Module 10: Using Subqueries

Contents:

Module Overview

At this point in the course, you have learned many aspects of the T-SQL SELECT statement, but each query you have written has been a single, self-contained statement. You can also use Microsoft® SQL Server® to nest one query within another—in other words, to form subqueries. In a subquery, the results of the inner query (subquery) are returned to the outer query. This can provide a great deal of flexibility for your query logic. In this module, you will learn to write several types of subqueries.

Objectives

After completing this module, you will be able to:

- Describe the uses for queries that are nested within other queries.
- Write self-contained subqueries that return scalar or multi-valued results.
- Write correlated subqueries that return scalar or multi-valued results.
- Use the EXISTS predicate to efficiently check for the existence of rows in a subquery.

Lesson 1: Writing Self-Contained Subqueries

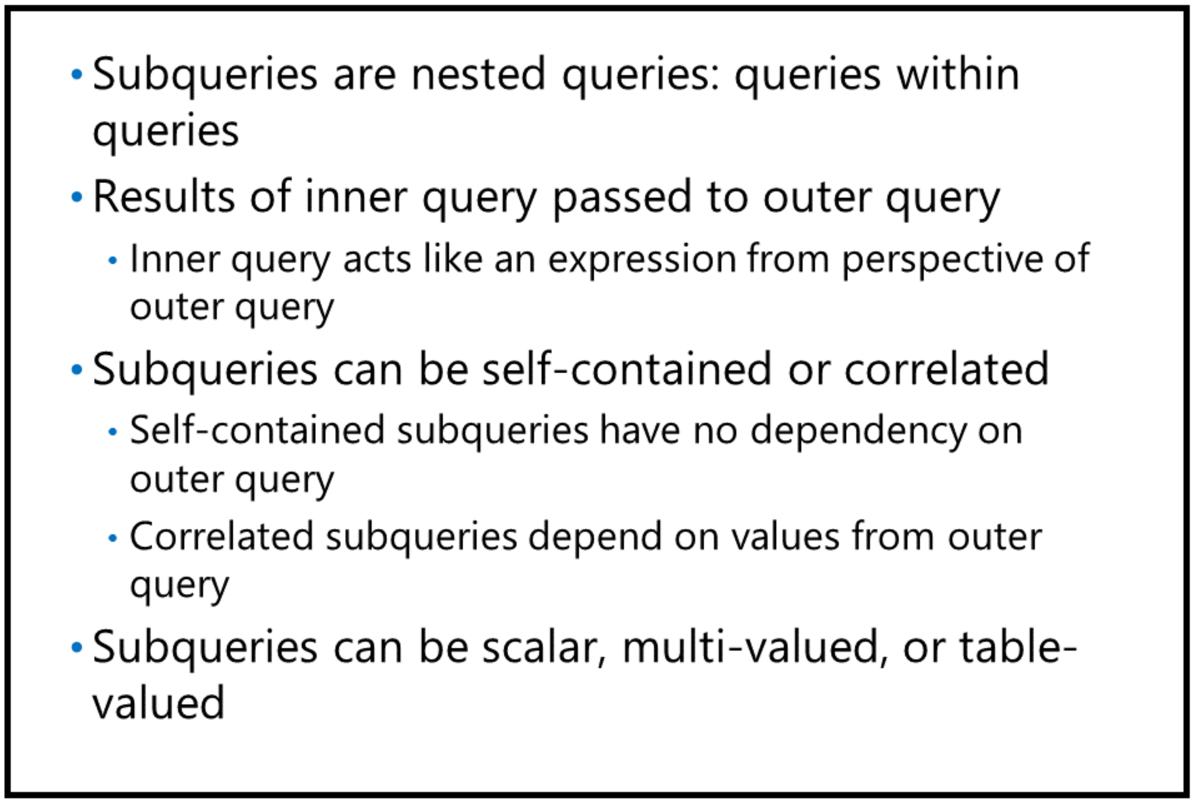
A subquery is a SELECT statement nested within another query. Being able to nest one query within another will enhance your ability to create effective queries in T-SQL. In this lesson, you will learn how to write self-contained queries, which are evaluated once, and provide their results to the outer query. You will learn how to write scalar subqueries, which return a single value, and multi-valued subqueries, which, as their name suggests, can return a list of values to the outer query.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe where subqueries may be used in a SELECT statement.
- Write queries that use scalar subqueries in the WHERE clause of a SELECT statement.
- Write queries that use multi-valued subqueries in the WHERE clause of a SELECT statement.

Working with Subqueries


- Subqueries are nested queries: queries within queries
- Results of inner query passed to outer query
  - Inner query acts like an expression from perspective of outer query
- Subqueries can be self-contained or correlated
  - Self-contained subqueries have no dependency on outer query
  - Correlated subqueries depend on values from outer query
- Subqueries can be scalar, multi-valued, or table-valued

A subquery is a SELECT statement nested, or embedded, within another query. The nested query, which is the subquery, is the inner query. The query containing the nested query is the outer query.

The purpose of a subquery is to return results to the outer query. The form of the results will determine whether the subquery is a scalar or multi-valued subquery:

- Scalar subqueries, like scalar functions, return a single value. Outer queries need to be written to process a single result.
- Multi-valued subqueries return a result much like a single-column table. Outer queries need to be written to handle multiple possible results.

In addition to the choice between scalar and multi-valued subqueries, you may choose to write self-contained subqueries or others that are correlated with the outer query:

- Self-contained subqueries can be written as stand-alone queries, with no dependencies on the outer query. A self-contained subquery is processed once, when the outer query runs and passes its results to that outer query.
- Correlated subqueries reference one or more columns from the outer query and therefore depend on it.

Correlated subqueries cannot be run separately from the outer query.

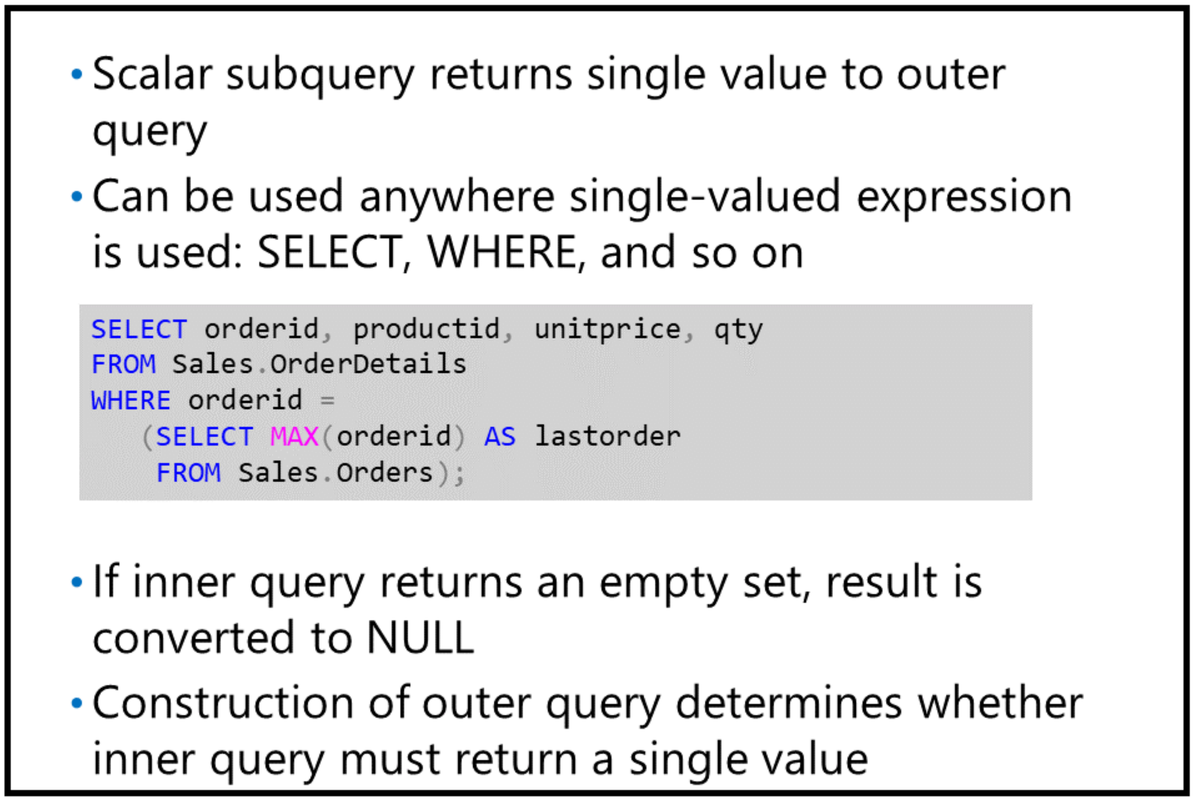
Note: You will learn about correlated subqueries later in this module.

For additional reading about subqueries, see the SQL Server Technical Documentation:

Subquery Fundamentals

http://aka.ms/f6uu08

Writing Scalar Subqueries



- Scalar subquery returns single value to outer query
- Can be used anywhere single-valued expression is used: SELECT, WHERE, and so on

```
SELECT orderid, productid, unitprice, qty
FROM Sales.OrderDetails
WHERE orderid =
   (SELECT MAX(orderid) AS lastorder
    FROM Sales.Orders);
```

- If inner query returns an empty set, result is converted to NULL
- Construction of outer query determines whether inner query must return a single value

A scalar subquery is an inner SELECT statement within an outer query, written to return a single value. Scalar subqueries may be used anywhere in an outer T-SQL statement where a single-valued expression is permitted—such as in a SELECT clause, a WHERE clause, a HAVING clause, or even a FROM clause.

To write a scalar subquery, consider the following guidelines:

- To denote a query as a subquery, enclose it in parentheses.
- Multiple levels of subqueries are supported in SQL Server. In this lesson, we will only consider two-level queries (one inner query within one outer query), but up to 32 levels are supported.
- If the subquery returns an empty set, the result of the subquery is converted and returned as a NULL. Ensure your outer query can gracefully handle a NULL, in addition to other expected results.

Inner Query

```
USE TSQL;
GO
SELECT MAX(orderid) AS lastorder
FROM Sales.Orders;
```

This returns:

```
lastorder
---------
11077
```

You will then write the outer query, using the value returned by the inner query.

Outer and Inner Query

```
SELECT orderid, productid, unitprice, qty
FROM Sales.OrderDetails
WHERE orderid =
        (SELECT MAX(orderid) AS lastorder
        FROM Sales.Orders);
```

This returns (partial result):

```
orderid     productid   unitprice             qty
----------- ----------- --------------------- ------
11077       2           19.00                 24
11077       3           10.00                 4
11077       4           22.00                 1
11077       6           25.00                 1
```

Test the logic of your subquery to ensure it will only return a single value. In the query above, because the outer query used an = operator in the predicate of the WHERE clause, and the subquery returned a single value, the query ran correctly. If an outer query is written to expect a single value, such as by using simple equality operators (=, <, >, and <>, for example), and the inner query returns more than one result, an error will be returned:

```
Msg 512, Level 16, State 1, Line 1
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=,
<, <= , >, >= or when the subquery is used as an expression.
```

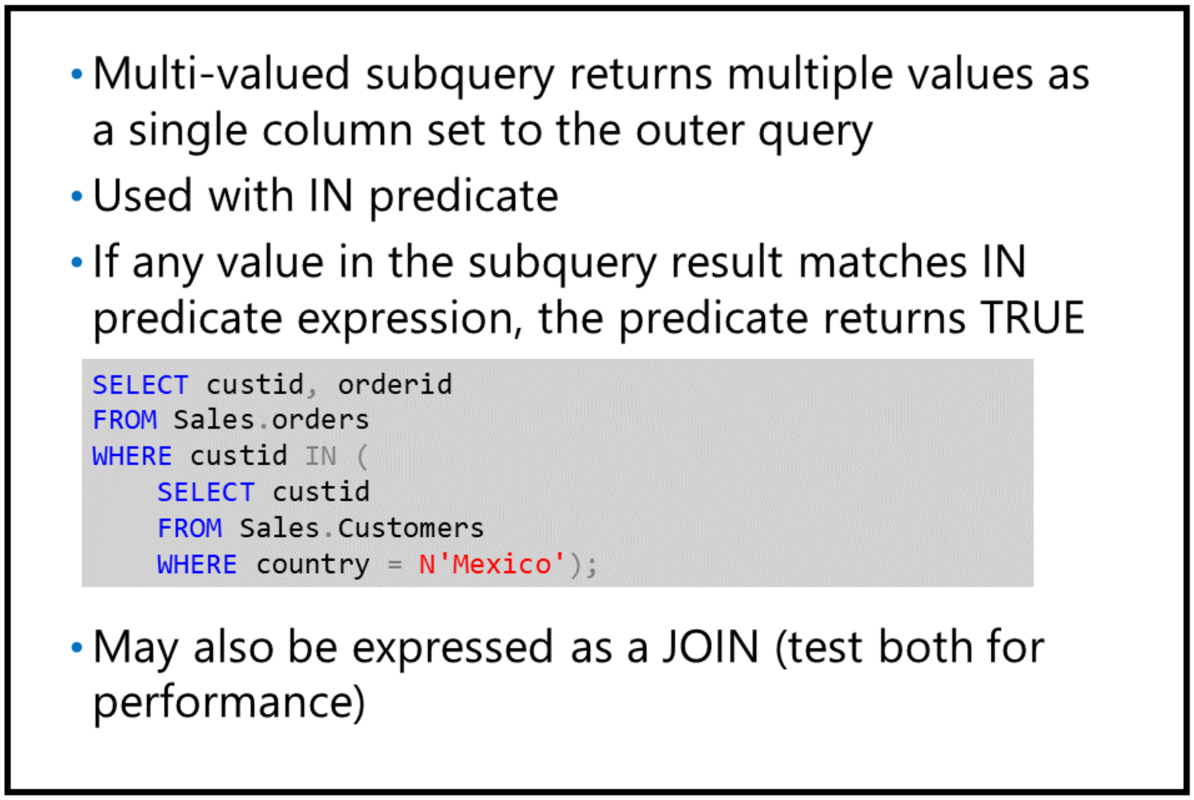
In the case of the Sales.Orders table, orderid is known to be a unique column, enforced in the structure of the table by a PRIMARY KEY constraint.

See *PRIMARY KEY Constraints* in the SQL Server Technical Documentation:

PRIMARY KEY Constraints

http://aka.ms/acq0rx

Writing Multi-Valued Subqueries



- Multi-valued subquery returns multiple values as a single column set to the outer query
- Used with IN predicate
- If any value in the subquery result matches IN predicate expression, the predicate returns TRUE

```
SELECT custid, orderid
FROM Sales.orders
WHERE custid IN (
    SELECT custid
    FROM Sales.Customers
    WHERE country = N'Mexico');
```

- May also be expressed as a JOIN (test both for performance)

As its name suggests, a multi-valued subquery may return more than one result, in the form of a single-column set.

Multi-Valued Subquery

```
SELECT custid, orderid
FROM Sales.orders
WHERE custid IN (
        SELECT custid
        FROM Sales.Customers
        WHERE country =N'Mexico');
```

In this example, if you were to execute only the inner query, you would return the following list of custids for customers in the country of Mexico:

```
custid
------
2
3
13
58
80
```

Logical Structure of Outer Query

```
SELECT custid, orderid
FROM Sales.orders
WHERE custid IN (2,3,13,58,80);
```

The outer query will continue to process the SELECT statement, with the following partial results:

```
custid orderid
------ -----------
2      10308
2      10625
3      10365
3      10507
3      10856
13     10259
58     10322
58     10354
```

As you continue to learn about writing T-SQL queries, you may find scenarios in which multi-valued subqueries are written as SELECT statements using JOINs.

Subquery Rewritten As a Join

```
SELECT c.custid, o.orderid
FROM Sales.Customers AS c JOIN Sales.Orders AS o
        ON c.custid = o.custid
WHERE c.country = N'Mexico‘;
```

Note: In some cases, the database engine will interpret a subquery as a JOIN and execute it accordingly. As you learn more about SQL Server internals, such as execution plans, you may be able to see your queries interpreted this way. For more information about execution plans and query performance, see Microsoft Course 20472-3: *Performance Tuning and Optimizing Microsoft SQL Server Databases*.

Demonstration: Writing Self-Contained Subqueries

In this demonstration, you will see how to write a nested subquery.

Demonstration Steps

Write a Nested Subquery

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:\Demofiles\Mod10** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.
4. At the command prompt, press any key.
5. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows® authentication.
6. On the **File** menu, point to **Open**, and then click **File**.
7. In the **Open File** dialog box, navigate to the **D:\Demofiles\Mod10\Demo** folder, and then double-click **Demo.ssmssln**.
8. In Solution Explorer, expand **Queries**, and then double-click **11 - Demonstration A.sql**.
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. Select the code under the comment **Step 4**, and then click **Execute**. Note the error message.
13. Select the code under the comment **Step 5**, and then click **Execute**.
14. Select the code under the comment **Step 6**, and then click **Execute**.
15. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You are troubleshooting a query. The outer query contains an inner query in its WHERE clause. The first inner query also contains a second inner query in its WHERE clause. Both inner queries are self-contained. The complete query returns an error. How should you approach this task?

Show solution Reset

Break the complete query into its constituent subqueries. Start with the innermost subquery and test to see if it returns the information you expect. Next, test the first inner query. Finally, test the complete query.

Lesson 2: Writing Correlated Subqueries

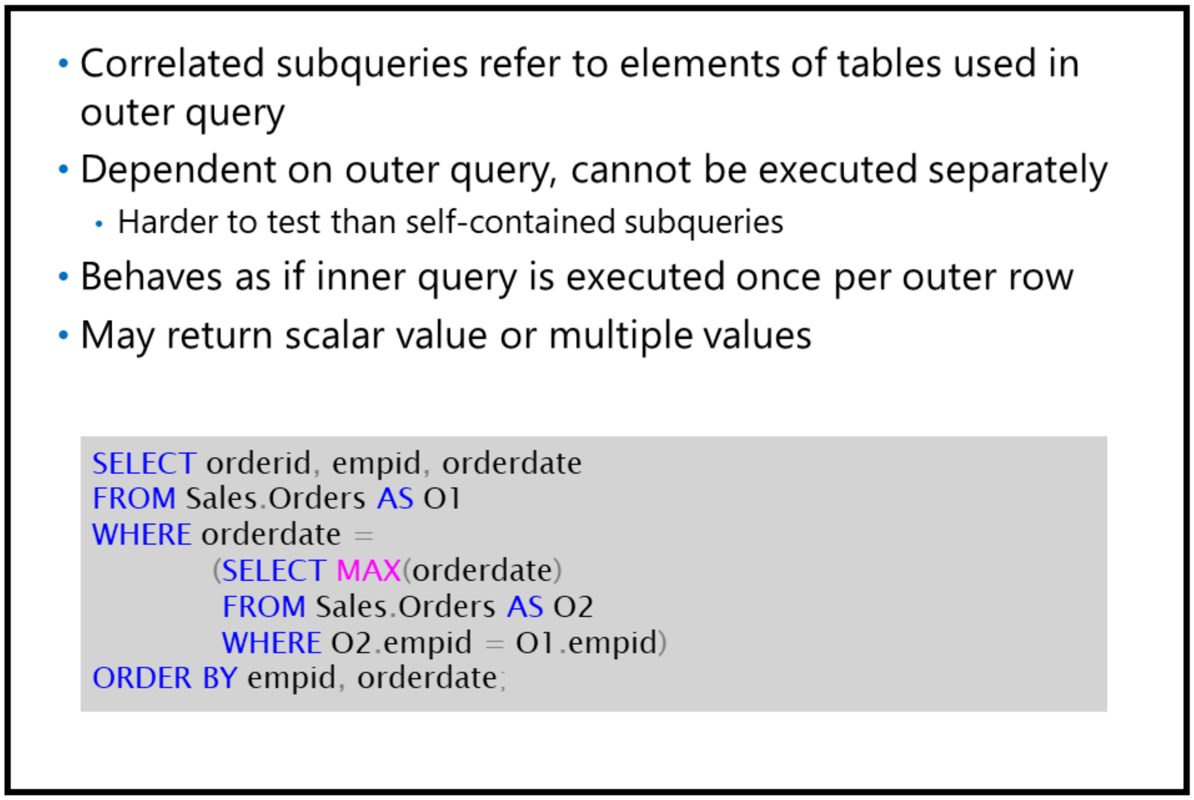
Earlier in this module, you learned how to write self-contained subqueries, in which the inner query is independent of the outer query, executes once, and returns its results to the outer query. Microsoft SQL Server also supports correlated subqueries, in which the inner query receives input from the outer query and conceptually executes once per row in it. In this lesson, you will learn how to write correlated subqueries, in addition to rewriting some types of correlated subqueries as JOINs for performance or logical efficiency.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how correlated subqueries are processed.
- Write queries that use correlated subqueries in a SELECT statement.
- Rewrite some correlated subqueries as JOINs.

Working with Correlated Subqueries



- Correlated subqueries refer to elements of tables used in outer query
- Dependent on outer query, cannot be executed separately
  - Harder to test than self-contained subqueries
- Behaves as if inner query is executed once per outer row
- May return scalar value or multiple values

```
SELECT orderid, empid, orderdate
FROM Sales.Orders AS O1
WHERE orderdate =
          (SELECT MAX(orderdate)
           FROM Sales.Orders AS O2
           WHERE O2.empid = O1.empid)
ORDER BY empid, orderdate;
```

Like self-contained subqueries, correlated subqueries are SELECT statements nested within an outer query. They may also be written as scalar or multi-valued subqueries. They are typically used to pass a value from the outer query to the inner query, to be used as a parameter there.

However, unlike self-contained subqueries, correlated subqueries depend on the outer query to pass values into the subquery as a parameter. This leads to some special considerations when planning their use:

- Correlated subqueries cannot be executed separately from the outer query. This complicates testing and debugging.
- Unlike self-contained subqueries which are processed once, correlated subqueries will run multiple times. Logically, the outer query runs first, and for each row returned, the inner query is processed.

The following example uses a correlated subquery to return the orders with the latest order date for each employee. The subquery accepts an input value from the outer query, uses the input in its WHERE clause, and returns a scalar result to the outer query. Line numbers have been added for use in the subsequent explanation. They do not indicate the order in which the steps are logically processed.

Correlated Subquery Example

```
1.      SELECT orderid, empid, orderdate
2.      FROM Sales.Orders AS O1
3.      WHERE orderdate =
4.              (SELECT MAX(orderdate)
5.               FROM Sales.Orders AS O2
6.               WHERE O2.empid = O1.empid)
7.      ORDER BY empid, orderdate;
```

Line No.	Statement	Description
1	SELECT orderid, empid, orderdate	Columns returned by the outer query.
2	FROM Sales.Orders AS O1	Source table for the outer query. Note the alias.
3	WHERE orderdate =	Predicate used to evaluate the outer rows against the result of the inner query.
4	(SELECT MAX(orderdate)	Column returned by the inner query. Aggregate function returns a scalar value.
5	FROM Sales.Orders AS O2	Source table for the inner query. Note the alias.
6	WHERE O2.empid = O1.empid)	Correlation of empid from the outer query to empid from the inner query. This value will be supplied for each row in the outer query.
7	ORDER BY empid, orderdate;	Sorts the results of the outer query.

The query returns the following results. Note that some employees appear more than once, because they are associated with multiple orders on the latest orderdate:

```
orderid empid orderdate
------- ----- -----------------------
11077   1     2008-05-06 00:00:00.000
11073   2     2008-05-05 00:00:00.000
11070   2     2008-05-05 00:00:00.000
11063   3     2008-04-30 00:00:00.000
11076   4     2008-05-06 00:00:00.000
11043   5     2008-04-22 00:00:00.000
```

```
11045    6      2008-04-23 00:00:00.000
11074    7      2008-05-06 00:00:00.000
11075    8      2008-05-06 00:00:00.000
11058    9      2008-04-29 00:00:00.000
```

Check Your Knowledge

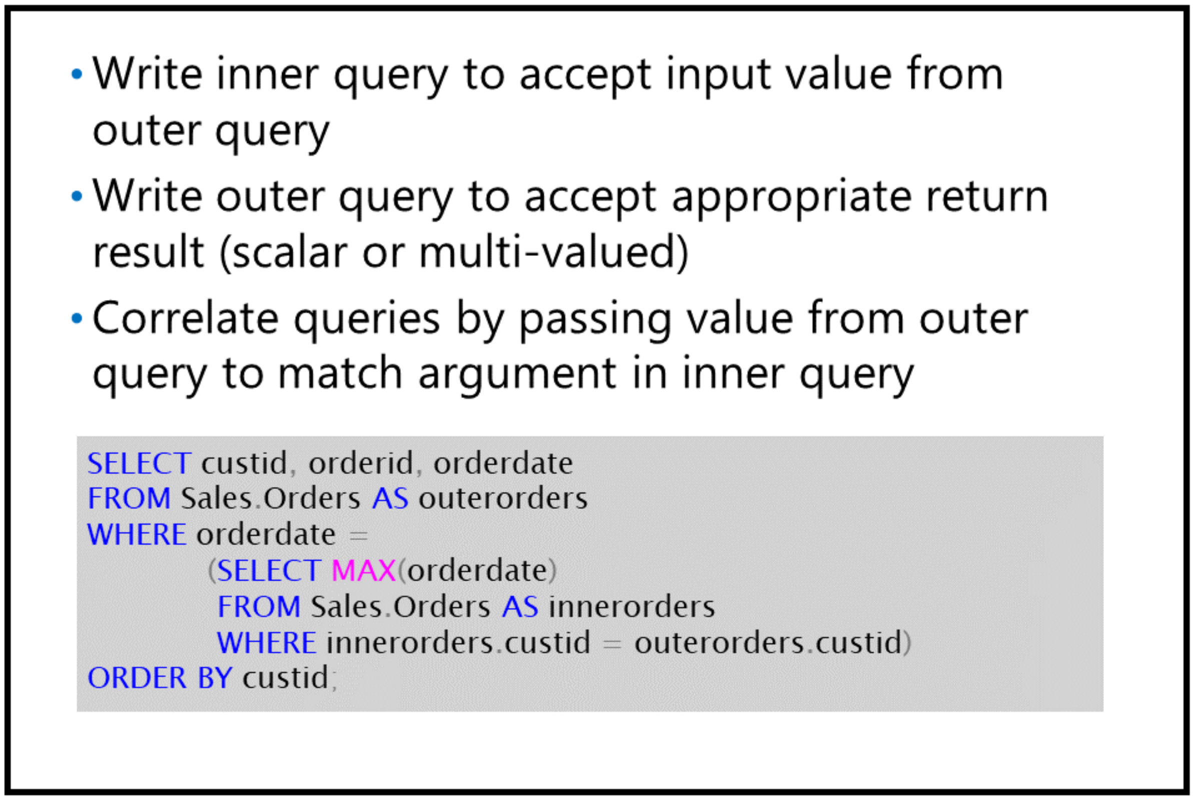
Discovery

Why can't a correlated subquery be executed separately from the outer query?

Show solution Reset

The subquery depends on input from the outer query for its values.

Writing Correlated Subqueries



- Write inner query to accept input value from outer query
- Write outer query to accept appropriate return result (scalar or multi-valued)
- Correlate queries by passing value from outer query to match argument in inner query

```
SELECT custid, orderid, orderdate
FROM Sales.Orders AS outerorders
WHERE orderdate =
          (SELECT MAX(orderdate)
           FROM Sales.Orders AS innerorders
           WHERE innerorders.custid = outerorders.custid)
ORDER BY custid;
```

To write correlated subqueries, consider the following guidelines:

- Write the outer query to accept the appropriate return result from the inner query. If the inner query will be scalar, you can use equality and comparison operators, such as =, <, >, and <>, in the WHERE clause. If the inner query might return multiple values, use an IN predicate. Plan to handle NULL results.
- Identify the column from the outer query that will be passed to the correlated subquery. Declare an alias for the table that is the source of the column in the outer query.
- Identify the column from the inner table that will be compared to the column from the outer table. Create an alias for the source table, as you did for the outer query.

- Write the inner query to retrieve values from its source, based on the input value from the outer query. For example, use the outer column in the WHERE clause of the inner query.

The correlation between the inner and outer queries occurs when the outer value is passed to the inner query for comparison. It's this correlation that gives the subquery its name.

For additional reading about correlated subqueries, see the SQL Server Technical Documentation:

Correlated Subqueries

http://aka.ms/hoxorm

Demonstration: Writing Correlated Subqueries

In this demonstration, you will see how to write a correlated subquery.

Demonstration Steps

Write a Correlated Subquery

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. Select the code under the comment **Step 4**, and then click **Execute**.
6. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Select the best answer

Which of the following statements about correlated subqueries is correct?

To troubleshoot a correlated subquery, execute the inner query first on its own, before placing it into the outer query.

In a correlated subquery, the inner query is run only once, regardless of the number of rows the outer query returns.

In a correlated subquery, the inner query uses data returned by the outer query.

In a correlated subquery, the inner query is executed first, the outer query second.

Check answer Show solution Reset

You cannot troubleshoot a correlated subquery by executing the inner query separately, because the inner query requires data from the outer query. In a correlated subquery, multiple values may be passed into the inner query, which

would require the inner query to run multiple times. The outer query must be executed first, so that data is available to pass to the inner query.

Lesson 3: Using the EXISTS Predicate with Subqueries

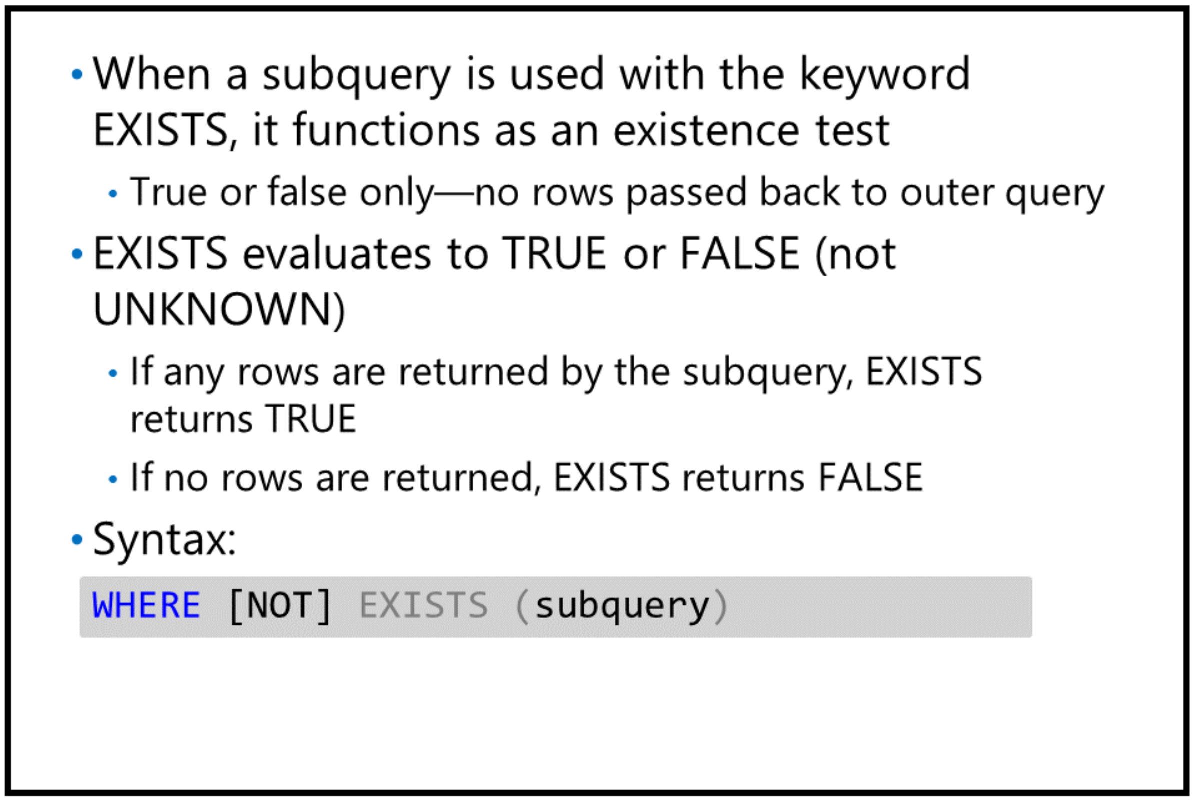
In addition to retrieving values from a subquery, SQL Server provides a mechanism for checking whether any results would be returned from a query. The EXISTS predicate evaluates whether rows exist, but rather than return them, it returns TRUE or FALSE. This is a useful technique for validating data without incurring the overhead of retrieving and counting the results.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how the EXISTS predicate combines with a subquery to perform an existence test.
- Write queries that use EXISTS predicates in a WHERE clause to test for the existence of qualifying rows.

Working with EXISTS



- When a subquery is used with the keyword EXISTS, it functions as an existence test
  - True or false only—no rows passed back to outer query
- EXISTS evaluates to TRUE or FALSE (not UNKNOWN)
  - If any rows are returned by the subquery, EXISTS returns TRUE
  - If no rows are returned, EXISTS returns FALSE
- Syntax:

```
WHERE [NOT] EXISTS (subquery)
```

When a subquery is invoked by an outer query using the EXISTS predicate, SQL Server handles the results of the subquery differently to how it does elsewhere in this module. Rather than retrieve a scalar value or a multi-valued list from the subquery, EXISTS simply checks to see if there are any rows in the results.

Conceptually, an EXISTS predicate is equivalent to retrieving the results, counting the rows returned, and comparing the count to zero. Compare the following queries, which will return details about employees who are associated with orders:

Using COUNT in a Subquery

```
SELECT empid, lastname
FROM HR.Employees AS e
WHERE (SELECT COUNT(*)
                FROM Sales.Orders AS O
                WHERE O.empid = e.empid)>0;
```

Using EXISTS in a Subquery

```
SELECT empid, lastname
FROM HR.Employees AS e
WHERE EXISTS(   SELECT *
                FROM Sales.Orders AS O
                WHERE O.empid = e.empid);
```

In the first example, the subquery must count every occurrence of each empid found in the Sales.Orders table, and compare the count results to zero, simply to indicate that the employee has associated orders.

In the second query, EXISTS returns TRUE for an empid as soon as one has been found in the Sales.Orders table —a complete accounting of each occurrence is unnecessary.

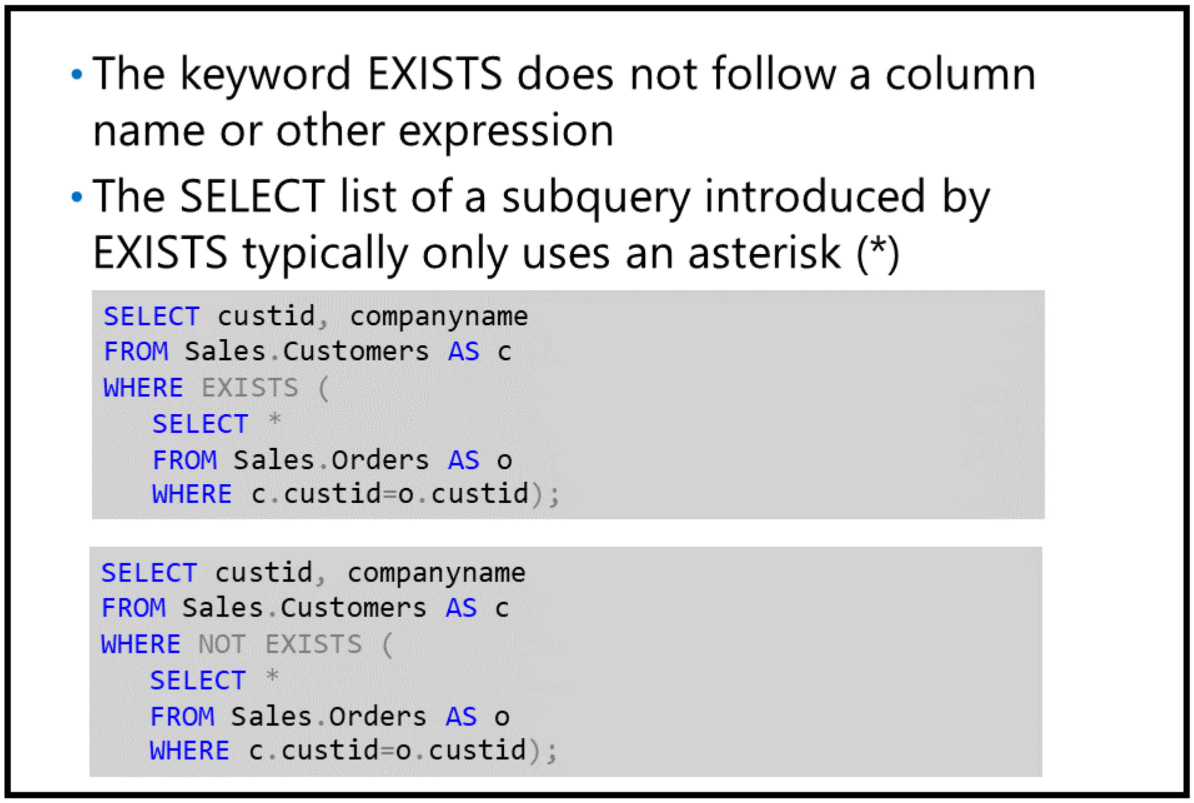
Note: From the perspective of logical processing, the two query forms are equivalent. From a performance perspective, the database engine may treat the queries differently as it optimizes them for execution. Consider testing each one for your own usage.

NOT EXISTS Example

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

Once again, SQL Server will not have to return data about the related orders for customers who have placed orders. If a customer ID is found in the Sales.Orders table, NOT EXISTS evaluates to FALSE and the evaluation quickly completes.

Writing Queries Using EXISTS with Subqueries



- The keyword EXISTS does not follow a column name or other expression
- The SELECT list of a subquery introduced by EXISTS typically only uses an asterisk (*)

```
SELECT custid, companyname
FROM Sales.Customers AS c
WHERE EXISTS (
    SELECT *
    FROM Sales.Orders AS o
    WHERE c.custid=o.custid);
```

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

To write queries that use EXISTS with subqueries, consider the following guidelines:

- The keyword EXISTS directly follows WHERE. No column name (or other expression) needs to precede it, unless NOT is also used.
- Within the subquery following EXISTS, the SELECT list only needs to contain (*). No rows are returned by the subquery, so no columns need to be specified.

See *Subqueries with EXISTS* in the SQL Server Technical Documentation:

Subqueries with EXISTS

http://aka.ms/q812le

Demonstration: Writing Subqueries Using EXISTS

In this demonstration, you will see how to write queries using EXISTS and NOT EXISTS.

Demonstration Steps

Write Queries Using EXISTS and NOT EXISTS

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 **Step 4a**, and then click **Execute**.
6. Select the code under the comment **Step 4b**, and then click **Execute**.
7. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Discovery

The Human Resources database has recently been extended to record the skills possessed by employees. Employees have added their skills to the database by using a web-based user interface. You want to find employees who have not yet added their skills. You have the following query:

```
SELECT e.EmployeeID, e.FirstName
FROM HumanResources.Employees AS e
WHERE NOT EXISTS (
  SELECT s.EmployeeID, s.SkillName, s.SkillCategory
  FROM HumanResources.Skills AS s
  WHERE e.EmployeeID = s.EmployeeID);
```

How can you improve the query?

Show solution Reset

In the inner query SELECT clause, replace the list of columns with "*". The original query would successfully return the desired results, but the list of columns is not necessary and clutters the query. You can use "*" to simplify the command. Because the EXISTS function does not return rows but only true or false, this does not negatively impact on the performance of the query.

Lab: Using Subqueries

Scenario

As a business analyst for Adventure Works, you are writing reports using corporate databases stored in SQL Server. You have been handed a set of business requirements for data and will write T-SQL queries to retrieve the specified data from the databases. Due to the complexity of some of the requests, you will need to embed subqueries into your queries to return results in a single query.

Objectives

After completing this lab, you will be able to:

- Write queries that use subqueries.
- Write queries that use scalar and multiresult set subqueries.
- Write queries that use correlated subqueries and the EXISTS predicate.

Lab Setup

Estimated Time: 60 minutes

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

User name: **AdventureWorks\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Queries That Use Self-Contained Subqueries

Scenario

The sales department needs some advanced reports to analyze sales orders. You will write different SELECT statements that use self-contained subqueries.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Write a SELECT Statement to Retrieve the Last Order Date
3. Write a SELECT Statement to Retrieve All Orders Placed on the Last Order Date
4. Observe the T-SQL Statement Provided by the IT Department
5. Write A SELECT Statement to Analyze Each Order's Sales as a Percentage of the Total Sales Amount

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\Lab10\Starter** folder as Administrator.

Detailed Steps ▲

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

1. Open the project file **D:\Labfiles\Lab10\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 maximum order date from the table **Sales.Orders**.
3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab10\Solution\52 - Lab Exercise 1 - Task 1 Result.txt.

Detailed Steps ▲

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

1. Write a SELECT statement to return the **orderid**, **orderdate**, **empid**, and **custid** columns from the **Sales.Orders** table. Filter the results to include only orders where the date order equals the last order date. (Hint: use the query in task 1 as a self-contained subquery.)
2. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab10\Solution\53 - Lab Exercise 1 - Task 2 Result.txt.

Detailed Steps ▲

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

1. The IT department has written a T-SQL statement that retrieves the orders for all customers whose contact name starts with a letter I:

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

2. Execute the query and observe the result.
3. Modify the query to filter customers whose contact name starts with a letter B.
4. Execute the query. What happened? What is the error message? Why did the query fail?
5. Apply the needed changes to the T-SQL statement so that it will run without an error.
6. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab10\Solution\54 - Lab Exercise 1 - Task 3 Result.txt.

Detailed Steps ▲

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

1. Write a SELECT statement to retrieve the **orderid** column from the **Sales.Orders** table and the following calculated columns:
   - **totalsalesamount** (based on the **qty** and **unitprice** columns in the **Sales.OrderDetails** table).
   - **salespctoftotal** (percentage of the total sales amount for each order divided by the total sales amount for all orders in a specific period).
2. Filter the results to include only orders placed in May 2008.
3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab10\Solution\55 - Lab Exercise 1 - Task 4 Result.txt.

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

Scenario

The marketing department would like to prepare materials for different groups of products and customers, based on historic sales information. You have to prepare different SELECT statements that use a subquery in the WHERE clause.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Retrieve Specific Products
2. Write a SELECT Statement to Retrieve Those Customers Without Orders
3. Add a Row and Rerun the Query That Retrieves Those Customers Without Orders

Detailed Steps ▲

Task 1: Write a SELECT Statement to Retrieve Specific Products

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. Filter the results to include only products that were sold in high quantities (more than 100) for a specific order line.

3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab10\Solution\62 - Lab Exercise 2 -Task 1 Result.txt.

Detailed Steps ▲

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

1. Write a SELECT statement to retrieve the **custid** and **contactname** columns from the **Sales.Customers** table. Filter the results to include only those customers who do not have any placed orders.
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab10\Solution\63 - Lab Exercise 2 - Task 2 Result.txt. Remember the number of rows in the results.

Detailed Steps ▲

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

1. The IT department has written a T-SQL statement that inserts an additional row in the **Sales.Orders** table. This row has a NULL in the **custid** column:

```
INSERT INTO Sales.Orders (
custid, empid, orderdate, requireddate, shippeddate, shipperid, freight,
shipname, shipaddress, shipcity, shipregion, shippostalcode, shipcountry)
VALUES
(NULL, 1, '20111231', '20111231', '20111231', 1, 0,
'ShipOne', 'ShipAddress', 'ShipCity', 'RA', '1000', 'USA')
```

2. Execute this query exactly as written inside a query window.
3. Copy the T-SQL statement you wrote in task 2 and execute it.
4. Observe the result. How many rows are in the result? Why?
5. Modify the T-SQL statement to retrieve the same number of rows as in task 2. (Hint: you have to remove the rows with an unknown value in the **custid** column.)
6. Execute the modified statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab10\Solution\64 - Lab Exercise 2 - Task 3 Result.txt.

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

Scenario

The sales department would like to have some additional reports to display different analyses of existing customers. Because the requests are complex, you will need to use correlated subqueries.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Retrieve the Last Order Date for Each Customer
2. Write a SELECT Statement That Uses the EXISTS Predicate to Retrieve Those Customers Without Orders
3. Write a SELECT Statement to Retrieve Customers Who Bought Expensive Products
4. Write a SELECT Statement to Display the Total Sales Amount and the Running Total Sales Amount for Each Order Year
5. Clean the Sales.Customers Table

Detailed Steps ▲

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

1. Open the T-SQL script **71 - Lab Exercise 3.sql**. Make sure you are connected to the **TSQL** database.
2. Write a SELECT statement to retrieve the **custid** and **contactname** columns from the **Sales.Customers** table. Add a calculated column named **lastorderdate** that contains the last order date from the **Sales.Orders** table for each customer. (Hint: you have to use a correlated subquery.)
3. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab10\Solution\72 - Lab Exercise 3 - Task 1 Result.txt.

Detailed Steps ▲

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

1. Write a SELECT statement to retrieve all customers that do not have any orders in the **Sales.Orders** table, similar to the request in exercise 2, task 3. However, this time use the EXISTS predicate to filter the results to include only those customers without an order. Also, you do not need to explicitly check that the **custid** column in the **Sales.Orders** table is not NULL.
2. Execute the written statement and compare the results that you achieved with the recommended results

shown in the file D:\Labfiles\Lab10\Solution\73 - Lab Exercise 3 - Task 2 Result.txt.

3. Why didn't you need to check for a NULL?

Detailed Steps ▲

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

1. Write a SELECT statement to retrieve the **custid** and **contactname** columns from the **Sales.Customers** table. Filter the results to include only customers who placed an order on or after April 1, 2008, and ordered a product with a price higher than $100.
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab10\Solution\74 - Lab Exercise 3 - Task 3 Result.txt.

Detailed Steps ▲

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

1. Running aggregates accumulate values over time. Write a SELECT statement to retrieve the following information for each year:
   - The order year.
   - The total sales amount.
   - The running total sales amount over the years. That is, for each year, return the sum of sales amount up to that year. So, for example, for the earliest year (2006), return the total sales amount; for the next year (2007), return the sum of the total sales amount for the previous year and 2007.
2. The SELECT statement should have three calculated columns:
   - **orderyear**, representing the order year. This column should be based on the **orderyear** column from the **Sales.Orders** table.
   - **totalsales**, representing the total sales amount for each year. This column should be based on the **qty** and **unitprice** columns from the **Sales.OrderDetails** table.
   - **runsales**, representing the running sales amount. This column should use a correlated subquery.
3. Execute the T-SQL code and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab10\Solution\75 - Lab Exercise 3 - Task 4 Result.txt.

Detailed Steps ▲

Task 5: Clean the Sales.Customers Table

1. Delete the row added in exercise 2 using the provided SQL statement:

```
DELETE Sales.Orders
WHERE custid IS NULL;
```

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

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

Module Review and Takeaways

In this module, you have learned how to:

- Describe the uses for queries that are nested within other queries.
- Write self-contained subqueries that return scalar or multi-valued results.
- Write correlated subqueries that return scalar or multi-valued results.
- Use the EXISTS predicate to efficiently check for the existence of rows in a subquery.

Review Question(s)

Check Your Knowledge

Discovery

Can a correlated subquery return a multi-valued set?

Show solution Reset

Yes.

Check Your Knowledge

Discovery

What type of subquery may be rewritten as a JOIN?

Show solution Reset

Correlated subqueries.

Check Your Knowledge

Discovery

Which columns should appear in the SELECT list of a subquery following the EXISTS predicate?

Show solution Reset

Only a * needs to be specified. No actual columns will be retrieved.