

Module 3: Retrieving Data

Contents

Overview	1
Retrieving Data by Using the SELECT Statement	2
Filtering Data	8
Formatting Result Sets	20
How Queries Are Processed	28
How Queries Are Cached Automatically	29
Performance Considerations	31
Recommended Practices	32
Lab A: Retrieving Data and Manipulating Result Sets	33
Review	45



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2000 Microsoft Corporation. All rights reserved.

Microsoft, BackOffice, MS-DOS, PowerPoint, Visual Studio, Windows, Windows Media, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted.

Other product and company names mentioned herein may be the trademarks of their respective owners.

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

Labs:
45 Minutes

This module provides students with the knowledge and skills to perform basic queries by using the SELECT statement, which includes sorting data, eliminating duplicates, and changing the format of the result set. The module concludes with a description of how queries are processed.

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

- Retrieve data from tables by using the SELECT statement.
- Filter data by using different search conditions to use with the WHERE clause.
- Format result sets.
- Describe how queries are processed.
- Describe performance considerations that affect retrieving data.

Materials and Preparation

Required Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2017A_03.ppt.
- The C:\MOC\2071A\Demo\Ex_03.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 all demonstrations.
- Complete the labs.

Module Strategy

Use the following strategy to present this module:

- **Retrieving Data by Using the SELECT Statement**
Explain how to retrieve specific columns and rows by using the SELECT statement and the WHERE clause.
- **Filtering Data**
Describe how to restrict the number of rows that are returned by using search conditions in the WHERE clause. Discuss comparison and logical operators, character strings, range of values, list of values, and unknown values.
- **Formatting Result Sets**
Describe how to format the result set to improve readability by sorting data, eliminating duplicate data, changing column names to aliases, and using literals. Explain that these formatting options do not change the data, only the presentation of it.
- **How Queries Are Processed**
Describe how queries are processed. Mention that all queries follow the same process before they execute and that Microsoft SQL Server™ 2000 can store some of the processing for subsequent execution of the same query. Then briefly describe the benefits of cached queries and the ways in which queries can be cached.
- **Performance Considerations**
Discuss some of the issues that affect the performance of SQL Server when you perform basic queries.

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 an overview of the module topics and objectives.

Lead-in

In this module, you will learn how to retrieve data by using basic queries.

- Retrieving Data by Using the SELECT Statement
- Filtering Data
- Formatting Result Sets
- How Queries Are Processed
- Performance Considerations

This module provides students with the knowledge and skills to perform basic queries by using the SELECT statement, which includes sorting data, eliminating duplicates, and changing the format of the result set. The module concludes with a description of how queries are processed.

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

- Retrieve data from tables by using the SELECT statement.
- Filter data by using different search conditions to use with the WHERE clause.
- Format results sets.
- Describe how queries are processed.
- Describe performance considerations that affect retrieving data.

◆ Retrieving Data by Using the SELECT Statement

Slide Objective

To list the topics that the following sections covers.

Lead-in

Retrieving data from tables includes using the SELECT statement, which involves specifying columns and rows.

- Using the SELECT Statement
- Specifying Columns
- Using the WHERE Clause to Specify Rows

Before you can work with data, you must select the data that you want to retrieve from your tables. You can use the SELECT statement to specify the columns and rows of data that you want to retrieve from tables.

Trainer Materials
for Microsoft Certified
Trainer Use Only

Using the SELECT Statement

Slide Objective

To discuss using the SELECT statement to retrieve data from a table.

Lead-in

Use the SELECT statement to retrieve data.

- **Select List Specifies the Columns**
- **WHERE Clause Specifies the Rows**
- **FROM Clause Specifies the Table**

Partial Syntax

```
SELECT [ALL | DISTINCT] <select_list>  
FROM {<table_source>} [,...n]  
WHERE <search_condition>
```

Use the SELECT statement to retrieve data.

Partial Syntax

```
SELECT [ ALL | DISTINCT ] <select_list>  
FROM {<table_source>} [,...n]  
[ WHERE <search_condition> ]
```

Use the SELECT statement to specify the columns and rows that you want to be returned from a table:

- The select list specifies the columns to be returned.
- The WHERE clause specifies the rows to return. When you use search conditions in the WHERE clause, you can restrict the number of rows by using comparison operators, character strings, and logical operators as search conditions.
- The FROM clause specifies the table from which columns and rows are returned.

Specifying Columns


Slide Objective

To show how to select columns within a table.

Lead-in

You can retrieve particular columns from a table by listing them in the select list.

```
USE northwind
SELECT employeeid, lastname, firstname, title
FROM employees
GO
```



employeeid	lastname	firstname	title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative
7	King	Robert	Sales Representative
8	Callahan	Laura	Inside Sales Coordinator
9	Dodsworth	Anne	Sales Representative

You can retrieve particular columns from a table by listing them in the select list.

The select list contains the columns, expressions, or keywords to select or the local variable to assign. The options that can be used in the select list include:

Partial Syntax

<select_list> ::=

```
{
  *
  | { table_name | view_name | table_alias }. *
  | { column_name | expression | IDENTITYCOL | ROWGUIDCOL }
    [ [AS] column_alias ]
  | column_alias = expression
} [,...n]
```

When you specify columns to retrieve, consider the following facts and guidelines:

- The select list retrieves and displays the columns in the specified order.
- Separate the column names with commas, except for the last column name.
- Avoid or minimize the use of an asterisk (*) in the select list. An asterisk is used to retrieve all columns from a table.

Example

This example retrieves the **employeeid**, **lastname**, **firstname**, and **title** columns of all employees from the **employees** table.

```
USE northwind
SELECT employeeid, lastname, firstname, title
FROM employees
GO
```

Result

employeeid	lastname	firstname	title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative
7	King	Robert	Sales Representative
8	Callahan	Laura	Inside Sales Coordinator
9	Dodsworth	Anne	Sales Representative

(9 row(s) affected)

Trainer Materials
for Microsoft Certified
Trainer Use Only

Using the WHERE Clause to Specify Rows


Slide Objective

To introduce how to retrieve rows by using the WHERE clause.

Lead-in

Using the WHERE clause, you can retrieve specific rows based on given search conditions.

```
USE northwind
SELECT employeeid, lastname, firstname, title
FROM employees
WHERE employeeid = 5
GO
```



employeeid	lastname	firstname	title
5	Buchanan	Steven	Sales Manager

Delivery Tip

Compare the result set from the previous slide to the one in this slide.

Point out that using the WHERE clause restricts the number of rows that are returned.

Using the WHERE clause, you can retrieve specific rows based on given search conditions. The search conditions in the WHERE clause can contain an unlimited list of predicates.

```
<search_condition> ::=
{ [ NOT ] <predicate> | ( <search_condition> ) }
[ { AND | OR } [ NOT ] { <predicate> | ( <search_condition> ) } ]
{ [,...n]
```

The predicate placeholder lists the expressions that can be included in the WHERE clause. The options that can be contained in a predicate include:

The syntax that is listed here is found in the "Search Condition (T-SQL)" topic in SQL Server Books Online, not in the "SELECT statement" topic.

```
<predicate> ::=
{
  expression { = | < | > | >= | <= } expression
  | string_expression [NOT] LIKE string_expression
  [ESCAPE 'escape_character']
  | expression [NOT] BETWEEN expression AND expression
  | expression IS [NOT] NULL
  | CONTAINS
  ( { column | * }, '<contains_search_condition>' )
  | FREETEXT ( { column | * }, 'freetext_string' )
  | expression [NOT] IN (subquery | expression [,...n])
  | expression { = | < | > | >= | <= }
  { ALL | SOME | ANY } (subquery)
  | EXISTS (subquery)
}
```

When you specify rows with the WHERE clause, consider the following facts and guidelines:

- Place single quotation marks around all **char**, **nchar**, **varchar**, **nvarchar**, **text**, **datetime**, and **smalldatetime** data.
- Use a WHERE clause to limit the number of rows that are returned when you use the SELECT statement.

Example

This example retrieves the **employeeid**, **lastname**, **firstname**, and **title** columns from the **employees** table for the employee with an **employeeid** of 5.

```
USE northwind
SELECT employeeid, lastname, firstname, title
FROM employees
WHERE employeeid = 5
GO
```

Result

employeeid	lastname	firstname	title
5	Buchanan	Steven	Sales Manager

(1 row(s) affected)

Trainer Materials
for Microsoft Certified
Trainer Use Only

◆ Filtering Data

Slide Objective

To describe the different types of search conditions to use with the WHERE clause.

Lead-in

You sometimes want to limit the results that a query returns.

- Using Comparison Operators
- Using String Comparisons
- Using Logical Operators
- Retrieving a Range of Values
- Using a List of Values as Search Criteria
- Retrieving Unknown Values

You sometimes want to limit the results that a query returns. You can limit the results by specifying search conditions in a WHERE clause to filter data. There is no limit to the number of search conditions that you can include in a SELECT statement. The following table describes the type of filter and the corresponding search condition that you can use to filter data.

Type of filter	Search condition
Comparison operators	=, >, <, >=, <=, and <>
String comparisons	LIKE and NOT LIKE
Logical operators: combination of conditions	AND, OR
Logical operator: negations	NOT
Range of values	BETWEEN and NOT BETWEEN
Lists of values	IN and NOT IN
Unknown values	IS NULL and IS NOT NULL

Using Comparison Operators

Slide Objective

To show how to retrieve subsets of rows by using comparison operators.

Lead-in

Use comparison operators, such as greater than (>), less than (<), and equal to (=) to select rows based on comparisons.

Example 1

```
USE northwind
SELECT lastname, city
FROM employees
WHERE country = 'USA'
GO
```



lastname	city
Davolio	Seattle
Fuller	Tacoma
Leverling	Kirkland
Peacock	Redmond
Callahan	Seattle

Use comparison operators to compare the values in a table to a specified value or expression. You also can use comparison operators to check for a condition. Comparison operators compare columns or variables of compatible data types. The comparison operators are listed in the following table.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Note Avoid the use of NOT in search conditions. They may slow data retrieval because all rows in a table are evaluated.

Example 1 This example retrieves the last name and city of employees who reside in the United States from the **employees** table.

```
USE northwind
SELECT lastname, city
FROM employees
WHERE country = 'USA'
GO
```

Result	lastname	city
	Davolio	Seattle
	Fuller	Tacoma
	Leverling	Kirkland
	Peacock	Redmond
	Callahan	Seattle

(5 row(s) affected)

Example 2 This example retrieves the **orderid** and **customerid** columns with order dates that are older than 8/1/96 from the **orders** table.

```
USE northwind
SELECT orderid, customerid
FROM orders
WHERE orderdate < '8/1/96'
GO
```

Result	orderid	customerid
	10248	VINET
	10249	TOMSP
	10250	HANAR
	10251	VICTE
	10252	SUPRD
	10253	HANAR
	.	.
	.	.
	.	.

(22 row(s) affected)

Using String Comparisons

Slide Objective

To show how to retrieve rows by using the LIKE search condition in combination with wildcard characters.

Lead-in

You can use the LIKE search condition to select rows by comparing character strings.

```
USE northwind
SELECT companyname
FROM customers
WHERE companyname LIKE '%Restaurant%'
GO
```

companyname

GROSELLA-Restaurante
Lonesome Pine Restaurant
Tortuga Restaurante

You can use the LIKE search condition in combination with wildcard characters to select rows by comparing character strings. When you use the LIKE search condition, consider the following facts:

- All characters in the pattern string are significant, including leading and trailing blank spaces.
- LIKE can be used only with data of the **char**, **nchar**, **varchar**, **nvarchar**, or **datetime** data types.

Types of Wildcard Characters

Use the following four wildcard characters to form your character string search criteria.

Wildcard	Description
%	Any string of zero or more characters
_	Any single character
[]	Any single character within the specified range or set
[^]	Any single character <i>not</i> within the specified range or set

Examples of the Use of Wildcard Characters

The following table lists examples of the use of wildcards with the LIKE search condition.

Expression	Returns
LIKE 'BR%'	Every name beginning with the letters BR
LIKE 'Br%'	Every name beginning with the letters Br
LIKE '%een'	Every name ending with the letters een
LIKE '%en%'	Every name containing the letters en
LIKE '_en'	Every three-letter name ending in the letters en
LIKE '[CK]%'	Every name beginning with the letter C or K
LIKE '[S-V]ing'	Every four-letter name ending in the letters ing and beginning with any single letter from S to V
LIKE 'M[^c]%'	Every name beginning with the letter M that does not have the letter c as the second letter

Example

This example retrieves companies from the **customers** table that have the word restaurant in their company names.

```
USE northwind
SELECT companyname
FROM customers
WHERE companyname LIKE '%Restaurant%'
GO
```

Result

companyname
GROSELLA-Restaurante
Lonesome Pine Restaurant
Tortuga Restaurante

(3 row(s) affected)

Using Logical Operators

Slide Objective


To show how to combine several expressions by using logical operators.

Lead-in

You may want to limit the number of rows that SQL Server returns when you execute a query. To do so, use logical operators to combine two or more expressions.

Example 1

```
USE northwind
SELECT productid, productname, supplierid, unitprice
FROM products
WHERE (productname LIKE 'T%' OR productid = 46)
AND (unitprice > 16.00)
GO
```



productid	productname	supplierid	unitprice
14	Tofu	6	23.25
29	Thüringer Rostbratwurst	12	123.79
62	Tarte au sucre	29	49.3

Use the logical operators AND, OR, and NOT to combine a series of expressions and to refine query processing. The results of a query may vary depending on the grouping of expressions and the order of the search conditions.

When you use logical operators, consider the following guidelines:

- Use the AND operator to retrieve rows that meet all of the search criteria.
- Use the OR operator to retrieve rows that meet any of the search criteria.
- Use the NOT operator to negate the expression that follows the operator.

Using Parentheses

Use parentheses when you have two or more expressions as the search criteria. Using parentheses allows you to:

- Group expressions.
- Change the order of evaluation.
- Make expressions more readable.

Order of Search Conditions

When you use more than one logical operator in a statement, consider the following facts:

- Microsoft® SQL Server™ 2000 evaluates the NOT operator first, followed by the AND operator and then the OR operator.
- The precedence order is from left to right if all operators in an expression are of the same level.

Example 1

The following example retrieves all products with product names that begin with the letter T or have a product identification number of 46, and that have a price greater than \$16.00.

Delivery Tip

Compare the queries in Example 1 and Example 2. Point out that the expressions are grouped differently and, therefore, produce different result sets.

```
USE northwind
SELECT productid, productname, supplierid, unitprice
FROM products
WHERE (productname LIKE 'T%' OR productid = 46)
AND (unitprice > 16.00)
GO
```

Result

productid	productname	supplierid	unitprice
14	Tofu	6	23.25
29	Thüringer Rostbratwurst	12	123.79
62	Tarte au sucre	29	49.3

(3 row(s) affected)

Example 2

The following example retrieves products with product names that begin with the letter T or that have a product identification number of 46 and a price greater than \$16.00. Compare the query in Example 1 to that in Example 2. Notice that because the expressions are grouped differently, the queries are processed differently and return different result sets.

```
USE northwind
SELECT productid, productname, supplierid, unitprice
FROM products
WHERE (productname LIKE 'T%')
OR (productid = 46 AND unitprice > 16.00)
GO
```

Result

productid	productname	supplierid	unitprice
54	Tourtière	25	7.45
62	Tarte au sucre	29	49.3
23	Tunnbröd	9	9
19	Teatime Chocolate Biscuits	8	9.2
14	Tofu	6	23.25
29	Thüringer Rostbratwurst	12	123.79

(6 row(s) affected)

Retrieving a Range of Values

Slide Objective

To show how to retrieve data by using the BETWEEN search condition.

Lead-in

To retrieve rows that are between a range of values, use the BETWEEN search condition.

Example 1

```
USE northwind
SELECT productname, unitprice
FROM products
WHERE unitprice BETWEEN 10 AND 20
GO
```



productname	unitprice
Chai	18
Chang	19
Aniseed Syrup	10
Genen Shouyu	15.5
Pavlova	17.45
Sir Rodney's Scones	10
...	...

Use the BETWEEN search condition in the WHERE clause to retrieve rows that are within a specified range of values. When you use the BETWEEN search condition, consider the following facts and guidelines:

- SQL Server includes the end values in the result set.
- Use the BETWEEN search condition rather than an expression that includes the AND operator with two comparison operators ($\geq x$ AND $\leq y$). However, to search for an exclusive range in which the returned rows do not contain the end values, use an expression that includes the AND operator with two comparison operators ($> x$ AND $< y$).
- Use the NOT BETWEEN search condition to retrieve rows outside of the specified range. Be aware that using NOT conditions may slow data retrieval.

Example 1

This example retrieves the product name and unit price of all products with a unit price between \$10.00 and \$20.00. Notice that the result set includes the end values.

```
USE northwind
SELECT productname, unitprice
FROM products
WHERE unitprice BETWEEN 10 AND 20
GO
```

Result

productname	unitprice
Chai	18
Chang	19
Aniseed Syrup	10
Genen Shouyu	15.5
Pavlova	17.45
Sir Rodney's Scones	10
.	
.	
.	
(29 row(s) affected)	

Example 2

This example retrieves the product name and unit price of all products with a unit price between \$10 and \$20. Notice that the result set excludes the end values.

```
USE northwind
SELECT productname, unitprice
FROM products
WHERE (unitprice > 10)
AND (unitprice < 20)
GO
```

Result

productname	unitprice
Chai	18
Chang	19
Genen Shouyu	15.5
Pavlova	17.45
.	
.	
.	
(25 row(s) affected)	

Using a List of Values as Search Criteria

Slide Objective


To show how to retrieve rows by using the IN search condition.

Lead-in

You may want to retrieve rows that match a specified list of values. To do so, use the IN search condition in the WHERE clause.

Example 1

```
USE northwind
SELECT companyname, country
FROM suppliers
WHERE country IN ('Japan', 'Italy')
GO
```



companyname	country
Tokyo Traders	Japan
Mayumi's	Japan
Formaggi Fortini s.r.l.	Italy
Pasta Buttini s.r.l.	Italy

Delivery Tip

Point out that SQL Server resolves Examples 1 and 2 in the same way, returning the same result set. Example 1 uses the IN search operator, while Example 2 uses two equal to (=) operators that are connected with the OR operator.

Use the IN search condition in the WHERE clause to retrieve rows that match a specified list of values. When you use the IN search condition, consider the following guidelines:

- Use either the IN search condition or a series of comparison expressions that are connected with an OR operator—SQL Server resolves them in the same way, returning identical result sets.
- Do not include a null value in the search condition. A null value in the search condition list evaluates to the expression, = NULL. This may return unpredicted result sets.
- Use the NOT IN search condition to retrieve rows that are not in your specified list of values. Be aware that using NOT conditions may slow data retrieval.

Example 1

This example produces a list of companies from the **suppliers** table that are located in Japan or Italy.

```
USE northwind
SELECT companyname, country
FROM suppliers
WHERE country IN ('Japan', 'Italy')
GO
```

Result

companyname	country
Tokyo Traders	Japan
Mayumi's	Japan
Formaggi Fortini s.r.l.	Italy
Pasta Buttini s.r.l.	Italy

(4 row(s) affected)

Example 2

This example also produces a list of companies from the **suppliers** table that are located in Japan or Italy. Notice that rather than using the IN search condition, two expressions that use the comparison operator are joined by the OR operator. The result set is identical to the result set in Example 1.

```
USE northwind
SELECT companyname, country
FROM suppliers
WHERE country = 'Japan' OR country = 'Italy'
GO
```

Result

companyname	country
Tokyo Traders	Japan
Mayumi's	Japan
Formaggi Fortini s.r.l.	Italy
Pasta Buttini s.r.l.	Italy

(4 row(s) affected)

Trainer Materials
for Microsoft Certified
Trainer Use Only

Retrieving Unknown Values

Slide Objective

To show how to retrieve rows that contain unknown values.

Lead-in

You can retrieve rows that contain unknown values by specifying IS NULL in the WHERE clause.

```
USE northwind
SELECT companyname, fax
FROM suppliers
WHERE fax IS NULL
GO
```



companyname	fax
Exotic Liquids	NULL
New Orleans Cajun Delights	NULL
Tokyo Traders	NULL
Cooperativa de Quesos 'Las Cabras'	NULL
...	...

A column has a null value if no value is entered during data entry and no default values are defined for that column. A null value is not the same as entries with a zero (a numerical value) or a blank (a character value).

Use the IS NULL search condition to retrieve rows in which information is missing from a specified column. When you retrieve rows that contain unknown values, consider the following facts and guidelines:

- Null values fail all comparisons because they do not evaluate equally with one another.
- You define whether columns allow null values in the CREATE TABLE statement.
- Use the IS NOT NULL search condition to retrieve rows that have known values in the specified columns.

Example

This example retrieves a list of companies from the **suppliers** table for which the **fax** column contains a null value.

```
USE northwind
SELECT companyname, fax
FROM suppliers
WHERE fax IS NULL
GO
```

Result

companyname	fax
Exotic Liquids	NULL
New Orleans Cajun Delights	NULL
Tokyo Traders	NULL
Cooperativa de Quesos 'Las Cabras'	NULL
.	.
.	.
.	.

(16 row(s) affected)

◆ Formatting Result Sets

Slide Objective

To show how to format result sets.

Lead-in

To make your result sets more readable, you can sort data, eliminate duplicate rows, change column names, or use literals.

- **Sorting Data**
- **Eliminating Duplicate Rows**
- **Changing Column Names**
- **Using Literals**

You can improve the readability of a result set by sorting the order in which the result set is listed, eliminating any duplicate rows, changing column names to column aliases, or using literals to replace result set values. These formatting options do not change the data, only the presentation of it.

Trainer Material
for Microsoft Certified
Trainer Use Only

Sorting Data

Slide Objective


To show how to use the ORDER BY clause.

Lead-in

After deciding which columns and rows to retrieve, you can sort the order of the result set by using the ORDER BY clause.

Example 1

```
USE northwind
SELECT productid, productname, categoryid, unitprice
FROM products
ORDER BY categoryid, unitprice DESC
GO
```



productid	productname	categoryid	unitprice
38	Cote de Blaye	1	263.5000
43	Ipoh Coffee	1	46.0000
2	Chang	1	19.0000
...
63	Vegie-spread	2	43.9000
8	Northwoods Cranberry Sauce	2	40.0000
61	Sirop d'érable	2	28.5000
...

Delivery Tip

Compare the specified columns in the ORDER BY clauses of Example 1 and Example 2. In Example 2, the specified columns in the ORDER BY clause are replaced with their respective positions in the select list. The result sets of Example 1 and Example 2 are identical.

Use the ORDER BY clause to sort rows in the result set in ascending (ASC) or descending (DESC) order. When you use the ORDER BY clause, consider the following facts and guidelines:

- The sort order is specified when SQL Server is installed. Execute the **sp_helpsort** system stored procedure to determine the sort order that was defined for the database during installation.
- SQL Server does not guarantee an order in the result set unless the order is specified with an ORDER BY clause.
- SQL Server sorts in ascending order by default.
- Columns that are included in the ORDER BY clause do not have to appear in the select list.
- Columns that are specified in the ORDER BY clause cannot exceed 8060 bytes.
- You can sort by column names, computed values, or expressions.
- You can refer to columns by their positions in the select list in the ORDER BY clause. The columns are evaluated in the same way and return the same result set.
- Do not use an ORDER BY clause on **text** or **image** columns.

Tip Using appropriate indexes can make ORDER BY sorts more efficient.

Example 1

This example retrieves the product identification, product name, category, and unit price of each product from the **products** table. By default, the result set is ordered by category in ascending order, and within each category the rows are ordered by unit price in descending order.

```
USE northwind
SELECT productid, productname, categoryid, unitprice
FROM products
ORDER BY categoryid, unitprice DESC
GO
```

Result

productid	productname	categoryid	unitprice
38	Côte de Blaye	1	263.5000
43	Ipoh Coffee	1	46.0000
2	Chang	1	19.0000
1	Chai	1	18.0000
35	Steeleye Stout	1	18.0000
39	Chartreuse verte	1	18.0000
76	Lakkalikööri	1	18.0000
70	Outback Lager	1	15.0000
34	Sasquatch Ale	1	14.0000
67	Laughing Lumberjack Lager	1	14.0000
75	Rhönbräu Klosterbier	1	7.7500
24	Guaraná Fantástica	1	4.5000
63	Vegie-spread	2	43.9000
8	Northwoods Cranberry Sauce	2	40.0000
61	Sirop d'érable	2	28.5000
6	Grandma's Boysenberry Spread	2	25.0000

.
.
.
(77 row(s) affected)

Example 2

This example is similar to Example 1. The only difference is that the numbers that follow the ORDER BY clause indicate the position of columns in the select list. SQL Server resolves both queries in the same way, returning the same result set.

```
USE northwind
SELECT productid, productname, categoryid, unitprice
FROM products
ORDER BY 3, 4 DESC
GO
```

Eliminating Duplicate Rows

Slide Objective

To show how to eliminate duplicate rows by using the DISTINCT search condition.

Lead-in

To eliminate duplicate rows from the result set, specify the DISTINCT clause in the SELECT statement.

Example 1

```
USE northwind
SELECT DISTINCT country
FROM suppliers
ORDER BY country
GO
```

country
Australia
Brazil
Canada
Denmark
Finland
France
Germany
Italy
Japan
Netherlands
Norway
Singapore
Spain
Sweden
UK
USA

Delivery Tip

Execute a query similar to that in the slide without using the DISTINCT clause.

Have students compare the number of rows that are returned.

If you require a list of unique values, use the DISTINCT clause to eliminate duplicate rows in the result set. When you use the DISTINCT clause, consider the following facts:

- All rows that meet the search condition that is specified in the SELECT statement are returned in the result set unless you have specified the DISTINCT clause.
- The combination of values in the select list determines distinctiveness.
- Rows that contain any unique combination of values are retrieved and returned in the result set.
- The DISTINCT clause sorts the result set in random order unless you have included an ORDER BY clause.
- If you specify a DISTINCT clause, the ORDER BY clause must include the columns listed in the result set.

Example 1

This example retrieves all rows from the **suppliers** table but displays each country name only once.

```
USE northwind
SELECT DISTINCT country
FROM suppliers
ORDER BY country
GO
```

Result**country**

Australia
Brazil
Canada
Denmark
Finland
France
Germany
Italy
Japan
Netherlands
Norway
Singapore
Spain
Sweden
UK
USA

(16 row(s) affected)

Example 2

This example does not specify the DISTINCT clause. All rows from the **suppliers** table are retrieved and listed in descending order. Notice that each instance of a country is displayed.

```
USE northwind
SELECT country
  FROM suppliers
 ORDER BY country
GO
```

Result**country**

Australia
Australia
Brazil
Canada
Canada
Denmark
Finland
France
France
France
Germany
Germany
Germany
Italy
Italy
Japan
Japan
Netherlands
Norway
Singapore
Spain
.
.
.

(29 row(s) affected)

Changing Column Names

Slide Objective

To show how you can change column names for readability.

Lead-in

Use aliases to change column names and to make your result sets more readable.

```
USE northwind
SELECT  firstname AS First, lastname AS Last
        ,employeeid AS 'Employee ID:'
FROM employees
GO
```



<i>First</i>	<i>Last</i>	<i>Employee ID:</i>
Nancy	Davolio	1
Andrew	Fuller	2
Janet	Leverling	3
Margaret	Peacock	4
Steven	Buchanan	5
Michael	Suyama	6
Robert	King	7
Laura	Callahan	8
Anne	Dodsworth	9

Create more readable column names by using the AS keyword to replace default column names with aliases in the select list.

Partial Syntax

```
SELECT column_name | expression AS column_heading
FROM table_name
```

When you change column names, consider the following facts and guidelines:

- By default, the result set displays the column names that are designated in the CREATE TABLE statement.
- Include single quotation marks for column names that contain blank spaces or that do not conform to SQL Server object naming conventions.
- You can create column aliases for computed columns that contain functions or string literals.
- You can include up to 128 characters in a column alias.

Example

This example retrieves a list of employees from the **employees** table. The specified column aliases replace the **firstname**, **lastname**, and **employeeid** columns. Notice that the Employee ID: alias is enclosed in single quotation marks because it contains a blank space.

```
USE northwind
SELECT  firstname AS First, lastname AS Last
        ,employeeid AS 'Employee ID:'
FROM employees
GO
```

Result

First	Last	Employee ID:
Nancy	Davolio	1
Andrew	Fuller	2
Janet	Leverling	3
Margaret	Peacock	4
Steven	Buchanan	5
Michael	Suyama	6
Robert	King	7
Laura	Callahan	8
Anne	Dodsworth	9

(9 row(s) affected)

Trainer Materials
for Microsoft Certified
Trainer Use Only

Using Literals

Slide Objective

To show how using literals in the SELECT statement makes result sets more readable.

Lead-in

You can make result set values more readable by using literals.

```
USE northwind
SELECT firstname, lastname,
       'Identification number:', employeeid
FROM employees
GO
```



First	Last	Employee ID:
Nancy	Davolio	Identification Number: 1
Andrew	Fuller	Identification Number: 2
Janet	Leverling	Identification Number: 3
Margaret	Peacock	Identification Number: 4
Steven	Buchanan	Identification Number: 5
Michael	Suyama	Identification Number: 6
Robert	King	Identification Number: 7
Laura	Callahan	Identification Number: 8
Anne	Dodsworth	Identification Number: 9

Literals are letters, numerals, or symbols that are used as specific values in a result set. You can include literals in the select list to make result sets more readable.

Partial Syntax

```
SELECT column_name | 'string literal' [, column_name | 'string_literal'...]
FROM table_name
```

Example

This example retrieves a list of employees from the **employees** table. Notice that the Identification number: character string precedes the **employeeid** column in the result set.

```
USE northwind
SELECT  firstname, lastname
        , 'Identification number:', employeeid
FROM employees
GO
```

Result

firstname	lastname	employeeid
Nancy	Davolio	Identification number: 1
Andrew	Fuller	Identification number: 2
Janet	Leverling	Identification number: 3
Margaret	Peacock	Identification number: 4
Steven	Buchanan	Identification number: 5
Michael	Suyama	Identification number: 6
Robert	King	Identification number: 7
Laura	Callahan	Identification number: 8
Anne	Dodsworth	Identification number: 9

(9 row(s) affected)

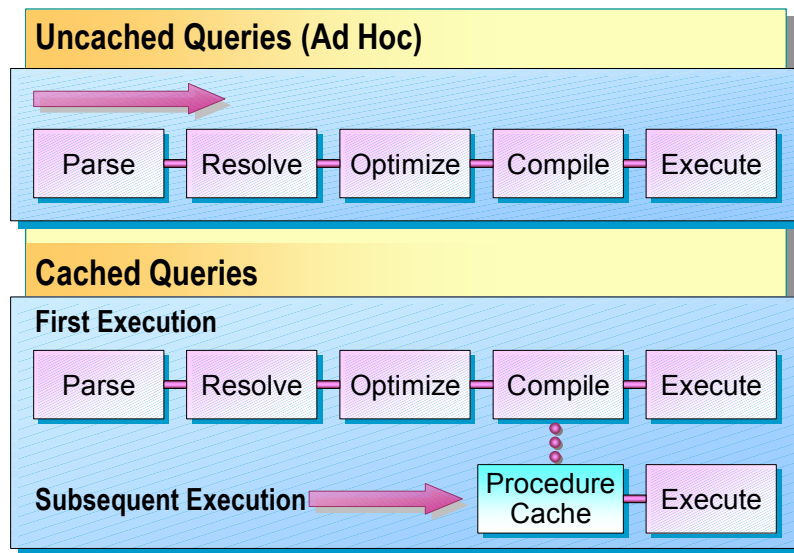
◆ How Queries Are Processed

Slide Objective

To describe how queries are processed.

Lead-in

All queries follow the same process before they are executed.



All queries follow the same process before they are executed. SQL Server can store some of the processing for subsequent execution of the same query.

Uncached Queries (Ad Hoc)

All queries are parsed, resolved, optimized, and compiled before they are executed.

Views are reduced (resolved) to a single statement. The statement referencing the view is merged with the view definition, creating one SELECT statement.

Process	Description
Parse	Checks syntax for accuracy.
Resolve	Validates that the names of the objects are present; determines object ownership permission.
Optimize	Determines the indexes to use and the join strategies.
Compile	Translates the query into an executable form.
Execute	Submits compiled requests for processing.

Cached Queries

To improve performance, SQL Server can save compiled query plans for reuse. Query plans are optimized instructions on how to process queries and access the data. The query plans are stored in the *procedure cache*, a temporary storage location for the currently executing version of a specific query.

How Queries Are Cached Automatically

Slide Objective

To describe how SQL Server caches queries automatically.

Lead-in

Queries are cached automatically under two conditions.

■ Ad Hoc Batches

```
USE northwind
SELECT * FROM products WHERE unitprice = $12.5
SELECT * FROM products WHERE unitprice = 12.5
SELECT * FROM products WHERE unitprice = $12.5
GO
```

■ Auto-Parameterization

```
USE library
SELECT * FROM member WHERE member_no = 7890
SELECT * FROM member WHERE member_no = 1234
SELECT * FROM member WHERE member_no = 7890
GO
```

Cached queries are saved in an area of memory called the *procedure cache*. Query definitions are cached automatically under two conditions—ad hoc batches and auto-parameterization. Automatic caching cannot be specified directly.

Ad Hoc Batches

SQL Server caches the plans from ad hoc batches. If a subsequent batch matches the text of the first batch, SQL Server uses the cached plan. This plan is limited to exact textual matches.

Example 1

The same query plan would be used for the first and third statements. The second statement would use a different query plan.

Delivery Tip

For an exact textual match, both the data and the data type must match. In this example, "\$12.5" is passed as a monetary data type, while "12.5" is passed as a floating point data type.

```
USE northwind
SELECT * FROM products WHERE unitprice = $12.5
SELECT * FROM products WHERE unitprice = 12.5
SELECT * FROM products WHERE unitprice = $12.5
GO
```

Auto-Parameterization

SQL Server attempts to determine the constants that are actually parameters and makes them into parameters automatically. If successful, later queries that follow the same template can use the same plan.

Example 2

Auto-parameterization uses the same query plan for all three of the following statements.

```
USE library
SELECT * FROM member WHERE member_no = 7890
SELECT * FROM member WHERE member_no = 1234
SELECT * FROM member WHERE member_no = 7890
GO
```

Trainer Materials
for Microsoft Certified
Trainer Use Only

Performance Considerations

Slide Objective

To discuss the performance considerations for performing basic queries.

Lead-in

Consider some of the issues that affect the performance of SQL Server when you perform basic queries.

- **Not Search Conditions May Slow Data Retrieval**
- **LIKE Search Conditions Slow Data Retrieval**
- **Exact Matches or Ranges May Speed Data Retrieval**
- **ORDER BY Clause May Slow Data Retrieval**

You should consider some of the issues that affect the performance of SQL Server when you perform basic queries:

- Use positive rather than negative search conditions. Negative search conditions—such as NOT BETWEEN, NOT IN, and IS NOT NULL—may slow data retrieval because all rows are evaluated.
- Avoid using the LIKE search condition if you can write a more specific query. Data retrieval may be slower when you use the LIKE search condition.
- Use exact matches or ranges as search conditions when possible. Again, specific queries perform better.
- Data retrieval may decrease if you use the ORDER BY clause because SQL Server must determine and sort the result set before it returns the first row.

Recommended Practices

Slide Objective

To list the recommended practices for performing basic queries.

Lead-in

The following recommended practices should help you perform basic queries.



Use the DISTINCT Clause to Eliminate Duplicate Rows in the Result Set



Improve the Readability of a Result Set by Changing Column Names or by Using Literals



Place a Comma Before the First Column Name in a Multi-Line Column List

The following recommended practices should help you perform basic queries:

- Use the DISTINCT clause to eliminate duplicate rows in result sets. All rows that meet the search conditions that are specified in the SELECT statement are returned in the result set unless you use the DISTINCT clause.
- Improve the readability of result sets by changing column names to column aliases or using literals to replace result set values. These formatting options change the presentation of the data, not the data itself.
- Place a comma before the first column name in a multiline column list. This style allows you to comment out or cut and paste single lines easily. For example, use the following style:

```
USE northwind
SELECT  firstname AS First
        ,lastname AS Last
        ,employeeid AS 'Employee ID:'
FROM employees
GO
```

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

Topic	Search on
Using character strings	“pattern matching”
Sorting result sets	“sort order”

Lab A: Retrieving Data and Manipulating Result Sets

Slide Objective

To introduce the lab.

Lead-in

In this lab, you will query SQL Server databases by using the SELECT statement and then format the result sets.



Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Perform queries on databases by using the SELECT statement.
- Sort the data and eliminate duplicate values in a result set.
- Format the result set by using column aliases and literals.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2071A\Labfiles\L03.
- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L03\Answers.
- The **library** database installed.

Lab Setup

None.

For More Information

If you require help in executing files, search Microsoft® SQL Server™ 2000 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 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 the 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: 45 minutes

Exercise 1

Retrieving Data

In this exercise, you will select specific data from tables in the **library** database. C:\Moc\2071A\Labfiles\L03\Answers contains completed scripts for this exercise.

► To select specific columns

In this procedure, you will write and execute a SELECT statement that retrieves the **title** and **title_no** columns from the **title** table.

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

Option	Value
User name	SQLAdminx (where <i>x</i> 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 **SQLAdminx**, 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. Write and execute a SELECT statement that retrieves the **title** and **title_no** columns from the **title** table. Answer_Columns.sql is a completed script for this step.

```
USE library
SELECT title, title_no
FROM title
GO
```

5. Save the SELECT statement as ANSI text with an .sql file name extension.
6. Save the result set with an .rpt file name extension.

Result

Your result should look similar to the following partial result set.

Title	title_no
Last of the Mohicans	1
The Village Watch-Tower	2
Self Help; Conduct & Perseverance	3
Songs of a Savoyard	4
.	
.	
.	
.	
(50 row(s) affected)	

► To select rows by using a comparison operator

In this procedure, you will write and execute a SELECT statement that retrieves data from specific rows by using a WHERE clause with a comparison operator. Answer_Comparison.sql is a completed script for this procedure.

- Write and execute a SELECT statement that retrieves the title of title number 10 from the **title** table.

You can execute the **sp_help** system stored procedure for the **title** table to find the correct column names.

```
USE library
SELECT title
FROM title
WHERE title_no = 10
GO
```

Result

Your result should look similar to the following result set.

title

The Night-Born
(1 row(s) affected)

► To select rows by using a range

In this procedure, you will write and execute a SELECT statement that retrieves data from specific rows by using a WHERE clause with a range. Answer_Range.sql is a completed script for this procedure.

- Write and execute a SELECT statement that retrieves the member numbers and assessed fines from the **loanhist** table for all members who have had fines between \$8.00 and \$9.00.

You can execute the **sp_help** system stored procedure for the **loanhist** table to find the correct column names.

```
USE library
SELECT member_no, fine_assessed
FROM loanhist
WHERE (fine_assessed BETWEEN $8.00 AND $9.00)
GO
```

Result

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

member_no	fine_assessed
7399	9
7399	9
7399	9
7399	9
.	.
.	.
.	.

(312 row(s) affected)

► To select rows by using a list of values

In this procedure, you will write and execute a SELECT statement that retrieves data from specific rows by using a WHERE clause that contains a list of values. Answer_InList.sql is a completed script for this procedure.

- Write and execute a SELECT statement that retrieves the title number and author from the **title** table for all books authored by Charles Dickens or Jane Austen. Use the IN operator as part of the SELECT statement.

```
USE library
SELECT author, title_no
FROM title
WHERE author IN ('Charles Dickens','Jane Austen')
GO
```

Result

Your result should look similar to the following result set.

author	title_no
Jane Austen	27
Charles Dickens	30
Charles Dickens	31
Jane Austen	41
Jane Austen	43

(5 row(s) affected)

► To select rows by using a character string comparison

In this procedure, you will write and execute a SELECT statement that retrieves data from specific rows that contain a character string similar to another character string. Answer_String.sql is a completed script for this procedure.

- Write and execute a SELECT statement that retrieves the title number and title from the **title** table for all rows that contain the character string “adventures” in the title. Use the LIKE operator in your query.

```
USE library
SELECT title_no, title
FROM title
WHERE title LIKE ('%Adventures%')
GO
```

Result

Your result should look similar to the following result set.

title_no	title
26	The Adventures of Robin Hood
44	Adventures of Huckleberry Finn

(2 row(s) affected)

► To select rows that contain null values

In this procedure, you will write and execute a SELECT statement that retrieves data from specific rows by using a WHERE clause that searches for null values. Answer_IsNull.sql is a completed script for this procedure.

- Write and execute a SELECT statement that retrieves the member number, assessed fine, and fine that is paid for loans that have unpaid fines from the **loanhist** table. Retrieve rows that have fines entered in the **fine_assessed** column and that have null values for the **fine_paid** column.

USE library

SELECT member_no, fine_assessed, fine_paid

FROM loanhist

WHERE (fine_assessed IS NOT NULL) AND (fine_paid IS NULL)

GO

Result

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

member_no	fine_assessed	fine_paid
4645	5.0000	NULL
4240	.0000	NULL
3821	1.0000	NULL
3389	9.0000	NULL

.

.

.

(1118 row(s) affected)

Trainer Materials
for Microsoft Certified
Trainer Use Only

Exercise 2

Manipulating Result Sets

In this exercise, you will write and execute queries that change the way that the data is displayed in the result set. You will use the **DISTINCT** keyword to eliminate duplicate rows and the **ORDER BY** keyword to sort the result set. Additionally, you will change the column names and presentation of data in result sets by using aliases and literals.

C:\Moc\2071A\Labfiles\L03\Answers contains completed scripts for this exercise.

► To eliminate duplicate rows from the result set

In this procedure, you will write and execute a query on the **adult** table that returns only unique combinations of cities and states in your result set. Answer_Duplicates.sql is a completed script for this procedure.

- Write and execute a query that retrieves all of the unique pairs of cities and states from the **adult** table. You should receive only one row in the result set for each city and state pair.

```
USE library
SELECT DISTINCT city, state
FROM adult
GO
```

Result

Your result should look similar to the following partial result set.

City	state
Salt Lake City	UT
Atlanta	GA
Tallahassee	FL
Washington	DC
.	
.	
.	
(23 row(s) affected)	

► To sort data

In this procedure, you will write and execute a query that retrieves the titles from the **title** table and lists them in alphabetical order. Answer_Sort.sql is a completed script for this procedure.

- Write and execute a query that retrieves a sorted list of all titles from the **title** table.

```
USE library
SELECT title
FROM title
ORDER BY title
GO
```

Result

Your result should look similar to the following partial result set.

Title

A Tale of Two Cities
 Adventures of Huckleberry Finn
 Ballads of a Bohemian
 Candide

.
 .
 .

(50 row(s) affected)

► **To compute data, return computed values, and use a column alias**

In this exercise, you will write and execute a query that returns the **member_no**, **isbn**, and **fine_assessed** columns from the **loanhist** table of all archived loans with a non-null value in the **fine_assessed** column. Then, you will create a new column in the result set that contains the computed value of the **fine_assessed** column multiplied by two, and you will use a column alias named **double fine**.

Answer_Computed.sql is a completed script for this procedure.

1. Write and execute a query that retrieves the **member_no**, **isbn**, and **fine_assessed** columns from the **loanhist** table of all archived loans a non-null value in the **fine_assessed** column.
2. Create a computed column that contains the value of the **fine_assessed** column multiplied by two.
3. Use the column alias 'double fine' for the computed column. Enclose the column alias within single quotation marks because it does not conform to the SQL Server object naming conventions.

USE library

```
SELECT member_no, isbn, fine_assessed
      , (fine_assessed * 2) AS 'double fine'
FROM   loanhist
WHERE  (fine_assessed IS NOT NULL)
GO
```

Result

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

member_no	isbn	fine_assessed	double fine
7399	101	9.0000	18.0000
6709	102	9.0000	18.0000

.
 .
 .

(1300 row(s) affected)

► **To format the result set of a column by using string functions**

In this procedure, you will write and execute a query that lists all members in the **member** table with the last name Anderson. Format the result set in lowercase characters and display a single column of e-mail names that consists of the member's first name, middle initial, and first two letters of the last name.

Answer_Formatting.sql is a completed script for this procedure.

1. Write and execute a query that generates a single column that contains the **firstname**, **middleinitial**, and **lastname** columns from the **member** table for all members with the last name Anderson.
2. Use the column alias **email_name**.
3. Modify the query to return a list of e-mail names with the member's first name, middle initial, and first two letters of the last name in lowercase characters. Use the SUBSTRING function to retrieve part of a string column. Use the LOWER function to return the result in lowercase characters. Use the addition (+) operator to concatenate the character strings.

```
USE library
SELECT LOWER(firstname + middleinitial
            +SUBSTRING(lastname, 1, 2) ) AS email_name
FROM member
WHERE lastname = 'anderson'
GO
```

Result

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

```
email_name
-----
Amyaan
Angelaan
Brianaan
Clairaan
.
.
.
(390 row(s) affected)
```

► **To format the result set of a column by using literals**

In this procedure, you will format the result set of a query for readability by using the CONVERT function and string literals. Answer_Literals.sql is a completed script for this procedure.

1. Write and execute a query that retrieves the **title** and **title_no** columns from the **title** table. Your result set should be a single column with the following format:

The title is: Poems, title number 7

This query returns a single column based on an expression that concatenates four elements:

- **The title is:** string constant
 - **title.title** column
 - **title number** string constant
 - **title.title_no** column
2. Use the CONVERT function to format the **title.title_no** column and the addition (+) operator to concatenate the character strings.

USE library

```
SELECT 'The title is: ' + title + ', title number ' +  
       CONVERT(char(6),title_no)  
FROM title  
GO
```

Result

Your result should look similar to the following partial result set.

```
The title is: Last of the Mohicans, title number 1  
The title is: The Village Watch-Tower, title number 2  
The title is: Self Help; Conduct & Perseverance, title number 3  
The title is: Songs of a Savoyard, title number 4  
The title is: Fall of the House of Usher, title number 5  
.  
.  
.  
(50 row(s) affected)
```

Exercise 3

Using System Functions

In this exercise, you will gather system information by using system functions. C:\Moc\2071A\Labfiles\L03\Answers contains completed scripts for this exercise.

► To determine the server process ID

In this procedure, you will observe current server activity and determine the activity that your session is generating. Answer_SPID.sql is a completed script for this procedure.

1. Execute the **sp_who** system stored procedure.

SQL Server displays all activity that is occurring on the server.

2. To determine which activity is yours, execute the following statement:

```
SELECT @@spid  
GO
```

The server process ID (spid) number of your process is returned in the results.

3. Execute the **sp_who** system stored procedure again, using your spid number as an additional parameter. (In the following statement, *n* represents your spid number.)

```
EXEC sp_who n  
GO
```

The activity related to your spid is displayed.

► To retrieve environmental information

In this procedure, you will determine which version of SQL Server that you are running and you will retrieve connection, database context, and server information. You will perform these tasks by using system functions.

Answer_Environment.sql is a completed script for this procedure.

1. Execute the following statement:

```
SELECT @@version  
GO
```

2. Execute the following statement:

```
SELECT USER_NAME(), DB_NAME(), @@servername  
GO
```

► **To retrieve metadata**

In this procedure, you will execute several queries to return the metadata from specific database objects by using information schema views. Remember that **information_schema** is a predefined database user that is the owner of the information schema views. Answer_Metadata.sql is a completed script for this procedure.

1. Execute the following statement to return a list of all the user-defined tables in a database:

```
USE library
SELECT *
  FROM information_schema.tables
 WHERE table_type = 'base table'
GO
```

2. Execute the following statement to return the primary key and foreign key columns for the **orders** table:

```
SELECT *
  FROM information_schema.key_column_usage
 WHERE table_name = 'orders'
GO
```

What column has a primary key defined on it?

orderid

Trainer Materials
for Microsoft Certified
Trainer Use Only

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.

- Retrieving Data by Using the SELECT Statement
- Filtering Data
- Formatting Result Sets
- How Queries Are Processed
- Performance Considerations

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

Ask students whether they need clarification on any topic.

You are the database administrator for a health care plan. The **physicians** table was created with the following statement:

```
CREATE TABLE dbo.physicians (  
    physician_no int IDENTITY (100, 2) NOT NULL ,  
    f_name varchar (25) NOT NULL ,  
    l_name varchar (25) NOT NULL ,  
    street varchar (50) NULL ,  
    city varchar (255) NULL ,  
    state varchar (255) NULL ,  
    postal_code varchar (7) NULL ,  
    co_pay money NOT NULL CONSTRAINT phys_co_pay DEFAULT (10)  
)
```

1. How would you retrieve information about physicians who have practices in the states of New York (NY), Washington (WA), Virginia (VA), or California (CA)?

Write a SELECT statement with a WHERE clause of the following type:

WHERE state = 'NY' OR state = 'WA' OR state = ...

Or, use a WHERE clause that includes the IN keyword:

WHERE state in ('NY', 'WA', 'VA', 'CA')

2. How can you generate a list of states that does not include any duplicate states in the result set?

Use the `DISTINCT` keyword as part of the `SELECT` statement.

3. How can you generate a column in your result set that lists the `co_pay` value plus a service charge of \$5.00 for each physician, and then alias this column as **Amt_Due**?

Use a computed column in the select list. Use an alias for the column name 'Amt_Due' = (co_pay + 5)

Trainer Materials
for Microsoft Certified
Trainer Use Only