Lab Answer Key: Module 12: Using Set Operators

Lab: Using Set Operators

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

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. In the **D:\Labfiles\Lab12\Starter** folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, wait for the script to finish, and then press any key.

Task 2: Write a SELECT Statement to Retrieve Specific Products

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine using Windows® authentication.
2. On the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab12\Starter\Project** folder, and then double-click **Project.ssmssln**.
4. In Solution Explorer, expand **Queries**, and then double-click **51 - Lab Exercise 1.sql**.
5. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
6. In the query pane, after the **Task 1** description, type the following query:

```
SELECT
productid, productname
FROM Production.Products
WHERE categoryid = 4;
```

7. Highlight the written query, and click **Execute**. Observe that the query retrieved 10 rows.

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

1. In the query pane, after the **Task 2** description, type the following query:

```
SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

2. Highlight the written query, and click **Execute**. Observe that the query retrieved four rows.

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

1. In the query pane, after the **Task 3** description, type the following query:

```
SELECT
productid, productname
FROM Production.Products
WHERE categoryid = 4

UNION

SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

2. Highlight the written query, and click **Execute**.
3. Observe the result. What is the total number of rows in the result? If you compare this number with an aggregate value of the number of rows from tasks 1 and 2, is there any difference? The total number of rows retrieved by the query is 12. This is two rows less than the aggregate value of rows from the query in task 1 (10 rows) and task 2 (four rows).
4. Highlight the previous query, and on the **Edit** menu, click **Copy**.
5. In the query window, click the line after the written T-SQL statement, and on the **Edit** menu, click **Paste**.
6. Modify the T-SQL statement by replacing the UNION operator with the UNION ALL operator. The query should look like this:


```
SELECT
productid, productname
```

```
FROM Production.Products
WHERE categoryid = 4

UNION ALL

SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

7. Highlight the modified query, and click **Execute**.

8. Observe the result. What is the total number of rows in the result? What is the difference between the UNION and UNION ALL operators? The total number of rows retrieved by the query is 14. It is the same as the aggregate value of rows from the queries in tasks 1 and 2. This is because UNION ALL is a multi-set operator that returns all rows that appear in any of the inputs, without really comparing rows and without eliminating duplicates. The UNION set operator removes the duplicate rows and the result consists of only distinct rows.

9. So, when should you use either UNION ALL or UNION when unifying two inputs? If a potential exists for duplicates and you need to return them, use UNION ALL. If a potential exists for duplicates but you need to return distinct rows, use UNION. If no potential exists for duplicates when unifying the two inputs, UNION and UNION ALL are logically equivalent. However, in such a case, using UNION ALL is recommended because it removes the overhead of SQL Server checking for duplicates.

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

1. In the query pane, after the **Task 4** description, type the following query:

```
SELECT
c1.custid, c1.contactname
FROM
(
SELECT TOP (10)
o.custid, c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.orderdate >= '20080101' AND o.orderdate < '20080201'
GROUP BY o.custid, c.contactname
ORDER BY SUM(o.val) DESC
) AS c1

UNION
```

```
SELECT c2.custid, c2.contactname
FROM
(
SELECT TOP (10)
o.custid, c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.orderdate >= '20080201' AND o.orderdate < '20080301'
GROUP BY o.custid, c.contactname
ORDER BY SUM(o.val) DESC
) AS c2;
```

2. Highlight the written query, and click **Execute**.

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

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

1. In Solution Explorer, double-click **61 - Lab Exercise 2.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, after the **Task 1** description, type the following query:

```
SELECT
p.productid, p.productname, o.orderid
FROM Production.Products AS p
CROSS APPLY
(
SELECT TOP(2)
d.orderid
FROM Sales.OrderDetails AS d
WHERE d.productid = p.productid
ORDER BY d.orderid DESC
) o
ORDER BY p.productid;
```

4. Highlight the written query, and click **Execute**.

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. Highlight the provided T-SQL code after the **Task 2** description, and click **Execute**.
2. In the query pane, type the following query after the provided T-SQL code:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
CROSS APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
ORDER BY c.custid;
```

   Tip: you can make the inline TVF (dbo.fnGetTop3ProductsForCustomer) more flexible by making the number of top rows to return an argument instead of fixing the number to three in the function's code.

3. Highlight the written query, and click **Execute**. Note that the query retrieves 265 rows.

Task 3: Use the OUTER APPLY Operator

1. Highlight the previous query in **Task 2**, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement by replacing the CROSS APPLY operator with the OUTER APPLY operator. The query should look like this:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
OUTER APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
ORDER BY c.custid;
```

4. Highlight the modified query, and click **Execute**.
5. Notice that the query retrieved 267 rows, which is two more rows than the previous query. Observe the result to see two rows with NULL in the columns from the inline TVF.

Task 4: Analyze the OUTER APPLY Operator

1. Highlight the previous query in **Task 3**, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 4** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement to search for a null productid. The query should look like this:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
OUTER APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
WHERE p.productid IS NULL;
```

4. Highlight the modified query, and click **Execute**.
5. Notice that the query now retrieves the two rows that do not occur in the CROSS APPLY query in Task 2.

Task 5: Remove the TVF Created for This Lab

- Highlight the provided T-SQL statement after the **Task 5** description, and click **Execute**.

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

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

1. In Solution Explorer, double-click **71 - Lab Exercise 3.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, after the **Task 1** description, type the following query:

```
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20;
```

4. Highlight the written query, and click **Execute**.

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

1. In the query pane, after the **Task 2** description, type the following query:

```
SELECT
custid
FROM Sales.Customers
WHERE country = 'USA'

EXCEPT

SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20;
```

2. Highlight the written query, and click **Execute**.

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

1. In the query pane, after the **Task 3** description, type the following query:

```
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING SUM(d.qty * d.unitprice) > 10000;
```

2. Highlight the written query, and click **Execute**.

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

1. Highlight the query from **Task 2**, and on the **Edit** menu, click **Copy**.

2. In the query window, click the line after the **Task 4** description, and on the **Edit** menu, click **Paste**.

3. Modify the first SELECT statement so that it selects all customers—not just those from the USA—and include the INTERSECT operator, adding the query from **Task 3**. The query should look like this:

```
SELECT
c.custid
FROM Sales.Customers AS c

EXCEPT

SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20

INTERSECT

SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING SUM(d.qty * d.unitprice) > 10000;
```

4. Highlight the modified query, and click **Execute**.

5. Observe that the total number of rows is 59. In business terms, can you explain which customers are part of the result? Because the INTERSECT operator is evaluated before the EXCEPT operator, the result consists of all customers, except those who bought more than 20 different products and spent more than $10,000.

Task 5: Change the Operator Precedence

1. Highlight the previous query in **Task 4**, and on the **Edit** menu, click **Copy**.

2. In the query window, click the line after the **Task 5** description, and on the **Edit** menu, click **Paste**.

3. Modify the T-SQL statement by adding a set of parentheses around the first two SELECT statements. The query should look like this:

```
(
SELECT
c.custid
```

```
FROM Sales.Customers AS c

EXCEPT

SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20
)

INTERSECT

SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING SUM(d.qty * d.unitprice) > 10000;
```

4. Highlight the provided T-SQL statement, and click **Execute**.
5. Observe that the total number of rows is nine. Is that different to the result of the query in task 4? Yes, because when you added the parentheses, the SQL Server engine first evaluated the EXCEPT operation, and then the INTERSECT operation. In business terms, this query retrieved all customers who did not buy more than 20 distinct products, and who spent more than $10,000.
6. What is the precedence among the set operators? SQL defines the following precedence among the set operations: INTERSECT precedes UNION and EXCEPT, while UNION and EXCEPT are considered equal. In a query that contains multiple set operations, INTERSECT operations are evaluated first, and then operations with the same precedence are evaluated, based on appearance order. Remember that set operations in parentheses are always processed first.
7. 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.