Lab Answer Key: Module 11: Using Table Expressions

Lab: Using Table Expressions

Exercise 1: Writing Queries That Use Views

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\Lab11\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 All Products for a Specific Category

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine 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\Lab11\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 then click **Execute**.
6. In the query pane, type the following query after the **Task 1** description:

```
SELECT
productid, productname, supplierid, unitprice, discontinued
FROM Production.Products
WHERE categoryid = 1;
```

7. Highlight the written query, and click **Execute**.
8. Modify the query to include the provided CREATE VIEW statement. The query should look like this:

```
CREATE VIEW Production.ProductsBeverages AS
SELECT
productid, productname, supplierid, unitprice, discontinued
```

```
FROM Production.Products
WHERE categoryid = 1;
```

9. Highlight the modified query, and click **Execute**.

Task 3: Write a SELECT Statement Against the Created View

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

```
SELECT
productid, productname
FROM Production.ProductsBeverages
WHERE supplierid = 1;
```

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

Task 4: Try to Use an ORDER BY Clause in the Created View

1. Highlight the provided T-SQL statement under the **Task 3** description, and then click **Execute**.
2. Observe the error message:

```
The ORDER BY clause is invalid in views, inline functions, derived tables, subqueries,
and common table expressions, unless TOP, OFFSET or FOR XML is also specified.
```

Why did the query fail? It failed because the view is supposed to represent a relation, and a relation has no order. You can only use the ORDER BY clause in the view if you specify the TOP, OFFSET, or FOR XML option. The reason you can use ORDER BY in special cases is that it serves a meaning other than presentation ordering to these special cases.

3. Modify the previous T-SQL statement by including the TOP (100) PERCENT option. The query should look like this:

```
ALTER VIEW Production.ProductsBeverages AS
SELECT TOP(100) PERCENT
productid, productname, supplierid, unitprice, discontinued
FROM Production.Products
WHERE categoryid = 1
ORDER BY productname;
```

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

5. Observe the result. If you now write a query against the Production.ProductsBeverages view, is it guaranteed that the retrieved rows will be sorted by productname? If you do not specify the ORDER BY clause in the T-SQL statement against the view, there is no guarantee that the retrieved rows will be sorted. It is important to remember that any order of the rows in the output is considered valid, and no specific order is guaranteed. Therefore, when querying a table expression, you should not assume any order unless you specify an ORDER BY clause in the outer query.

Task 5: Add a Calculated Column to the View

1. Highlight the provided T-SQL statement under the **Task 4** description, and then click **Execute**.

2. Observe the error message:

```
Create View or Function failed because no column name was specified for column 6.
```

Why did the query fail? It failed because each column must have a unique name. In the provided T-SQL statement, the last column does not have a name.

3. Modify the T-SQL statement to include the column name pricetype. The query should look like this:

```
ALTER VIEW Production.ProductsBeverages AS
SELECT
        productid, productname, supplierid, unitprice, discontinued,
        CASE WHEN unitprice > 100. THEN N'high' ELSE N'normal' END AS pricetype
FROM Production.Products
WHERE categoryid = 1;
```

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

Task 6: Remove the Production.ProductsBeverages View

- Highlight the provided T-SQL statement under the **Task 5** description and click **Execute**.

Result: After this exercise, you should know how to use a view in T-SQL statements.

Exercise 2: Writing Queries That Use Derived Tables

Task 1: Write a SELECT Statement Against a Derived Table

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
p.productid, p.productname
FROM
(
SELECT
productid, productname, supplierid, unitprice, discontinued,
CASE WHEN unitprice > 100. THEN N'high' ELSE N'normal' END AS pricetype
FROM Production.Products
WHERE categoryid = 1
) AS p
WHERE p.pricetype = N'high';
```

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

Task 2: Write a SELECT Statement to Calculate the Total and Average Sales Amount

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

```
SELECT
c.custid,
SUM(c.totalsalesamountperorder) AS totalsalesamount,
AVG(c.totalsalesamountperorder) AS avgsalesamount
FROM
(
SELECT
o.custid, o.orderid, SUM(d.unitprice * d.qty) AS totalsalesamountperorder
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails d ON d.orderid = o.orderid
GROUP BY o.custid, o.orderid
) AS c
GROUP BY c.custid;
```

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

Task 3: Write a SELECT Statement to Retrieve the Sales Growth Percentage

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

```
SELECT
cy.orderyear,
cy.totalsalesamount AS curtotalsales,
py.totalsalesamount AS prevtotalsales,
(cy.totalsalesamount - py.totalsalesamount) / py.totalsalesamount * 100. AS
percentgrowth
FROM
(
SELECT
YEAR(orderdate) AS orderyear, SUM(val) AS totalsalesamount
FROM Sales.OrderValues
GROUP BY YEAR(orderdate)
) AS cy
LEFT OUTER JOIN
(
SELECT
YEAR(orderdate) AS orderyear, SUM(val) AS totalsalesamount
FROM Sales.OrderValues
GROUP BY YEAR(orderdate)
) AS py ON cy.orderyear = py.orderyear + 1
ORDER BY cy.orderyear;
```

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

Result: After this exercise, you should be able to use derived tables in T-SQL statements.

Exercise 3: Writing Queries That Use CTEs

Task 1: Write a SELECT Statement That Uses a CTE

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

```
WITH ProductsBeverages AS
(
SELECT
```

```
productid, productname, supplierid, unitprice, discontinued,
CASE WHEN unitprice > 100. THEN N'high' ELSE N'normal' END AS pricetype
FROM Production.Products
WHERE categoryid = 1
)
SELECT
productid, productname
FROM ProductsBeverages
WHERE pricetype = N'high';
```


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

Task 2: Write a SELECT Statement to Retrieve the Total Sales Amount for Each Customer

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

```
WITH c2008 (custid, salesamt2008) AS
(
SELECT
custid, SUM(val)
FROM Sales.OrderValues
WHERE YEAR(orderdate) = 2008
GROUP BY custid
)
SELECT
c.custid, c.contactname, c2008.salesamt2008
FROM Sales.Customers AS c
LEFT OUTER JOIN c2008 ON c.custid = c2008.custid;
```


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

Task 3: Write a SELECT Statement to Compare the Total Sales Amount for Each Customer Over the Previous Year

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

```
WITH c2008 (custid, salesamt2008) AS
(
SELECT
custid, SUM(val)
```

```
FROM Sales.OrderValues
WHERE YEAR(orderdate) = 2008
GROUP BY custid
),
c2007 (custid, salesamt2007) AS
(
SELECT
custid, SUM(val)
FROM Sales.OrderValues
WHERE YEAR(orderdate) = 2007
GROUP BY custid
)
SELECT
c.custid, c.contactname,
c2008.salesamt2008,
c2007.salesamt2007,
COALESCE((c2008.salesamt2008 - c2007.salesamt2007) / c2007.salesamt2007 * 100., 0) AS
percentgrowth
FROM Sales.Customers AS c
LEFT OUTER JOIN c2008 ON c.custid = c2008.custid
LEFT OUTER JOIN c2007 ON c.custid = c2007.custid
ORDER BY percentgrowth DESC;
```

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

Result: After this exercise, you should have an understanding of how to use a CTE in a T-SQL statement.

Exercise 4: Writing Queries That Use Inline TVFs

Task 1: Write a SELECT Statement to Retrieve the Total Sales Amount for Each Customer

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
custid, SUM(val) AS totalsalesamount
FROM Sales.OrderValues
WHERE YEAR(orderdate) = 2007
GROUP BY custid;
```

4.

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

5. Create an inline TVF using the provided code. Add the previous query, putting it after the function's RETURN clause. In the query, replace the order date of 2007 with the function's input parameter @orderyear. The resulting T-SQL statement should look like this:

```
CREATE FUNCTION dbo.fnGetSalesByCustomer
(@orderyear AS INT) RETURNS TABLE
AS
RETURN
SELECT
custid, SUM(val) AS totalsalesamount
FROM Sales.OrderValues
WHERE YEAR(orderdate) = @orderyear
GROUP BY custid;
```

This T-SQL statement will create an inline TVF named dbo.fnGetSalesByCustomer.

6. Highlight the written T-SQL statement, and click **Execute**.

Task 2: Write a SELECT Statement Against the Inline TVF

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

```
SELECT
custid, totalsalesamount
FROM dbo.fnGetSalesByCustomer(2007);
```

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

Task 3: Write a SELECT Statement to Retrieve the Top Three Products Based on the Total Sales Value for a Specific Customer

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

```
SELECT TOP(3)
d.productid,
MAX(p.productname) AS productname,
SUM(d.qty * d.unitprice) AS totalsalesamount
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
WHERE custid = 1
GROUP BY d.productid
ORDER BY totalsalesamount DESC;
```

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

3. Create an inline TVF using the provided code. Add the previous query, putting it after the function's RETURN clause. In the query, replace the constant custid value of 1 with the function's input parameter @custid. The resulting T-SQL statement should look like this:

```
CREATE FUNCTION dbo.fnGetTop3ProductsForCustomer
(@custid AS INT) RETURNS TABLE
AS
RETURN
SELECT TOP(3)
d.productid,
MAX(p.productname) AS productname,
SUM(d.qty * d.unitprice) AS totalsalesamount
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
WHERE custid = @custid
GROUP BY d.productid
ORDER BY totalsalesamount DESC;
```

4. To test the inline TVF, add the following query after the CREATE FUNCTION and GO statement:

```
SELECT
p.productid,
p.productname,
p.totalsalesamount
FROM dbo.fnGetTop3ProductsForCustomer(1) AS p;
```

5. Highlight the CREATE FUNCTION statement and the written query, and click **Execute**.

Task 4: Using Inline TVFs, Write a SELECT Statement to Compare the Total Sales Amount for Each Customer Over the Previous Year

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

```
SELECT
c.custid, c.contactname,
c2008.totalsalesamount AS salesamt2008,
c2007.totalsalesamount AS salesamt2007,
COALESCE((c2008.totalsalesamount - c2007.totalsalesamount) / c2007.totalsalesamount *
100., 0) AS percentgrowth
FROM Sales.Customers AS c
LEFT OUTER JOIN dbo.fnGetSalesByCustomer(2007) AS c2007 ON c.custid = c2007.custid
LEFT OUTER JOIN dbo.fnGetSalesByCustomer(2008) AS c2008 ON c.custid = c2008.custid;
```

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

Task 5: Remove the Created Inline TVFs

- Highlight the provided T-SQL statement under the **Task 5** description and click **Execute**.

Result: After this exercise, you should know how to use inline TVFs in T-SQL statements.