Grouping and Aggregation

Exercise Handout

Learn. To Change.

Contents

Overview

In this set of exercises, you will first work with the Orders table from the Northwind database to find out general information about the orders on the system.

In the second exercise, you will start to investigate individual customers' ordering habits.

In the third exercise you will work with the [Order Details] table to start drilling down into exactly what products were ordered by customers and filtering the results of aggregated queries.

Objectives

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

- write a query that calculates single aggregates.
- write a query that calculates aggregates for groups of information.
- write a query that aggregates, groups and filters on calculated values.

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Launch SQL Server Management Studio.
3. Connect to the server.

Exercise 1: Basic aggregates

You want to learn more about the data stored in the Orders table, specifically those placed by Nancy Davolio.

In this exercise, you will use basic aggregates.

The main tasks for this exercise are as follows:

1. Write a query that selects a count of the orders.
2. Modify the query so that it also calculates the earliest and latest dates on which orders were placed.
3. Modify the query to only calculate the values for Nancy's orders (employee id 1).

Task 1: Write a query that selects a count of rows

1. Create a new query and save it with a name of "OrderAnalysis.sql".
2. Write a query that uses the Northwind database and displays a count of all the rows in the Orders table. Alias the count to call it "NumberOfOrders".
3. Execute the query and verify that the result set contains a single value of 830.

Task 2: Write a query that selects maximums and minimums

1. Modify your existing query.
2. Add an aggregate to the select list that calculates the minimum OrderDate value. Alias it as "EarliestOrder".
3. Add another aggregate that calculates the maximum OrderDate value. Alias it as "LatestOrder".
4. Execute the query and verify that the answers are 830, 4th July 1996 and 6th May 1998.

Task 3: Write a query that selects aggregates for only one employee

1. Modify your existing query.
2. Add a WHERE clause to your query so that it only includes rows with an EmployeeID equal to 1.
3. Execute the query and verify that the answers are now 123, 17th July 1996 and 6th May 1998.

Exercise 2: Grouping aggregates

You have been asked to write a query on the number of orders placed by each customer. The query is to be sorted by the number of orders placed, from highest to lowest.

In this exercise, you will use the GROUP BY clause and an ORDER BY.

The main tasks for this exercise are as follows:

1. Write a query that counts the number of OrderIDs in the Orders table.
2. Modify the query so that it includes the CustomerID and groups the results on that column.
3. Modify the query to sort in reverse order on the number of orders placed.

Task 1: Create a query that counts orders

1. Create a new query and save it with a name of “CustomerOrders.sql”.
2. Write a query that uses the Northwind database and displays a count of all the OrderIDs in the Orders table. Alias the aggregate as “NumberOfOrders”.
3. Execute the query and verify that it returns a value of 830.

Task 2: Write a query that groups orders based on the customer’s ID

1. Modify the existing query.
2. Add the CustomerID column to the query’s select list.
3. NOTE: At this point, the query won’t work.
4. Add a GROUP BY clause to the query that groups the results based on the CustomerID column.
5. Execute the query and verify that it returns 89 rows, the top one being ALFKI with a NumberOfOrders of 6.

Task 3: Write a query that sorts order counts in descending order

1. Modify the existing query.
2. Add an ORDER BY clause to the query that sorts on the NumberOfOrders column in descending order.
3. Execute the query and verify that it returns 89 rows, the top one now being SAVEA with a count of 31.

Exercise 3: Aggregating calculated values and filtering aggregates

Northwind Traders are trying to see exactly how much customers are spending on products.

In this exercise, you will use aggregate functions on calculated columns.

The main tasks for this exercise are as follows:

1. Write a query that sums the Quantity column of the [Order Details] table, grouped by Product IDs.
2. Modify the query so that it sums the Quantity times the UnitPrice and sorts the results in descending order of those values.
3. Modify the query to filter the results to only include those products with a total value of less than or equal to 5000.

Task 1: Write a query that sums quantities of products sold

1. Create a new query and save it with a name of "ProductSales.sql".
2. Write a query that uses the Northwind database and selects the ProductID and the sum of the Quantity columns from the [Order Details] table. Alias the aggregate column as "TotalSold"
3. Group the results on the ProductID column.
4. Execute the query and verify that it returns 77 rows, the first row being product ID 23, with a TotalSold of 580.

Task 2: Write a query that sums a calculation

1. Modify the existing query.
2. Modify the SUM aggregate so that it adds up the value of the Quantity column multiplied by the UnitPrice column for each product. Change the alias name to "TotalValue".
3. Sort the results on the TotalValue column, in descending order.
4. Execute the query and verify that the top-selling product is productid 38, with a total sales value of 149984.20.

Task 3: Write a query that filters aggregate values

1. Modify the existing query.
2. Add a HAVING clause to the query to only return the rows with a TotalValue of less than or equal to 5000.
3. REMEMBER: just like with a WHERE clause, the actual column named TotalValue doesn’t exist yet in your HAVING, so you’ll need to re-use the calculation.
4. Execute the query and verify that you now see only 16 rows, the first of which is product 23 with a value of 4840.20.

Learn. To Change.

QA.com