# Lab Answer Key: Module 10: Using Subqueries

# Lab: Using Subqueries

## Exercise 1: Writing Queries That Use Self-Contained Subqueries

### 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\Lab10\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
4. At the command prompt, press any key.

### Task 2: Write a SELECT Statement to Retrieve the Last Order Date

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 click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab10\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 click **Execute**.
6. In the query pane, type the following query after the **Task 1** description:

```
SELECT MAX(orderdate) AS lastorderdate
FROM Sales.Orders;
```

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

### Task 3: Write a SELECT Statement to Retrieve All Orders Placed on the Last Order Date

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

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
orderdate = (SELECT MAX(orderdate) FROM Sales.Orders);
```

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



**Task 4: Observe the T-SQL Statement Provided by the IT Department**

1. Highlight the provided T-SQL statement under the **Task 3** description, and click **Execute**.
2. Modify the query to filter customers whose contact name starts with the letter B. Your query should look like this:

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
custid =
(
SELECT custid
FROM Sales.Customers
WHERE contactname LIKE N'B%'
);
```

3. Highlight the written query, and click **Execute**.
4. Observe the error message:

```
Subquery returned more than 1 value. This is not permitted when the subquery follows =,
!=, <, <= , >, >= or when the subquery is used as an expression.
```

Why did the query fail? It failed because the subquery returned more than one row. To fix this problem, you should replace the = operator with an IN operator.

5. Modify the query so that it uses the IN operator. Your query should look like this:

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
```

```
custid IN
(
SELECT custid
FROM Sales.Customers
WHERE contactname LIKE N'B%'
);
```

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



**Task 5: Write A SELECT Statement to Analyze Each Order's Sales as a Percentage of the Total Sales Amount**

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

```
SELECT
o.orderid,
SUM(d.qty * d.unitprice) AS totalsalesamount,
SUM(d.qty * d.unitprice) /
(
SELECT SUM(d.qty * d.unitprice)
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080501' AND orderdate < '20080601'
) * 100. AS salespctoftotal
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080501' AND orderdate < '20080601'
GROUP BY o.orderid;
```

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



**Result**: After this exercise, you should be able to use self-contained subqueries in T-SQL statements.

## Exercise 2: Writing Queries That Use Scalar and Multiresult Subqueries

**Task 1: Write a SELECT Statement to Retrieve Specific Products**

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, type the following query after the **Task 1** description:

```
SELECT
productid, productname
FROM Production.Products
WHERE
productid IN
(
SELECT productid
FROM Sales.OrderDetails
WHERE qty > 100
);
```

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

### Task 2: Write a SELECT Statement to Retrieve Those Customers Without Orders

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

```
SELECT
custid, contactname
FROM Sales.Customers
WHERE custid NOT IN
(
SELECT custid
FROM Sales.Orders
);
```

2. Highlight the written query, and click **Execute**.
3. Observe the result. Notice there are two customers without an order.

### Task 3: Add a Row and Rerun the Query That Retrieves Those Customers Without Orders

1. Highlight the provided T-SQL statement under the **Task 3** description, and click **Execute**. This code inserts an additional row that has a NULL in the **custid** column of the **Sales.Orders** table.
2. Highlight the query in **Task 2**, and on the **Edit** menu, click **Copy**.
3. In the query window, under the **Task 3** description, click the line after the provided T-SQL statement, and on the **Edit** menu, click **Paste**.

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

5. Notice that you have an empty result despite having two rows when you first ran the query in task 2. Why did you have an empty result this time? There is an issue with the NULL in the new row you added because the **custid** column is the only one that is part of the subquery. The IN operator supports three-valued logic (TRUE, FALSE, UNKNOWN). Before you apply the NOT operator, the logical meaning of UNKNOWN is that you can't tell for sure whether the customer ID appears in the set, because the NULL could represent that customer ID as well as anything else. As a more tangible example, consider the expression 22 NOT IN (1, 2, NULL). If you evaluate each individual expression in the parentheses to its truth value, you will get NOT (FALSE OR FALSE OR UNKNOWN), which translates to NOT UNKNOWN, which evaluates to UNKNOWN. The tricky part is that negating UNKNOWN with the NOT operator still yields UNKNOWN; and UNKNOWN is filtered out in a query filter. In short, when you use the NOT IN predicate against a subquery that returns at least one NULL, the outer query always returns an empty set.

6. To solve this problem, modify the T-SQL statement so that the subquery does not return NULLs. Your query should look like this:

```
SELECT
custid, contactname
FROM Sales.Customers
WHERE custid NOT IN
(
SELECT custid
FROM Sales.Orders
WHERE custid IS NOT NULL
);
```

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

**Result**: After this exercise, you should know how to use multiresult subqueries in T-SQL statements.

## Exercise 3: Writing Queries That Use Correlated Subqueries and an EXISTS Predicate

### Task 1: Write a SELECT Statement to Retrieve the Last Order Date for Each Customer

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, type the following query after the **Task 1** description:

```
SELECT
c.custid, c.contactname,
(
SELECT MAX(o.orderdate)
```

```
FROM Sales.Orders AS o
WHERE o.custid = c.custid
) AS lastorderdate
FROM Sales.Customers AS c;
```

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



**Task 2: Write a SELECT Statement That Uses the EXISTS Predicate to Retrieve Those Customers Without Orders**

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

```
SELECT c.custid, c.contactname
FROM Sales.Customers AS c
WHERE
NOT EXISTS (SELECT * FROM Sales.Orders AS o WHERE o.custid = c.custid);
```

2. Highlight the written query, and click **Execute**.
3. Notice that you achieved the same result as the modified query in exercise 2, task 3, but without a filter to exclude NULLs. Why didn't you need to explicitly filter out NULLs? The EXISTS predicate uses two-valued logic (TRUE, FALSE) and checks only if the rows specified in the correlated subquery exist. Another benefit of using the EXISTS predicate is better performance. The SQL Server engine knows it is enough to determine whether the subquery returns at least one row or none, so it doesn't need to process all qualifying rows.

**Task 3: Write a SELECT Statement to Retrieve Customers Who Bought Expensive Products**

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

```
SELECT c.custid, c.contactname
FROM Sales.Customers AS c
WHERE
EXISTS (
SELECT *
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.custid = c.custid
AND d.unitprice > 100.
AND o.orderdate >= '20080401'
);
```

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

### Task 4: Write a SELECT Statement to Display the Total Sales Amount and the Running Total Sales Amount for Each Order Year

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

```
SELECT
YEAR(o.orderdate) as orderyear,
SUM(d.qty * d.unitprice) AS totalsales,
(
SELECT SUM(d2.qty * d2.unitprice)
FROM Sales.Orders AS o2
INNER JOIN Sales.OrderDetails AS d2 ON d2.orderid = o2.orderid
WHERE YEAR(o2.orderdate) <= YEAR(o.orderdate)
) AS runsales
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY YEAR(o.orderdate)
ORDER BY orderyear;
```

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

### Task 5: Clean the Sales.Customers Table

- Under the **Task 5** description, highlight the provided T-SQL statement, and then click **Execute**.

**Result**: After this exercise, you should have an understanding of how to use a correlated subquery in T-SQL statements.