Lab Answer Key: Module 9: Grouping and Aggregating Data

Lab: Grouping and Aggregating Data

Exercise 1: Writing Queries That Use the GROUP BY Clause

Task 1: Prepare the Lab Environment

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. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

Task 2: Write a SELECT Statement to Retrieve Different Groups of Customers

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
2. On the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab09\Starter\Project** folder, and then double-click **Project.ssmssln**.
4. In Solution Explorer, expand **Queries**, and then double-click **51 - Lab Exercise 1.sql**.
5. In the query pane, highlight the statement **USE TSQL;**, and click **Execute**.
6. In the query pane, type the following query after the **Task 2** description:

```
SELECT
o.custid, c.contactname
FROM Sales.Orders AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.empid = 5
GROUP BY o.custid, c.contactname;
```

7. Highlight the written query, and click **Execute**.

Task 3: Add an Additional Column From the Sales.Customers Table

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement so that it adds an additional column. Your query should look like this:

```
SELECT
o.custid, c.contactname, c.city
FROM Sales.Orders AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.empid = 5
GROUP BY o.custid, c.contactname;
```

4. Highlight the written query, and click **Execute**.
5. Observe the error message:

```
Column 'Sales.Customers.city' is invalid in the select list because it is not contained
in either an aggregate function or the GROUP BY clause.
```

Why did the query fail?

In a grouped query, there will be an error if you refer to an attribute that is not in the GROUP BY list (such as the city column) or not an input to an aggregate function in any clause that is processed after the GROUP BY clause.

6. Modify the SQL statement to include the city column in the GROUP BY clause. Your query should look like this:

```
SELECT
o.custid, c.contactname, c.city
FROM Sales.Orders AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.empid = 5
GROUP BY o.custid, c.contactname, c.city;
```

7. Highlight the written query, and click **Execute**.

Task 4: Write a SELECT Statement to Retrieve the Customers with Orders for Each Year

1. In the query pane, type the following query after the **Task 4** description:

```
SELECT
custid, YEAR(orderdate) AS orderyear
FROM Sales.Orders
WHERE empid = 5
GROUP BY custid, YEAR(orderdate)
ORDER BY custid, orderyear;
```

2. Highlight the written query, and click **Execute**.

Task 5: Write a SELECT Statement to Retrieve Groups of Product Categories Sold in a Specific Year

1. In the query pane, type the following query after the **Task 5** description:

```
SELECT
c.categoryid, c.categoryname
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
INNER JOIN Production.Products AS p ON p.productid = d.productid
INNER JOIN Production.Categories AS c ON c.categoryid = p.categoryid
WHERE orderdate >= '20080101' AND orderdate < '20090101'
GROUP BY c.categoryid, c.categoryname;
```

2. Highlight the written query, and click **Execute**.

Note: Important note regarding the use of the DISTINCT clause:
In all the tasks in Exercise 1, you could use the DISTINCT clause in the SELECT clause as an alternative to using a grouped query. This is possible because aggregate functions are not being requested.

Result: After this exercise, you should be able to use the GROUP BY clause in the T-SQL statement.

Exercise 2: Writing Queries That Use Aggregate Functions

Task 1: Write a SELECT statement to Retrieve the Total Sales Amount Per Order

1. In Solution Explorer, double-click **61 - Lab Exercise 2.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, type the following query after the **Task 1** description:

```
SELECT
o.orderid, o.orderdate, SUM(d.qty * d.unitprice) AS salesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.orderid, o.orderdate
ORDER BY salesamount DESC;
```

4. Highlight the written query, and click **Execute**.

Task 2: Add Additional Columns

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 2** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement so that it adds extra columns. Your query should look like this:

```
SELECT
o.orderid, o.orderdate,
SUM(d.qty * d.unitprice) AS salesamount,
COUNT(*) AS noofoderlines,
AVG(d.qty * d.unitprice) AS avgsalesamountperorderline
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.orderid, o.orderdate
ORDER BY salesamount DESC;
```

4. Highlight the written query, and click **Execute**.

Task 3: Write a SELECT Statement to Retrieve the Sales Amount Value Per Month

1. In the query pane, type the following query after the **Task 3** description:

```
SELECT
YEAR(orderdate) * 100 + MONTH(orderdate) AS yearmonthno,
SUM(d.qty * d.unitprice) AS saleamountpermonth
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY YEAR(orderdate), MONTH(orderdate)
ORDER BY yearmonthno;
```

2. Highlight the written query, and click **Execute**.

Task 4: Write a SELECT Statement to List All Customers with the Total Sales Amount and Number of Order Lines Added

1. In the query pane, type the following query after the **Task 4** description:

```
SELECT
c.custid, c.contactname,
SUM(d.qty * d.unitprice) AS totalsalesamount,
MAX(d.qty * d.unitprice) AS maxsalesamountperorderline,
COUNT(*) AS numberofrows,
COUNT(o.orderid) AS numberoforderlines
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON o.custid = c.custid
LEFT OUTER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY c.custid, c.contactname
ORDER BY totalsalesamount;
```

2. Highlight the written query, and click **Execute**.
3. Observe the result. Notice that the values in the **numberofrows** and **numberoforderlines** columns are different. Why? All aggregate functions ignore NULLs except COUNT(*), which is why you received the value 1 for the **numberofrows** column. When you used the **orderid** column in the COUNT function, you received the value 0 because the **orderid** is NULL for customers without an order.

Exercise 3: Writing Queries That Use Distinct Aggregate Functions

Task 1: Modify a SELECT Statement to Retrieve the Number of Customers

1. In Solution Explorer, double-click **71 - Lab Exercise 3.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. Highlight the provided T-SQL statement after the **Task 1** description, and click **Execute**.
4. Observe the result. Notice that the number of orders is the same as the number of customers. Why? You are using the aggregate COUNT function on the **orderid** and **custid** columns and, because every order has a customer, the COUNT function returns the same value. It does not matter if there are multiple orders for the same customer, because you are not using a DISTINCT clause inside the aggregate function. To get the correct number of distinct customers, you can modify the provided T-SQL statement to include a DISTINCT clause.

5. Modify the provided T-SQL statement to include a DISTINCT clause. The query should look like this:

```
SELECT
YEAR(orderdate) AS orderyear,
COUNT(orderid) AS nooforders,
COUNT(DISTINCT custid) AS noofcustomers
FROM Sales.Orders
GROUP BY YEAR(orderdate);
```

6. Highlight the written query and click **Execute**.

Task 2: Write a SELECT Statement to Analyze Segments of Customers

1. In the query pane, type the following query after the **Task 2** description:

```
SELECT
SUBSTRING(c.contactname,1,1) AS firstletter,
COUNT(DISTINCT c.custid) AS noofcustomers,
COUNT(o.orderid) AS nooforders
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON o.custid = c.custid
GROUP BY SUBSTRING(c.contactname,1,1)
ORDER BY firstletter;
```

2. Highlight the written query, and click **Execute**.

Task 3: Write a SELECT Statement to Retrieve Additional Sales Statistics

1. In the query pane, type the following query after the **Task 3** description:

```
SELECT
c.categoryid, c.categoryname,
SUM(d.qty * d.unitprice) AS totalsalesamount, COUNT(DISTINCT o.orderid) AS nooforders,
SUM(d.qty * d.unitprice) / COUNT(DISTINCT o.orderid) AS avgsalesamountperorder
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
INNER JOIN Production.Products AS p ON p.productid = d.productid
INNER JOIN Production.Categories AS c ON c.categoryid = p.categoryid
WHERE orderdate >= '20080101' AND orderdate < '20090101'
GROUP BY c.categoryid, c.categoryname;
```

2. Highlight the written query, and click **Execute**.

Result: After this exercise, you should have an understanding of how to apply a DISTINCT aggregate function.

Exercise 4: Writing Queries That Filter Groups with the HAVING Clause

Task 1: Write a SELECT Statement to Retrieve the Top 10 Customers

1. In Solution Explorer, double-click **81 - Lab Exercise 4.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, type the following query after the **Task 1** description:

```
SELECT TOP (10)
o.custid,
SUM(d.qty * d.unitprice) AS totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING SUM(d.qty * d.unitprice) > 10000
ORDER BY totalsalesamount DESC;
```

4. Highlight the written query, and click **Execute**.

Task 2: Write a SELECT Statement to Retrieve Specific Orders

1. In the query pane, type the following query after the **Task 2** description:

```
SELECT
o.orderid,
o.empid,
SUM(d.qty * d.unitprice) as totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080101' AND o.orderdate < '20090101'
GROUP BY o.orderid, o.empid;
```

2. Highlight the written query, and click **Execute**.

Task 3: Apply Additional Filtering

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement to apply additional filtering. Your query should look like this:

```
SELECT
o.orderid,
o.empid,
SUM(d.qty * d.unitprice) as totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080101' AND o.orderdate < '20090101'
GROUP BY o.orderid, o.empid
HAVING SUM(d.qty * d.unitprice) >= 10000;
```

4. Highlight the written query, and click **Execute**.
5. Modify the T-SQL statement to include an additional filter to retrieve only orders handled by the employee whose ID is 3. Your query should look like this:

```
SELECT
o.orderid,
o.empid,
SUM(d.qty * d.unitprice) as totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE
o.orderdate >= '20080101' AND o.orderdate <= '20090101'
AND o.empid = 3
GROUP BY o.orderid, o.empid
HAVING SUM(d.qty * d.unitprice) >= 10000;
```

In this query, the predicate logic is applied in the WHERE clause. You could also write the predicate logic inside the HAVING clause. Which do you think is better?

Unlike with **orderdate** filtering, with **empid** filtering, the result is going to be correct either way because you are filtering by an element that appears in the GROUP BY list. Conceptually, it seems more intuitive to filter as early as possible. This query then applies the filtering in the WHERE clause because it will be logically applied before the GROUP BY clause. Do not forget, though, that the actual processing in the SQL Server engine could be different.

6. Highlight the written query, and click **Execute**.

Task 4: Retrieve the Customers with More Than 25 Orders

1. In the query pane, type the following query after the **Task 4** description:

```
SELECT
o.custid,
MAX(orderdate) AS lastorderdate,
SUM(d.qty * d.unitprice) AS totalsalesamount
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT o.orderid) > 25;
```

2. Highlight the written query, and click **Execute**.
3. Close SQL Server Management Studio without saving any files.

Result: After this exercise, you should have an understanding of how to use the HAVING clause.