MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 4: Grouping and Summarizing Data

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:
45 Minutes

This module provides students with the skills to group and summarize data by using aggregate functions. These skills include using the GROUP BY and HAVING clauses to summarize and group data and using the ROLLUP and CUBE operators with the GROUPING function to group data and summarize values for those groups. This module also introduces how to use the COMPUTE and COMPUTE BY clauses to generate summary reports and to list the TOP *n* values in a result set.

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

- Use the TOP *n* keyword to retrieve a list of the specified top values in a table.
- Generate a single summary value by using aggregate functions.
- Organize summary data for a column by using aggregate functions with the GROUP BY and HAVING clauses.
- Generate summary data for a table by using aggregate functions with the GROUP BY clause and the ROLLUP or CUBE operator.
- Generate control-break reports by using the COMPUTE and COMPUTE BY clauses.

Materials and Preparation

Required Materials

To teach this course, you need the following materials:

- Microsoft® PowerPoint® file 2071A_04.ppt.
- The C:\Moc\2071A\Demo\Ex_04.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:

- Listing the TOP *n* Values

  Introduce using the TOP *n* keyword to list only the first *n* rows or *n* percent of a result set. Although the TOP *n* keyword is not ANSI-standard, it is useful, for example, to list a company's top selling products.

- Using Aggregate Functions

  Discuss the use of aggregate functions in summarizing data. Encourage caution in using aggregate functions with null values because the result sets may not be representative of the data. Using aggregate functions is the basis for the remaining topics that are presented in this module.

- GROUP BY Fundamentals

  Explain the benefits of using aggregate functions with the GROUP BY clause to organize rows into groups and to summarize those groups. The HAVING clause is used with the GROUP BY clause to restrict the rows that are returned. Use the graphic images to compare the use of the GROUP BY and HAVING clauses.

- Generating Aggregate Values Within Result Sets

  Introduce the use of the ROLLUP and CUBE operators to generate detail and summary values in the result set. Both operators provide data in a standard relational format that can be used for other applications.

  Discuss how to use the GROUPING function to determine whether the values in the result set are detail values or a summary. Point out that on the slides, the NULLs that are displayed in the result sets represent summary values.

- Using the COMPUTE and COMPUTE BY Clauses

  Mention the COMPUTE and COMPUTE BY clauses within the context of using these clauses to print basic reports or verify client results. Do not spend too much time on these clauses, because they are not ANSI-standard and they generate result sets in a non-relational format. Use the graphic image to compare result sets when the COMPUTE and COMPUTE BY clauses are used.

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*.

Module Setup

The C:\Moc\2071A\Batches\2071A_R04.sql script, which adds the **orderhist** table to the **Northwind** database, is normally executed as part of the Classroom Setup. When you customize the course, you must ensure that this script is executed so that the examples in the module function correctly.

Lab Setup

There are no special setup requirements that affect this lab.

Lab Results

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

Overview

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

Lead-in
You may want to group or summarize data when you retrieve it.

- **Listing the TOP *n* Values**
- **Using Aggregate Functions**
- **GROUP BY Fundamentals**
- **Generating Aggregate Values Within Result Sets**
- **Using the COMPUTE and COMPUTE BY Clauses**

You may want to group or summarize data when you retrieve it.

This module provides students with the skills to group and summarize data by using aggregate functions. These skills include using the GROUP BY and HAVING clauses to summarize and group data and using the ROLLUP and CUBE operators with the GROUPING function to group data and summarize values for those groups. This module also introduces how to use the COMPUTE and COMPUTE BY clauses to generate summary reports and to list the TOP *n* values in a result set.

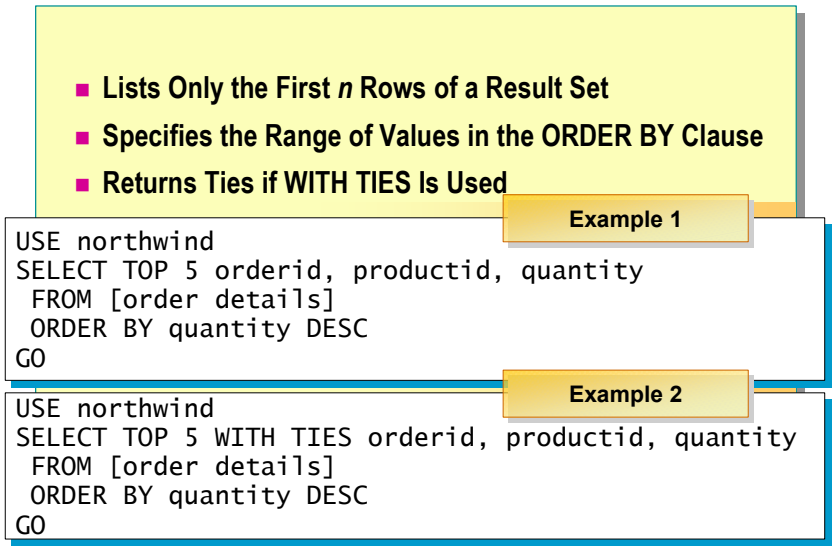
After completing this module, you will be able to:

- Use the TOP *n* keyword to retrieve a list of the specified top values in a table.
- Generate a single summary value by using aggregate functions.
- Organize summary data for a column by using aggregate functions with the GROUP BY and HAVING clauses.
- Generate summary data for a table by using aggregate functions with the GROUP BY clause and the ROLLUP or CUBE operator.
- Generate control-break reports by using the COMPUTE and COMPUTE BY clauses.

Listing the TOP *n* Values

Topic Objective
To describe how to list the top *n* summary values.

Lead-in
Use the TOP *n* keyword to list only the first *n* rows of a result set.



Instructor Note
Appropriate indexes can increase the efficiency of sorts and groupings. This course does not cover indexing in detail; for more information on indexing, see course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Use the TOP *n* keyword to list only the first *n* rows or *n* percent of a result set. Although the TOP *n* keyword is not ANSI-standard, it is useful, for example, to list a company's top selling products.

When you use the TOP *n* or TOP *n* PERCENT keyword, consider the following facts and guidelines:

- Specify the range of values in the ORDER BY clause. If you do not use an ORDER BY clause, Microsoft® SQL Server™ 2000 returns rows that satisfy the WHERE clause in no particular order.
- Use an unsigned integer following the TOP keyword.
- If the TOP *n* PERCENT keyword yields a fractional row, SQL Server rounds to the next integer value.
- Use the WITH TIES clause to include ties in your result set. Ties result when two or more values are the same as the last row that is returned in the ORDER BY clause. Your result set may therefore include any number of rows.

Note You can use the WITH TIES clause only when an ORDER BY clause exists.

Example 1

This example uses the TOP *n* keyword to find the five products with the highest quantities that are ordered in a single order. Tied values are excluded from the result set.

```
USE northwind
SELECT TOP 5 orderid, productid, quantity
 FROM [order details]
 ORDER BY quantity DESC
GO
```

Result

orderid	productid	quantity
10764	39	130
11072	64	130
10398	55	120
10451	55	120
10515	27	120

```
(5 row(s) affected)
```

Example 2

This example uses the TOP *n* keyword and the WITH TIES clause to find the five products with the highest quantities that are ordered in a single order. The result set lists a total of 10 products, because additional rows with the same values as the last row also are included. Compare the following result set to the result set in Example 1.

Delivery Tip
Compare the following result set to the result set in Example 1.

```
USE northwind
SELECT TOP 5 WITH TIES orderid, productid, quantity
 FROM [order details]
 ORDER BY quantity DESC
GO
```

Result

orderid	productid	quantity
10764	39	130
11072	64	130
10398	55	120
10451	55	120
10515	27	120
10595	61	120
10678	41	120
10711	53	120
10776	51	120
10894	75	120

```
(10 row(s) affected)
```

Using Aggregate Functions

Topic Objective
To demonstrate the use of aggregate functions for producing summary data.

Lead-in
Use aggregate functions to calculate column values and to include those values in your result set.

Aggregate function	Description
AVG	Average of values in a numeric expression
COUNT	Number of values in an expression
COUNT (*)	Number of selected rows
MAX	Highest value in the expression
MIN	Lowest value in the expression
SUM	Total values in a numeric expression
STDEV	Statistical deviation of all values
STDEVP	Statistical deviation for the population
VAR	Statistical variance of all values
VARP	Statistical variance of all values for the population

Functions that calculate averages and sums are called *aggregate functions*. When an aggregate function is executed, SQL Server summarizes values for an entire table or for groups of columns within the table, producing a single value for each set of rows for the specified columns:

- You can use aggregate functions with the SELECT statement or in combination with the GROUP BY clause.
- With the exception of the COUNT(*) function, all aggregate functions return a NULL if no rows satisfy the WHERE clause. The COUNT(*) function returns a value of zero if no rows satisfy the WHERE clause.

Tip Index frequently aggregated columns to improve query performance. For example, if you aggregate frequently on the **quantity** column, indexing on the **quantity** column improves aggregate operations.

The data type of a column determines the functions that you can use with it. The following table describes the relationships between functions and data types.

Function	Data type
COUNT	COUNT is the only aggregate function that can be used on columns with **text**, **ntext**, or **image** data types.
MIN and MAX	You cannot use the MIN and MAX functions on columns with **bit** data types.
SUM and AVG	You can use only the SUM and AVG aggregate functions on columns with **int**, **smallint**, **tinyint**, **decimal**, **numeric**, **float**, **real**, **money**, and **smallmoney** data types. When you use the SUM or AVG function, SQL Server treats the **smallint** or **tinyint** data types as an **int** data type value in your result set.

Partial Syntax

```
SELECT [ ALL | DISTINCT ]
  [ TOP n [PERCENT] [ WITH TIES] ] <select_list>
  [ INTO new_table ]
  [ FROM <table_sources> ]
  [ WHERE <search_conditions> ]
  [ [ GROUP BY [ALL] group_by_expression [,…n]]
  [HAVING <search_conditions> ]
  [ WITH { CUBE | ROLLUP } ]
  ]
  [ ORDER BY { column_name [ ASC | DESC ] } [,…n] ]
  [ COMPUTE
  { { AVG | COUNT | MAX | MIN | SUM } (expression) } [,…n]
  [ BY expression [,…n]
  ]
```

Example 1

This example calculates the average unit price of all products in the **products** table.

```
USE northwind
SELECT AVG(unitprice)
 FROM products
GO
```

Result

```
28.8663

(1 row(s) affected)
```

Example 2

This example adds all rows in the **quantity** column in the **order details** table.

```
USE northwind
SELECT SUM(quantity)
 FROM [order details]
GO
```

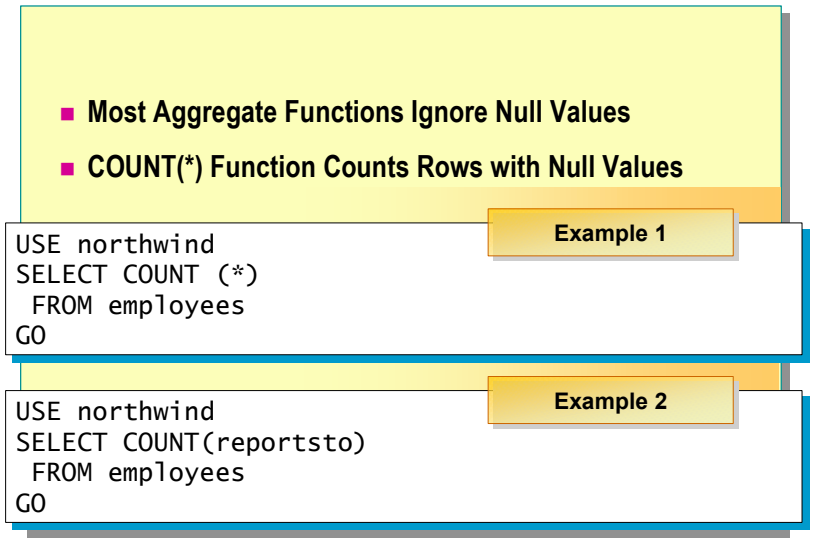
Result

```
51317

(1 row(s) affected)
```

Using Aggregate Functions with Null Values

Topic Objective
To discuss the behavior of null values when they are used with aggregate functions.

Lead-in
You may receive unexpected results if you use aggregate functions with null values.



Null values can cause aggregate functions to produce unexpected results. For example, if you execute a SELECT statement that includes a COUNT function on a column that contains 18 rows, two of which contain null values, your result set returns a total of 16 rows. SQL Server ignores the two rows that contain null values.

Therefore, use caution when using aggregate functions on columns that contain null values, because the result set may not be representative of your data. However, if you decide to use aggregate functions with null values, consider the following facts:

- SQL Server aggregate functions, with the exception of the COUNT (*) function, ignore null values in columns.
- The COUNT (*) function counts all rows, even if every column contains a null value. For example, if you execute a SELECT statement that includes the COUNT (*) function on a column that contains a total of 18 rows, two of which contain null values, your result set returns a total of 18 rows.

Example 1

This example lists the number of employees in the **employees** table.

```
USE northwind
SELECT COUNT(*)
 FROM employees
GO
```

Result

```
9

(1 row(s) affected)
```

Example 2

This example lists the number of employees who do not have a null value in the **reportsto** column in the **employees** table, indicating that a reporting manager is defined for that employee.

```
USE northwind
SELECT COUNT(reportsto)
 FROM employees
GO
```

Result

```
8

(1 row(s) affected)
```

GROUP BY Fundamentals

Topic Objective
To provide an overview of the clauses that summarize values for a column.

Lead-in
You typically use aggregate functions in conjunction with the GROUP BY and HAVING clauses.

- **Using the GROUP BY Clause**
- **Using the GROUP BY Clause with the HAVING Clause**

By itself, an aggregate function produces a single summary value for all rows in a column.

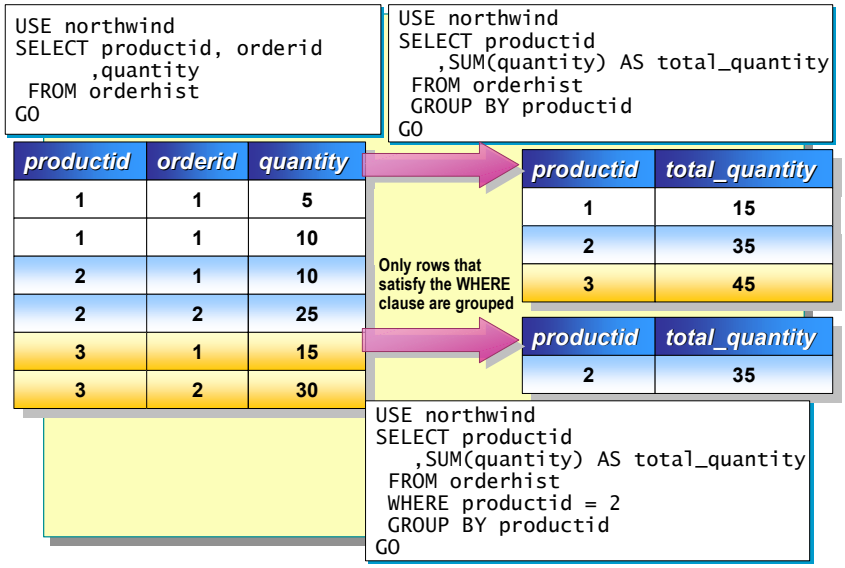
If you want to generate summary values for a column, use aggregate functions with the GROUP BY clause. Use the HAVING clause with the GROUP BY clause to restrict the groups of rows that are returned in the result set.

Note Using the GROUP BY clause does not guarantee a sort order. If you want the results to be sorted, include the ORDER BY clause.

Using the GROUP BY Clause

Topic Objective
To explain how to use the GROUP BY clause to summarize data.

Lead-in
Use the GROUP BY clause on columns or expressions to organize rows into groups and to summarize those groups.



Use the GROUP BY clause on columns or expressions to organize rows into groups and to summarize those groups. For example, use the GROUP BY clause to determine the quantity of each product that was ordered for all orders.

Delivery Tip
The **orderhist** table is specifically created for the examples in this module. This is also included in the Student Materials compact disc.

Compare the result sets in the slide. The table on the left lists all of the rows in the **orderhist** table.

The table on the top right uses the GROUP BY clause to group all **productid** column data and present the total quantity that is ordered for each group.

The table on the bottom right uses the GROUP BY clause and the WHERE clause to further restrict the number of rows returned.

When you use the GROUP BY clause, consider the following facts and guidelines:

- SQL Server produces a column of values for each defined group.
- SQL Server returns only single rows for each group that you specify; it does not return detail information.
- All columns that are specified in the GROUP BY clause must be included in the select list.
- If you include a WHERE clause, SQL Server groups only the rows that satisfy the WHERE clause conditions.
- You can have up to 8,060 bytes in the column list of the GROUP BY clause.
- Do not use the GROUP BY clause on columns that contain multiple null values because the null values are processed as a group.
- Use the ALL keyword with the GROUP BY clause to display all rows with null values in the aggregate columns, regardless of whether the rows satisfy the WHERE clause.

Note The **orderhist** table is specifically created for the examples in this module. The Ordhist.sql script, which is included on the Student Materials compact disc, can be executed to add this table to the **Northwind** database.

Example 1

This example returns information about orders from the **orderhist** table. The query groups and lists each product ID and calculates the total quantity ordered. The total quantity is calculated with the SUM aggregate function and displays one value for each product in the result set.

```
USE northwind
SELECT productid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid
GO
```

Result

productid	total_quantity
1	15
2	35
3	45

```
(3 row(s) affected)
```

Example 2

This example adds a WHERE clause to the query in Example 1. This query restricts the rows to product ID 2 and then groups these rows and calculates the total quantity ordered. Compare this result set to that in Example 1.

```
USE northwind
SELECT productid, SUM(quantity) AS total_quantity
 FROM orderhist
 WHERE productid = 2
 GROUP BY productid
GO
```

Result

productid	total_quantity
2	35

```
(1 row(s) affected)
```


Example 3

This example returns information about orders from the **order details** table. This query groups and lists each product ID and then calculates the total quantity ordered. The total quantity is calculated with the SUM aggregate function and displays one value for each product in the result set. This example does not include a WHERE clause and, therefore, returns a total for each product ID.

```
USE northwind
SELECT productid, SUM(quantity) AS total_quantity
 FROM [order details]
 GROUP BY productid
GO
```

Result

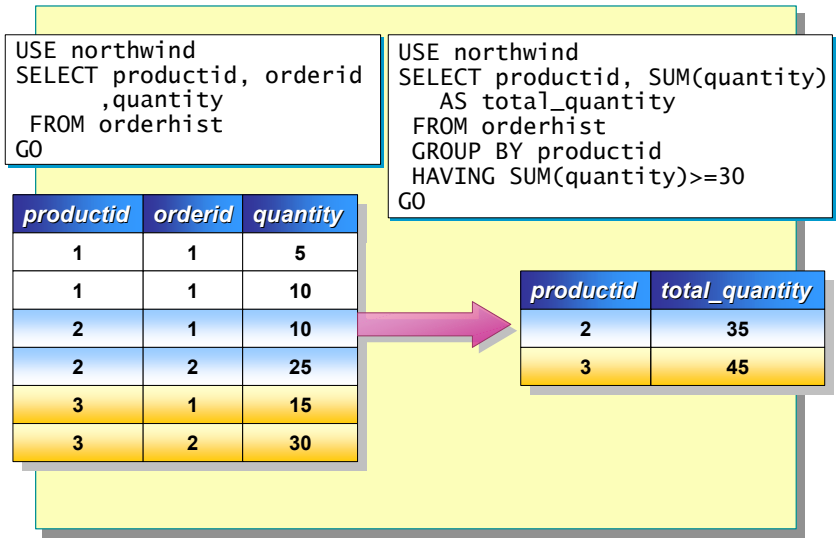
productid	total_quantity
61	603
3	328
32	297
.	
.	
.	

```
(77 row(s) affected)
```

Using the GROUP BY Clause with the HAVING Clause

Topic Objective
To explain how to use the HAVING clause to summarize data further, based on groups.

Lead-in
You can use the HAVING clause to set conditions on groups to include in a result set.



Use the HAVING clause on columns or expressions to set conditions on the groups included in a result set. The HAVING clause sets conditions on the GROUP BY clause in much the same way that the WHERE clause interacts with the SELECT statement.

Delivery Tip
Point out the search condition defined in the HAVING clause in the example in the slide.

The table on the right groups all **productid** column data but presents only the total quantity that is ordered for the groups that meet the HAVING clause search condition.

When you use the HAVING clause, consider the following facts and guidelines:

- Use the HAVING clause only with the GROUP BY clause to restrict the grouping. Using the HAVING clause without the GROUP BY clause is not meaningful.
- You can have up to 128 conditions in a HAVING clause. When you have multiple conditions, you must combine them with logical operators (AND, OR, or NOT).
- You can reference any of the columns that appear in the select list.
- Do not use the ALL keyword with the HAVING clause because the HAVING clause overrides the ALL keyword and returns groups that satisfy only the HAVING clause.

Example 1

This example lists each group of products from the **orderhist** table that has orders of 30 or more units.

```
USE northwind
SELECT productid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid
 HAVING SUM(quantity) >=30
GO
```

Result

productid	total_quantity
2	35
3	45

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

Example 2

This example lists the product ID and quantity for products that have orders for more than 1,200 units.

```
USE northwind
 SELECT productid, SUM(quantity) AS total_quantity
 FROM [order details]
 GROUP BY productid
 HAVING SUM(quantity) > 1200
GO
```

Result

productid	total_quantity
59	1496
56	1263
60	1577
31	1397

```
(4 row(s) affected)
```

Generating Aggregate Values Within Result Sets

Topic Objective
To provide an overview of summarizing values for a table by using the ROLLUP and CUBE operators.

Lead-in
Use the GROUP BY clause with the ROLLUP and CUBE operators to generate aggregate values within result sets. If you do so, you most likely use the GROUPING function to interpret the result set.

- **Using the GROUP BY Clause with the ROLLUP Operator**
- **Using the GROUP BY Clause with the CUBE Operator**
- **Using the GROUPING Function**

Use the GROUP BY clause with the ROLLUP and CUBE operators to generate aggregate values within result sets. The ROLLUP or CUBE operators can be useful for cross-referencing information within a table without having to write additional scripts.

When you use the ROLLUP or CUBE operators, use the GROUPING function to identify the detail and summary values in the result set.

Using the GROUP BY Clause with the ROLLUP Operator

Topic Objective
To explain how to use the ROLLUP operator to summarize data in a table.

Lead-in
Use the ROLLUP operator to summarize data in a table.

```
USE northwind
SELECT productid, orderid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH ROLLUP
 ORDER BY productid, orderid
GO
```

productid	orderid	total_quantity	Description
NULL	NULL	95	Grand total
1	NULL	15	Summarizes only rows for **productid 1**
1	1	5	Detail value for **productid 1**, **orderid 1**
1	2	10	Detail value for **productid 1**, **orderid 2**
2	NULL	35	Summarizes only rows for **productid 2**
2	1	10	Detail value for **productid 2**, **orderid 1**
2	2	25	Detail value for **productid 2**, **orderid 2**
3	NULL	45	Summarizes only rows for **productid 3**
3	1	15	Detail value for **productid 3**, **orderid 1**
3	2	30	Detail value for **productid 3**, **orderid 2**

Delivery Tip
Point out that the NULLs in the example on the slide indicate that those particular rows are created only as a result of the ROLLUP operator.

Use the GROUP BY clause with the ROLLUP operator to summarize group values. The GROUP BY clause with the ROLLUP operator provides data in a standard relational format.

For example, you could generate a result set that includes the quantity that is ordered for each product for each order, the total quantity that is ordered for each product, and the grand total of all products.

When you use the GROUP BY clause with the ROLLUP operator, consider the following facts and guidelines:

- SQL Server processes data from right to left, along the list of columns that are specified in the GROUP BY clause. SQL Server then applies the aggregate function to each group.
- SQL Server adds a row to the result set that displays cumulative aggregates, such as a running sum or a running average. These cumulate aggregates are indicated with a NULL in the result set.
- You can have up to 10 grouping expressions when you use the ROLLUP operator.
- You cannot use the ALL keyword with the ROLLUP operator.
- When you use the ROLLUP operator, ensure that the columns that follow the GROUP BY clause have a relationship that is meaningful in your business environment.

Example 1

This example lists all rows from the **orderhist** table and summary quantity values for each product.

Delivery Tip
The examples in this topic build on one another so that students can understand how ROLLUP builds upon GROUP BY.

```
USE northwind
SELECT productid, orderid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH ROLLUP
 ORDER BY productid, orderid
GO
```

Result

productid	orderid	total_quantity
NULL	NULL	95
1	NULL	15
1	1	5
1	2	10
2	NULL	35
2	1	10
2	2	25
3	NULL	45
3	1	15
3	2	30

(10 row(s) affected)

Example 2

This example returns information about orders from the **order details** table. This query contains a SELECT statement with a GROUP BY clause without the ROLLUP operator. The example returns a list of the total quantity that is ordered for each product on each order, for orders with an **orderid** less than 10250.

```
USE northwind
SELECT orderid, productid, SUM(quantity) AS total_quantity
 FROM [order details]
 WHERE orderid < 10250
 GROUP BY orderid, productid
 ORDER BY orderid, productid
GO
```

Result

orderid	productid	total_quantity
10248	11	12
10248	42	10
10248	72	5
10249	14	9
10249	51	40

(5 row(s) affected)

Example 3

This example adds the ROLLUP operator to the statement in Example 2. The result set includes the total quantity for:

- Each product for each order (also returned by the GROUP BY clause without the ROLLUP operator).
- All products for each order.
- All products for all orders (grand total).

Notice in the result set that the row that contains NULL in both the **productid** and **orderid** columns represents the grand total quantity for all orders for all products. The rows that contain NULL in the **productid** column represent the total quantity of a product for the order in the **orderid** column.

```
USE northwind
SELECT orderid, productid, SUM(quantity) AS total_quantity
 FROM [order details]
 WHERE orderid < 10250
 GROUP BY orderid, productid
 WITH ROLLUP
 ORDER BY orderid, productid
GO
```

Result

orderid	productid	total_quantity
NULL	NULL	76
10248	NULL	27
10248	11	12
10248	42	10
10248	72	5
10249	NULL	49
10249	14	9
10249	51	40

```
(8 row(s) affected)
```

Using the GROUP BY Clause with the CUBE Operator

Topic Objective
To explain how to use the CUBE operator to summarize data in a table.

Lead-in
The CUBE operator differs from the ROLLUP operator in that it creates all possible combinations of groups based on the GROUP BY clause and then applies aggregate functions.

```
USE northwind
SELECT productid, orderid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH CUBE
 ORDER BY productid, orderid
GO
```

The CUBE operator produces two more summary values than the ROLLUP operator

productid	orderid	total_quantity	Description
NULL	NULL	95	Grand total
NULL	1	30	Summarizes all rows for **orderid 1**
NULL	2	65	Summarizes all rows for **orderid 2**
1	NULL	15	Summarizes only rows for **productid 1**
1	1	5	Detail value for **productid 1**, **orderid 1**
1	2	10	Detail value for **productid 1**, **orderid 2**
2	NULL	35	Summarizes only rows for **productid 2**
2	1	10	Detail value for **productid 2**, **orderid 1**
2	2	25	Detail value for **productid 2**, **orderid 2**
3	NULL	45	Summarizes only rows for **productid 3**
3	1	15	Detail value for **productid 3**, **orderid 1**
3	2	30	Detail value for **productid 3**, **orderid 2**

Delivery Tip
Point out that the NULLs in the result set in the example on the slide indicate that those particular rows are created as a result of the CUBE operator.

Use the GROUP BY clause with the CUBE operator to create and summarize all possible combinations of groups based on the GROUP BY clause. Use the GROUP BY clause with the ROLLUP operator to provide data in a standard relational format.

When you use the GROUP BY clause with CUBE operator, consider the following facts and guidelines:

- If you have *n* columns or expressions in the GROUP BY clause, SQL Server returns 2^{n-1} possible combinations in the result set.
- The NULLs in the result set indicate that those particular rows are created as a result of the CUBE operator.
- You can include up to 10 grouping expressions when you use the CUBE operator.
- You cannot use the ALL keyword with the CUBE operator.
- When you use the CUBE operator, ensure that the columns that follow the GROUP BY clause have a relationship that is meaningful in your business environment.

Example

This example returns a result that provides the quantity for each product for each order, total quantity for all products for each order, total quantity for each product for all orders, and a grand total quantity for all products for all orders.

```
USE northwind
SELECT productid, orderid, SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH CUBE
 ORDER BY productid, orderid
GO
```

Result

productid	orderid	total_quantity
NULL	NULL	95
NULL	1	30
NULL	2	65
1	NULL	15
1	1	5
1	2	10
2	NULL	35
2	1	10
2	2	25
3	NULL	45
3	1	15
3	2	30

```
(12 row(s) affected)
```

Using the GROUPING Function

Topic Objective
To explain how the GROUPING function works.

Lead-in
Use the GROUPING function with either the ROLLUP or CUBE operator to distinguish between the detail and summary values in your result set.

```
SELECT productid, GROUPING (productid)
      ,orderid, GROUPING (orderid)
      ,SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH CUBE
 ORDER BY productid, orderid
GO
```

productid		orderid		total_quantity
NULL	1	NULL	1	95
NULL	1	1	0	30
NULL	1	2	0	65
1	0	NULL	1	15
1	0	1	0	5
1	0	2	0	10
2	0	NULL	1	35
2	0	1	0	10
2	0	2	0	25
3	0	NULL	1	45
3	0	1	0	15
3	0	2	0	30

1 represents summary values in the preceding column

0 represents detail values in the preceding column

Delivery Tip
Point out that the result set in the example on the slide is similar to that in the previous slide with one important exception: the GROUPING function is used and two extra columns are included in the result set. The 1 represents summary values, and the 0 represents detail values in the preceding column.

Use the GROUPING function with either the ROLLUP or CUBE operator to distinguish between the detail and summary values in your result set. Using the GROUPING function helps to determine whether the NULLs that appear in your result set are actual null values in the base tables or whether the ROLLUP or CUBE operator generated the row.

When you use the GROUPING function, consider the following facts and guidelines:

- SQL Server produces new columns in the result set for each column that is specified in the GROUPING function.
- SQL Server returns a value of 1 to represent ROLLUP or CUBE summary values in the result set.
- SQL Server returns a value of 0 to represent detail values in the result set.
- You can specify the GROUPING function only on columns that exist in the GROUP BY clause.
- Use the GROUPING function to assist in referencing your result sets programmatically.

Example 1

This example returns a result that provides the quantity for each product for each order, total quantity for all products for each order, total quantity for each product for all orders, and a grand total quantity for all products for all orders. The GROUPING function distinguishes the rows in the result set that the CUBE operator generates.

```
USE northwind
SELECT productid, GROUPING (productid)
      ,orderid, GROUPING (orderid)
      ,SUM(quantity) AS total_quantity
 FROM orderhist
 GROUP BY productid, orderid
 WITH CUBE
 ORDER BY productid, orderid
GO
```

Result

productid		orderid		total_quantity
NULL	1	NULL	1	95
NULL	1	1	0	30
NULL	1	2	0	65
1	0	NULL	1	15
1	0	1	0	5
1	0	2	0	10
2	0	NULL	1	35
2	0	1	0	10
2	0	2	0	25
3	0	NULL	1	45
3	0	1	0	15
3	0	2	0	30

```
(12 row(s) affected)
```

Example 2

This example uses the GROUPING function on the **productid** and **orderid** columns that are listed in the GROUP BY clause. The result set has an additional column after the **productid** and **orderid** columns. The GROUPING function returns a 1 when the values in that particular column have been grouped together by the CUBE operator. The result set includes the total quantity for each product for each order, each product for all orders, all products for each order, and the grand total quantity for all products for all orders.

Notice in the result set that the rows that contain NULL in both the **productid** and the **orderid** columns represent the grand total quantity of all products for all orders. Rows that contain NULL in the **productid** column represent the total quantity for all products for each order. Rows that contain NULL in the **orderid** column represent the total quantity for a product for all orders.

```
USE northwind
SELECT orderid, GROUPING(orderid), productid
      ,GROUPING(productid), SUM(quantity) AS total_quantity
 FROM [order details]
 WHERE orderid < 10250
 GROUP BY orderid, productid
 WITH CUBE
 ORDER BY orderid, productid
GO
```

Result

orderid		productid		total_quantity
NULL	1	NULL	1	76
NULL	1	11	0	12
NULL	1	14	0	9
NULL	1	42	0	10
NULL	1	51	0	40
NULL	1	72	0	5
10248	0	NULL	1	27
10248	0	11	0	12
10248	0	42	0	10
10248	0	72	0	5
10249	0	NULL	1	49
10249	0	14	0	9
10249	0	51	0	40

```
(13 row(s) affected)
```

Using the COMPUTE and COMPUTE BY Clauses

Topic Objective
To explain the purpose of using the COMPUTE and COMPUTE BY clauses.

Lead-in
While the COMPUTE and COMPUTE BY clauses are not ANSI-standard, you may want to use them to print basic reports or to verify results of applications that you are writing.

COMPUTE

```
USE northwind
SELECT productid, orderid
      ,quantity
FROM orderhist
ORDER BY productid, orderid
COMPUTE SUM(quantity)
GO
```

productid	orderid	quantity
1	1	5
1	2	10
2	1	10
2	2	25
3	1	15
3	2	30
	sum	95

COMPUTE BY

```
USE northwind
SELECT productid, orderid, quantity
 FROM orderhist
 ORDER BY productid, orderid
 COMPUTE SUM(quantity) BY productid
 COMPUTE SUM(quantity)
GO
```

productid	orderid	quantity
1	1	5
1	2	10
	sum	15
2	1	10
2	2	25
	sum	35
3	1	15
3	2	30
	sum	45
	sum	95

Delivery Tip
These clauses are not recommended for building applications. However, they can be useful for testing applications.

The COMPUTE and COMPUTE BY clauses generate extra summary rows of data in a non-relational format that is not ANSI-standard. While it is useful for viewing, the output is not well suited for generating result sets to use with other applications.

For example, you may want to use COMPUTE and COMPUTE BY to print basic reports quickly or to verify results of applications that you are writing. However, other tools, such as Crystal Reports or Microsoft Access, offer richer reporting capabilities.

If you use the COMPUTE and COMPUTE BY clauses, consider the following facts:

- You cannot include **text**, **ntext**, or **image** data types in a COMPUTE or COMPUTE BY clause.
- You cannot adjust the format of your result set. For example, if you use the SUM aggregate function, SQL Server displays the word sum in your result set. You cannot change it to read summary.

Generating a Report with Detail and Summary Values for a Column

The COMPUTE clause produces detailed rows and a single aggregate value for a column. When you use the COMPUTE clause, consider the following facts and guidelines:

- You can use multiple COMPUTE clauses with the COMPUTE BY clause in a single statement.
- SQL Server requires that you specify the same columns in the COMPUTE clause that are listed in the select list.
- Do not use the SELECT INTO statement in the same statement as a COMPUTE clause because statements that include COMPUTE do not generate relational output.

Example 1

This example lists each row in the **orderhist** table and generates a grand total for all products that are ordered.

```
USE northwind
SELECT productid, orderid, quantity
 FROM orderhist
 ORDER BY productid, orderid
 COMPUTE SUM(quantity)
GO
```

Result

productid	orderid	total_quantity
1	1	5
1	2	10
2	1	10
2	2	25
3	1	15
3	2	30

```
                                    sum
                                    ==========
                                    95

7 row(s) affected
```

Generating a Report with Detail and Summary Values for Subset of Groups

The COMPUTE BY clause generates detail rows and multiple summary values. Summary values are generated when column values change. Use COMPUTE BY for data that is easily categorized. When you use the COMPUTE BY clause, consider the following facts and guidelines:

- You should use an ORDER BY clause with the COMPUTE BY clause so that rows are grouped together.
- Specify the column names after the COMPUTE BY clause to determine which summary values that SQL Server generates.
- The columns listed after the COMPUTE BY clause must be identical to or a subset of those that are listed after the ORDER BY clause. They must be listed in the same order (left-to-right), start with the same expression, and not skip any expressions.

Example 2

This example lists each row in the **orderhist** table, generates a total that is ordered for each product, and a grand total of all products that are ordered.

```
USE northwind
SELECT productid, orderid, quantity
 FROM orderhist
 ORDER BY productid, orderid
 COMPUTE SUM(quantity) BY productid
 COMPUTE SUM(quantity)
GO
```

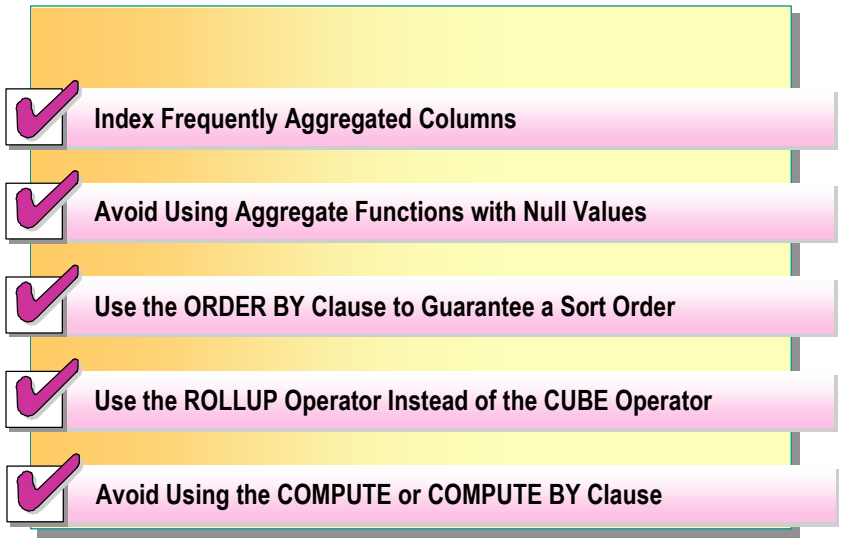
Result

```
productid   orderid     total_quantity
----------- ----------- --------------
1           1           5
1           2           10

                        sum
                        ==========
                        15


2           1           10
2           2           25

                        sum
                        ==========
                        35


3           1           15
3           2           30

                        sum
                        ==========
                        45


                        sum
                        ==========
                        95

10 row(s) affected
```

Recommended Practices

Topic Objective
To list the recommended practices for summarizing data.

Lead-in
To get the most out of using clauses and operators to summarize data, you should consider these recommended practices.



Instructor Note
This course does not cover indexing in detail. For more information on indexing, see course 2073A, *Programming a Microsoft SQL Server 2000 Database.*

When you use clauses and operators to summarize data, consider the following recommended practices:

- Index frequently aggregated columns to improve query performance. For example, adding the **quantity** column to an index improves aggregate operations, such as those in the examples in this module, even when you use the ROLLUP operator.
- Avoid using aggregate functions with columns that contain null values because the result set may not be representative of your data.
- Use the ORDER BY clause to guarantee a sort order in the result set. If you do not use the ORDER BY clause, SQL Server does not guarantee a sort order.
- Use the ROLLUP operator whenever possible because it is more efficient than the CUBE operator. The ROLLUP operator is efficient because it summarizes data as the detail data is processed. The CUBE operator can be resource intensive because of the large number of calculations that it performs.
- Use the COMPUTE or COMPUTE BY clause because it is useful for viewing and printing result sets to test your applications. However, because they generate extra summary rows of data in a non-relational format, the output is not well suited for production databases.

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

Topic	Search on
Summarizing calculations	"aggregate system functions"

Lab A: Grouping and Summarizing Data

Topic Objective
To prepare students for the lab.

Lead-in
In these exercises, you will group and summarize data by using aggregate functions with the GROUP BY, HAVING, COMPUTE, and COMPUTE BY clauses and the ROLLUP and CUBE operators.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Use the GROUP BY and HAVING clauses to summarize data by groups.
- Use the ROLLUP and CUBE operators and GROUPING function to generate summary data.
- Use the COMPUTE and COMPUTE BY clauses to generate control-break reports, grand totals, and averages.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2071A\Labfiles\L04.
- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L04\Answers.

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 **Northwind** 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: 45 minutes

Exercise 1
Using the TOP *n* Keyword

In this exercise, you will use the TOP *n* keyword and the WITH TIES clause to return the top number or percent of rows from a result set. C:\Moc\2071A\Labfiles\L04\Answers contains completed scripts for this exercise.

► To use the TOP *n* keyword to list the top rows of a result set

In this procedure, you will modify a script so that it returns the first ten rows of a query. Answer_TopN1.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 **northwind**.
4. Open and review the C:\Moc\2071A\Labfiles\L04\TopN.sql script, which is a query that calculates the total sale amount for each order in the **order details** table, and returns the results in descending order.
5. Modify the query described in step 4 so that the query returns the first ten rows.

```
USE northwind
 SELECT TOP 10
      orderid
     ,(unitprice * quantity) AS totalsale
 FROM [order details]
 ORDER BY (unitprice * quantity) DESC
GO
```

6. Execute the query to verify that it returns ten rows.

Result

Your result will look similar to the following result set.

```
orderid     totalsale
10865       15810.0000
10981       15810.0000
10353       10540.0000
10417       10540.0000
10889       10540.0000
10424       10329.2000
10897       9903.2000
10372       8432.0000
10540       7905.0000
10816       7905.0000

(10 row(s) affected)
```

► To list the top values of a result set using the TOP *n* keyword

In this procedure, you will use the TOP *n* keyword to list the top values of a result set. Answer_TopN2.sql is a completed script for this procedure.

1. Modify the query described in step 5 of the previous procedure to return the top ten products (including ties) having the highest total quantity.

```
USE northwind
 SELECT TOP 10 WITH TIES
       orderid
      ,(unitprice * quantity) AS totalsale
 FROM [order details]
 ORDER BY (unitprice * quantity) DESC
GO
```

2. Execute the query to verify that it returns eleven rows.

Result

Your result will look similar to the following result set.

```
orderid     totalsale
----------- ---------------------------------------
10865       15810.0000
10981       15810.0000
10353       10540.0000
10417       10540.0000
10889       10540.0000
10424       10329.2000
10897       9903.2000
10372       8432.0000
10540       7905.0000
10816       7905.0000
10817       7905.0000

(11 row(s) affected)
```

3. Why were more rows returned from the query that asked for the top 10 values that included ties?

   The TOP 10 operator specifies that only the first ten rows should be returned. The TOP 10 WITH TIES operator specifies that all rows with values that are in the list of the top 10 values should be returned, regardless of how many rows that might be.

Exercise 2
Using the GROUP BY and HAVING Clauses

In this exercise, you will use the GROUP BY and HAVING clauses to summarize data from the **Northwind** database. C:\Moc\2071A\Labfiles\L04\Answers contains completed scripts for this exercise.

► To use the GROUP BY clause to summarize data

In this procedure, you will open a script that contains a query that includes the GROUP BY clause. Then you will modify the query to obtain different results.

1. Open and review the C:\Moc\2071A\Labfiles\L04\Groupby.sql script, which is a query that calculates the total quantity of items ordered for two different categories of items in the **order details** table.
2. Execute the query to review the results.

Result Your result will look similar to the following result set.

```
categoryid      total_quantity
----------      --------------
1               9532
2               5298

(2 row(s) affected)
```

► To calculate the total quantity for each category

Answer_Groupby1.sql is a completed script for this procedure.

1. Modify the script from step 1 of the previous procedure to summarize the quantity by category for all products, regardless of category.

```
USE northwind
SELECT categoryid, SUM(quantity) AS total_quantity
 FROM [order details] AS od
 INNER JOIN products AS p
  ON od.productid = p.productid
 GROUP BY categoryid
GO
```

2. Execute the query to review the results.

Result Your result will look similar to the following result set.

```
categoryid      total_quantity
----------      --------------
1h              9532
2               5298
3               7906
4               9149
5               4562
6               4199
7               2990
8               7681

(8 row(s) affected)
```

► To calculate the total quantity for each order

In this procedure, you will calculate the total quantity for each order. Answer_Groupby2.sql is a completed script for this procedure.

1. Modify the script from step 1 of the previous procedure to summarize the quantity by **orderid** for all products, regardless of category.

```
USE northwind
SELECT orderid, SUM(quantity) AS total_quantity
 FROM [order details] AS od
 INNER JOIN products AS p
  ON od.productid = p.productid
 GROUP BY orderid
GO
```

2. Execute the query to review the results.

Result

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

```
ordered     total_quantity
----------- --------------
10248       27
10249       49
10250       60
.
.
.
11075       42
11076       50
11077       72

(830 row(s) affected)
```

► To calculate the number of orders with more than 250 units ordered

In this procedure, you will calculate the number of orders with more than 250 units ordered. Answer_Groupby3.sql is a completed script for this procedure.

1. Modify the script from step 1 of the previous procedure to summarize the quantity by **orderid** for all products, regardless of category, and only return orders that had more than 250 units ordered.

```
USE northwind
SELECT orderid, SUM(quantity) AS total_quantity
 FROM [order details] AS od
 INNER JOIN products AS p
  ON od.productid = p.productid
 GROUP BY orderid
 HAVING SUM(quantity) > 250
GO
```

2. Execute the query to review the results.

Result

Your result will look similar to the following result set.

```
ordered     total_quantity
----------- --------------
10515       286
10612       263
10658       255
10678       280
10847       288
10895       346
10990       256
11030       330

(8 row(s) affected)
```

Exercise 3
Using the ROLLUP and CUBE Operators

In this exercise, you will use the ROLLUP and CUBE operators to generate summary data. You also will use the GROUPING function to determine the result rows that are summaries. C:\Moc\2071A\Labfiles\L04\Answers contains completed scripts for this exercise.

► To use the ROLLUP operator to generate summary results

In this procedure, you will use the ROLLUP operator with the GROUP BY and HAVING clauses to generate summary results. Answer_Rollup1.sql is a completed script for this procedure.

1. Open and review the C:\Moc\2071A\Labfiles\L04\Rollup.sql script, which is a query that summarizes the quantity of items that were ordered by **productid** and **ordered**, and performs a rollup calculation.
2. Modify the query from step 1 to limit the result to product number 50 by using a WHERE clause, and then execute the query.

```
USE northwind
SELECT productid, orderid, SUM(quantity) AS total_quantity
 FROM [order details]
 WHERE productid = 50
 GROUP BY productid, orderid
  WITH ROLLUP
 ORDER BY productid, orderid
GO
```

3. Execute the query to review the results. Make note of the rows that have null values.

Result

Your result will look similar to the following result set.

productid	orderid	total_quantity
NULL	NULL	235
50	NULL	235
50	10350	15
50	10383	15
50	10429	40
50	10465	25
50	10637	25
50	10729	40
50	10751	20
50	10920	24
50	10948	9
50	11072	22

(12 row(s) affected)

1. What is the significance of the null values in the **productid** and **orderid** columns?

   The null values in a row indicate that the value in the total_quantity column for that row is the sum of all of the total_quantity values without grouping on the column that has the null value.

   For example, the total_quantity value in the row where productid and orderid are both null is the sum of all of the total_quantity values in the table.

   _____

   _____

► To use the CUBE operator to generate summary results

In this procedure, you will use the CUBE operator and the GROUPING function to distinguish between summary and detail rows in the result set. Answer_Cube1.sql is a completed script for this procedure.

1. Open and review the C:\Moc\2071A\Labfiles\L04\Rollup.sql script, which is a query that summarizes the quantity of items that were ordered by **productid** and **ordered**, and performs a rollup calculation.
2. Modify the query from step 1 to use the CUBE operator instead of the ROLLUP operator. Also, use the GROUPING function on the **productid** and **orderid** columns to distinguish between summary and detail rows in the result set, and then execute the query.

```
USE northwind
SELECT productid
      ,GROUPING(productid)
      ,orderid
      ,GROUPING(orderid)
      ,SUM(quantity) AS total_quantity
 FROM [order details]
 WHERE productid = 50
 GROUP BY productid, orderid
  WITH CUBE
 ORDER BY productid, orderid
GO
```

3. Execute the query to review the results.

Result

Your result will look similar to the following result set.

productid		orderid		total_quantity
NULL	1	NULL	1	235
NULL	1	10350	0	15
NULL	1	10383	0	15
NULL	1	10429	0	40
NULL	1	10465	0	25
NULL	1	10637	0	25
NULL	1	10729	0	40
NULL	1	10751	0	20
NULL	1	10920	0	24
NULL	1	10948	0	9
NULL	1	11072	0	22
50	0	NULL	1	235
50	0	10350	0	15
50	0	10383	0	15
50	0	10429	0	40
50	0	10465	0	25
50	0	10637	0	25
50	0	10729	0	40
50	0	10751	0	20
50	0	10920	0	24
50	0	10948	0	9
50	0	11072	0	22

```
(22 row(s) affected)
```

Which rows are summaries?

The rows with the number 1 in a GROUPING function column.

Which rows are summaries by product? By order?

If a number 1 is present in the column generated by the GROUPING function for the productid column, the row is a summary by order. The productid for that row is NULL because it is a summary row rather than a detail row that contains a NULL. The row with a number 1 in the orderid GROUPING column is a summary row for product number 50. The row with a number 1 in both GROUPING columns is a grand total.

Exercise 4
Using the COMPUTE and COMPUTE BY Clauses

In this exercise, you will use the COMPUTE and COMPUTE BY clauses to generate control-break reports and end-of-report totals and averages.

C:\Moc\2071A\Labfiles\L04\Answers contains completed scripts for this exercise.

► To use the COMPUTE clause to generate reports

In this procedure, you will modify an existing query by adding the COMPUTE and COMPUTE BY clauses to generate subtotals and grand totals. Answer_Compute1.sql is a completed script for this procedure.

1. Open and review the C:\Moc\2071A\Labfiles\L04\Compute.sql script, which is a query that returns the **orderid** and **quantity** ordered for all orders with an **orderid** > 11070.
2. Modify the query from step 1 to generate a grand total for the quantity column using the COMPUTE clause.

```
USE northwind
SELECT orderid, quantity
 FROM [order details]
 WHERE orderid >= 11070
 COMPUTE SUM(quantity)
GO
```

3. Execute the query to review the results.

Result

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

```
ordered     quantity
----------- -----------
11070       40
11070       20
11070       30
.
.
.
11077       24
11077       4
11077       1

            Sum
            ==========
            543

(45 row(s) affected)
```

► To use the COMPUTE BY clause to generate reports

In this procedure, you will modify an existing query by using the COMPUTE BY clause to generate grand totals. Answer_Compute2.sql is a completed script for this procedure.

1. Open and review the C:\Moc\2071A\Labfiles\L04\Compute.sql script, which is a query that returns the **orderid** and **quantity** ordered for all orders with an **orderid** > 11070.
2. Modify the query from step 1 to generate a control-break report that provides the total quantity for order numbers 11075 and 11076.

```
USE northwind
 SELECT orderid, quantity
 FROM [order details]
 WHERE orderid in ( 11075, 11076 )
 ORDER BY orderid
 COMPUTE SUM(quantity) BY orderid
GO
```

3. Execute the query to review the results.

Result

Your result will look similar to the following result set.

```
ordered     quantity
----------- -----------
11075       10
11075       30
11075       2

sum
==========
42

11076       20
11076       20
11076       10

sum
==========
50

(8 row(s) affected)
```

► To add total quantity and average quantity to the end of the control-break report

In this procedure, you will add total quantity and average quantity to the end of the control-break report. Answer_Compute3.sql is a completed script for this procedure.

1. Modify the query from step 1 of the previous procedure to add total quantity and average quantity to the end of the control-break report.

```
USE northwind
SELECT orderid, quantity
 FROM [order details]
 WHERE orderid in ( 11075, 11076 )
 ORDER BY orderid
 COMPUTE SUM(quantity) BY orderid
 COMPUTE SUM(quantity)
 COMPUTE AVG(quantity)
GO
```

2. Execute the query to review the results.

Result

Your result will look similar to the following result set. Notice that this result set is similar to that of step 3 of the previous procedure, with the addition of end-of-report totals (total quantity and average quantity).

```
orderid     quantity
----------- -----------
11075       10
11075       30
11075       2
            Sum
            ==========
            42

11076       20
11076       20
11076       10
            Sum
            ==========
            50

            Sum
            ==========
            92

            Avg
            ==========
            15

(10 row(s) affected
```

Review

Topic Objective
To reinforce module objectives by reviewing important topics.

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

- Listing the TOP *n* Values
- Using Aggregate Functions
- GROUP BY Fundamentals
- Generating Aggregate Values Within Result Sets
- Using the COMPUTE and COMPUTE BY Clauses

Use these questions to review module topics.

Ask students whether they have any questions.

1. An employee in the marketing department has asked you to provide summary data for product sales. She needs all breakfast cereals summarized by type (hot, cold, or low-fat), manufacturer, and size of the store where the product was sold (small, medium, or large). Assuming that a single table holds all this information, what clauses or operators might you use with the SELECT statement? Why?

   The GROUP BY clause with the CUBE operator is the best answer.

   The GROUP BY and HAVING clauses provide only one level of summaries (or groups).

   The ROLLUP operator provides summaries for one category.

   The CUBE operator provides summaries for multiple categories.

   You also could use the COMPUTE or COMPUTE BY clauses to generate basic reports.

2. Your manager has asked you to deliver a file that includes all of the data from Question 1 to another development group that is responsible for report generation and graphing tools. Would using the COMPUTE and COMPUTE BY clauses be appropriate for this task? Why or why not?

   No. The COMPUTE and COMPUTE BY clauses generate extra summary rows of data in a non-relational format. While it is useful for viewing, the output is not well suited for generating result sets to use with other applications. You could use the GROUP BY clause and the CUBE or ROLLUP operator to provide data in a standard relational format that other clients can use easily.

3. You are reviewing the results of a SELECT statement that used the GROUP BY clause and the CUBE operator. You see null values in the result set, and you know that null values are allowed in the tables that the SELECT statement uses. How can you distinguish between detail rows and summary rows with null values?

   Use the GROUPING function on the columns that allow null values. A value of 1 appears in the column generated by the GROUPING function if that row is a summary row.

4. You need to provide a list of the top 100 products, as well as the products that are in the bottom five percent of sales. Can you use the SELECT TOP *n* [PERCENT] statement to answer each question? Are there other ways to answer the questions?

   Yes, you can use the SELECT TOP *n* [PERCENT] statement to answer each question. The first question would be answered with the SELECT TOP 100...ORDER BY...DESC statement so that the items with the highest quantity sold would be at the top of the list.

   The second question would be answered with the SELECT TOP 5 PERCENT...ORDER BY...ASC statement so that the items with the lowest quantity sold would be at the top of the list.