left set but not the right
 - The order in which sets are specified matters



```
-- only rows from Employees will be returned
SELECT country, region, city FROM HR.Employees
EXCEPT
SELECT country, region, city FROM Sales.Customers;
```

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

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

- APPLY is a table operator used in the FROM clause
- Two forms—CROSS APPLY and OUTER APPLY
- Operates on two input tables, referred to as left and right
- Right table may be any table expression including a derived table or a table-valued function

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

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

- CROSS APPLY applies the right table source to each row in the left table source
 - Only rows with results in both the left table source and right table source are returned
- Most INNER JOIN statements can be rewritten as CROSS APPLY statements

```
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;
```

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
```



```

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:

is document belong
sebas...

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)

-wed!

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**:

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

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

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

This document

- OUTER APPLY applies the right table source to each row in the left table source
 - All rows from the left table source are returned—values from the right table source are returned where they exist, otherwise NULL is returned
- Most LEFT OUTER JOIN statements can be rewritten as OUTER APPLY statements

```
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;
```

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

- CROSS APPLY and OUTER APPLY allow query expressions that could not appear in a JOIN to return as part of a single result set
 - For example, table-valued functions (TVFs)

```
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;
```

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:

This doc.

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 WJFP	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

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 ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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 ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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 ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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

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

Check Your Knowledge

Discovery

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

Show solution Reset