MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 6: Working with Subqueries

Contents

Microsoft

Project Lead: Cheryl Hoople
Instructional Designer: Cheryl Hoople
Technical Lead: LeRoy Tuttle
Program Manager: LeRoy Tuttle
Graphic Artist: Kimberly Jackson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Editorial Contributor: Elizabeth Reese
Copy Editor: Bill Jones (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Tools Specialist: Julie Challenger
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: Testing Testing 123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
45 Minutes

Lab:
30 Minutes

This module presents advanced query techniques, which include nested and correlated subqueries. It describes when and how to use a subquery and how to use subqueries to break down and perform complex queries.

At the end of this module, you will be able to:

- Describe when and how to use a subquery.
- Use subqueries to break down and perform complex queries.

Materials and Preparation

Required Materials

To teach this course, you need the following materials:

- Microsoft® PowerPoint® file 2017A_06.ppt.
- The C:\Moc\Demo\Ex_06.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials.
- Complete the lab.

Module Strategy

Use the following strategy to present this module:

- Introduction to Subqueries

  Define subqueries and present basic facts and guidelines related to using them. Point out that subqueries may be less efficient than joins because subqueries specify the order in which to retrieve data. Joins allow the query optimizer in Microsoft SQL Server™ 2000 to retrieve data in the most efficient way.

- Using a Subquery as a Derived Table

  Describe how a derived table is a special use of a subquery in a FROM clause to which an alias or user-specified name refers. Explain when to use it. Review the example.

- Using a Subquery as an Expression

  Describe when and how to use a subquery as an expression. Review the example.

- Using a Subquery to Correlate Data

  Discuss how correlated queries are processed. Use the graphic to illustrate how correlated subqueries are evaluated. Point out the difference between a correlated subquery and a nested subquery. In a correlated subquery, the inner query is evaluated repeatedly, once for each row of the outer query.

  Describe how to use a subquery to correlated data by mimicking JOIN and HAVING clauses. Review the examples.

- Using a Subquery with EXISTS and NOT EXISTS

  Present the EXISTS and NOT EXISTS keywords in the context of their use with correlated subqueries. Review the example.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2071A, *Querying Microsoft SQL Server 2000 with Transact-SQL*.

Lab Setup

There are no lab setup requirements that affect replication or customization.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Overview

Slide Objective
To provide a brief overview of the topics covered in this module.

Lead-in
In this module, you will learn about advanced query techniques.

- **Introduction to Subqueries**
- **Using a Subquery as a Derived Table**
- **Using a Subquery as an Expression**
- **Using a Subquery to Correlate Data**
- **Using the EXISTS and NOT EXISTS Clauses**

This module presents advanced query techniques, which include nested and correlated subqueries, and how they can be used to modify data. It describes when and how to use a subquery and how to use subqueries to break down and perform complex queries.

At the end of this module, you will be able to:

- Describe when and how to use a subquery.
- Use subqueries to break down and perform complex queries.

Introduction to Subqueries

Slide Objective
To discuss whether to use subqueries.

Lead-in
Subqueries are a series of SELECT statements. Often, you can rewrite subqueries as joins.

- **Why to Use Subqueries**
  - To break down a complex query into a series of logical steps
  - To answer a query that relies on the results of an other query
- **Why to Use Joins Rather Than Subqueries**
  - SQL Server executes joins faster than subqueries
- **How to Use Subqueries**

A *subquery* is a SELECT statement nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery. Often you can rewrite subqueries as joins and use subqueries in place of an expression.

An *expression* is a combination of identifiers, values, and operators that SQL Server evaluates to obtain a result.

Why to Use Subqueries

You use subqueries to break down a complex query into a series of logical steps and, as a result, to solve a problem with a single statement. Subqueries are useful when your query relies on the results of another query.

Why to Use Joins Rather Than Subqueries

Often, a query that contains subqueries can be written as a join. Query performance may be similar with a join and a subquery. The query optimizer usually optimizes subqueries so that it uses the sample execution plan that a semantically equivalent join would use. The difference is that a subquery may require the query optimizer to perform additional steps, such as sorting, which may influence the processing strategy.

Using joins typically allows the query optimizer to retrieve data in the most efficient way. If a query does not require multiple steps, it may not be necessary to use a subquery.

Delivery Tip
Review each fact and guideline to consider when using subqueries.

How to Use Subqueries

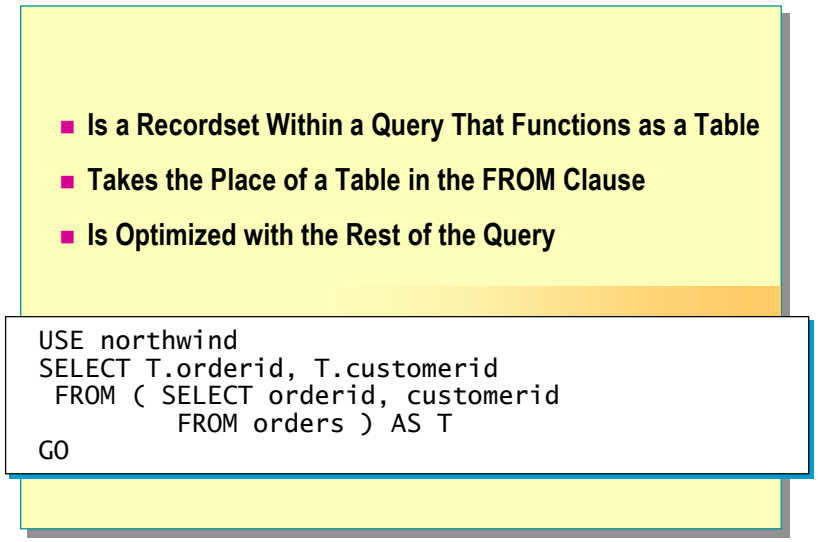
When you decide to use subqueries, consider the following facts and guidelines:

- You must enclose subqueries in parentheses.
- You can use a subquery in place of an expression as long as a single value or list of values is returned. You can use a subquery that returns a multi-column record set in place of a table or to perform the same function as a join.
- You cannot use subqueries that retrieve columns that contain **text** and **image** data types.
- You can have subqueries within subqueries, nesting up to 32 levels. The limit varies based on available memory and the complexity of other expressions in the query. Individual queries may not support nesting up to 32 levels.

Using a Subquery as a Derived Table

Slide Objective
To describe how to use a subquery as a derived table.

Lead-in
You create a derived table by using a subquery in place of a table in a FROM clause.



You create a *derived table* by using a subquery in place of a table in a FROM clause. A derived table is a special use of a subquery in a FROM clause to which an alias or user-specified name refers. The result set of the subquery in the FROM clause forms a table that the outer SELECT statement uses.

Example

This example uses a subquery to create a derived table in the inner part of the query that the outer part queries. The derived table itself is functionally equivalent to the whole query, but it is separated for illustrative purposes.

```
USE northwind
SELECT T.orderid, T.customerid
 FROM ( SELECT orderid, customerid
        FROM orders ) AS T
GO
```

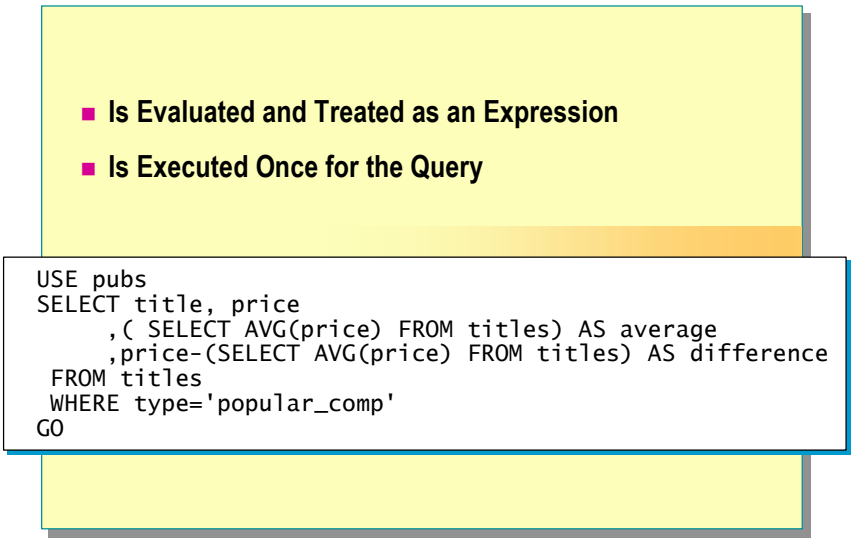
When used as a derived table, consider that a subquery:

- Is a recordset within a query that functions as a table.
- Takes the place of a table in the FROM clause.
- Is optimized with the rest of the query.

Using a Subquery as an Expression

Slide Objective
To describe how to use a subquery as an expression.

Lead-in
You can substitute a subquery wherever you use an expression in SELECT, UPDATE, INSERT, and DELETE statements.



Delivery Tip
Point out that subqueries that return a list of values replace an expression in a WHERE clause that contains the IN keyword.

In Transact-SQL, you can substitute a subquery wherever you use an expression. The subquery must evaluate to a scalar value, or to a single column list of values. Subqueries that return a list of values replace an expression in a WHERE clause that contains the IN keyword.

When used as an expression, consider that a subquery:

- Is evaluated and treated as an expression. The query optimizer often evaluates an expression as equivalent to a join connecting to a table that has one row.
- Is executed once for the entire statement.

Example

This example returns the price of a popular computer book, the average price of all books, and the difference between the price of the book and the average price of all books.

```
USE pubs
SELECT title, price
      ,(SELECT AVG(price) FROM titles) AS average
      ,price-(SELECT AVG(price) FROM titles) AS difference
 FROM titles
 WHERE type='popular_comp'
GO
```

◆ Using a Subquery to Correlate Data

Slide Objective
To describe how to use a subquery to correlate data.

Lead-in
A correlated subquery can be used as a dynamic expression that changes for each row of an outer query.

- **Evaluating a Correlated Subquery**
- **Mimicking a JOIN Clause**
- **Mimicking a HAVING Clause**

You can use a correlated subquery as a dynamic expression that changes for each row of an outer query.

The query processor performs the subquery for each row in the outer query, one row at a time, which is in turn evaluated as an expression for that row and passed to the outer query. The correlated subquery is effectively a JOIN between the dynamically executed subquery and the row from the outer query.

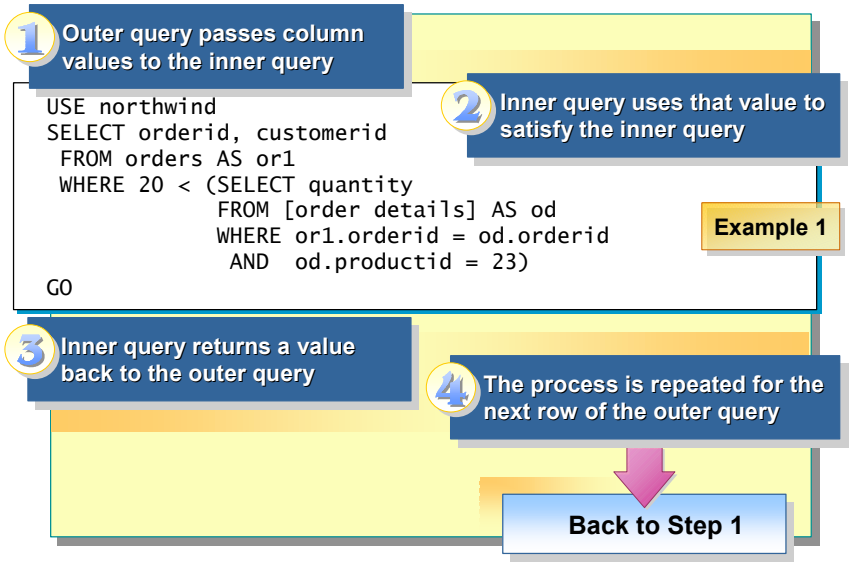
You can typically rewrite a query in a number of ways and still obtain the same results. Correlated subqueries break down complex queries into two or more simple, related queries.

Tip You can easily recognize correlated subqueries. A column from a table inside the subquery is compared to a column from a table outside the subquery.

Evaluating a Correlated Subquery

Slide Objective
To discuss how correlated subqueries are processed.

Lead-in
When you create a correlated subquery, the inner subqueries are evaluated repeatedly, once for each row of the outer query.



Delivery Tip
Compare correlated subqueries to nested subqueries.

When you create a correlated subquery, the inner subqueries are evaluated repeatedly, once for each row of the outer query:

- SQL Server executes the inner query for each row that the outer query selects.
- SQL Server compares the results of the subquery to the results outside the subquery.

Example 1

This example returns a list of customers who ordered more than 20 pieces of product number 23.

```
USE northwind
SELECT orderid, customerid
 FROM orders AS or1
 WHERE 20 < (SELECT quantity
             FROM [order details] AS od
             WHERE or1.orderid = od.orderid
              AND od.productid = 23)
GO
```

Result

orderid	customerid
10337	FRANK
10348	WANDK
10396	FRANK
10402	ERNSH
10462	CONSH
.	
.	
.	

(11 row(s) affected)

Correlated subqueries return a single value or a list of values for each row specified by the FROM clause of the outer query. The following steps describe how the correlated subquery is evaluated in example 1:

1. The outer query passes a column value to the inner query.

   The column value that the outer query passes to the inner query is the **orderid**. The outer query passes the first **orderid** in the **orders** table to the inner query.

2. The inner query uses the values that the outer query passes.

   Each **orderid** in the **orders** table is evaluated to determine whether an identical **orderid** is found in the **order details** table. If the first **orderid** matches an **orderid** in the **order details** table and that **orderid** purchased product number 23, then the inner query returns that **orderid** to the outer query.

3. The inner query returns a value back to the outer query.

   The WHERE clause of the outer query further evaluates the **orderid** that purchased product number 23 to determine whether the quantity ordered exceeds 20.

4. The process is repeated for the next row of the outer query.

   The outer query passes the second **orderid** in the **orders** table to the inner query, and SQL Server repeats the evaluation process for that row.

Example 2

This example returns a list of products and the largest order ever placed for each product in the **order details** table. Notice that this correlated subquery references the same table as the outer query; the optimizer will generally treat this as a self-join.

```
USE northwind
SELECT DISTINCT productid, quantity
 FROM [order details] AS ord1
 WHERE quantity = ( SELECT MAX(quantity)
                    FROM [order details] AS ord2
                    WHERE ord1.productid = ord2.productid )
GO
```


Result

productid	quantity
50	40
67	40
4	50
9	50
11	50
.	
.	
.	

(77 row(s) affected)

Mimicking a JOIN Clause

Slide Objective
To describe how to use a correlated subquery to mimic a JOIN.

Lead-in
You can use a correlated subquery to produce the same results as a JOIN.

- **Correlated Subqueries Can Produce the Same Result as a JOIN Clause**
- **Joins Let the Query Optimizer Determine How to Correlate Data Most Efficiently**

Example 1

```
USE pubs
SELECT DISTINCT t1.type
 FROM titles AS t1
 WHERE t1.type IN
   (SELECT t2.type
     FROM titles AS t2
     WHERE t1.pub_id <> t2.pub_id)
GO
```

Delivery Tip
The key to understanding correlated subquery syntax is understanding the use of table aliases. The table aliases show you which tables are correlated.

You can use a correlated subquery to produce the same results as a JOIN, for example, selecting data from a table referenced in the outer query.

Note You usually can rephrase correlated subqueries as joins. Using joins rather than correlated subqueries allows the query optimizer to determine the most efficient way to correlate the data.

Example 1

This example uses a correlated subquery to find the types of books published by more than one publisher. To prevent ambiguity, aliases are required to distinguish the two different roles in which the **titles** table appears.

```
USE pubs
SELECT DISTINCT t1.type
 FROM titles AS t1
 WHERE t1.type IN
   (SELECT t2.type
     FROM titles AS t2
     WHERE t1.pub_id <> t2.pub_id)
GO
```

Result

```
type
------------
business
psychology

(2 row(s) affected)
```

Example 2

This example returns the same results as example 1 by using a self-join instead of a correlated subquery.

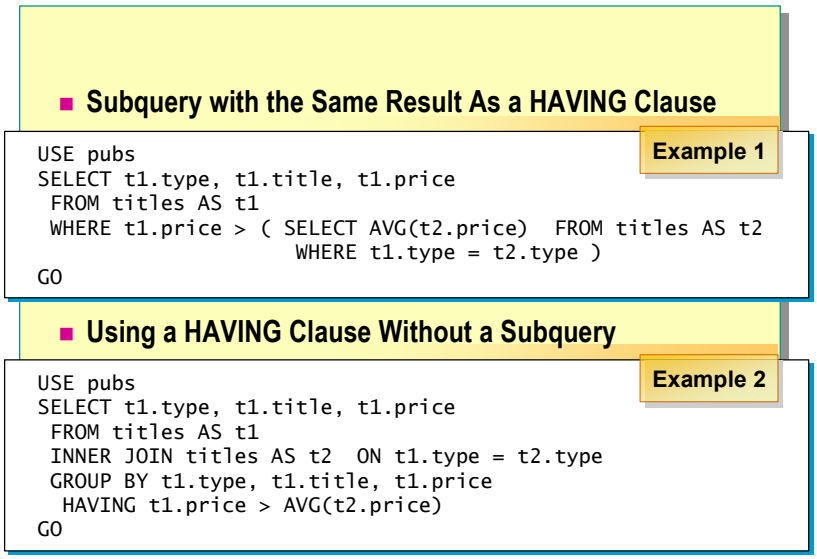
```
USE pubs
SELECT DISTINCT t1.type
 FROM titles AS t1
 INNER JOIN titles AS t2
  ON t1.type = t2.type
 WHERE t1.pub_id <> t2.pub_id
GO
```

Delivery Tip
Use SQL Query Analyzer to execute both JOIN examples and show the different execution plans.

Mimicking a HAVING Clause

Slide Objective
To describe how to mimic a HAVING clause.

Lead-in
You can use a correlated subquery to produce the same results as a query that uses the HAVING clause.



You can use a correlated subquery to produce the same results as a query that uses the HAVING clause.

Example 1

This example finds all titles that have a price greater than the average price for books of the same type. For each possible value of **t1**, SQL Server evaluates the subquery and includes the row in the results if the price value of that row is greater than the calculated average. It is not necessary to group by type explicitly, because the rows for which average price is calculated are restricted by the WHERE clause in the subquery.

Delivery Tip
Use SQL Query Analyzer to execute both examples and verify that they produce the same results.

```
USE pubs
SELECT t1.type, t1.title, t1.price
 FROM titles AS t1
 WHERE t1.price > ( SELECT AVG(t2.price)
                   FROM titles AS t2
                   WHERE t1.type = t2.type )
GO
```

Resulttype	title
Business	The Busy Executive's Database Guide
Business	Straight Talk About Computers
mod_cook	Silicon Valley Gastronomic Treats
popular_comp	But Is It User Friendly?
Psychology	Computer Phobic AND Non-Phobic Individuals: Behavior Variations
Psychology	Prolonged Data Deprivation: Four Case Studies
trad_cook	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean

```
(7 row(s) affected)
```

Example 2

This example produces the same result set as example 1, but uses a self-join with GROUP BY and HAVING clauses.

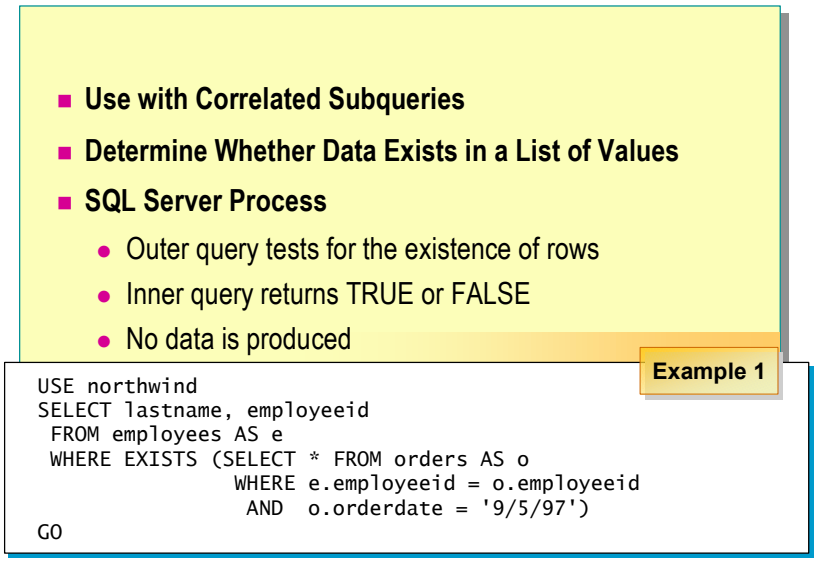
```
USE pubs
SELECT t1.type, t1.title, t1.price
 FROM titles AS t1
 INNER JOIN titles AS t2
  ON t1.type = t2.type
 GROUP BY t1.type, t1.title, t1.price
  HAVING t1.price > AVG(t2.price)
GO
```

Note You can write correlated subqueries that produce the same results as a JOIN or HAVING clause, but the query processor may not implement them in the same manner.

Using the EXISTS and NOT EXISTS Clauses

Slide Objective
To discuss how the EXISTS and NOT EXISTS operators are used with correlated subqueries.

Lead-in
You can use the EXISTS and NOT EXISTS operators to determine whether data exists in a list of values.



You can use the EXISTS and NOT EXISTS operators to determine whether data exists in a list of values.

Use with Correlated Subqueries

Use the EXISTS and NOT EXISTS operators with correlated subqueries to restrict the result set of an outer query to rows that satisfy the subquery. The EXISTS and NOT EXISTS operators return TRUE or FALSE, based on whether rows are returned for subqueries.

Determine Whether Data Exists in a List of Values

When a subquery is introduced with the EXISTS operator, SQL Server tests whether data that matches the subquery exists. No rows are actually retrieved. SQL Server terminates the retrieval of rows when it knows that at least one row satisfies the WHERE condition in the subquery.

SQL Server Process

When SQL Server processes subqueries that use the EXISTS or NOT EXISTS operator:

- The outer query tests for the existence of rows that the subquery returns.
- The subquery returns either a TRUE or FALSE value based on the given condition in the query.
- The subquery does not produce any data.

Partial Syntax

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

Example 1

This example uses a correlated subquery with an EXISTS operator in the WHERE clause to return a list of employees who took orders on 4/10/2000.

Delivery Tip
Execute these two examples with STATISTICS TIME set ON to compare the processing time.

```
USE northwind
SELECT lastname, employeeid
 FROM employees AS e
 WHERE EXISTS ( SELECT * FROM orders AS o
                 WHERE e.employeeid = o.employeeid
                  AND o.orderdate = '9/5/1997' )
GO
```

Result

lastname	employeeid
Peacock	4
King	7

```
(2 row(s) affected)
```

Example 2

This example returns the same result set as example 1 and shows that you could use a join operation rather than a correlated subquery. Note that the query needs the DISTINCT keyword to return only a single row for each employee.

```
USE northwind
SELECT DISTINCT lastname, e.employeeid
 FROM orders AS o
 INNER JOIN employees AS e
  ON o.employeeid = e.employeeid
 WHERE o.orderdate = '9/5/1997'
GO
```

Result

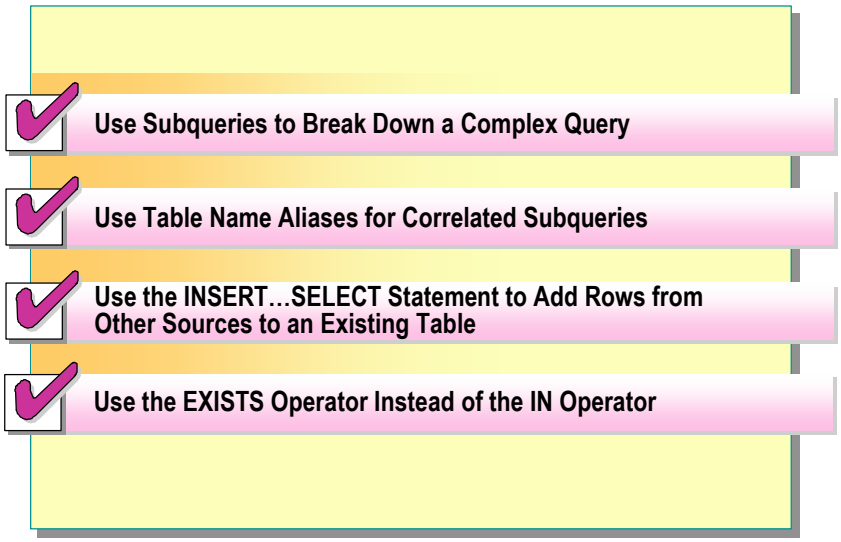
lastname	employeeid
Peacock	4
King	7

```
(2 row(s) affected)
```

Recommended Practices

Slide Objective
To list the recommended practices for data retrieval and modification.

Lead-in
The following recommended practices should help you perform advanced queries.



The following recommended practices should help you perform advanced queries:

- Use subqueries to break down a complex query. You can solve a problem with a single statement by using subqueries. Subqueries are useful when your query relies on the results of another query.
- Use table name aliases for correlated subqueries. SQL Server requires that aliases be used to reference the ambiguous table names in order to distinguish between the inner and outer tables.
- Use the INSERT…SELECT statement to add rows from other sources to an existing table. Using the INSERT…SELECT statement is more efficient than writing multiple, single-row INSERT statements.
- Use the EXISTS operator instead of the IN operator wherever possible so that it is not necessary to retrieve the full result set of the subquery.

Additional information on the following topics is available in SQL Server Books Online.

Topic	Search on
Using subqueries	“creating subqueries”
Correlating tables	“using table aliases”
	“creating table aliases”
Using a subquery instead of an expression	“subqueries used in place of an expression”

Lab A: Working with Subqueries

Slide Objective
To introduce the lab.

Lead-in
In this lab, you will write and execute subqueries used as an expression, a join, and to correlate data.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Use a subquery as a derived table.
- Use a subquery as an expression.
- Use a subquery to correlate data.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2071A\Labfiles\L06.
- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L06\Answers.
- The **library** database installed.

Lab Setup

None.

For More Information

If you require help in executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **library** database schema.
- SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious nwtraders.msft domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.x.1
SQLAdmin2	Denver	192.168.x.2
SQLAdmin3	Perth	192.168.x.3
SQLAdmin4	Brisbane	192.168.x.4
SQLAdmin5	Lisbon	192.168.x.5
SQLAdmin6	Bonn	192.168.x.6
SQLAdmin7	Lima	192.168.x.7
SQLAdmin8	Santiago	192.168.x.8
SQLAdmin9	Bangalore	192.168.x.9
SQLAdmin10	Singapore	192.168.x.10
SQLAdmin11	Casablanca	192.168.x.11
SQLAdmin12	Tunis	192.168.x.12
SQLAdmin13	Acapulco	192.168.x.13
SQLAdmin14	Miami	192.168.x.14
SQLAdmin15	Auckland	192.168.x.15
SQLAdmin16	Suva	192.168.x.16
SQLAdmin17	Stockholm	192.168.x.17
SQLAdmin18	Moscow	192.168.x.18
SQLAdmin19	Caracas	192.168.x.19
SQLAdmin20	Montevideo	192.168.x.20
SQLAdmin21	Manila	192.168.x.21
SQLAdmin22	Tokyo	192.168.x.22
SQLAdmin23	Khartoum	192.168.x.23
SQLAdmin24	Nairobi	192.168.x.24

Estimated time to complete this lab: 30 minutes

Exercise 1
Using a Subquery as a Derived Table

In this exercise, you will write a query that uses a derived table and joins the derived table to another table. You will also separate the query into individual steps to show how a derived table is processed.
C:\Moc\2071A\Labfiles\L06\Answers contains completed scripts for this exercise.

► To execute a query that uses a derived table

In this procedure, you will write and execute a query that uses a derived table and returns the **juvenile.adult_member_no** column and the number of juveniles for each adult member who has more than three juvenile members.

Answer_DerivedTab.sql is a completed script for this procedure.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the nwtraders.msft classroom domain)
Password	**Password**

2. Open SQL Query Analyzer and, if requested, log in to the (local) server with Microsoft Windows® Authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

3. In the **DB** list, click **library**.

4. Type the following query that uses a derived table.

```
USE Library
SELECT d.adult_member_no, a.expr_date, d.No_Of_Children
 FROM adult AS a
 INNER JOIN (
            SELECT adult_member_no, COUNT(*) AS No_Of_Children
             FROM juvenile
             GROUP BY adult_member_no
             HAVING COUNT(*) > 3
            ) AS d
   ON a.member_no = d.adult_member_no
GO
```

5. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set. The number of rows returned may vary.

```
adult_member_no   expr_date                 No_Of_Children
1                 2000-06-22 12:43:51.800   4
3                 2000-06-24 12:43:51.800   4
67                2000-06-27 12:43:51.800   4

(28 row(s) affected)
```

Note When you answer questions later in this exercise, remember that this is the result set of the original derived table query.

► To write the derived table query as two separate queries

In this procedure, you will rewrite and execute the previous query as two separate queries to show how the query that uses a derived table is processed.

1. Type the following query that returns the **adult_member_no** column data, calculates the number of children that each adult member has, and returns only the rows containing adult members that have more than three children from the **juvenile** table.

```
USE Library
SELECT adult_member_no, COUNT(*) AS No_Of_Children
 FROM juvenile
 GROUP BY adult_member_no
 HAVING COUNT(*) > 3
GO
```

2. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set. The number of rows returned may vary.

```
adult_member_no   No_Of_Children
1                 4
3                 4
5                 4

(248 row(s) affected)
```

3. Compare the results of the query in step 1 of this procedure and the original derived table query from the previous procedure.

   What are the similarities between the two results?

   Both queries return 248 rows in the result set. Both queries also return the adult_member_no and No_Of_Children columns.

4. Type the following query that retrieves the **expr_date** column data from the **adult** table.

```
USE library
SELECT a.expr_date
 FROM adult AS a
GO
```

5. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set.

```
expr_date
2000-06-22 12:43:51.800
2000-06-24 12:43:51.800
2000-06-26 12:43:51.800

(5000  row(s) affected)
```

6. Compare the results of the query in step 4 of this procedure and the original derived table query.

What are the similarities between the two results?

Both queries return the expr_date column.

What are the differences between the two results?

The preceding query returns 5000 rows, while the query using the derived table only returns 248 rows.

► To rewrite the derived table query by using a join

In this procedure, you will rewrite and execute the original derived table query as a join of two separate queries to show that you can obtain the same results as using a derived table.

1. Type the following query.

```
USE Library
SELECT j.adult_member_no, a.expr_date
      ,COUNT(*) AS No_Of_Children
 FROM adult AS a
 INNER JOIN juvenile AS j
  ON a.member_no = j.adult_member_no
 GROUP BY adult_member_no, expr_date
 HAVING COUNT(*) > 3
GO
```

2. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set. The number of rows returned may vary.

```
adult_member_no   expr_date                 No_Of_Children
1                 2000-06-22 12:43:51.800   4
3                 2000-06-24 12:43:51.800   4
5                 2000-06-27 12:43:51.800   4

(248 row(s) affected)
```

3. Compare the results of the query in step 1 of this procedure and the results of the original derived table query.

   Do both queries return the same results?

   Yes.

Exercise 2
Using a Subquery as an Expression

In this exercise, you will write queries that use single values and multiple values to restrict the result sets of the outer query and to combine multiple processing steps into one SELECT statement.
C:\Moc\2071A\Labfiles\L06\Answers contains completed scripts for this exercise.

► To use a single-value subquery

In this procedure, you will write and execute a query that returns **member.firstname**, **member.lastname**, **loanhist.isbn**, and **loanhist.fine_paid** for members who have paid the highest recorded fines for all books.
Answer_Highpay.sql is a completed script for this procedure.

1. Type a query that returns the largest recorded value in the **loanhist.fine_paid** column.

```
USE Library
SELECT MAX(fine_paid)
 FROM loanhist
GO
```

2. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set.

```
8.0000

(1 row(s) affected)

Warning: Null value is eliminated by an aggregate or
other SET operation.
```

► To use a single-value subquery as part of a search condition

In this procedure, you will use a single-value subquery as part of a search condition.

1. Write a query that joins the **member** and **loanhist** tables and returns the **firstname**, **lastname**, **isbn**, and **fine_paid** for each row.
2. Use the query from step 1 of the previous procedure as selection criteria in the WHERE clause to return only those rows from the join in which the fine that is paid equals the largest value that was ever recorded for all books.
3. Include the DISTINCT keyword in your query to eliminate entries for members who have paid this fine on several occasions.

```
USE library
SELECT DISTINCT firstname, lastname, isbn, fine_paid
 FROM member AS m
 INNER JOIN loanhist AS lh
  ON m.member_no = lh.member_no
 WHERE lh.fine_paid = (SELECT MAX(fine_paid) FROM loanhist)
GO
```

4. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following result set. The number of rows returned may vary.

```
Firstname      lastname       isbn        fine_paid
Michael        Nash           883         8.0000
Robert         Rothenberg     330         8.0000

(2 row(s) affected)

Warning: Null value is eliminated by an aggregate or other
SET operation.
```

► To use a query to make a list of values

In this procedure, you will write and execute a query on the **title**, **loan**, and **reservation** tables that returns four columns: **title_no**, **title**, **isbn**, and **Total Reserved**. The **Total Reserved** column is the per-isbn (book) count of books on reserve with more than 50 reservations and less than five copies of the book. Group the results by **title_no**, **title**, and **isbn**. Answer_SubqIn.sql is a completed script for this procedure.

1. Write a query that returns the isbn numbers of books from the **reservations** table that have more than fifty reservations.

```
USE library
SELECT isbn
 FROM reservation
 GROUP BY isbn
 HAVING COUNT(*)> 50
GO
```

2. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following partial result set. The number of rows returned may vary.

```
Isbn
1
43
246
288
330
.
.
.
(11 row(s) affected)
```

► To use a multiple-value subquery

1. Write an outer query that returns the **title_no**, **title**, **isbn**, and **Total Reserved** columns in which the **Total Reserved** column is the number of records for each group of **title_no**, **title**, and **isbn**. To do this:
   a. Restrict the rows that form the groups in the outer query by specifying books that have less than five copies.
   b. Use the IN keyword as part of the WHERE clause against the list of values generated by the query in step 1 of the previous procedure.

```
USE Library
SELECT t.title_no, title, l.isbn
      ,count(*) AS 'Total Reserved'
 FROM title AS t
 INNER JOIN loan AS l
  ON t.title_no = l.title_no
 INNER JOIN reservation AS r
  ON r.isbn = l.isbn
 WHERE r.isbn IN
      ( SELECT isbn
         FROM reservation
         GROUP BY isbn
         HAVING COUNT(*)> 50 )
  AND  l.copy_no < 5
 GROUP BY t.title_no, title, l.isbn
GO
```

2. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following partial result set. The number of rows returned may vary.

```
title_no  Title                          isbn   Total Reserved
1         Last of the Mohicans           1      197
25        The Black Tulip                246    196
33        The First 100,000 Prime Numbers 330   196
.
.
.
8 row(s) affected)
```

Exercise 3
Using a Subquery to Correlate Data

In this exercise, you will write queries that use correlated subqueries to restrict the result set of the outer query and to combine multiple processing steps into one SELECT statement. C:\Moc\2071A\Labfiles\L06\Answers contains completed scripts for this exercise.

► To use a correlated subquery

In this procedure, you will create a query that uses a correlated subquery to calculate a value, based on data from the outer query, and then uses that value as part of a comparison. You will query the **member** and **loanhist** tables to return a list of library members who have fines that total more than $5.00. A correlated subquery calculates the fines for each member. Answer_Fineof5.sql is a completed script for this procedure.

Note You also can write this query with a join and a GROUP BY or HAVING clause instead of a correlated subquery. Answer_Finejoin.sql is a completed script for this alternate solution.

---

1. Write a query that returns the **member_no** and **lastname** columns of the **member** table, by using a table alias for the **member** table.

```
USE Library
SELECT member_no, lastname
 FROM member AS m
GO
```

2. Execute the query to verify that it returns the desired results.
3. Write a query that calculates the total fines for each member as recorded in the **loanhist** table. To do this:
   a. Use an alias for the **loanhist** table.
   b. Correlate the **member.member_no** column of the outer query to the **loanhist.member_no** column of the inner query in a subquery.
   c. Use a comparison operator in the WHERE clause of the outer query to select those members who have fines that total more than $5.00.

```
USE library
SELECT member_no, lastname
 FROM member AS m
 WHERE 5 < ( SELECT SUM(fine_assessed)
              FROM loanhist AS lh
              WHERE m.member_no = lh.member_no )
GO
```

4. Execute the query to verify that it returns the desired results.

Result

Your result will look similar to the following partial result set. The number of rows returned may vary.

```
member_no      lastname
-------------- ------------------
204            Graff
372            Miksovsky
1054           Miksovsky
1094           O'Brian
.
.
.
(41 row(s) affected)

Warning: Null value is eliminated by an aggregate or other
SET operation.
```

Review

Slide Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Introduction to Subqueries
- Using a Subquery as a Derived Table
- Using a Subquery as an Expression
- Using a Subquery to Correlate Data
- Using the EXISTS and NOT EXISTS Clauses

Use this scenario to answer these questions and review module topics.

Ask students whether they need clarification on any topic. The Duluth Mutual Life health care organization has a database that tracks information about doctors and their patients. The database includes the following tables.

Doctor table

Column	Data type and constraints
doc_id	char(9), PRIMARY KEY
fname	char(20)
lname	char(25)
specialty	char(25)
phone	char(10)

Patient table

Column	Data type and constraints
pat_id	char(9), PRIMARY KEY
fname	char(20)
lname	char(25)
insurance_company	char(25)
phone	char(10)

Casefile table

Column	Data type and constraints
admission_date	**datetime,** PRIMARY KEY **(composite)**
pat_id	**char(9)**, PRIMARY KEY **(composite),** FOREIGN KEY **to patient.pat_id**
doc_id	**char(9)**, FOREIGN KEY **to doctor.doc_id**
diagnosis	**varchar(150)**

Based on this table structure, answer the following questions.

1. How, with a single query, can you produce a list of all cases that were admitted on the first chronological date in the database?

   Use a single-value subquery with the MIN function to determine the oldest date of admission. Compare the result of the subquery to the admission date for each case with the WHERE clause.

2. You want to know the total number of hospital admissions, listed by patient name. How can you determine this? What are the advantages or disadvantages of your method?

   You could write a SELECT statement with a correlated subquery that calculates the total admissions for each patient by using the COUNT function.

```
SELECT pat_id, pat_name
 ,(SELECT count(*) FROM casefile C WHERE C.pat_id =
P.pat_id)
FROM patient AS P
```

   This could also be done by using a join with the GROUP BY clause and the COUNT function. The subquery method might be less efficient than the GROUP BY method, but it is logically clearer.