Module 14: Pivoting and Grouping Sets

Contents:

Module Overview

This module discusses more advanced manipulations of data, building on the basics you have learned so far in the course. First, you will learn how to use the PIVOT and UNPIVOT operators to change the orientation of data from column-oriented to row-oriented and back. Next, you will learn how to use the GROUPING SET subclause of the GROUP BY clause to specify multiple groupings in a single query. This will include the use of the CUBE and ROLLUP subclauses of GROUP BY to automate the setup of grouping sets.

Objectives

After completing this module, you will be able to:

- Write queries that pivot and unpivot result sets.
- Write queries that specify multiple groupings with grouping sets.

Lesson 1: Writing Queries with PIVOT and UNPIVOT

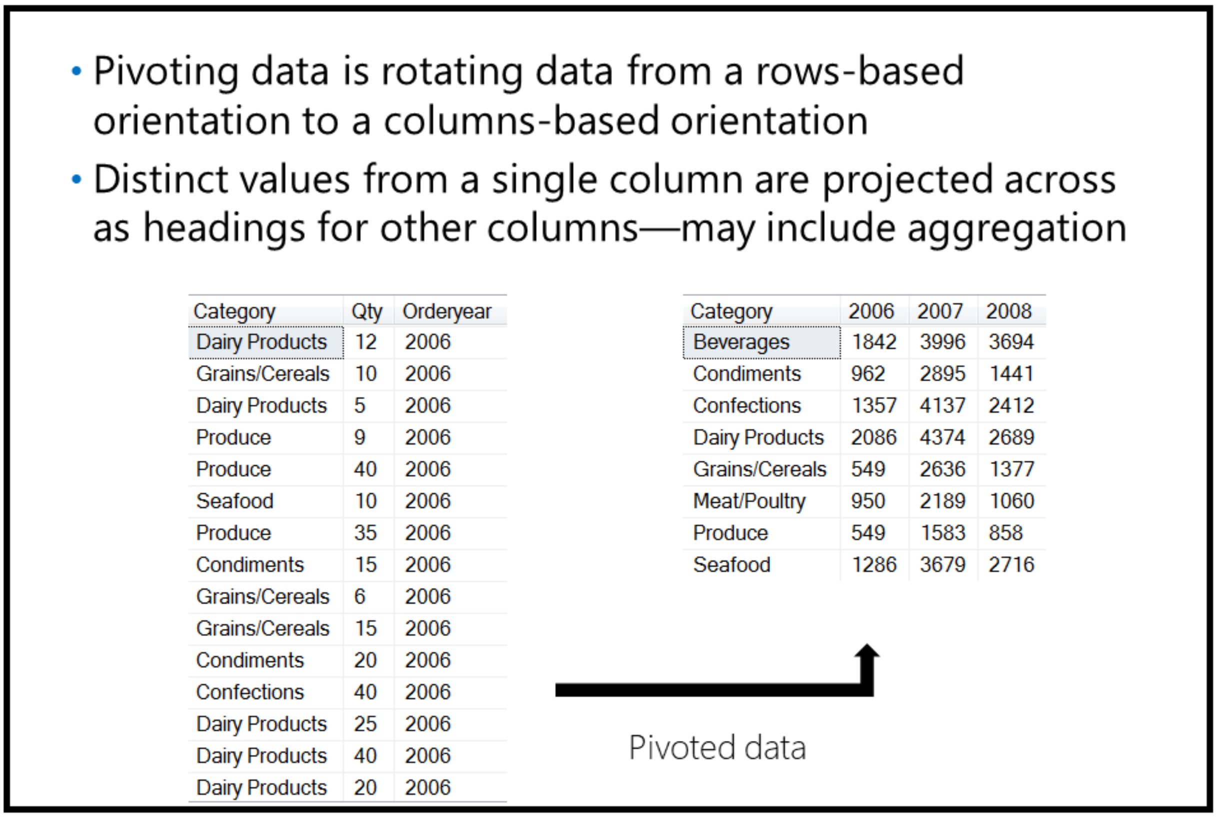
Sometimes you may need to present data in a different orientation to how it is stored, with respect to row and column layout. For example, some data may be easier to compare if you can arrange values across columns of the same row. In this lesson, you will learn how to use the T-SQL PIVOT operator to accomplish this. You will also learn how to use the UNPIVOT operator to return the data to a rows-based orientation.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how pivoting data can be used in T-SQL queries.
- Write queries that pivot data from rows to columns using the PIVOT operator.
- Write queries that unpivot data from columns back to rows using the UNPIVOT operator.

What Is Pivoting?



- Pivoting data is rotating data from a rows-based orientation to a columns-based orientation
- Distinct values from a single column are projected across as headings for other columns—may include aggregation

Category	Qty	Orderyear
Dairy Products	12	2006
Grains/Cereals	10	2006
Dairy Products	5	2006
Produce	9	2006
Produce	40	2006
Seafood	10	2006
Produce	35	2006
Condiments	15	2006
Grains/Cereals	6	2006
Grains/Cereals	15	2006
Condiments	20	2006
Confections	40	2006
Dairy Products	25	2006
Dairy Products	40	2006
Dairy Products	20	2006

Category	2006	2007	2008
Beverages	1842	3996	3694
Condiments	962	2895	1441
Confections	1357	4137	2412
Dairy Products	2086	4374	2689
Grains/Cereals	549	2636	1377
Meat/Poultry	950	2189	1060
Produce	549	1583	858
Seafood	1286	3679	2716

Pivoting data in SQL Server rotates its display from a rows-based orientation to a columns-based orientation. It does this by consolidating values in a column to a list of distinct values, and then projecting that list across as column headings. Typically, this includes aggregation to column values in the new columns.

For example, the partial source data below lists repeating values for Category and Orderyear, along with values for Qty, for each instance of a Category/Orderyear pair:

```
Category        Qty    Orderyear
--------------- ------ -----------
Dairy Products  12     2006
Grains/Cereals  10     2006
Dairy Products  5      2006
Seafood         2      2007
Confections     36     2007
Condiments      35     2007
Beverages       60     2007
Confections     55     2007
Condiments      16     2007
Produce         15     2007
Dairy Products  60     2007
Dairy Products  20     2007
Confections     24     2007
...
Condiments      2      2008

 (2155 row(s) affected)
```

To analyze this by category and year, you might want to arrange the values to be displayed as follows, summing the Qty column along the way:

```
Category       2006 2007 2008
-------------- ---- ---- ----
Beverages      1842 3996 3694
Condiments     962  2895 1441
Confections    1357 4137 2412
Dairy Products 2086 4374 2689
```

```
Grains/Cereals 549  2636 1377
Meat/Poultry   950  2189 1060
Produce        549  1583 858
Seafood        1286 3679 2716

(8 row(s) affected)
```

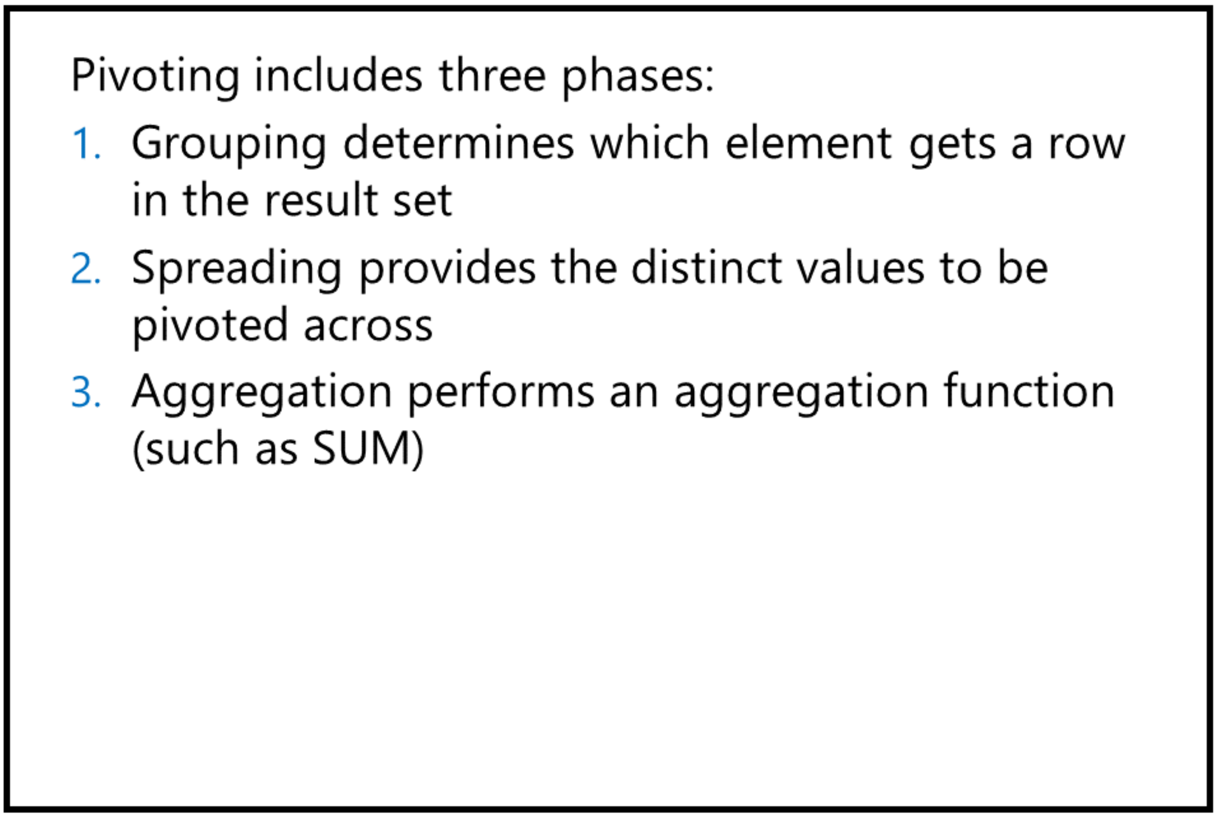
In the pivoting process, each distinct year was created as a column header, and values in the Qty column were grouped by Category and aggregated. This is a very useful technique in many scenarios.

For more information, see the SQL Server Technical Documentation:

Using PIVOT and UNPIVOT

http://go.microsoft.com/fwlink/?LinkID=402781

Elements of PIVOT



Pivoting includes three phases:

1. Grouping determines which element gets a row in the result set
2. Spreading provides the distinct values to be pivoted across
3. Aggregation performs an aggregation function (such as SUM)

The T-SQL PIVOT table operator, introduced in Microsoft® SQL Server® 2005, operates on the output of the FROM clause in a SELECT statement. To use PIVOT, you need to supply three elements to the operator:

- **Grouping**: in the FROM clause, you need to provide the input columns. From those columns, PIVOT will determine which column(s) will be used to group the data for aggregation. This is based on looking at which columns are not being used as other elements in the PIVOT operator.
- **Spreading**: you need to provide a comma-delimited list of values to be used as the column headings for the pivoted data. The values need to occur in the source data.
- **Aggregation**: you need to provide an aggregation function (SUM, and so on) to be performed on the grouped rows.

Additionally, you need to assign a table alias to the result table of the PIVOT operator. The following example shows the elements in place:

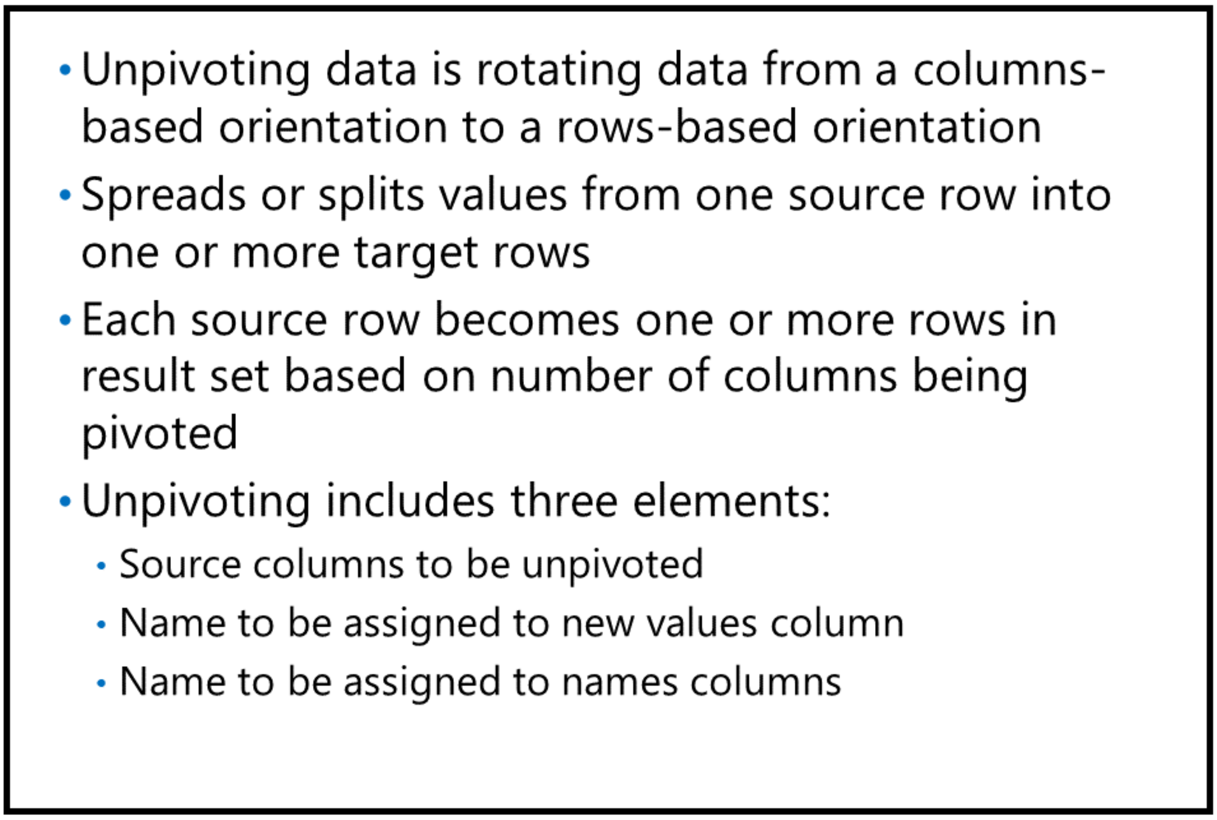
PIVOT Example

```
SELECT  Category, [2006],[2007],[2008]
FROM  ( SELECT  Category, Qty, Orderyear FROM Sales.CategoryQtyYear)
AS D
      PIVOT(SUM(qty) FOR orderyear IN ([2006],[2007],[2008])) AS
pvt;
```

Note: Any attributes in the source subquery, that are not used for aggregation or spreading, will be used as grouping elements—be sure that no unnecessary attributes are included in the subquery.

One of the challenges in writing queries using PIVOT is the need to supply a fixed list of spreading elements to the PIVOT operator, such as the specific order year values above. Later in this course, you will learn how to write dynamically-generated queries, which may help you write PIVOT queries with more flexibility.

Writing Queries with UNPIVOT



- Unpivoting data is rotating data from a columns-based orientation to a rows-based orientation
- Spreads or splits values from one source row into one or more target rows
- Each source row becomes one or more rows in result set based on number of columns being pivoted
- Unpivoting includes three elements:
  - Source columns to be unpivoted
  - Name to be assigned to new values column
  - Name to be assigned to names columns

Unpivoting data is the logical reverse of pivoting data. Instead of turning rows into columns, unpivot turns columns into rows. This is a technique useful in taking data that has already been pivoted (with or without using a T-SQL PIVOT operator) and returning it to a row-oriented tabular display. SQL Server provides the UNPIVOT table operator to accomplish this.

When unpivoting data, one or more columns is defined as the source to be converted into rows. The data in those columns is spread, or split, into one or more new rows, depending on how many columns are being unpivoted.

In the following source data, three columns will be unpivoted. Each Orderyear value will be copied into a new row and associated with its Category value. Any NULLs will be removed in the process and no row is created:

```
Category        2006 2007 2008
--------------- ---- ---- ----
Beverages       1842 3996 3694
Condiments      962  2895 1441
Confections     1357 4137 2412
```

```
Dairy Products  2086  374 2689
Grains/Cereals  549  2636 1377
Meat/Poultry    950  2189 1060
Produce         549  1583 858
Seafood         1286 3679 2716
```

For each intersection of Category and Orderyear, a new row will be created, as in these partial results:

```
category        qty  orderyear
--------------- ---- ---------
Beverages       1842 2006
Beverages       3996 2007
Beverages       3694 2008
Condiments      962  2006
Condiments      2895 2007
Condiments      1441 2008
Confections     1357 2006
Confections     4137 2007
Confections     2412 2008
```

Note: Unpivoting does not restore the original data. Detail-level data was lost during the aggregation process in the original pivot. UNPIVOT has no ability to allocate values to return to original detail values.

To use the UNPIVOT operator, you need to provide three elements:

- Source columns to be unpivoted.
- A name for the new column that will display the unpivoted values.
- A name for the column that will display the names of the unpivoted values.

Note: As with PIVOT, you will define the output of the UNPIVOT table operator as a derived table and provide its name.

UNPIVOT Example

```
SELECT category, qty, orderyear
FROM Sales.PivotedCategorySales
UNPIVOT(qty FOR orderyear IN([2006],[2007],[2008])) AS unpvt;
```

The partial results:

```
category        qty         orderyear
--------------- ----------- ---------
Beverages       1842        2006
Beverages       3996        2007
Beverages       3694        2008
Condiments      962         2006
Condiments      2895        2007
Condiments      1441        2008
Confections     1357        2006
Confections     4137        2007

Confections     2412        2008
Dairy Products  2086        2006
Dairy Products  4374        2007
Dairy Products  2689        2008
```

Demonstration: Writing Queries with PIVOT and UNPIVOT

In this demonstration, you will see how to use PIVOT and UNPIVOT.

Demonstration Steps

Use PIVOT and UNPIVOT

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod14\Setup.cmd** as an administrator.
3. At the command prompt, type **y**, and then press Enter.
4. When the script completes, close the command prompt window.
5. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
6. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod14\Demo** folder.
7. In Solution Explorer, open the **11 - Demonstration A.sql** script file.
8. Select the code under the comment **Step 1**, and then click **Execute**.
9. Select the code under the comment **Step 2**, and then click **Execute**.
10. Select the code under the comment **Step 3**, and then click **Execute**.
11. Select the code under the comment **Step 4**, and then click **Execute**.
12. Select the code under the comment **Step 5**, and then click **Execute**.
13. Select the code under the comment **Step 6**, and then click **Execute**.
14. Select the code under the comment **Step 7**, and then click **Execute**.
15. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have the following query:

SELECT category, qty, orderyear

FROM Sales.PivotedCategorySales

UNPIVOT(qty FOR orderyear) AS unpvt;
In this query, you have provided a name for the new column that will display the unpivoted values (“qty”). You have also provided a name for the column that will display the names of the unpivoted values (orderyear). What else must you provide for the UNPIVOT query to execute?

Show solution Reset

Lesson 2: Working with Grouping Sets

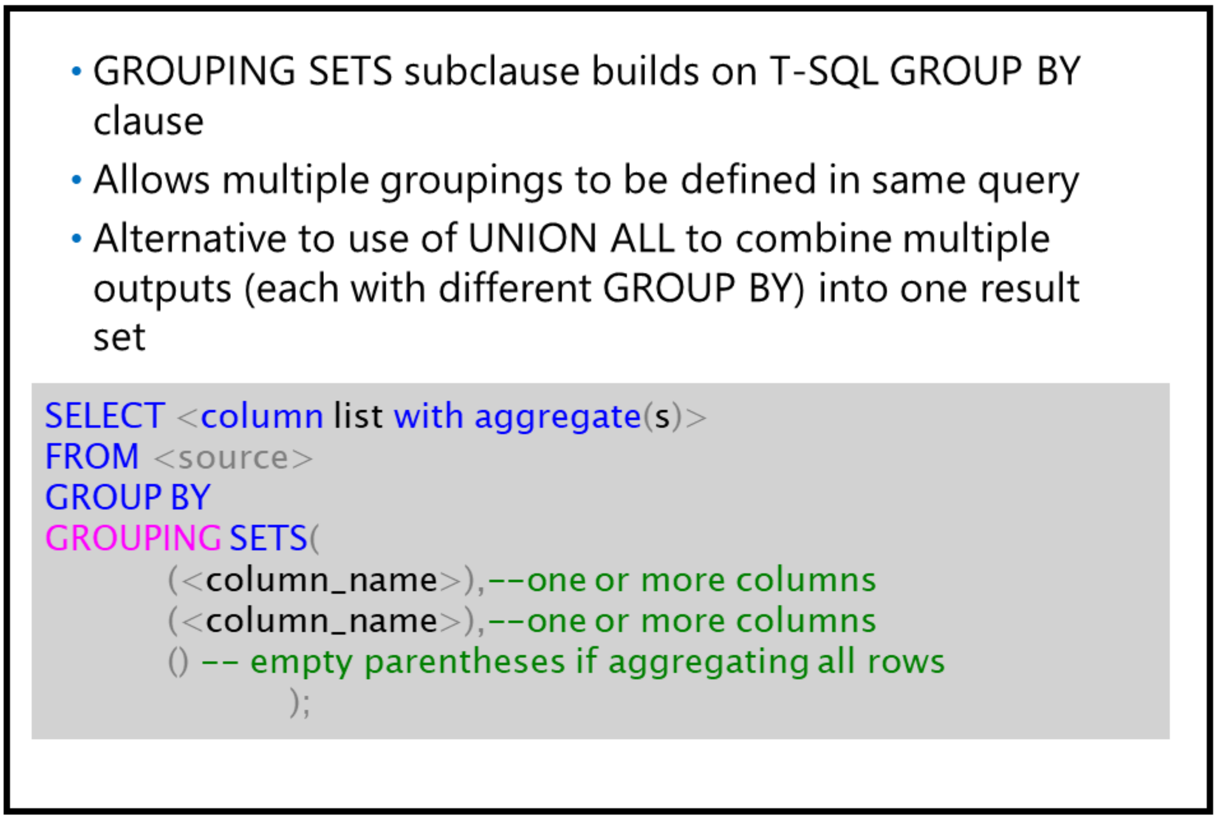
As you learned earlier in this course, you can use the GROUP BY clause in a SELECT statement to arrange rows in groups, typically to support aggregations. However, if you need to group by different attributes at the same time, for example to report at different levels, you will need multiple queries combined with UNION ALL. SQL Server 2008 and later provides the GROUPING SETS subclause to GROUP BY, which enables multiple sets to be returned in the same query.

Lesson Objectives

After completing this lesson, you will be able to:

- Write queries using the GROUPING SETS subclause.
- Write queries that use ROLLUP AND CUBE.
- Write queries that use the GROUPING_ID function.

Writing Queries with Grouping Sets



- GROUPING SETS subclause builds on T-SQL GROUP BY clause
- Allows multiple groupings to be defined in same query
- Alternative to use of UNION ALL to combine multiple outputs (each with different GROUP BY) into one result set

```
SELECT <column list with aggregate(s)>
FROM <source>
GROUP BY
GROUPING SETS(
        (<column_name>),--one or more columns
        (<column_name>),--one or more columns
        () -- empty parentheses if aggregating all rows
                );
```

If you need to produce aggregates of multiple groupings in the same query, you can use the GROUPING SETS subclause of the GROUP BY clause.

GROUPING SETS provide an alternative to using UNION ALL to combine results from multiple individual queries, each with its own GROUP BY clause.

GROUPING SETS Syntax

```
SELECT <column list with aggregate(s)>
FROM <source>
GROUP BY
GROUPING SETS(
        (<column_name>),--one or more columns
        (<column_name>),--one or more columns
        () -- empty parentheses if aggregating all rows
                );
```

With GROUPING SETS, you can specify which attributes to group on and their order. If you want to group on any possible combination of attributes instead, see the topic on CUBE and ROLLUP later in this lesson.

GROUPING SETS Example

```
Code Example Content
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY
        GROUPING SETS((Category),(Cust),())
ORDER BY Category, Cust;
```

The results:

```
Category    Cust TotalQty
----------- ---- --------
NULL        NULL 999
NULL        1    80
NULL        2    12
NULL        3    154
NULL        4    241
NULL        5    512
Beverages   NULL 513
Condiments  NULL 114
Confections NULL 372
```

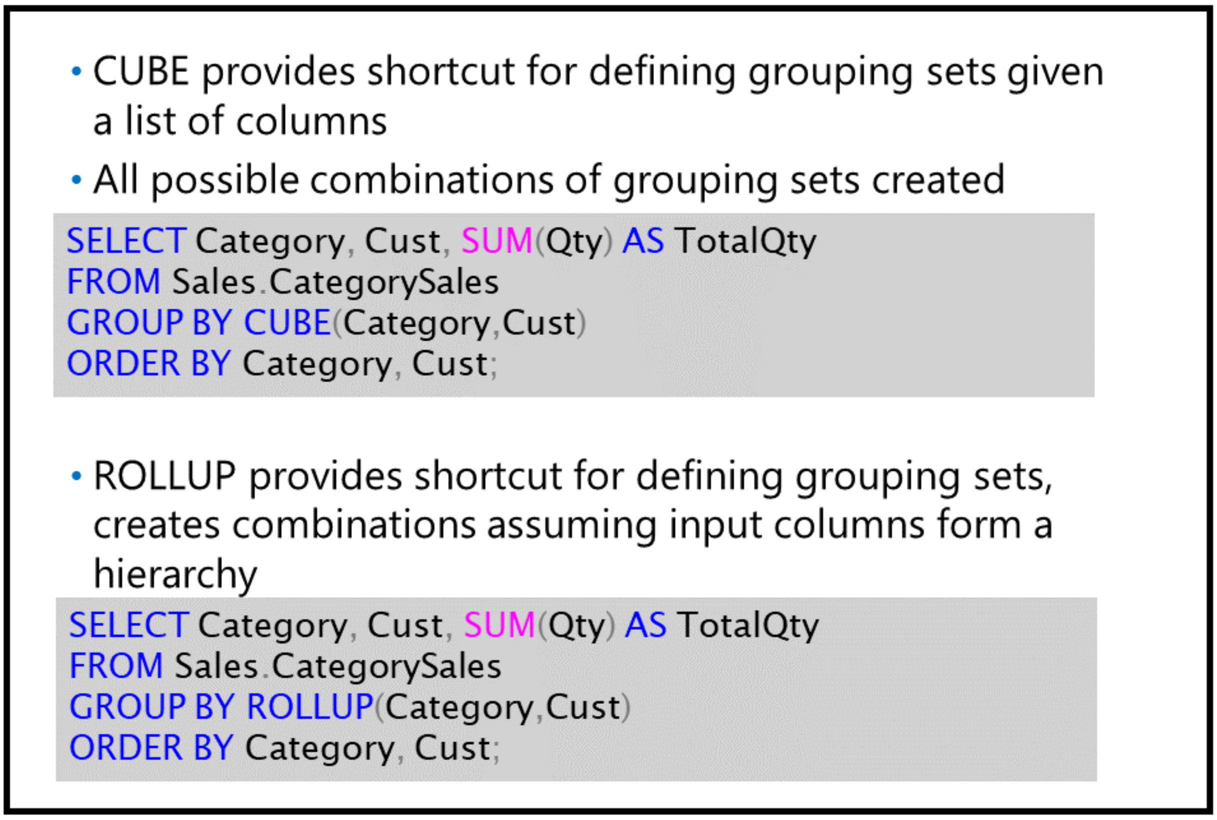
Note the presence of NULLs in the results. NULLs may be returned because a NULL was stored in the underlying source, or because it is a placeholder in a row generated as an aggregate result. For example, in the previous results, the first row displays NULL, NULL, 999. This represents a grand total row. The NULL in the Category and Cust columns are placeholders because neither Category nor Cust take part in the aggregation.

For more information, see Using GROUP BY with ROLLUP, CUBE, and GROUPING SETS in the SQL Server Technical Documentation:

Using GROUP BY with ROLLUP, CUBE, and GROUPING SETS

http://go.microsoft.com/fwlink/?LinkID=402782

CUBE and ROLLUP



- CUBE provides shortcut for defining grouping sets given a list of columns
- All possible combinations of grouping sets created

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust)
ORDER BY Category, Cust;
```

- ROLLUP provides shortcut for defining grouping sets, creates combinations assuming input columns form a hierarchy

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY ROLLUP(Category,Cust)
ORDER BY Category, Cust;
```

Like GROUPING SETS, the CUBE and ROLLUP subclauses also enable multiple groupings for aggregating data. However, CUBE and ROLLUP do not need you to specify each set of attributes to group. Instead, given a set of columns, CUBE will determine all possible combinations and output groupings. ROLLUP creates combinations, assuming the input columns represent a hierarchy. Therefore, CUBE and ROLLUP can be thought of as shortcuts to GROUPING SETS.

To use CUBE, append the keyword CUBE to the GROUP BY clause and provide a list of columns to group.

CUBE Example

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust);
```

This will output groupings for the following combinations: (Category, Cust), (Cust, Category), (Cust), (Category) and the aggregate on all empty ().

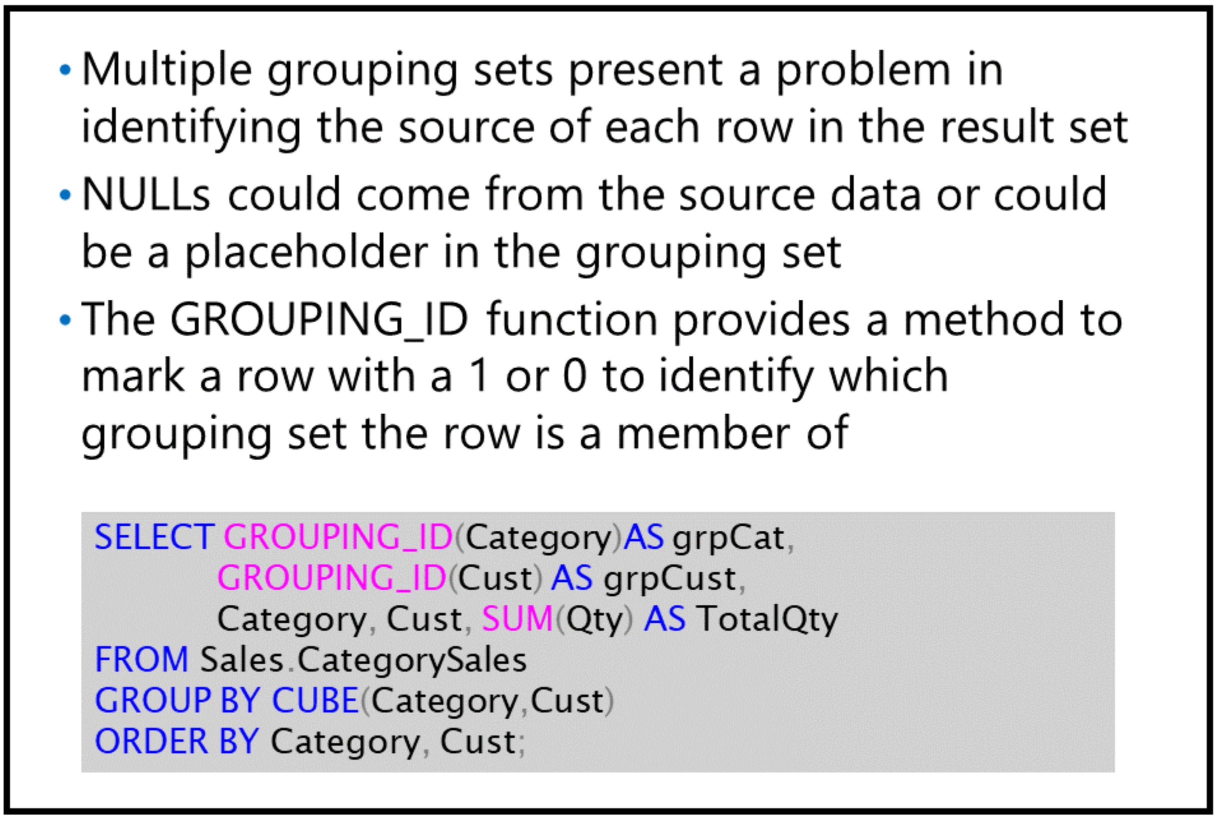
ROLLUP Example

```
SELECT Category, Subcategory, Product, SUM(Qty) AS TotalQty
FROM Sales.ProductSales
GROUP BY ROLLUP(Category,Subcategory, Product);
```

This will output groupings for the following combinations: (Category, Subcategory, Product), (Category, Subcategory), (Category), and the aggregate on all empty (). Note that the order in which columns are supplied is significant: ROLLUP assumes that the columns are listed in an order that expresses a hierarchy.

Note: The example just given is for illustration only. Object names do not correspond to the sample database supplied with the course.

GROUPING_ID



- Multiple grouping sets present a problem in identifying the source of each row in the result set
- NULLs could come from the source data or could be a placeholder in the grouping set
- The GROUPING_ID function provides a method to mark a row with a 1 or 0 to identify which grouping set the row is a member of

```
SELECT GROUPING_ID(Category)AS grpCat,
          GROUPING_ID(Cust) AS grpCust,
          Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust)
ORDER BY Category, Cust;
```

As you have seen, multiple grouping sets allow you to combine different levels of aggregation in the same query. You have also learned that SQL Server will mark placeholder values with NULL if a row does not take part in a grouping set. In a query with multiple sets, however, how do you know whether a NULL marks a placeholder or comes from the underlying data? If it marks a placeholder for a grouping set, which set? The GROUPING_ID function can help you provide additional information to answer these questions.

Grouping Sets with NULLs Example

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY
GROUPING SETS((Category),(Cust),())
ORDER BY Category, Cust;
```

The partial results:

```
Category        Cust        TotalQty
```

```
--------------- ----------- --------
NULL            NULL        999
NULL            1           80

NULL            2           12
NULL            3           154
NULL            4           241
NULL            5           512
Beverages       NULL        513
Condiments      NULL        114
Confections     NULL        372
```

At a glance, it might be difficult to determine why a NULL appears in a column.

GROUPING_ID Example

```
SELECT
        GROUPING_ID(Category)AS grpCat,
        GROUPING_ID(Cust) AS grpCust,
        Category, Cust, SUM(Qty) AS TotalQty

FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust);
```

The partial results:

```
grpCat      grpCust     Category        Cust        TotalQty
----------- ----------- --------------- ----------- -----------
0           0           Beverages       1           36
0           0           Condiments      1           44
1           0           NULL            1           80
0           0           Beverages       2           5
0           0           Confections     2           7
1           0           NULL            2           12
```

```
0           0           Beverages       3           105
0           0           Condiments      3           4
0           0           Confections     3           45
1           0           NULL            3           154
...
1           1           NULL            NULL        999
0           1           Beverages       NULL        513
0           1           Condiments      NULL        114
0           1           Confections     NULL        372
```

As you can see, the GROUPING_ID function returns a 1 when a row is aggregated as part of the current grouping set and a 0 when it is not. In the first row, both grpCat and grpCust return 0; therefore, the row is part of the grouping set (Category, Cust).

GROUPING_ID can also take multiple columns as inputs and return a unique integer bitmap, comprised of combined bits, per grouping set. For more information, see Microsoft Docs:

GROUPING_ID (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402787

SQL Server also provides a GROUPING function, which accepts only one input to return a bit. For more information, see *GROUPING (Transact-SQL)* in Microsoft Docs:

GROUPING (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402788

Demonstration: Using Grouping Sets

In this demonstration, you will see how to use the CUBE and ROLLUP subclauses.

Demonstration Steps

Use the CUBE and ROLLUP Subclauses

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Select the code under the comment **Step 6**, and then click **Execute**.
8. Select the code under the comment **Step 7**, and then click **Execute**.
9. Select the code under the comment **Step 8**, and then click **Execute**.
10. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Select the best answer

You have the following query:
SELECT e.Department, e.Country, COUNT(EmployeeID) AS Staff
FROM HumanResources.Employees AS e
You want to find out how many staff are in each country and how many staff are in each department. You also want to find out how many staff are in Sales in the US, and so on, with all departments in all countries where the company operates. Choose the most succinct grouping technique for this query:

GROUPING SETS

CUBE

ROLLUP

You cannot return the required data with GROUPING. Instead, use multiple queries and a UNION element.

Check answer Show solution Reset

Lab: Pivoting and Grouping Sets

Scenario

As a business analyst for Adventure Works, you will be writing reports using corporate databases stored in SQL Server. You have been given a set of business requirements for data and you will write T-SQL queries to retrieve the specified data from the databases. The business requests are analytical in nature. To fulfill those requests, you will need to provide crosstab reports and multiple aggregates based on different granularities. Therefore, you will need to use pivoting techniques and grouping sets in your T-SQL code.

Objectives

After completing this lab, you will be able to:

- Write queries that use the PIVOT operator.
- Write queries that use the UNPIVOT operator.
- Write queries that use the GROUPING SETS, CUBE, and ROLLUP subclauses.

Lab Setup

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Queries That Use the PIVOT Operator

Scenario

The sales department would like to have a crosstab report, displaying the number of customers for each customer group and country. They would like to display each customer group as a new column. You will write different SELECT statements using the PIVOT operator to achieve the required result.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Write a SELECT Statement to Retrieve the Number of Customers for a Specific Customer Group
3. Specify the Grouping Element for the PIVOT Operator
4. Use a Common Table Expression (CTE) to Specify the Grouping Element for the PIVOT Operator
5. Write a SELECT Statement to Retrieve the Total Sales Amount for Each Customer and Product Category

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should be able to use the PIVOT operator in T-SQL statements.

Exercise 2: Writing Queries That Use the UNPIVOT Operator

Scenario

You will now create multiple rows by turning columns into rows.

The main tasks for this exercise are as follows:

1. Create and Query the Sales.PivotCustGroups View
2. Write a SELECT Statement to Retrieve a Row for Each Country and Customer Group
3. Remove the Created Views

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should know how to use the UNPIVOT operator in your T-SQL statements.

Exercise 3: Writing Queries That Use the GROUPING SETS, CUBE, and ROLLUP Subclauses

Scenario

You have to prepare SELECT statements to retrieve a unified result set with aggregated data for different combinations of columns. First, you have to retrieve the number of customers for all possible combinations of the country and city columns. Instead of using multiple T-SQL statements with a GROUP BY clause and then unifying them with the UNION ALL operator, you will use a more elegant solution using the GROUPING SETS subclause of the GROUP BY clause.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement That Uses the GROUPING SETS Subclause to Return the Number of Customers for Different Grouping Sets
2. Write a SELECT Statement That Uses the CUBE Subclause to Retrieve Grouping Sets Based on Yearly, Monthly, and Daily Sales Values

3. Write the Same SELECT Statement Using the ROLLUP Subclause

4. Analyze the Total Sales Value by Year and Month

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should have an understanding of how to use the GROUPING SETS, CUBE, and ROLLUP subclauses in T-SQL statements.

Module Review and Takeaways

In this module, you have learned how to:

- Write queries that pivot and unpivot result sets.
- Write queries that specify multiple groupings with grouping sets.

Review Question(s)

Check Your Knowledge

Discovery

Once a dataset has been pivoted with aggregation, can the original detail rows be restored with an unpivot operation?

Show solution Reset

Check Your Knowledge

Discovery

What are the possible sources of NULLs returned by a query using grouping sets to create aggregations?

Show solution Reset

Check Your Knowledge

Discovery

Which subclause infers a hierarchy of columns to create meaningful grouping sets?

Show solution Reset