Lab Answer Key: Module 1: Introduction to Microsoft SQL Server 2014

Lab: Working with SQL Server 2014 Tools

Exercise 1: Working with SQL Server Management Studio
Task 1: Open Microsoft SQL Server Management Studio
1.
Start SQL Server Management Studio.
2.
In the Connect to Server window, click Cancel.
3.
Close the Object Explorer   clicking the close icon.
4.
Close the Solution Explorer window by clicking the close icon.
5.
Open the Object Explorer window byselecting Object Explorer on theView menu (or press F8 on the keyboard).
6.
Open the Solution Explorer window by selecting Solution Explorer on the View menu
(or press Ctrl+Alt+L on the keyboard).

Task 2: Configure the Editor Settings
1.
On the Tools menu, select Options toopen the Options window in SQL Server Management Studio.
2.
Expand the Environment option andselect Fonts and Colors. In the Showsettings for box, select Text Edit orand set the font size in the Size box to14.
3.
In the left pane, expand the TextEditor option, expand the Transact-
SQL option, and select IntelliSense.Under Transact-
SQL IntelliSenseSettings, clear the EnableIntelliSense check box.
4.
In the left pane, select the Tabs optionunder Text Editor and Transact-
SQL. In the Tab frame, change theTab size property to 6.
5.
In the left pane, expand QueryResults, expand SQL Server, andselect Results to Grid. Enable theoption In clude column headers whencopying or saving the results byselecting the check box.
6.
Accept the changes by clicking the OK button.
Result: After this exercise, you shouldhave opened SSMS and configured editorsettings.

Exercise 2: Creating and Organizing T-SQL scripts

Task 1: Create a Project
1.
On the File menu, select New and click Project.
2.
In the New Project window, typeMyFirstProject in the Name text boxand D:\Labfiles\Lab01\Starter in th eLocation text box. Click the OKbutton to create the new project.
3.
In the Solution Explorer window, right-
click the Queries folder underMyFirstProject and select NewQuery.
4.
In the Connect to Database Engine Dialog box, click Cancel.
5.
Right-
click the query fileSQLQuery1.sql under the Queriesfolder, choose Rename, and typeMyFirstQueryFile.s ql as the newname for the file.
6.
On the File menu, select Save All.

Task 2: Add an Additional Query File
1.
In the Solution Explorer window, right-
click the Queries folder underMyFirstProject and select NewQuery.
2.
In the Connect to Database Engine Dialog box, click Cancel.
3.
In the Queries folder, right-
click thequery file SQLQuery1.sql, chooseRename, and typeMySecondQueryFile.sql as the newname for the file.
4.
Click File Explorer on the taskbar.
5.
In Windows Explorer, navigate to thefolder D:\Labfiles\Lab01\Starter\MyFirstProject\MyFirstProject and observe the created files.
6.
In the Solution Explorer window inSQL Server Management Studio, right-
click the query fileMySecondQueryFile.sql and selectRemove. When the confirmationdialog appears, clic k the Removebutton.
7.
In Windows Explorer, press F5 torefresh the Windows Explorerwindow and notice that the fileMySecond QueryFile.sql is still there.

8.
In the Solution Explorer window inSSMS, right-
click the query fileMyFirstQueryFile.sql and selectRemove. When the confirmationdialog appears, click t
he Deletebutton.
9.
In Windows Explorer, press F5 torefresh the Windows Explorerwindow. Notice that the fileMyFirstQuery
File.sql was deletedfrom the file system.
Task 3: Reopen the Created Project
1.
On the File menu, select Save All.
2.
On the File menu, select Exit to close the project and SSMS.
3.
Click Start, click the Down arrow, expand All Programs, scroll to theright to the Microsoft SQL Server 201
4 group, and click SQL Server2014 Management Studio.
4.
In the Connect to Server window, click Cancel.
5.
On the File menu, click Open and click Project / Solution. In theOpen Project window, select the projectD
:\Labfiles\Lab01\Starter\MyFirstProject\MyFirstProject.ssmssln.
6.
Click Open.
7.
In Windows Explorer, navigate to the folderD:\Labfiles\Lab01\Starter\MyFirstProject\MyFirstProject. Dr
ag the fileMySecondQueryFile.sql to the Queries folder in the SolutionExplorer window in SSMS.
8.
On the File menu, select Save All.
Result: After this exercise, you shouldhave a basic understanding of how tocreate a project in SSMS and
add queryfiles to it.

Exercise 3: Using Books Online
Task 1: Launch Books Online
1.
On the virtual machine, press theWindows key, and type manage helpsettings and click Manage HelpSet
tings.
2.
In the Help Library Managerwindow, select Choose online or localhelp.
3.
Under Set your preferred helpexperience, click I want to use onlinehelp and click the OK button toconfir
m.
4.
Click the Exit button to leave theHelp Library Manager window.
Task 2: Use Books Online

On the virtual machine, press the Windowskey, and type sql server documentationand click SQL Server Documentation.
If Online Help Consent dialog box appears, click Yes to continue.
In the left pane of the Books Online forSQL Server 2014 website, expand BooksOnline for SQL Server 201 4 and expandGetting Started (SQL Server 2014).
Click Documentation for SQL Server2014 Tools and Add-in Components.
Browse the help article.
Shut down Internet Explorer.
Result: After this exercise, you shouldhave a basic understanding of how to findinformation in Books Onl ine.
--end mod 1

Lab: Introduction to T-SQL Querying

Exercise 1: Executing Basic SELECT Statements
Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab02\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: Execute the T-SQL Script
1.
On the virtual machine, on the Taskbar, click SQL Server Management Studio.

2.
In the Connect to Server window, in the Server name text box, type MIA-SQL.

3.
Click the Options button. Under Connection Properties, select <Browse server> in the Connect to database list. Choose Yes when prompted for the connection to the database. Under User Databases, select the TSQL database. Click OK.

4.
Click the Login tab, select Windows Authentication in the Authentication list, and click Connect.

5.
On the File menu, click Open and click Project / Solution. In the OpenProject window, select project D:\Labfiles\Lab02\Starter\Project\Project.ssmssln.

6.
In Solution Explorer, double-click 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

7.
When the query window opens, click the Execute button on the toolbar (or press F5 on the keyboard). You will notice that the TSQL database is selected in the Available Databases box. The Available Databases box displays the current database context under which the T-SQL script will run. This information is also visible on the status bar.

Task 3: Execute a Part of the T-SQL Script
1.
Highlight the following statement under the task 2 description:

```
SELECT
firstname, lastname, city, country
FROM HR.Employees;
```
To highlight it, you can move the pointer over the statement while pressing the left mouse button or use the arrow keys to move the pointer while pressing the Shift key.

2.
Click Execute (or press F5). It is very important to understand that you can highlight a specific part of the code inside the T-SQL script and execute that part alone. If you click Execute without selecting any part of the code, the whole T-SQL script will be executed. If you highlight a specific part of the code by mistake, the SQL Server will attempt to run only that part.

Result: After this exercise, you should know how to open the T-SQL script and execute the whole script or just a specific statement inside it.

Exercise 2: Executing Queries That Filter Data Using Predicates
Task 1: Execute the T-SQL Script
1.
Close SQL Server Management Studio.

2.
If prompted to save the files, click No.

3.
On the virtual machine, on the Taskbar, click SQL Server 2014 Management Studio.

4.
In the Connect to Server window, type MIA-SQL in the Server name text box.

5.
Click the Options button. Under Connection Properties, select <Browse server> in the Connect to database list. Choose Yes when prompted for the connection to the database. Under System Databases, select the master database.

6.
Click OK and click Connect.

7.
On the File menu, click Open and click Project / Solution. In the OpenProject window, select project D:\Labfiles\Lab02\Starter\Project\Project.ssmssln.

8.
In Solution Explorer, double-click 61 - Lab Exercise 2.sql.

9.
When the query window opens, click Execute.

10.
Notice that you get the error message:

Invalid object name 'HR.Employees'.:
Why do you think this happened? This error is very common when you are beginning to learn T-SQL.

The message tells you that SQL Server could not find the object HR.Employees. This is because the current database context is set to the master database (look at the Available Databases box where the current database is displayed), but the IT department supplied T-SQL scripts to be run against the TSQL database. So, you need to change the database context from master to TSQL. You will learn how to change the database context in the next task.

Task 2: Apply Needed Changes and Execute the T-SQL Script
1.
In the Available Databases box, select TSQL to change the database context.

2.
Click Execute.

3.
Notice that the result from the SELECT statement returns fewer rows than the one in exercise 1. That is because it has a predicate in the WHERE clause to filter out all rows that do not have the value USA in the column country. Only rows for which the logical expression evaluates to TRUE are returned by the WHERE phase to the subsequent logical query processing phase.

Task 3: Uncomment the USE Statement

1.
In the script 61 - Lab Exercise 2.sql, find the line:

--USE TSQL;
2.
Delete the first two characters, so that the line looks like this:

USE TSQL;
By deleting these two characters, you have removed the comment mark. Now the line will not be ignored by SQL Server.

3.
On the File menu, click Save 61 - Lab Exercise 2.sql.

4.
On the File menu, click Close. This will close the T-SQL script.

5.
In Solution Explorer, double-click 61 - Lab Exercise 2.sql.

6.
Click Execute.

7.
Observe the results. Why did the script execute with no errors? The script now includes the uncommented USE TSQL; statement. When you execute the whole T-SQL script, the USE statement applies the database context to the TSQL database. The next statement in the T-SQL script then executes against the TSQL database.

Result: After this exercise, you should have a basic understanding of database context and how to change it.

Exercise 3: Executing Queries That Sort Data Using ORDER BY
Task 1: Execute the T-SQL Script
1.
In Solution Explorer, double-click 71 - Lab Exercise 3.sql.

2.
Click Execute.

3.
Notice that the result window is empty. All the statements inside the T-SQL script are commented out, so SQL Server ignores all the statements inside the T-SQL script.

Task 2: Uncomment the Needed T-SQL Statements and Execute Them
1.
Locate the line:

--USE TSQL;
2.
Delete the two characters before the USE statement. The line should now look like this:

USE TSQL;
3.
Locate the block comment start element /* after the task 1 description and delete it.

4.
Locate the block comment end element */ and delete it. Any text residing within a block starting with /* and ending with */ is treated as a block comment and is ignored by SQL Server.

5.
Highlight the statement:

USE TSQL;
GO
Click Execute. The database context is now changed to the TSQL database.

6.
Highlight the statement:

SELECT
firstname, lastname, city, country
FROM HR.Employees
WHERE country = 'USA'
ORDER BY lastname;

7.
Click Execute.

8.
Observe the result and notice that the rows are sorted by the lastname column in ascending order.

Result: After this exercise, you should have an understanding of how comments can be specified inside T-SQL scripts.--end Mod 2
Lab Answer Key: Module 3: Writing SELECT Queries

Lab: Writing Basic SELECT Statements
Exercise 1: Writing Simple SELECT Statements

Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to

20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab03\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: View all the Tables in the TSQL Database in Object Explorer
1.
On the virtual machine, on the Taskbar, click SQL Server 2014 Management Studio.

2.
In the Connect to Server window, in the Server name text box, type MIA-SQL.

3.
Click the Options button. Under Connection Properties, select <Browse server> in the Connect to database list. Choose Yes when prompted for the connection to the database. Under User Databases, select the TSQL database. Click OK.

4.
Click the Login tab, select Windows Authentication in the Authentication list, and click Connect.

5.
In Object Explorer, expand the server MIA-SQL, expand Databases, expand the database TSQL, and expand Tables.

6.
Under Tables, notice that there are four table objects in the Sales schema:

Sales.Customers

Sales.OrderDetails

Sales.Orders

Sales.Shippers

Task 3: Write a Simple SELECT Statement

1.
On the File menu, click Open and click Project/Solution.

2.
In the Open Project window, open the project D:\Labfiles\Lab03\Starter\Project\Project.ssmssln.

3.
In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

4.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

5.
In the query pane, type the following query after the task 2 description:

```
SELECT*
FROM Sales.Customers;
```

6.
Highlight the query you typed in step 5 and click Execute.

7.
In the query pane, type the following code after the first query:

```
SELECT*
FROM
```

8.
In Object Explorer, select the Sales.Customers table under MIA-SQL, TSQL, Tables. Using the mouse, drag the selected table into the query pane, after the FROM clause. Then add a semicolon to the end of the SELECT statement.

Your finished query should look like this:

```
SELECT*
FROM [Sales].[Customers];
```

9.
Highlight the written query and click Execute.

Task 4: Write a SELECT Statement that Includes Specific Columns
1.
In Object Explorer, expand the Sales.Customers table under MIA-SQL, TSQL, and Tables.

2.
Expand Columns and observe all the columns in the Sales.Customers table.

3.
In the query pane, type the following query after the task 3 description:

```
SELECT
contactname, address, postalcode, city, country
FROM Sales.Customers;
```

4.
Highlight the written query and click Execute.

5.
Observe the result. How many rows are affected by the last query? There are multiple ways to answer this question using SQL Server Management Studio. One way is to select the previous query and click Execute. The total number of rows affected by the executed query is written in the Results pane under the Messages tab:

```
(91 row(s) affected)
```

Another way is to look at the status bar displayed below the Results pane. On the left side of the status bar, there is a message stating: "Query executed successfully." On the right side, the total number of rows affected by the current query is displayed (91 rows).

Result: After this exercise, you should know how to create simple SELECT statements to analyze existing tables.

Exercise 2: Eliminating Duplicates Using DISTINCT
Task 1: Write a SELECT Statement that Includes a Specific Column
1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECTcountry
FROM Sales.Customers;
```

4.
Highlight the written query and click Execute.

5.

Observe that you have multiple rows with the same values. This occurs because the Sales.Customers table has multiple rows with the same value for the country column.

Task 2: Write a SELECT Statement that Uses the DISTINCT Clause
1.
Highlight the previous query. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 2 description.

3.
Modify the query by typing DISTINCT after the SELECT clause. Your query should look like this:

```
SELECT DISTINCT
country
FROM Sales.Customers;
```

4.
Highlight the written query and click Execute.

5.
Observe the result and answer these questions:

How many rows did the query in task 1 return?

To answer this question, you can highlight the query written under the task 1 description, click Execute, and read the Results pane. (If you forgot how to access this pane, look at task 4 in exercise 1.) The number of rows affected by the query is 91.

How many rows did the query in Task 2 return?

To answer this question, you can highlight the query written under the task 2 description, click Execute, and read the Results pane. The number of rows affected by the query is 21. This means that there are 21 distinct values for the country column in the Sales.Customers table.

Under which circumstances do the following queries against the Sales.Customers table return the same result?

```
SELECT city, region FROM Sales.Customers;
SELECT DISTINCT city, region FROM Sales.Customers;
```

Both queries would return the same number of rows if all combinations of values in the city and region columns in the Sales.Customers table are unique. If they are not unique, the first query would return more rows than the second one with the DISTINCT clause.

Is the DISTINCT clause applied to all columns specified in the query or just the first column?

The DISTINCT clause is always applied to all columns specified in the SELECT list. It is very important to remember that the DISTINCT clause does not apply to just the first column in the list.

Result: After this exercise, you should have an understanding of how to return only the different (distinct) rows in the result set of a query.

Exercise 3: Using Table and Column Aliases
Task 1: Write a SELECT Statement that Uses a Table Alias
1.
In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECTc.contactname, c.contacttitle
FROM Sales.Customers AS c;
```

Tip: To use the IntelliSense feature when entering column names in a SELECT statement, you can use keyboard shortcuts. To enable IntelliSense, press Ctrl+Q+I. To list all the alias members, position your pointer after the alias and dot (for example, after "c.") and press Ctrl+J.

4.
Highlight the written query and click Execute.

Task 2: Write A SELECT Statement That Uses Column Aliases
1.
In the query pane, type the following query after the task 2 description:

```
SELECTc.contactname AS Name, c.contacttitle AS Title, c.companyname AS [Company Name]
FROM Sales.Customers AS c;
```

Observe that the column alias [Company Name] is enclosed in square brackets. Column names and aliases that have embedded spaces or reserved keywords must be delimited. This example uses square brackets as the delimiter, but you can also use the ANSI SQL standard delimiter of double quotes, as in "Company Name".

2.
Highlight the written query and click Execute.

Task 3: Write a SELECT Statement that Uses a Table Alias and a Column Alias

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

```
SELECT p.productname AS [Product Name]
FROM Production.Products AS p;
```

2.
Highlight the written query and click Execute.

Task 4: Analyze and Correct the Query
1.
Highlight the written query under the task 4 description and click Execute.

2.
Observe the result. Note that only one column is retrieved. The problem is that the developer forgot to add a comma after the first column name, so SQL Server treated the second word after the first column name as an alias. For this reason, it is a best practice to always use AS when specifying aliases. That way, it is easier to spot such errors.

3.
Correct the query by adding a comma after the first column name. The corrected query should look like this:

```
SELECT city, country
FROM Sales.Customers;
```

Result: After this exercise, you will know how to use aliases for table and column names.

Exercise 4: Using a Simple CASE Expression
Task 1: Write a SELECT Statement
1.
In Solution Explorer, double-click the query 81 - Lab Exercise 4.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECT p.categoryid, p.productname
FROM Production.Products AS p;
```

4.
Highlight the written query and click Execute.

Task 2: Write a SELECT Statement that Uses a CASE Expression

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

```
SELECT p.categoryid, p.productname,
CASE
WHEN p.categoryid = 1 THEN 'Beverages'
WHEN p.categoryid = 2 THEN 'Condiments'
WHEN p.categoryid = 3 THEN 'Confections'
WHEN p.categoryid = 4 THEN 'Dairy Products'
WHEN p.categoryid = 5 THEN 'Grains/Cereals'
WHEN p.categoryid = 6 THEN 'Meat/Poultry'
WHEN p.categoryid = 7 THEN 'Produce'
WHEN p.categoryid = 8 THEN 'Seafood'
ELSE 'Other'
END AS categoryname
FROM Production.Products AS p;
```

This query uses a CASE expression to add a new column. Note that, when you have a dynamic list of possible values, you usually store them in a separate table. However, for this example, a static list of values is being supplied.

2.
Highlight the written query and click Execute.

Task 3: Write a SELECT Statement that Uses a CASE Expression to differentiate Campaign-Focused Products
1.
Highlight the previous query. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 3 description.

3.
Add a new column using an additional CASE expression. Your query should look like this:

```
SELECTp.categoryid, p.productname,
CASE
WHEN p.categoryid = 1 THEN 'Beverages'
WHEN p.categoryid = 2 THEN 'Condiments'
WHEN p.categoryid = 3 THEN 'Confections'
WHEN p.categoryid = 4 THEN 'Dairy Products'
WHEN p.categoryid = 5 THEN 'Grains/Cereals'
WHEN p.categoryid = 6 THEN 'Meat/Poultry'
WHEN p.categoryid = 7 THEN 'Produce'
WHEN p.categoryid = 8 THEN 'Seafood'
ELSE 'Other'
END AS categoryname,
CASE
WHEN p.categoryid IN (1, 7, 8) THEN 'Campaign Products'
ELSE 'Non-Campaign Products'
END AS iscampaign
FROM Production.Products AS p;
```

4.
Highlight the written query and click Execute.

5.
In the result, observe that the first CASE expression uses the simple form, whereas the second uses the searched form.

Result: After this exercise, you should know how to use CASE expressions to write simple conditional logic.

--end Mod 3

Lab Answer Key: Module 4: Querying Multiple Tables

Lab: Querying Multiple Tables
Exercise 1: Writing Queries That Use Inner Joins
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and

20461C-MIA-SQL virtual machines are both running, and then log on to

20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab04\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 that Uses an Inner Join
1.

On the virtual machine, on the Taskbar, click SQL Server 2014 Management Studio. In the Connect to Server window, type MIA-SQL in the Server name text box and click Connect.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab04\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
SELECT
p.productname, c.categoryname
FROM Production.Products AS p
INNER JOIN Production.Categories AS c ON p.categoryid = c.categoryid;
```

7.

Highlight the written query and click Execute.

8.

Observe the result and answer these questions:

o

Which column did you specify as a predicate in the ON clause of the join? Why?

In this query, the categoryid column is the predicate. By intuition, most people would say that it is the predicate because this column exists in both input tables. By the way, using the same name for columns that contain the same data but in different tables is a good practice in data modeling. Another possibility is to check for referential integrity through primary and foreign key information using SQL Server Management Studio. If there are no primary or foreign key constraints, you will have to acquire information about the data model from the developer.

o

Let us say that there is a new row in the Production.Categories table and this new product category does not have any products associated with it in the Production.Products table. Would this row be included in the result of the SELECT statement written under the task 1 description?

No, because an inner join retrieves only the matching rows based on the predicate from both input tables. Since the new value for the categoryid is not present in the categoryid column in the Production.Products table, there would be no matching rows in the result of the SELECT statement.

Result: After this exercise, you should know how to use an inner join between two tables.

Exercise 2: Writing Queries That Use Multiple-Table Inner Joins
Task 1: Execute the T-SQL Statement
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

Highlight the written query under the Task 1 description and click Execute.

4.

Observe that you get the error message:

Ambiguous column name 'custid'.
5.

This error occurred because the custid column appears in both tables and you have to specify from which table you would like to retrieve the column values.

Task 2: Apply the Needed Changes and Execute the T-SQL Statement
1.

Copy the T-SQL statement from Task 1. In the query window, click the line after the Task 2 description. On the toolbar, click Paste.

2.

Add the column prefix “Customers” to the existing query so that it looks like this:

```
SELECT
Customers.custid, contactname, orderid
FROM Sales.Customers
INNER JOIN Sales.Orders ON Customers.custid = Orders.custid;
```
3.

Highlight the modified query and click Execute.

Task 3: Change the Table Aliases
1.

Highlight the previous query. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 3 description.

3.

Modify the T-SQL statement to use table aliases. Your query should look like this:

```
SELECT
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

4.

Highlight the written query and click Execute.

5.

Compare the results with the Task 2 results.

6.

Modify the T-SQL statement to include a full source table name as the column prefix. Your query should now look like this:

```
SELECT
Customers.custid, Customers.contactname, Orders.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

7.

Highlight the written query and click Execute.

8.

Observe that you get the error messages:

```
Msg 4104, Level 16, State 1, Line 2
The multi-part identifier "Customers.custid" could not be found.
Msg 4104, Level 16, State 1, Line 2
The multi-part identifier "Customers.contactname" could not be found.
Msg 4104, Level 16, State 1, Line 2
The multi-part identifier "Orders.orderid" could not be bound.
```

You received these error messages as, because you are using a different table alias, the full source table name you are referencing as a column prefix no longer exists. Remember that the SELECT clause is evaluated after the FROM clause, so you must use the table aliases when specifying columns in the SELECT clause.

9.

Modify the SELECT statement so that it uses the correct table aliases. Your query should look like this:

```
SELECT
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

Task 4: Add an Additional Table and Columns

1.

In the query pane, type the following query after the task 4 description:

```
SELECT
c.custid, c.contactname, o.orderid, d.productid, d.qty, d.unitprice
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid;
```

2.

Highlight the written query and click Execute.

3.

Observe the result. Remember that, when you have a multiple-table inner join, the logical query processing is different from the physical implementation. In this case, it means that you cannot guarantee the order in which the SQL Server optimizer will process the tables. For example, you cannot guarantee that the Sales.Customers table will be joined first with the Sales.Orders table, and then with the Sales.OrderDetails table.

Result: After this exercise, you should have a better understanding of why aliases are important and how to do a multiple-table join.

Exercise 3: Writing Queries That Use Self Joins

Task 1: Write a Basic SELECT Statement

1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid
FROM HR.Employees AS e;
```

4.

Highlight the written query and click Execute.

5.

Observe that the query retrieved nine rows.

Task 2: Write a Query that Uses a Self Join

1.

Highlight the previous query. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 2 description.

3.

Modify the query by adding a self join to get information about the managers. The query should look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid;
```

4.

Highlight the written query and click Execute.

5.

Observe that the query retrieved eight rows and answer these questions:

o

Is it mandatory to use table aliases when writing a statement with a self join? Can you use a full source table name as an alias?

You must use table aliases. You cannot use the full source table name as an alias when referencing both input tables. Eventually, you could use a full source table name as an alias for one input table and another alias for the second input table.

o

Why did you get fewer rows in the result from the T-SQL statement under the task 2 description compared to the result from the T-SQL statement under the task 1 description?

In task 2's T-SQL statement, the inner join used an ON clause based on manager information (column mgrid). The employee who is the CEO has a missing value in the mgrid column so this row is not included in the result.

Result: After this exercise, you should have an understanding of how to write T-SQL statements that use self joins.

Exercise 4: Writing Queries That Use Outer Joins
Task 1: Write a SELECT Statement that Uses an Outer Join
1.

In Solution Explorer, double-click the query 81 - Lab Exercise 4.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid;
```
4.

Highlight the written query and click Execute.

5.

Inspect the result. Notice that the custid 22 and custid 57 rows have a missing value in the orderid column. This is because there are no rows in the Sales.Orders table for these two values of the custid column. In business terms, this means that there are currently no orders for these two customers.

Result: After this exercise, you should have a basic understanding of how to write T-SQL statements that use outer joins.

Exercise 5: Writing Queries That Use Cross Joins
Task 1: Execute the T-SQL Statement
1.

In Solution Explorer, double-click the query 91 - Lab Exercise 5.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

Highlight the T-SQL code under the task 1 description and click Execute. Don't worry if you do not understand the provided T-SQL code, as it is used here to provide a more realistic example for a cross join in the next task.

Task 2: Write a SELECT Statement that Uses a Cross Join
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
e.empid, e.firstname, e.lastname, c.calendardate
FROM HR.Employees AS e
CROSS JOIN HR.Calendar AS c;
```
2.

Highlight the written query and click Execute.

3.

Observe that the query retrieved 3,285 rows and that there are nine rows in the HR.Employees table. Because a cross join produces a Cartesian product of both inputs, it means that there are 365 (3,285/9) rows in the HR.Calendar table.

Task 3: Drop the HR.Calendar Table
1.

Highlight the written query under the task 3 description and click Execute.

Result: After this exercise, you should have an understanding of how to write T-SQL statements that use cross joins..

--end mod 4

Lab Answer Key: Module 5: Sorting and Filtering Data

Lab: Sorting and Filtering Data
Exercise 1: Writing Queries That Filter Data Using a WHERE Clause
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab05\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 that Uses a WHERE Clause
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine instance using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab05\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
country = N'Brazil';
```
7.

Notice the use of the N prefix for the character literal. This prefix is used because the country column is a Unicode data type. When expressing a Unicode character literal, you need to specify the character N (for National) as a prefix. You will learn more about data types in the next module.

8.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement that Uses an IN Predicate in the WHERE Clause
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
country IN (N'Brazil', N'UK', N'USA');
```
2.

Highlight the written query and click Execute.

Task 4: Write a SELECT Statement that Uses a LIKE Predicate in the WHERE Clause
1.

In the query pane, type the following query after the task 3 description:

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
contactname LIKE N'A%';
```
2.

Remember that the percent sign (%) wildcard represents a string of any size (including an empty string), whereas the underscore (_) wildcard represents a single character.

3.

Highlight the written query and click Execute.

Task 5: Observe the T-SQL Statement Provided by the IT Department
1.

Highlight the T-SQL statement provided under the task 4 description and click Execute.

2.

Highlight the provided T-SQL statement. On the toolbar, click Edit and then Copy.

3.

In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 4 description.

4.

Modify the query so that it looks like this:

```
SELECT
c.custid, c.companyname, o.orderid
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE
c.city = N'Paris';
```
5.

Highlight the modified query and click Execute.

6.

Observe the result. Is it the same as that of the first SQL statement? The result is not the same. When you specify the predicate in the ON clause, the left outer join preserves all the rows from the left table (Sales.Customers) and adds only the matching rows from the right table (Sales.Orders), based on the predicate in the ON clause. This means that all the customers will show up in the output, but only the ones from Paris will have matching orders. When you specify the predicate in the WHERE clause, the query will filter only the Paris customers. So, be aware that, when you use an outer join, the result of a query in which the predicate is specified in the ON clause can differ from the result of a query in which the predicate is specified in the WHERE clause. (When using an inner join, the results are always the same.) This is because the ON predicate is matching—it defines which rows from the non-preserved side to match to those from the preserved side. The WHERE predicate is a filtering predicate—if a row from either side doesn't satisfy the WHERE predicate, the row is filtered out.

Task 6: Write a SELECT Statement to Retrieve those Customers Without Orders

1.

In the query pane, type the following query after the task 5 description:

```
SELECT
c.custid, c.companyname
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE o.custid IS NULL;
```

2.

It is important to note that, when you are looking for a NULL, you should use the IS NULL, not the equality operator. The equality operator will always return UNKNOWN when you compare something to a NULL. It will even return UNKNOWN when you compare two NULLs.

3.

The choice of which attribute to filter from the non-preserved side of the join is also important. You should choose an attribute that can only have a NULL when the row is an outer row (for example, a NULL originating from the base table). For this purpose, three cases are safe to consider:

o

A primary key column. A primary key column cannot be NULL. Therefore, a NULL in such a column can only mean that the row is an outer row.

o

A join column. If a row has a NULL in the join column, it is filtered out by the second phase of the join. So a NULL in such a column can only mean that it is an outer row.

o

A column defined as NOT NULL. A NULL in a column that is defined as NOT NULL can only mean that the row is an outer row.

4.

Highlight the written query and click Execute.

Result: After this exercise, you should be able to filter rows of data from one or more tables by using WHERE predicates with logical operators.

Exercise 2: Writing Queries That Sort Data Using an ORDER BY Clause
Task 1: Write a SELECT Statement that Uses an ORDER BY Clause
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
c.custid, c.contactname, o.orderid, o.orderdate
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE
o.orderdate >= '20080401'
ORDER BY
o.orderdate DESC, c.custid ASC;
```

Notice the date filter. It uses a literal (constant) of a date. SQL Server recognizes "20080401" as a character string literal and not as a date and time literal, but because the expression involves two operands of different types, one needs to be implicitly converted to the other's type. In this example, the character string literal is converted to the column's data type (DATETIME) because character strings are considered lower in terms of data type precedence, with respect to date and time data types.

Also notice that the character string literal follows the format "yyyymmdd". Using this format is a best practice because SQL Server knows how to convert it to the correct date, regardless of the language settings.

4.

Highlight the written query and click Execute.

Task 2: Apply the Needed Changes and Execute the T-SQL Statement
1.

Highlight the written query under the task 2 description and click Execute.

2.

Observe the error message:

Invalid column name 'mgrlastname'.

3.

This error occurred because the WHERE clause is evaluated before the SELECT clause and, at that time, the column did not have an alias. To fix this problem, you must use the source column name with the appropriate table alias. Modify the T-SQL statement to look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
WHERE
m.lastname = N'Buck';
```

4.
Highlight the written query and click Execute.

Task 3: Order the Result by the firstname Column
1.
Highlight the previous query. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 3 description.

3.
Modify the T-SQL statement to include an ORDER BY clause that uses the source column name of m.firstname. Your query should look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
ORDER BY
m.firstname;
```

4.

Highlight the written query and click Execute.

5.

Modify the ORDER BY clause so that it uses the alias for the same column (mgrfirstname). Your query should look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
ORDER BY
mgrfirstname;
```

6.

Highlight the written query and click Execute.

7.

Observe the result. Why were you able to use a source column or alias column name? You can use either one because the ORDER BY clause is evaluated after the SELECT clause and the alias for the column name is known.

Result: After this exercise, you should know how to use an ORDER BY clause.

Exercise 3: Writing Queries That Filter Data Using the TOP Option
Task 1: Writing Queries That Filter Data Using the TOP Clause
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT TOP (20)
orderid, orderdate
FROM Sales.Orders
```

```
ORDER BY orderdate DESC;
```

4.

Highlight the written query and click Execute.

Task 2: Use the OFFSET-FETCH Clause to Implement the Same Task

1.

In the query pane, type the following query after the task 2 description:

```
SELECT
orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC
OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY;
```

2.

Remember that the OFFSET-FETCH clause was a new functionality in SQL Server 2012. Unlike the TOP clause, the OFFSET-FETCH clause must be used with the ORDER BY clause.

3.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Retrieve the Most Expensive Products

1.

In the query pane, type the following query after the task 3 description:

```
SELECT TOP (10) PERCENT
productname, unitprice
FROM Production.Products
ORDER BY unitprice DESC;
```

2.

Implementing this task with the OFFSET-FETCH clause is possible but not easy because, unlike TOP, OFFSET-FETCH does not support a PERCENT option.

3.

Highlight the written query and click Execute.

Result: After this exercise, you should have an understanding of how to apply the TOP option in the SELECT clause of a T-SQL statement.

Exercise 4: Writing Queries That Filter Data Using the OFFSET-FETCH Clause
Task 1: OFFSET-FETCH Clause to Fetch the First 20 Rows
1.
In Solution Explorer, double-click the query 81 - Lab Exercise 4.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECT
custid, orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid
OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY;
```

4.
Highlight the written query and click Execute.

Task 2: Use the OFFSET-FETCH Clause to Skip the First 20 Rows
1.
In the query pane, type the following query after the task 2 description:

```
SELECT
custid, orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

2.

Highlight the written query and click Execute.

Task 3: Write a Generic Form of the OFFSET-FETCH Clause for Paging
1.

Solution: OFFSET (@pagenum - 1) * @pagesize ROWS FETCH NEXT @pagesize ROWS ONLY.

--end Mod 5

Lab Answer Key: Module 6: Working with SQL Server 2014 Data Types

Lab: Working with SQL Server 2014 Data Types
Exercise 1: Writing Queries That Return Date and Time Data
Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab06\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 Distinct Customers
1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab06\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.
In the query pane, type the following query after the task 1 description:

```
SELECT
CURRENT_TIMESTAMP AS currentdatetime,
CAST(CURRENT_TIMESTAMP AS DATE) AS currentdate,
CAST(CURRENT_TIMESTAMP AS TIME) AS currenttime,
YEAR(CURRENT_TIMESTAMP) AS currentyear,
MONTH(CURRENT_TIMESTAMP) AS currentmonth,
DAY(CURRENT_TIMESTAMP) AS currentday,
```

DATEPART(week, CURRENT_TIMESTAMP) AS currentweeknumber,
DATENAME(month, CURRENT_TIMESTAMP) AS currentmonthname;
This query uses the CURRENT_TIMESTAMP function to return the current date and time. You can also use the SYSDATETIME function to get a more precise time element compared to the CURRENT_TIMESTAMP function.

Note that you cannot use the alias currentdatetime as the source in the second column calculation because SQL Server supports a concept called all-at-once operations. This means that all expressions appearing in the same logical query processing phase are evaluated as if they occurred at the same point in time. This concept explains why, for example, you cannot refer to column aliases assigned in the SELECT clause within the same SELECT clause, even if it seems intuitive that you should be able to.

7.
Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Return the Data Type date
1.
In the query pane, type the following queries after the task 2 description. The first query uses SQL Server 2014's new DATEFROMPARTS function:

SELECT DATEFROMPARTS(2011, 12, 11) AS somedate;
SELECT CAST('20111211' AS DATE) AS somedate;
SELECT CONVERT(DATE, '12/11/2011', 101) AS somedate;
2.
Highlight the written queries and click Execute.

Task 4: Write a SELECT Statement that Uses Different Date and Time Functions
1.
In the query pane, type the following query after the task 3 description:

SELECT
DATEADD(month, 3, CURRENT_TIMESTAMP) AS threemonths,
DATEDIFF(day, CURRENT_TIMESTAMP, DATEADD(month, 3, CURRENT_TIMESTAMP)) AS diffdays,
DATEDIFF(week, '19920404', '20110916') AS diffweeks,
DATEADD(day, -DAY(CURRENT_TIMESTAMP) + 1, CURRENT_TIMESTAMP) AS firstday;
2.

Highlight the written query and click Execute.

Task 5: Observe the Table Provided by the IT Department
1.
Highlight the written query under the task 4 description and click Execute.

2.

In the query pane, type the following queries after the task 4 description:

```
SELECT
isitdate,
CASE WHEN ISDATE(isitdate) = 1 THEN CONVERT(DATE, isitdate) ELSE NULL END AS converteddate
FROM Sales.Somedates;
--Uses the TRY_CONVERT function:
SELECT
isitdate,
TRY_CONVERT(DATE, isitdate) AS converteddate
FROM Sales.Somedates;
```

The second query uses the TRY_CONVERT function. This function returns a value casted to the specified data type if the casting succeeds; otherwise, it returns NULL. Do not worry if you do not recognize the type conversion functions as they will be covered in the next module.

3.
Highlight the written queries and click Execute.

4.
Observe the result and answer these questions:

What is the difference between the SYSDATETIME and CURRENT_TIMESTAMP functions?

There are two main differences. First, the SYSDATETIME function provides a more precise time element compared to the CURRENT_TIMESTAMP function. Second, the SYSDATETIME function returns the data type datetime2(7), whereas the CURRENT_TIMESTAMP returns the data type datetime.

What is a language-neutral format for the data type date?

You can use the format 'YYYYMMDD' or 'YYYY-MM-DD'.

Result: After this exercise, you should be able to retrieve date and time data using T-SQL.

Exercise 2: Writing Queries That Use Date and Time Functions
Task 1: Write a SELECT Statement to Retrieve All Distinct Customers
1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

SELECT DISTINCT
custid
FROM Sales.Orders
WHERE
YEAR(orderdate) = 2008
AND MONTH(orderdate) = 2;
4.

Highlight the written query and click Execute.

5.

Note that you could also write a query that uses a range format, which would better utilize indexing. The query would then look like this:

SELECT DISTINCT
custid
FROM Sales.Orders
WHERE
orderdate >= '20080201'
AND orderdate < '20080301';
Task 2: Write a SELECT Statement to Calculate the First and Last Day of the Month
1.

In the query pane, type the following query after the task 2 description:

SELECT
CURRENT_TIMESTAMP AS currentdate,
DATEADD (day, 1, EOMONTH(CURRENT_TIMESTAMP, -1)) AS firstofmonth,
EOMONTH(CURRENT_TIMESTAMP) AS endofmonth;
2.

This query uses the EOMONTH function, which was new in SQL Server 2012.

3.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Retrieve the Orders Placed in the Last Five Days of the Ordered Month
1.

In the query pane, type the following query after the task 3 description:

SELECT
orderid, custid, orderdate
FROM Sales.Orders
WHERE
DATEDIFF(
day,
orderdate,
EOMONTH(orderdate)
) < 5;
2.

Highlight the written query and click Execute.

Task 4: Write a SELECT Statement to Retrieve All Distinct Products Sold in the First 10 Weeks of the Year 2007
1.

In the query pane, type the following query after the task 4 description:

SELECT DISTINCT
d.productid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE
DATEPART(week, orderdate) <= 10
AND YEAR(orderdate) = 2007;
2.

Highlight the written query and click Execute.

Result: After this exercise, you should know how to use the date and time functions.

Exercise 3: Writing Queries That Return Character Data
Task 1: Write a SELECT Statement to Concatenate Two Columns
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
contactname + N' (city: ' + city + N')' AS contactwithcity
FROM Sales.Customers;
```
4.

Highlight the written query and click Execute.

Task 2: Add an Additional Column and Treat NULL as an Empty String
You will have to use the COALLESCE function which will be covered in module 8. You can either look at the solution or skip this task.

1.

In the query pane, type the following query after the task 2 description:

```
SELECT
contactname + N' (city: ' + city + N', region: ' + COALESCE(region, '') + N')' AS fullcontact
FROM Sales.Customers;
```
This query uses the COALESCE function to replace a NULL with an empty string. The next module will include more examples of how to handle a NULL.

2.

Highlight the written query and click Execute.

3.

Note that you can also use the CONCAT function to concatenate strings. It also replaces a NULL with an empty string. The query using the CONCAT function would look like this:

```
SELECT
CONCAT(contactname, N' (city: ', city,  N', region: ', region, N')') AS fullcontact
FROM Sales.Customers;
```
Task 3: Write a SELECT Statement to Retrieve All Customers Based on the First Character in the Contact Name
1.

In the query pane, type the following query after the task 3 description:

```
SELECT contactname, contacttitle
FROM Sales.Customers
```

```
WHERE contactname LIKE N'[A-G]%'
ORDER BY contactname;
```
2.

Highlight the written query and click Execute.

Result: After this exercise, you should have an understanding of how to concatenate character data.

Exercise 4: Writing Queries That Use Character Functions
Task 1: Write a SELECT Statement that Uses the SUBSTRING Function
1.

In Solution Explorer, double-click the query 81 - Lab Exercise 4.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
contactname,
SUBSTRING(contactname, 0, CHARINDEX(N',', contactname)) AS lastname
FROM Sales.Customers;
```
4.

Highlight the written query and click Execute.

Task 2: Extend the SUBSTRING Function to Retrieve the First Name
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
REPLACE(contactname, ',', '') AS newcontactname,
SUBSTRING(contactname, CHARINDEX(N',', contactname)+1, LEN(contactname)-CHARINDEX(N',',
contactname)+1) AS firstname
FROM Sales.Customers;
```
2.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Change the Customer IDs
1.

In the query pane, type the following query after the task 3 description:

```
SELECT
custid,
N'C' + RIGHT(REPLICATE('0', 5) + CAST(custid AS VARCHAR(5)), 5) AS custnewid
FROM Sales.Customers;
```
2.

Highlight the written query and click Execute.

3.

Note that you can also use the FORMAT function. The query would then look like this:

```
SELECT
FORMAT(custid, N'\C00000')
FROM Sales.Customers;
```
Task 4: Challenge: Write a SELECT Statement to Return the Number of Character Occurrences
1.

In the query pane, type the following query after the task 4 description:

```
SELECT
contactname,
LEN(contactname) - LEN(REPLACE(contactname, 'a', '')) AS numberofa
FROM Sales.Customers
ORDER BY numberofa DESC;
```
This elegant solution first returns the number of characters in the contact name, and then subtracts the number of characters in the contact name without the character 'a'. The result is stored in a new column named numberofa.

2.

Highlight the written query and click Execute.

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

--end Mod 6

Lab Answer Key: Module 7: Using DML to Modify Data

Lab: Using DML to Modify Data
Exercise 1: Inserting Records with DML
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab07\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: Insert a Row
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

In the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab07\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 41 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
INSERT INTO HR.Employees(Title, titleofcourtesy,FirstName, Lastname,hiredate, birthdate, address, city, country, phone)
VALUES
('Sales Representative',
'Mr',
'Laurence',
'Grider',
'04/04/2013',
'10/25/1975',
'1234 1st Ave. S.E. ',
'Seattle',
'USA',
'(206)555-0105');
```

7.

Click Execute.

Task 3: Insert a Row with SELECT

1.

Click New Query.

2.

In the query pane, type the following query:

```
INSERT INTO Sales.Customers(
companyname,
contactname,
contacttitle,
address,
city,
region,
postalcode,
country,
phone,
fax)
SELECT * FROM dbo.PotentialCustomers;
```

3.

Click Execute.

Result: After successfully completing this exercise, you will have one new employee and three new customers.

Exercise 2: Update and Delete Records Using DML
Task 1: Update Rows
1.

Click New Query.

2.

In the query pane, type the following query:

```
UPDATE Sales.Customers
SET contacttitle='Sales Consultant'
WHERE city='Berlin'
AND contacttitle='Sales Representative';
```

3.

Click Execute.

Task 2: Delete Rows
1.

Click New Query.

2.

In the query pane, type the following query:

```
DELETE FROM dbo.PotentialCustomers
WHERE contactname IN(
'Taylor, Maurice',
'Mallit, Ken',
'Tiano, Mike');
```

3.

Click Execute.

Result: After successfully completing this exercise, you will have updated all the records in the Customers table which have a city of Berlin and a contacttitle of Sales Representative to have a contacttitle of Sales Consultant. You will also have deleted the three records in the PotentialCustomers table, which have already been added to the Customers table.
--end Mod 7

Lab Answer Key: Module 8: Using Built-In Functions

Lab: Using Built-In Functions
Exercise 1: Writing Queries That Use Conversion Functions
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab08\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 that Uses the CAST or CONVERT Function
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab08\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
SELECT N'The unit price for the ' + productname + N' is ' + CAST(unitprice AS NVARCHAR(10)) + N' $.' AS
productdesc
FROM Production.Products;
```

This query uses the CAST function rather than the CONVERT function. It is better to use the CAST function because it is an ANSI SQL standard. You should use the CONVERT function only when you need to apply a specific style during a conversion.

7.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Filter Rows Based on Specific Date Information
1.

In the query pane, type the following query after the task 2 description:

```
SELECT orderid, orderdate, shippeddate, COALESCE(shipregion, 'No region') AS shipregion
FROM Sales.Orders
WHERE
orderdate >= CONVERT(DATETIME, '4/1/2007', 101)
AND orderdate <= CONVERT(DATETIME, '11/30/2007', 101)
AND shippeddate > DATEADD(DAY, 30, orderdate);
```

2.

Highlight the written query and click Execute.

3.

Note that you could also write a solution using the PARSE function. The query would look like this:

```
SELECT orderid, orderdate, shippeddate, COALESCE(shipregion, 'No region') AS shipregion
FROM Sales.Orders
WHERE
orderdate >= PARSE('4/1/2007' AS DATETIME USING 'en-US')
AND orderdate <= PARSE('11/30/2007' AS DATETIME USING 'en-US')
AND shippeddate > DATEADD(DAY, 30, orderdate);
```

Task 4: Write a SELECT Statement to Convert the Phone Number Information to an Integer Value
1.

In the query pane, type the following query after the task 3 description:

SELECT
CONVERT(INT, REPLACE(REPLACE(REPLACE(REPLACE(phone, N'-', N''), N'(', ''), N')', ''), ' ', '')) AS phonenoasint
FROM Sales.Customers;
This query is trying to use the CONVERT function to convert phone numbers that include characters such as hyphens and parentheses into an integer value.

2.

Highlight the written query and click Execute.

3.

Observe the error message:

Conversion failed when converting the nvarchar value '67.89.01.23' to data type int.
Because you want to retrieve rows without conversion errors and have a NULL for those that produce a conversion error, you can use the TRY_CONVERT function.

4.

Modify the query to use the TRY_CONVERT function. The query should look like this:

SELECT
TRY_CONVERT(INT, REPLACE(REPLACE(REPLACE(REPLACE(phone, N'-', N''), N'(', ''), N')', ''), ' ', '')) AS phonenoasint
FROM Sales.Customers;
5.

Highlight the written query and click Execute. Observe the result. The rows that could not be converted have a NULL.

Result: The unit price for the Product HHYDP is 18.00 $.

Exercise 2: Writing Queries That Use Logical Functions
Task 1: Write a SELECT Statement to Mark Specific Customers Based on their Country and Contact Title
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
IIF(country = N'Mexico' AND contacttitle = N'Owner', N'Target group', N'Other') AS segmentgroup,
custid, contactname
FROM Sales.Customers;
```
4.

The IIF function was new in SQL Server 2012. It was added mainly to support migrations from Microsoft Access to SQL Server. You can always use a CASE expression to achieve the same result.

5.

Highlight the written query and click Execute.

Task 2: Modify the T-SQL Statement to Mark Different Customers
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
IIF(contacttitle = N'Owner' OR region IS NOT NULL, N'Target group', N'Other') AS segmentgroup,  custid,
contactname
FROM Sales.Customers;
```
2.

Highlight the written query and click Execute.

Task 3: Create Four Groups of Customers
1.

In the query pane, type the following query after the task 3 description:

```
SELECT CHOOSE(custid % 4 + 1, N'Group One', N'Group Two', N'Group Three', N'Group Four') AS
segmentgroup, custid, contactname
FROM Sales.Customers;
```
2.

Highlight the written query and click Execute.

Result: After this exercise, you should know how to use the logical functions.

Exercise 3: Writing Queries That Test for Nullability
Task 1: Write a SELECT Statement to Retrieve the Customer Fax Information
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT contactname, COALESCE(fax, N'No information') AS faxinformation
FROM Sales.Customers;
```
This query uses the COALESCE function to retrieve customers' fax information.

4.

Highlight the written query and click Execute.

5.

In the query pane, type the following query after the previous query:

```
SELECT contactname, ISNULL(fax, N'No information') AS faxinformation
FROM Sales.Customers;
```
This query uses the ISNULL function. What is the difference between the ISNULL and COALESCE functions? COALESCE is a standard ANSI SQL function and ISNULL is not. So you should use the COALESCE function.

6.

Highlight the written query and click Execute.

Task 2: Write a Filter for a Variable that Could Be a Null
1.

Highlight the query provided under the task 2 description and click Execute.

2.

Highlight the previous query. On the toolbar, click Edit and then Copy.

3.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste. You have now copied the previous query to the same query window after the task 2 description.

4.

Modify the query so that it looks like this:

```
DECLARE @region AS NVARCHAR(30) = NULL;
SELECT
custid, region
FROM Sales.Customers
WHERE region = @region OR (region IS NULL AND @region IS NULL);
```

5.

Highlight the modified query and click Execute.

6.

Test the modified query by setting the @region parameter to N'WA'. The T-SQL expression should look like this:

```
DECLARE @region AS NVARCHAR(30) = N'WA';
SELECT
custid, region
FROM Sales.Customers
WHERE region = @region OR (region IS NULL AND @region IS NULL);
```

7.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Return All the Customers that Do Not Have a Two-Character Abbreviation for the Region

1.

In the query pane, type the following query after the task 3 description:

```
SELECT custid, contactname, city, region
FROM Sales.Customers
WHERE
region IS NULL
OR LEN(region) <> 2;
```

2.

Highlight the written query and click Execute.

Result: After this exercise, you should have an understanding of how to test for nullability.

--end Mod 8

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 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

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, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab09\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 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. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then 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:

o

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.

o

Why did the query fail? In a grouped query, you will get 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 3 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 4 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.

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 the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste.

3.

Modify the T-SQL statement so that it adds additional 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 the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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, since 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. If you want to get the correct number of distinct customers, you have to 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 the query 81 - Lab Exercise 4.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 3 description. On the toolbar, click Edit and then 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.

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

--end Mod 9

Lab Answer Key: Module 10: Using Subqueries

Lab: Using Subqueries
Exercise 1: Writing Queries That Use Self-Contained Subqueries
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab10\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 the Last Order Date
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab10\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
SELECT MAX(orderdate) AS lastorderdate
FROM Sales.Orders;
```

7.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Retrieve All Orders Placed on the Last Order Date

1.

In the query pane, type the following query after the task 2 description:

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
orderdate = (SELECT MAX(orderdate) FROM Sales.Orders);
```

2.

Highlight the written query and click Execute.

Task 4: Observe the T-SQL Statement Provided by the IT Department

1.

Highlight the provided T-SQL statement under the task 3 description and click Execute.

2.

Modify the query to filter customers whose contact name starts with the letter B. Your query should look like this:

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
custid =
(
SELECT custid
```

FROM Sales.Customers
WHERE contactname LIKE N'B%'
);
3.

Highlight the written query and click Execute.

4.

Observe the error message:

Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression.
Why did the query fail? It failed because the subquery returned more than one row. To fix this problem, you should replace the = operator with an IN operator.

5.

Modify the query so that it uses the IN operator. Your query should look like this:

```
SELECT
orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE
custid IN
(
SELECT custid
FROM Sales.Customers
WHERE contactname LIKE N'B%'
);
```
6.

Highlight the written query and click Execute.

Task 5: Write A SELECT Statement to Analyze Each Order's Sales as a Percentage of the Total Sales Amount
1.

In the query pane, type the following query after the task 4 description:

```
SELECT
o.orderid,
SUM(d.qty * d.unitprice) AS totalsalesamount,
SUM(d.qty * d.unitprice) /
```

```
(
SELECT SUM(d.qty * d.unitprice)
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080501' AND orderdate < '20080601'
) * 100. AS salespctoftotal
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.orderdate >= '20080501' AND orderdate < '20080601'
GROUP BY o.orderid;
```

2.

Highlight the written query and click Execute.

Result: After this exercise, you should be able to use self-contained subqueries in T-SQL statements.

Exercise 2: Writing Queries That Use Scalar and Multi-Result Subqueries
Task 1: Write a SELECT Statement to Retrieve Specific Products
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
productid, productname
FROM Production.Products
WHERE
productid IN
(
SELECT productid
FROM Sales.OrderDetails
WHERE qty > 100
);
```

4.

Highlight the written query and click Execute.

Task 2: Write a SELECT Statement to Retrieve Those Customers Without Orders
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
custid, contactname
FROM Sales.Customers
WHERE custid NOT IN
(
SELECT custid
FROM Sales.Orders
);
```
2.

Highlight the written query and click Execute.

3.

Observe the result. Notice there are two customers without an order.

Task 3: Add a Row and Rerun the Query That Retrieves Those Customers Without Orders
1.

Highlight the provided T-SQL statement under the task 3 description and click Execute. This code inserts an additional row that has a NULL in the custid column of the Sales.Orders table.

2.

Highlight the query in task 2. On the toolbar, click Edit and then Copy.

3.

In the query window, click the line after the provided T-SQL statement. On the toolbar, click Edit and then Paste.

4.

Highlight the written query and click Execute.

5.

Notice that you have an empty result despite getting two rows when you first ran the query in task 2. Why did you get an empty result this time? There is an issue with the NULL in the new row you added because the custid column is the only one that is part of the subquery. The IN operator supports three-valued logic (TRUE, FALSE, UNKNOWN). Before you apply the NOT operator, the logical meaning of UNKNOWN is that you can't tell for sure whether the customer ID appears in the set, because the NULL could represent that customer ID as well as anything else. As a more tangible example, consider the expression 22 NOT IN (1, 2, NULL). If you evaluate each individual expression in the parentheses to its truth value, you will get NOT (FALSE OR FALSE OR UNKNOWN), which translates to NOT UNKNOWN, which evaluates to UNKNOWN. The tricky part is that negating UNKNOWN with the NOT operator still yields UNKNOWN, and UNKNOWN is filtered out in a query filter. In short, when you use the NOT IN predicate against a subquery that returns at least one NULL, the outer query always returns an empty set.

6.

To solve this problem, modify the T-SQL statement so that the subquery does not return NULLs. Your query should look like this:

```
SELECT
custid, contactname
FROM Sales.Customers
WHERE custid NOT IN
(
SELECT custid
FROM Sales.Orders
WHERE custid IS NOT NULL
);
```
7.

Highlight the modified query and click Execute.

Result: After this exercise, you should know how to use multi-result subqueries in T-SQL statements.

Exercise 3: Writing Queries That Use Correlated Subqueries and an EXISTS Predicate
Task 1: Write a SELECT Statement to Retrieve the Last Order Date for Each Customer
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
c.custid, c.contactname,
(
SELECT MAX(o.orderdate)
FROM Sales.Orders AS o
WHERE o.custid = c.custid
) AS lastorderdate
FROM Sales.Customers AS c;
```

4.

Highlight the written query and click Execute.

Task 2: Write a SELECT Statement That Uses the EXISTS Predicate to Retrieve Those Customers Without Orders

1.

In the query pane, type the following query after the task 2 description:

```
SELECT c.custid, c.contactname
FROM Sales.Customers AS c
WHERE
NOT EXISTS (SELECT * FROM Sales.Orders AS o WHERE o.custid = c.custid);
```

2.

Highlight the written query and click Execute.

3.

Notice that you achieved the same result as the modified query in exercise 2, task 3, but without a filter to exclude NULLs. Why didn't you need to explicitly filter out NULLs? The EXISTS predicate uses two-valued logic (TRUE, FALSE) and checks only if the rows specified in the correlated subquery exists. Another benefit of using the EXISTS predicate is better performance. The SQL Server engine knows it is enough to determine whether the subquery returns at least one row or none, so it doesn't need to process all qualifying rows.

Task 3: Write a SELECT Statement to Retrieve Customers Who Bought Expensive Products

1.

In the query pane, type the following query after the task 3 description:

```
SELECT c.custid, c.contactname
FROM Sales.Customers AS c
WHERE
```

```
EXISTS (
SELECT *
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
WHERE o.custid = c.custid
AND d.unitprice > 100.
AND o.orderdate >= '20080401'
);
```

2.

Highlight the written query and click Execute.

Task 4: Write a SELECT Statement to Display the Total Sales Amount and the Running Total Sales Amount for Each Order Year

1.

In the query pane, type the following query after the task 4 description:

```
SELECT
YEAR(o.orderdate) as orderyear,
SUM(d.qty * d.unitprice) AS totalsales,
(
SELECT SUM(d2.qty * d2.unitprice)
FROM Sales.Orders AS o2
INNER JOIN Sales.OrderDetails AS d2 ON d2.orderid = o2.orderid
WHERE YEAR(o2.orderdate) <= YEAR(o.orderdate)
) AS runsales
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY YEAR(o.orderdate)
ORDER BY orderyear;
```

2.

Highlight the written query and click Execute.

Task 5: Clean the Sales.Customers Table

1.

Highlight the provided T-SQL statement and click Execute.

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

--end Mod 10

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 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

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, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab11\Starter\Project\Project.ssmssln.

4.

In Solution Explorer expand the Queries node, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

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 click Execute.

2.

Observe the error message:

Results: 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 click Execute.

2.

Observe the error message:

Results: 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
1.

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 the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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 the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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 the query 81 - Lab Exercise 4.sql.

2.

When the query window opens, highlight the statement USE TSQL; and 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
1.

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.

--end Mod 11

Lab Answer Key: Module 12: Using Set Operators

Lab: Using Set Operators
Exercise 1: Writing Queries That Use UNION Set Operators and UNION ALL Multi-Set Operators
Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab12\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 Specific Products
1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab12\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, expand Queries folder and double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.
In the query pane, type the following query after the task 1 description:

```
SELECT
productid, productname
FROM Production.Products
WHERE categoryid = 4;
```

7.
Highlight the written query and click Execute. Observe that the query retrieved 10 rows.

Task 3: Write a SELECT Statement to Retrieve All Products with a Total Sales Amount of More than $50,000

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

```
SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

2.
Highlight the written query and click Execute. Observe that the query retrieved four rows.

Task 4: Merge the Results from Task 1 and Task 2
1.
In the query pane, type the following query after the task 3 description:

```
SELECT
productid, productname
FROM Production.Products
WHERE categoryid = 4
UNION
SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

2.
Highlight the written query and click Execute.

3.
Observe the result. What is the total number of rows in the result? If you compare this number with an aggregate value of the number of rows from tasks 1 and 2, is there any difference? The total number of rows retrieved by the query is 12. This is two rows less than the aggregate value of rows from the query in task 1 (10 rows) and task 2 (four rows).

4.
Highlight the previous query. On the toolbar, click Edit and then Copy.

5.
In the query window, click the line after the written T-SQL statement. On the toolbar, click Edit and then Paste.

6.
Modify the T-SQL statement by replacing the UNION operator with the UNION ALL operator. The query should look like this:

```
SELECT
productid, productname
FROM Production.Products
WHERE categoryid = 4
UNION ALL
SELECT
d.productid, p.productname
FROM Sales.OrderDetails d
INNER JOIN Production.Products p ON p.productid = d.productid
GROUP BY d.productid, p.productname
HAVING SUM(d.qty * d.unitprice) > 50000;
```

7.
Highlight the modified query and click Execute.

8.
Observe the result. What is the total number of rows in the result? What is the difference between the UNION and UNION ALL operators? The total number of rows retrieved by the query is 14. It is the same as the aggregate value of rows from the queries in tasks 1 and 2. This is because UNION ALL is a multi-set operator that returns all rows that appear in any of the inputs, without really comparing rows and without eliminating duplicates. The UNION set operator removes the duplicate rows and the result consists of only distinct rows.

9.
So, when should you use either UNION ALL or UNION when unifying two inputs? If a potential exists for duplicates and you need to return them, use UNION ALL. If a potential exists for duplicates but you need to return distinct rows, use UNION. If no potential exists for duplicates when unifying the two inputs, UNION and UNION ALL are logically equivalent. However, in such a case, using UNION ALL is recommended because it removes the overhead of SQL Server checking for duplicates.

Task 5: Write a SELECT Statement to Retrieve the Top 10 Customers by Sales Amount for January 2008 and February 2008

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

```
SELECT
c1.custid, c1.contactname
FROM
(
SELECT TOP (10)
o.custid, c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.orderdate >= '20080101' AND o.orderdate < '20080201'
GROUP BY o.custid, c.contactname
ORDER BY SUM(o.val) DESC
) AS c1
UNION
SELECT c2.custid, c2.contactname
FROM
(
SELECT TOP (10)
o.custid, c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE o.orderdate >= '20080201' AND o.orderdate < '20080301'
GROUP BY o.custid, c.contactname
ORDER BY SUM(o.val) DESC
) AS c2;
```

2.
Highlight the written query and click Execute.

Result: After this exercise, you should know how to use the UNION and UNION ALL set operators in T-SQL statements.

Exercise 2: Writing Queries That Use the CROSS APPLY and OUTER APPLY Operators
Task 1: Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Last Two Orders for Each Product

1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECT
p.productid, p.productname, o.orderid
FROM Production.Products AS p
CROSS APPLY
(
SELECT TOP(2)
d.orderid
FROM Sales.OrderDetails AS d
WHERE d.productid = p.productid
ORDER BY d.orderid DESC
) o
ORDER BY p.productid;
```

4.
Highlight the written query and click Execute.

Task 2: Write a SELECT Statement That Uses the CROSS APPLY Operator to Retrieve the Top Three Products, Based on Sales Revenue, for Each Customer
1.
Highlight the provided T-SQL code after the task 2 description and click Execute.

2.
In the query pane, type the following query after the provided T-SQL code:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
CROSS APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
ORDER BY c.custid;
```

Tip: You can make the inline TVF (dbo.fnGetTop3ProductsForCustomer) more flexible by making the number of top rows to return an argument instead of fixing the number to three in the function's code.

3.
Highlight the written query and click Execute. The query retrieved 265 rows.

Task 3: Use the OUTER APPLY Operator
1.
Highlight the previous query in task 2. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste.

3.
Modify the T-SQL statement by replacing the CROSS APPLY operator with the OUTER APPLY operator. The query should look like this:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
OUTER APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
ORDER BY c.custid;
```

4.
Highlight the modified query and click Execute.

5.
Notice that the query retrieved 267 rows, which is two more rows than the previous query. If you observe the result, you will notice two rows with NULL in the columns from the inline TVF.

Task 4: Analyze the OUTER APPLY Operator
1.
Highlight the previous query in task 3. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste.

3.
Modify the T-SQL statement to search for a null productid. The query should look like this:

```
SELECT
c.custid, c.contactname, p.productid, p.productname, p.totalsalesamount
FROM Sales.Customers AS c
OUTER APPLY dbo.fnGetTop3ProductsForCustomer (c.custid) AS p
WHERE p.productid IS NULL;
```

4.
Highlight the modified query and click Execute.

5.

Notice that the query now retrieves the two rows that do not occur in the CROSS APPLY query in Task 2.

Task 5: Remove the Created Inline TVF
1.

Highlight the provided T-SQL statement after the task 5 description and click Execute.

Result: After this exercise, you should be able to use the CROSS APPLY and OUTER APPLY operators in your T-SQL statements.

Exercise 3: Writing Queries That Use the EXCEPT and INTERSECT Operators
Task 1: Write a SELECT Statement to Return All Customers Who Bought More than 20 Distinct Products
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20;
```
4.

Highlight the written query and click Execute.

Task 2: Write a SELECT Statement to Retrieve All Customers from the USA, Except Those Who Bought More than 20 Distinct Products
1.

In the query pane, type the following query after the task 2 description:

```
SELECT
custid
FROM Sales.Customers
WHERE country = 'USA'
EXCEPT
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20;
```

2.

Highlight the written query and click Execute.

Task 3: Write a SELECT Statement to Retrieve Customers Who Spent More than $10,000
1.

In the query pane, type the following query after the task 3 description:

```
SELECT
o.custid
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;
```
2.

Highlight the written query and click Execute.

Task 4: Write a SELECT Statement That Uses the EXCEPT and INTERSECT Operators
1.

Highlight the query from task 2. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste.

3.

Modify the first SELECT statement so that it selects all customers – not just those from the USA – and include the INTERSECT operator, adding the query from task 3. The query should look like this:

```
SELECT
c.custid
FROM Sales.Customers AS c
EXCEPT
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20
INTERSECT
```

```
SELECT
o.custid
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;
```

4.

Highlight the modified query and click Execute.

5.

Observe that the total number of rows is 59. In business terms, can you explain in which customers are part of the result? Because the INTERSECT operator is evaluated before the EXCEPT operator, the result consists of all customers, except those who bought more than 20 different products and spent more than $10,000.

Task 5: Change the Operator Precedence

1.

Highlight the previous query in task 4. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 5 description. On the toolbar, click Edit and then Paste.

3.

Modify the T-SQL statement by adding a set of parentheses around the first two SELECT statements. The query should look like this:

```
(
SELECT
c.custid
FROM Sales.Customers AS c
EXCEPT
SELECT
o.custid
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid
GROUP BY o.custid
HAVING COUNT(DISTINCT d.productid) > 20
)
INTERSECT
```

```
SELECT
o.custid
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;
```

4.

Highlight the provided T-SQL statement and click Execute.

5.

Observe that the total number of rows is nine. Is that different to the result of the query in task 4? Yes, because when you added the parentheses, the SQL Server engine first evaluated the EXCEPT operation, and then the INTERSECT operation. In business terms, this query retrieved all customers who did not buy more than 20 distinct products and who spent more than $10,000.

6.

What is the precedence among the set operators? SQL defines the following precedence among the set operations: INTERSECT precedes UNION and EXCEPT, while UNION and EXCEPT are considered equal. In a query that contains multiple set operations, INTERSECT operations are evaluated first, and then operations with the same precedence are evaluated, based on appearance order. Remember that set operations in parentheses precede all.

Result: After this exercise, you should have an understanding of how to use the EXCEPT and INTERSECT operators in T-SQL statements.

--end Mod 12

Lab Answer Key: Module 13: Using Window Ranking, Offset, and Aggregate Functions

Lab: Using Window Ranking, Offset, and Aggregate Functions
Exercise 1: Writing Queries That Use Ranking Functions
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab13\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 That Uses the ROW_NUMBER Function to Create a Calculated Column
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab13\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, type the following query after the task 1 description:

```
SELECT
orderid,
orderdate,
val,
ROW_NUMBER() OVER (ORDER BY orderdate) AS rowno
FROM Sales.OrderValues;
```
7.

Highlight the written query and click Execute.

Task 3: Add an Additional Column Using the RANK Function
1.

Highlight the previous query. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste.

3.

Modify the T-SQL statement by adding an additional calculated column. The query should look like this:

```
SELECT
orderid,
orderdate,
val,
ROW_NUMBER() OVER (ORDER BY orderdate) AS rowno,
RANK() OVER (ORDER BY orderdate) AS rankno
FROM Sales.OrderValues;
```

4.

Highlight the written query and click Execute.

5.

Observe the results. What is the difference between the RANK and ROW_NUMBER functions? The ROW_NUMBER function provides unique sequential integer values within the partition. The RANK function assigns the same ranking value to rows with the same values in the specified sort columns when the ORDER BY list is not unique. Also, the RANK function skips the next number if there is a tie in the ranking value.

Task 4: Write A SELECT Statement to Calculate a Rank, Partitioned by Customer and Ordered by the Order Value

1.

In the query pane, type the following query after the task 3 description:

```
SELECT
orderid,
orderdate,
custid,
val,
RANK() OVER (PARTITION BY custid ORDER BY val DESC) AS orderrankno FROM Sales.OrderValues;
```

2.

Highlight the written query and click Execute.

Task 5: Write a SELECT Statement to Rank Orders, Partitioned by Customer and Order Year, and Ordered by the Order Value

1.

In the query pane, type the following query after the task 4 description:

```
SELECT
custid,
val,
YEAR(orderdate) as orderyear,
RANK() OVER (PARTITION BY custid, YEAR(orderdate) ORDER BY val DESC) AS orderrankno
FROM Sales.OrderValues;
```

2.

Highlight the written query and click Execute.

Task 6: Filter Only Orders with the Top Two Ranks

1.

Highlight the previous query. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 5 description. On the toolbar, click Edit and then Paste.

3.

Modify the T-SQL statement to look like this:

```
SELECT
s.custid,
s.orderyear,
s.orderrankno,
s.val
FROM
(
SELECT
custid,
val,
YEAR(orderdate) as orderyear,
RANK() OVER (PARTITION BY custid, YEAR(orderdate) ORDER BY val DESC) AS orderrankno
FROM Sales.OrderValues
```

) AS s
WHERE s.orderrankno <= 2;
4.

Highlight the written query and click Execute.

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

Exercise 2: Writing Queries That Use Offset Functions
Task 1: Write a SELECT Statement to Retrieve the Next Row Using a Common Table Expression (CTE)
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
WITH OrderRows AS
(
SELECT
orderid,
orderdate,
ROW_NUMBER() OVER (ORDER BY orderdate, orderid) AS rowno,
val
FROM Sales.OrderValues
)
SELECT
o.orderid,
o.orderdate,
o.val,
o2.val as prevval,
o.val - o2.val as diffprev
FROM OrderRows AS o
LEFT OUTER JOIN OrderRows AS o2 ON o.rowno = o2.rowno + 1;
```
4.

Highlight the written query and click Execute.

Task 2: Add a Column to Display the Running Sales Total

1.

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

```
SELECT
orderid,
orderdate,
val,
LAG(val) OVER (ORDER BY orderdate, orderid) AS prevval,
val - LAG(val) OVER (ORDER BY orderdate, orderid) AS diffprev
FROM Sales.OrderValues;
```

2.

Highlight the written query and click Execute.

Task 3: Analyze the Sales Information for the Year 2007

1.

In the query pane, type the following query after the task 3 description:

```
WITH SalesMonth2007 AS
(
SELECT
MONTH(orderdate) AS monthno,
SUM(val) AS val
FROM Sales.OrderValues
WHERE orderdate >= '20070101' AND orderdate < '20080101'
GROUP BY MONTH(orderdate)
)
SELECT
monthno,
val,
(LAG(val, 1, 0) OVER (ORDER BY monthno) + LAG(val, 2, 0) OVER (ORDER BY monthno) + LAG(val, 3, 0)
OVER (ORDER BY monthno)) / 3 AS avglast3months,
val - FIRST_VALUE(val) OVER (ORDER BY monthno ROWS UNBOUNDED PRECEDING) AS diffjanuary,
LEAD(val) OVER (ORDER BY monthno) AS nextval
FROM SalesMonth2007;
```

2.

Highlight the written query and click Execute.

Result: After this exercise, you should be able to use the offset functions in your T-SQL statements.

Exercise 3: Writing Queries That Use Window Aggregate Functions

Task 1: Write a SELECT Statement to Display the Contribution of Each Customer's Order Compared to That Customer's Total Purchase

1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following query after the task 1 description:

```
SELECT
custid,
orderid,
orderdate,
val,
100. * val / SUM(val) OVER (PARTITION BY custid) AS percoftotalcust
FROM Sales.OrderValues
ORDER BY custid, percoftotalcust DESC;
```

4.

Highlight the written query and click Execute.

Task 2: Add a Column to Display the Running Sales Total

1.

Highlight the previous query. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste.

3.

Modify the T-SQL statement by adding an additional calculated column. The query should look like this:

```
SELECT
custid,
orderid,
orderdate,
val,
```

100. * val / SUM(val) OVER (PARTITION BY custid) AS percoftotalcust,
SUM(val) OVER (PARTITION BY custid
ORDER BY orderdate, orderid
ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) AS runval
FROM Sales.OrderValues;
4.

Highlight the written query and click Execute.

Task 3: Analyze the Year-to-Date Sales Amount and Average Sales Amount for the Last Three Months
1.

In the query pane, type the following query after the task 3 description:

```
WITH SalesMonth2007 AS
(
SELECT
MONTH(orderdate) AS monthno,
SUM(val) AS val
FROM Sales.OrderValues
WHERE orderdate >= '20070101' AND orderdate < '20080101'
GROUP BY MONTH(orderdate)
)
SELECT
monthno,
val,
AVG(val) OVER (ORDER BY monthno ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS
avglast3months,
SUM(val) OVER (ORDER BY monthno ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
AS ytdval
FROM SalesMonth2007;
```

2.

Highlight the written query and click Execute.

Result: After this exercise, you should have a basic understanding of how to use window aggregate functions in T-SQL statements.

--end Mod 13

Lab Answer Key: Module 14: Pivoting and Grouping Sets

Lab: Pivoting and Grouping Sets
Exercise 1: Writing Queries That Use the PIVOT Operator
Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab14\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 the Number of Customers for a Specific Customer Group
1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab14\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar.

6.
Highlight the following provided T-SQL code:

```
CREATE VIEW Sales.CustGroups AS
SELECT
custid,
CHOOSE(custid % 3 + 1, N'A', N'B', N'C') AS custgroup,
Country
FROM Sales.Customers;
```

7.
Click Execute. This code creates a view named Sales.CustGroups.

8.
In the query pane, type the following query after the provided T-SQL code:

```
SELECT
custid,
custgroup,
country
FROM Sales.CustGroups;
```

9.
Highlight the written query and click Execute.

10.
Modify the written T-SQL code by applying the PIVOT operator. The query should look like this:

```
SELECT
country,
p.A,
p.B,
p.C
FROM Sales.CustGroups
PIVOT (COUNT(custid) FOR custgroup IN (A, B, C)) AS p;
```

11.
Highlight the written query and click Execute.

Task 3: Specify the Grouping Element for the PIVOT Operator
1.
Highlight the following provided T-SQL code after the Task 2 description:

```
ALTER VIEW Sales.CustGroups AS
SELECT
custid,
CHOOSE(custid % 3 + 1, N'A', N'B', N'C') AS custgroup,
country,
city,
contactname
FROM Sales.Customers;
```

2.
Click Execute. This code modifies the view by adding two additional columns.

3.
Highlight the last query in task 1. On the toolbar, click Edit and then Copy.

4.
In the query window, click the line after the provided T-SQL code. On the toolbar, click Edit and then Paste. The query should look like this:

```
SELECT
country,
p.A,
p.B,
p.C
FROM Sales.CustGroups
PIVOT (COUNT(custid) FOR custgroup IN (A, B, C)) AS p;
```

5.
Highlight the copied query and click Execute.

6.
Observe the result. Is this result the same as that from the query in task 1? The result is not the same. More rows were returned after you modified the view.

7.
Modify the copied T-SQL statement to include additional columns from the view. The query should look like this:

```
SELECT
country,
city,
contactname,
p.A,
p.B,
p.C
FROM Sales.CustGroups
PIVOT (COUNT(custid) FOR custgroup IN (A, B, C)) AS p;
```

8.
Highlight the written query and click Execute.

9.
Notice that you received the same result as the previous query. Why did you get the same number of rows? The PIVOT operator assumes that all the columns except the aggregation and spreading elements are part of the grouping columns.

Task 4: Use a Common Table Expression (CTE) to Specify the Grouping Element for the PIVOT Operator
1.

In the query pane, type the following query after the task 3 description:

```
WITH PivotCustGroups AS
(
SELECT
custid,
country,
custgroup
FROM Sales.CustGroups
)
SELECT
country,
p.A,
p.B,
p.C
FROM PivotCustGroups
PIVOT (COUNT(custid) FOR custgroup IN (A, B, C)) AS p;
```

2.
Highlight the written query and click Execute.

3.
Observe the result. Is it the same as the result of the last query in task 1? Can you explain why? The result is the same. In this task, the CTE has provided three possible columns to the PIVOT operator. In task 1, the view also provided three columns to the PIVOT operator.

4.
Why do you think it is beneficial to use a CTE when using the PIVOT operator? When using the PIVOT operator, you cannot directly specify the grouping element since SQL Server automatically assumes that all columns should be used as grouping elements, with the exception of the spreading and aggregation elements. With a CTE, you can specify the exact columns and therefore control which columns to use for the grouping.

Task 5: Write a SELECT Statement to Retrieve the Total Sales Amount for Each Customer and Product Category

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

```
WITH SalesByCategory AS
(
SELECT
o.custid,
d.qty * d.unitprice AS salesvalue,
c.categoryname
FROM Sales.Orders AS o
INNER JOIN Sales.OrderDetails AS d ON o.orderid = d.orderid
INNER JOIN Production.Products AS p ON p.productid = d.productid
INNER JOIN Production.Categories AS c ON c.categoryid = p.categoryid
WHERE o.orderdate >= '20080101' AND o.orderdate < '20090101'
)
SELECT
custid,
p.Beverages,
p.Condiments,
p.Confections,
p.[Dairy Products],
p.[Grains/Cereals],
p.[Meat/Poultry],
p.Produce,
p.Seafood
FROM SalesByCategory
PIVOT (SUM(salesvalue) FOR categoryname
IN (Beverages, Condiments, Confections, [Dairy Products], [Grains/Cereals], [Meat/Poultry], Produce,
Seafood)) AS p;
```

2.
Highlight the written query and click Execute.

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
Task 1: Create and Query the Sales.PivotCustGroups View

1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

3.
Highlight the following provided T-SQL code:

```
CREATE VIEW Sales.PivotCustGroups AS
WITH PivotCustGroups AS
(
SELECT
custid,
country,
custgroup
FROM Sales.CustGroups
)
SELECT
country,
p.A,
p.B,
p.C
FROM PivotCustGroups
PIVOT (COUNT(custid) FOR custgroup IN (A, B, C)) AS p;
```

4.
Click Execute. This code creates a view named Sales.PivotCustGroups.

5.
In the query pane, type the following query after the provided T-SQL code:

```
SELECT
country, A, B, C
FROM Sales.PivotCustGroups;
```

6.
Highlight the written query and click Execute.

Task 2: Write a SELECT Statement to Retrieve a Row for Each Country and Customer Group
1.
In the query pane, type the following query after the Task 2 descriptions:

```
SELECT
custgroup,
country,
numberofcustomers
FROM Sales.PivotCustGroups
UNPIVOT (numberofcustomers FOR custgroup IN (A, B, C)) AS p;
```

2.
Highlight the written query and click Execute.

Task 3: Remove the Created Views
1.
Highlight the provided T-SQL statement after Task 3 description and click Execute.

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
Task 1: Write a SELECT Statement That Uses the GROUPING SETS Subclause to Return the Number of Customers for Different Grouping Sets
1.
In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECT
country,
city,
COUNT(custid) AS noofcustomers
FROM Sales.Customers
GROUP BY
GROUPING SETS
(
(country, city),
(country),
(city),
()
);
```
4.
Highlight the written query and click Execute.

Task 2: Write a SELECT Statement That Uses the CUBE Subclause to Retrieve Grouping Sets Based on Yearly, Monthly, and Daily Sales Values

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

```
SELECT
YEAR(orderdate) AS orderyear,
MONTH(orderdate) AS ordermonth,
DAY(orderdate) AS orderday,
SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY
CUBE (YEAR(orderdate), MONTH(orderdate), DAY(orderdate));
```

2.
Highlight the written query and click Execute.

Task 3: Write the Same SELECT Statement Using the ROLLUP Subclause

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

```
SELECT
YEAR(orderdate) AS orderyear,
MONTH(orderdate) AS ordermonth,
DAY(orderdate) AS orderday,
SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY
ROLLUP (YEAR(orderdate), MONTH(orderdate), DAY(orderdate));
```

2.
Highlight the written query and click Execute.

3.
Observe the result. What is the difference between the ROLLUP and CUBE subclauses of the GROUP BY clause? Like the CUBE subclause, the ROLLUP subclause provides an abbreviated way to define multiple grouping sets. However, unlike CUBE, ROLLUP doesn't produce all possible grouping sets that can be defined based on the input members—it produces a subset of those. ROLLUP assumes a hierarchy among the input members and produces all grouping sets that make sense, considering the hierarchy. In other words, while CUBE(a, b, c) produces all eight possible grouping sets out of the three input members, ROLLUP(a, b, c) produces only four grouping sets, assuming the hierarchy a>b>c. ROLLUP(a, b, c) is the equivalent of specifying GROUPING SETS((a, b, c), (a, b), (a), ()).

Which is the more appropriate subclause to use in this example? Since year, month, and day form a hierarchy, the ROLLUP clause is more suitable. There is probably not much interest in showing aggregates for a month irrespective of year, but the other way around is interesting.

Task 4: Analyze the Total Sales Value by Year and Month
1.
In the query pane, type the following query after the task 4 description:

```
SELECT
GROUPING_ID(YEAR(orderdate), MONTH(orderdate)) as groupid,
YEAR(orderdate) AS orderyear,
MONTH(orderdate) AS ordermonth,
SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY
ROLLUP (YEAR(orderdate), MONTH(orderdate))
ORDER BY groupid, orderyear, ordermonth;
```

2.
Highlight the written query and click Execute.

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

--end Mod 14

Lab Answer Key: Module 15: Executing Stored Procedures

Lab: Executing Stored Procedures
Exercise 1: Using the EXECUTE Statement to Invoke Stored Procedures
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab15\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: Create and Execute a Stored Procedure
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab15\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, expand the Queries node then double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

Highlight the following T-SQL code under the task 1 description:

```
CREATE PROCEDURE Sales.GetTopCustomers AS
SELECT TOP(10)
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC;
```

7.

Click Execute. You have created a stored procedure named Sales.GetTopCustomers.

8.

In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXECUTE Sales.GetTopCustomers;
```

9.

Highlight the written T-SQL code and click Execute. You have executed the stored procedure.

Task 3: Modify the Stored Procedure and Execute It

1.

Highlight the following T-SQL code after the task 2 description:

```
ALTER PROCEDURE Sales.GetTopCustomers AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

2.

Click Execute. You have modified the Sales.GetTopCustomers stored procedure.

3.

In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXECUTE Sales.GetTopCustomers;
```

4.

Highlight the written T-SQL code and click Execute. You have executed the modified stored procedure.

5.

Compare both the code and the result of the two versions of the stored procedure. What is the difference between them? In the modified version, the TOP option has been replaced with the OFFSET-FETCH option. Despite this change, the result is the same.

If some applications had been using the stored procedure in task 1, would they still work properly after the change you applied in task 2? Yes, since the result from the stored procedure is still the same. This demonstrates the huge benefit of using stored procedures as an additional layer between the database and the application/middle tier. Even if you change the underlying T-SQL code, the application would work properly without any changes. There are also other benefits of using stored procedures in terms of performance (for example, caching and reuse of plans) and security (for example, preventing SQL injections).

Result: After this exercise, you should be able to invoke a stored procedure using the EXECUTE statement.

Exercise 2: Passing Parameters to Stored Procedures
Task 1: Execute a Stored Procedure with a Parameter for Order Year
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

Highlight the following T-SQL code under the task 1 description:

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int
AS
SELECT
```

c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
4.

Click Execute. You have modified the Sales.GetTopCustomers stored procedure to accept the parameter @orderyear. Notice that the modified stored procedure uses a predicate in the WHERE clause that isn't a search argument. This predicate was used to keep things simple. The best practice is to avoid such filtering because it does not allow efficient use of indexing. A better approach would be to use the DATETIMEFROMPARTS function to provide a search argument for orderdate:

WHERE o.orderdate >= DATETIMEFROMPARTS(@orderyear, 1, 1, 0, 0, 0, 0)
AND o.orderdate < DATETIMEFROMPARTS(@orderyear + 1, 1, 1, 0, 0, 0, 0)
5.

In the query pane, type the following T-SQL code after the previous T-SQL code:

EXECUTE Sales.GetTopCustomers @orderyear = 2007;
Notice that you are passing the parameter by name as this is considered the best practice. There is also support for passing parameters by position. For example, the following EXECUTE statement would retrieve the same result as the T-SQL code you just typed:

EXECUTE Sales.GetTopCustomers 2007;
6.

Highlight the written T-SQL code and click Execute.

7.

After the previous T-SQL code, type the following T-SQL code to execute the stored procedure for the order year 2008:

EXECUTE Sales.GetTopCustomers @orderyear = 2008;
8.

Highlight the written T-SQL code and click Execute.

9.

After the previous T-SQL code, type the following T-SQL code to execute the stored procedure without specifying a parameter:

```
EXECUTE Sales.GetTopCustomers;
```

10.

Highlight the written T-SQL code and click Execute.

11.

Observe the error message:

Procedure or function 'GetTopCustomers' expects parameter '@orderyear', which was not supplied.
This error message is telling you that the @orderyear parameter was not supplied.

12.

Suppose that an application named MyCustomers is using the exercise 1 version of the stored procedure. Would the modification made to the stored procedure in this exercise impact the usability of the GetCustomerInfo application? Yes. The exercise 1 version of the stored procedure did not need a parameter, whereas the version in this exercise does not work without a parameter. To avoid problems, you can add a default parameter to the stored procedure. That way, the MyCustomers application does not have to be changed to support the @orderyear parameter.

Task 2: Modify the Stored Procedure to have a Default Value for the Parameter
1.

Highlight the following T-SQL code under the task 2 description:

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int = NULL
AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear OR @orderyear IS NULL
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

2.

Click Execute. You have modified the Sales.GetTopCustomers stored procedure to have a default value (NULL) for the @orderyear parameter. You have also included an additional logical expression to the WHERE clause.

3.

In the query pane, type the following T-SQL code after the previous one:

```
EXECUTE Sales.GetTopCustomers;
```
This code tests the modified stored procedure by executing it without specifying a parameter.

4.

Highlight the written query and click Execute.

5.

Observe the result. How do the changes to the stored procedure in task 2 influence the MyCustomers application and the design of future applications? The changes enable the MyCustomers application to use the modified stored procedure, and no changes need to be made to the application. The changes add new possibilities for future applications because the modified stored procedure accepts the order year as a parameter.

Task 3: Pass Multiple Parameters to the Stored Procedure
1.

Highlight the following T-SQL code under the task 3 description:

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int = NULL,
@n int = 10
AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear OR @orderyear IS NULL
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT @n ROWS ONLY;
```
2.

Click Execute. You have modified the Sales.GetTopCustomers stored procedure to have an additional parameter named @n. You can use this parameter to specify how many customers to retrieve. The default value is 10.

3.

After the previous T-SQL code, type the following T-SQL code to execute the modified stored procedure:

```
EXECUTE Sales.GetTopCustomers;
```
4.

Highlight the written query and click Execute.

5.

After the previous T-SQL code, type the following T-SQL code to retrieve the top five customers for the year 2008:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2008, @n = 5;
```
6.

Highlight the written query and click Execute.

7.

After the previous T-SQL code, type the following T-SQL code to retrieve the top 10 customers for the year 2007:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2007;
```
8.

Highlight the written query and click Execute.

9.

After the previous T-SQL code, type the following T-SQL code to retrieve the top 20 customers:

```
EXECUTE Sales.GetTopCustomers @n = 20;
```
10.

Highlight the written query and click Execute.

11.

Do the applications using the stored procedure need to be changed because another parameter was added? No changes need to be made to the application.

Task 4: Return the Result from a Stored Procedure Using the OUTPUT Clause

1.

Highlight the following T-SQL code under the task 4 description:

```
ALTER PROCEDURE Sales.GetTopCustomers
@customerpos int = 1,
@customername nvarchar(30) OUTPUT
AS
SET @customername = (
SELECT
c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY SUM(o.val) DESC
OFFSET @customerpos - 1 ROWS FETCH NEXT 1 ROW ONLY
);
```

2.

Click Execute.

3.

Find the following DECLARE statement in the provided code:

```
DECLARE @outcustomername nvarchar(30);
```

This statement declares a parameter named @outcustomername.

4.

After the DECLARE statement, add code that uses the OUTPUT clause to return the stored procedure's result as a variable named @outcustomername. Your code, together with the provided DECLARE statement, should look like this:

```
DECLARE @outcustomername nvarchar(30);
EXECUTE Sales.GetTopCustomers @customerpos = 1, @customername = @outcustomername OUTPUT;
SELECT @outcustomername AS customername;
```

5.

Highlight all three T-SQL statements and click Execute.

Result: After this exercise, you should know how to invoke stored procedures that have parameters.

Exercise 3: Executing System Stored Procedures
Task 1: Execute the Stored Procedure sys.sp_help
1.

In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

In the query pane, type the following T-SQL code after the task 1 description:

```
EXEC sys.sp_help;
```
4.

Highlight the written query and click Execute.

5.

In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXEC sys.sp_help N'Sales.Customers';
```
6.

Highlight the written query and click Execute.

Task 2: Execute the Stored Procedure sys.sp_helptext
1.

In the query pane, type the following T-SQL code after the task 2 description:

```
EXEC sys.sp_helptext N'Sales.GetTopCustomers';
```
2.

Highlight the written query and click Execute.

Task 3: Execute the Stored Procedure sys.sp_columns
1.

In the query pane, type the following T-SQL code after the task 3 description:

EXEC sys.sp_columns @table_name = N'Customers', @table_owner = N'Sales';
2.

Highlight the written query and click Execute.

Task 4: Drop the Created Stored Procedure
1.

Highlight the provided T-SQL statement under the task 4 description and click Execute.

Result: After this exercise, you should have a basic knowledge of invoking different system-stored procedures.

--end Mod 15

Lab Answer Key: Module 16: Programming with T-SQL

Lab: Programming with T-SQL
Exercise 1: Declaring Variables and Delimiting Batches
Task 1: Prepare the Lab Environment
1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab16\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: Declare a Variable and Retrieve the Value
1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab16\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, expand Queries, and then double-click the query 51 - Lab Exercise 1.sql (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.
In the query pane, type the following T-SQL code after the task 1 description:

```
DECLARE @num int = 5;
SELECT @num AS mynumber;
```

7.
Highlight the written T-SQL code and click Execute.

8.
In the query pane, type the following T-SQL code after the previous one:

```
DECLARE
@num1 int,
@num2 int;
SET @num1 = 4;
SET @num2 = 6;
SELECT @num1 + @num2 AS totalnum;
```

9.
Highlight the written T-SQL code and click Execute.

Task 3: Set the Variable Value Using a SELECT Statement
1.
In the query pane, type the following T-SQL code after the task 2 description:

```
DECLARE @empname nvarchar(30);
SET @empname = (SELECT firstname + N' ' + lastname FROM HR.Employees WHERE empid = 1);
SELECT @empname AS employee;
```
2.
Highlight the written T-SQL code and click Execute.

3.
Observe the result. What would happen if the SELECT statement was returning more than one row? You would get an error because the SET statement requires you to use a scalar subquery to pull data from a table. Remember that a scalar subquery fails at runtime if it returns more than one value.

Task 4: Use a Variable in the WHERE Clause

1.
In the query pane, type the following T-SQL code after the task 3 description:

```
DECLARE
@empname nvarchar(30),
@empid int;
SET @empid = 5;
SET @empname = (SELECT firstname + N' ' + lastname FROM HR.Employees WHERE empid = @empid);
SELECT @empname AS employee;
```

2.
Highlight the written T-SQL code and click Execute.

3.
Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab16\Solution\55 - Lab Exercise 1 - Task 3 Result.txt.

4.
Change the @empid variable's value from 5 to 2 and execute the modified T-SQL code to observe the changes.

Task 5: Use Variables with Batches
1.
Highlight the T-SQL code in task 4. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste.

3.
In the code you just copied, add the batch delimiter GO before this statement:

```
SELECT @empname AS employee;
```

4.
Make sure your T-SQL code looks like this:

```
DECLARE
@empname nvarchar(30),
@empid int;
SET @empid = 5;
SET @empname = (SELECT firstname + N' ' + lastname FROM HR.Employees WHERE empid = @empid)
GO
```

```
SELECT @empname AS employee;
```

5.
Highlight the written T-SQL code and click Execute.

6.
Observe the error:

```
Must declare the scalar variable "@empname".
```

7.
Can you explain why the batch delimiter caused an error? Variables are local to the batch in which they are defined. If you try to refer to a variable that was defined in another batch, you get an error saying that the variable was not defined. Also, keep in mind that GO is a client command, not a server T-SQL command.

Result: After this exercise, you should know how to declare and use variables in T-SQL code.

Exercise 2: Using Control-of-Flow Elements
Task 1: Write Basic Conditional Logic

1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

3.
In the query pane, type the following T-SQL code after the task 1 description:

```
DECLARE
@i int = 8,
@result nvarchar(20);
IF @i < 5
SET @result = N'Less than 5'
ELSE IF @i <= 10
SET @result = N'Between 5 and 10'
ELSE if @i > 10
SET @result = N'More than 10'
ELSE
SET @result = N'Unknown';
SELECT @result AS result;
```

4.
Highlight the written T-SQL code and click Execute.

5.
In the query pane, type the following T-SQL code:

```
DECLARE
@i int = 8,
@result nvarchar(20);
SET @result =
CASE
WHEN @i < 5 THEN
N'Less than 5'
WHEN @i <= 10 THEN
N'Between 5 and 10'
WHEN @i > 10 THEN
N'More than 10'
ELSE
N'Unknown'
END;
SELECT @result AS result;
```

This code uses a CASE expression and only one SET expression to get the same result as the previous T-SQL code. Remember to use a CASE expression when it is a matter of returning an expression. However, if you need to execute multiple statements, you cannot replace IF with CASE.

6.
Highlight the written T-SQL code and click Execute.

Task 2: Check the Employee Birthdate
1.
In the query pane, type the following T-SQL code after the task 2 description:

```
DECLARE
@birthdate date,
@cmpdate date;
SET @birthdate = (SELECT birthdate FROM HR.Employees WHERE empid = 5);
SET @cmpdate = '19700101';
IF @birthdate < @cmpdate
PRINT 'The person selected was born before January 1, 1970'
ELSE
PRINT 'The person selected was born on or after January 1, 1970';
```

2.
Highlight the written T-SQL code and click Execute.

Task 3: Create and Execute a Stored Procedure

1.
Highlight the following T-SQL code under the task 3 description:

```
CREATE PROCEDURE Sales.CheckPersonBirthDate
@empid int,
@cmpdate date
AS
DECLARE
@birthdate date;
SET @birthdate = (SELECT birthdate FROM HR.Employees WHERE empid = @empid);
IF @birthdate < @cmpdate
PRINT 'The person selected was born before ' + FORMAT(@cmpdate, 'MMMM d, yyyy', 'en-US');
ELSE
PRINT 'The person selected was born on or after ' + FORMAT(@cmpdate, 'MMMM d, yyyy', 'en-US');
```

2.
Click Execute. You have created a stored procedure named Sales.CheckPersonBirthDate. It has two parameters: @empid, which you use to specify an employee ID, and @cmpdate, which you use as a comparison date.

3.
In the query pane, type the following T-SQL code after the provided T-SQL code:

```
EXECUTE Sales.CheckPersonBirthDate @empid = 3, @cmpdate = '19900101';
```

4.
Highlight the written T-SQL code and click Execute.

Task 4: Execute a Loop Using the WHILE Statement
1.
In the query pane, type the following T-SQL code after the task 4 description:

```
DECLARE @i int = 1;
WHILE @i <= 10
BEGIN
PRINT @i;
SET @i = @i + 1;
END;
```

2.
Highlight the written T-SQL code and click Execute.

Task 5: Remove the Stored Procedure
1.
Highlight the following T-SQL code under the task 5 description:

```
DROP PROCEDURE Sales.CheckPersonBirthDate;
```
2.
Click Execute.

Result: After this exercise, you should know how to control the flow of the elements inside the T-SQL code.

Exercise 3: Using Variables in a Dynamic SQL Statement
Task 1: Write a Dynamic SQL Statement That Does Not Use a Parameter
1.
In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

3.
In the query pane, type the following T-SQL code after the task 1 description:

```
DECLARE @SQLstr nvarchar(200);
SET @SQLstr = N'SELECT empid, firstname, lastname FROM HR.Employees';
EXECUTE sys.sp_executesql @statement = @SQLstr;
```
4.
Highlight the written T-SQL code and click Execute.

Task 2: Write a Dynamic SQL Statement That Uses a Parameter
1.
Highlight the T-SQL code in task 1. On the toolbar, click Edit and then Copy.

2.
In the query window, click the line after the task 2 description. On the toolbar, click Edit and then Paste.

3.
Modify the T-SQL code to look like this:

```
DECLARE
@SQLstr nvarchar(200),
@SQLparam nvarchar(100);
SET @SQLstr = N'SELECT empid, firstname, lastname FROM HR.Employees WHERE empid = @empid';
SET @SQLparam = N'@empid int';
EXECUTE sys.sp_executesql @statement = @SQLstr, @params = @SQLparam, @empid = 5;
```

4.
Highlight the written T-SQL code and click Execute.

Result: After this exercise, you should have a basic knowledge of generating and invoking dynamic SQL statements.

Exercise 4: Using Synonyms
Task 1: Create and Use a Synonym for a Table
1.
In Solution Explorer, double-click the query 81 - Lab Exercise 4.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

3.
In the query pane, type the following T-SQL code after the task 1 description:

```
CREATE SYNONYM dbo.Person
FOR AdventureWorks.Person.Person;
```

4.
Highlight the written T-SQL code and click Execute. You have created a synonym named dbo.Person.

5.
In the query pane, type the following SELECT statement after the previous T-SQL code:

```
SELECT FirstName, LastName
FROM dbo.Person;
```

6.
Highlight the written query and click Execute.

Task 2: Drop the Synonym
1.
Highlight the following T-SQL code under the task 2 description:

```
DROP SYNONYM dbo.Person;
```

2.
Click Execute.

Result: After this exercise, you should know how to create and use a synonym.

--end Mod 16

Lab Answer Key: Module 17: Implementing Error Handling

Lab: Implementing Error Handling
Exercise 1: Redirecting Errors with TRY/CATCH
Task 1: Prepare the Lab Environment

1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab17\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 Basic TRY/CATCH Construct

1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab17\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, expand Queries and then double-click the 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.
Highlight the following SELECT statement under the task 1 description:

```
SELECT CAST(N'Some text' AS int);
```

7.
Click Execute. Notice the conversion error.

8.
Write a TRY/CATCH construct. Your T-SQL code should look like this:

```
BEGIN TRY
SELECT CAST(N'Some text' AS int);
END TRY
BEGIN CATCH
PRINT 'Error';
END CATCH;
```

9.
Highlight the written T-SQL code and click Execute.

Task 3: Display an Error Number and an Error Message
1.
Highlight the following T-SQL code under the task 2 description:

```
DECLARE @num varchar(20) = '0';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
END CATCH;
```

2.
Click Execute. Notice that you did not get an error because you used the TRY/CATCH construct.

3.
Modify the T-SQL code by adding two PRINT statements. The T-SQL code should look like this:

```
DECLARE @num varchar(20) = '0';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
PRINT 'Error Number: ' + CAST(ERROR_NUMBER() AS varchar(10));
PRINT 'Error Message: ' + ERROR_MESSAGE();
END CATCH;
```

4.
Highlight the T-SQL code and click Execute.

5.
Change the value of the @num variable to look like this:

```
DECLARE @num varchar(20) = 'A';
```

6.
Highlight the T-SQL code and click Execute. Notice that you get a different error number and message.

7.
Change the value of the @num variable to look like this:

```
DECLARE @num varchar(20) = ' 1000000000';
```

8.
Highlight the T-SQL code and click Execute. Notice that you get a different error number and message.

Task 4: Add Conditional Logic to a CATCH Block
1.
Modify the T-SQL code in Task 3 to look like this:

```
DECLARE @num varchar(20) = 'A';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
IF ERROR_NUMBER() IN (245, 8114)
BEGIN
PRINT 'Handling conversion error...'
END
ELSE
BEGIN
PRINT 'Handling non-conversion error...';
END;
PRINT 'Error Number: ' + CAST(ERROR_NUMBER() AS varchar(10));
PRINT 'Error Message: ' + ERROR_MESSAGE();
END CATCH;
```

2.
Highlight the written query and click Execute.

3.
Change the value of the @num variable to look like this:

```
DECLARE @num varchar(20) = '0';
```

4.
Highlight the T-SQL code and click Execute.

Task 5: Execute a Stored Procedure in the CATCH Block
1.
Highlight the following T-SQL code under the task 4 description:

```
CREATE PROCEDURE dbo.GetErrorInfo AS
PRINT 'Error Number: ' + CAST(ERROR_NUMBER() AS varchar(10));
PRINT 'Error Message: ' + ERROR_MESSAGE();
PRINT 'Error Severity: ' + CAST(ERROR_SEVERITY() AS varchar(10));
PRINT 'Error State: ' + CAST(ERROR_STATE() AS varchar(10));
 PRINT 'Error Line: ' + CAST(ERROR_LINE() AS varchar(10));
 PRINT 'Error Proc: ' + COALESCE(ERROR_PROCEDURE(), 'Not within procedure');
```

2.
Click Execute. You have created a stored procedure named dbo.GetErrorInfo.

3.
Modify the T-SQL code under TRY/CATCH to look like this:

```
DECLARE @num varchar(20) = '0';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
EXECUTE dbo.GetErrorInfo;
END CATCH;
```

4.
Highlight the written T-SQL code and click Execute.

Exercise 2: Using THROW to Pass an Error Message Back to a Client
Task 1: Re-Throw the Existing Error Back to a Client

1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

3.
Modify the T-SQL code under the Task 1 description to look like this:

```
DECLARE @num varchar(20) = '0';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
EXECUTE dbo.GetErrorInfo; THROW;
END CATCH;
```

4.
Highlight the written T-SQL code and click Execute.

Task 2: Add an Error Handling Routine

1.
Modify the T-SQL code under the Task 2 description to look like this:

```
DECLARE @num varchar(20) = 'A';
BEGIN TRY
PRINT 5. / CAST(@num AS numeric(10,4));
END TRY
BEGIN CATCH
EXECUTE dbo.GetErrorInfo;
IF ERROR_NUMBER() = 8134
BEGIN
PRINT 'Handling devision by zero...';
END
ELSE
BEGIN
PRINT 'Throwing original error';
THROW;
END;
END CATCH;
```

2.

Highlight the written T-SQL code and click Execute.

Task 3: Add a Different Error Handling Routine
1.

Find the following T-SQL code under the task 3 description:

```
DECLARE @msg AS varchar(2048);
SET @msg = 'You are doing the exercise for Module 17 on ' + FORMAT(CURRENT_TIMESTAMP, 'MMMM d, yyyy', 'en-US') + '. It''s not an error but it means that you are near the final module!';
```

2.

After the provided code, add a THROW statement. The completed T-SQL code should look like this:

```
DECLARE @msg AS varchar(2048);
SET @msg = 'You are doing the exercise for Module 17 on ' + FORMAT(CURRENT_TIMESTAMP, 'MMMM
d, yyyy', 'en-US') + '. It''s not an error but it means that you are near the final module!';
THROW 50001, @msg, 1;
```

3.

Highlight the written T-SQL code and click Execute.

Task 4: Remove the Stored Procedure

1.

Highlight the provided T-SQL statement under the task 4 description and click Execute.

Result: After this exercise, you should know how to throw an error to pass messages back to a client.

--end Mod 17

Lab Answer Key: Module 18: Implementing Transactions

Lab: Implementing Transactions
Exercise 1: Controlling Transactions with BEGIN, COMMIT, and ROLLBACK
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab18\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: Commit a Transaction
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab18\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard).

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

Modify the T-SQL code under the task 1 description by adding the BEGIN TRAN and COMMIT TRAN statements. Your T-SQL code should look like this:

```
BEGIN TRAN;
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101', '20110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 553344', 10);
COMMIT TRAN;
```

7.

Highlight the written T-SQL code and click Execute.

8.

In the query pane, type the following query after the previous T-SQL code:

```
SELECT empid, lastname, firstname
FROM HR.Employees
ORDER BY empid DESC;
```

9.

Highlight the written query and click Execute.

Task 3: Delete the Previously Inserted Rows from the HR.Employees Table

1.

Highlight the following T-SQL code under the task 2 description:

```
DELETE HR.Employees
WHERE empid IN (10, 11);
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```

2.

Click Execute.

Task 4: Open a Transaction and Use the ROLLBACK Statement

1.

Modify the T-SQL code under the task 3 description by adding the BEGIN TRAN statement. Your T-SQL code should look like this:

```
BEGIN TRAN;
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101', '20110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 553344', 10);
```

2.

Highlight the written T-SQL code and click Execute.

3.

In the query pane, type the following query after the previous T-SQL code:

```
SELECT empid, lastname, firstname
FROM HR.Employees
ORDER BY empid DESC;
```

4.

Highlight the written query and click Execute.

5.

In the query pane, type the following statement after the SELECT statement:

```
ROLLBACK TRAN;
```

6.

Highlight the written statement and click Execute.

7.

Again, highlight the SELECT statement shown in step 3 and click Execute.

Task 5: Clear the Modifications Against the HR.Employees Table

1.

Highlight the following T-SQL code after the task 4 description:

```
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```

2.

Click Execute.

Result: After this exercise, you should be able to control a transaction using the BEGIN TRAN, COMMIT, and ROLLBACK statements.

Exercise 2: Adding Error Handling to a CATCH Block
Task 1: Observe the Provided T-SQL Code
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

Highlight only the following SELECT statement under the task 1 description:

```
SELECT empid, lastname, firstname
FROM HR.Employees
ORDER BY empid DESC;
```

4.

Click Execute.

5.

In the provided T-SQL code, highlight the code between the BEGIN TRAN and COMMIT TRAN statements. Your highlighted T-SQL code should look like this:

```
BEGIN TRAN;
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101', '10110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 553344', 10);
```

COMMIT TRAN;
6.

Click Execute. Notice there is a conversion error in the second INSERT statement.

7.

Again, highlight the SELECT statement shown in step 3 and click Execute.

Task 2: Delete the Previously Inserted Row in the HR.Employees Table
1.

Highlight the following T-SQL code under the task 2 description:

```
DELETE HR.Employees
WHERE empid IN (10, 11);
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```
2.

Click Execute.

Task 3: Abort Both INSERT Statements If an Error Occurs
1.

Modify the T-SQL code under the task 3 description to look like this:

```
BEGIN TRY
BEGIN TRAN;
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate, hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101', '10110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 553344', 10);
PRINT 'Commit the transaction...';
COMMIT TRAN;
END TRY
BEGIN CATCH
IF @@TRANCOUNT > 0
BEGIN
PRINT 'Rollback the transaction...';
ROLLBACK TRAN;
```

```
END
END CATCH;
```
2.

Highlight the modified T-SQL code and click Execute.

3.

In the query pane, type the following query after the modified T-SQL code:

```
SELECT empid, lastname, firstname
FROM HR.Employees ORDER BY empid DESC;
```
4.

Highlight the written query and click Execute.

Task 4: Clear the Modifications Against the HR.Employees Table

1.

Highlight the following T-SQL code under the task 4 description:

```
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```
2.

Click Execute.

Result: After this exercise, you should have a basic understanding of how to control a transaction inside a TRY/CATCH block to efficiently handle possible errors.

--end Mod 18

Lab Answer Key: Module 19: Appendix A Improving Query Performance

Lab: Improving Query Performance
Exercise 1: Viewing Query Execution Plans
Task 1: Prepare the Lab Environment
1.

Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.

In the D:\Labfiles\Lab19\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: Create and populate the sample table Sales.TempOrders
1.

Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.

On the File menu, click Open and click Project/Solution.

3.

In the Open Project window, open the project D:\Labfiles\Lab19\Starter\Project\Project.ssmssln.

4.

In Solution Explorer, double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.

When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.

In the query pane, highlight the T-SQL code after the task 1 description and click Execute.

Task 3: Show estimated and actual execution plans
1.

In the query pane, type the following query after the task 2 description:

```
SELECT orderid, custid, orderdate
FROM Sales.TempOrders;
```
2.

Highlight the written query and click Display Estimated Execution Plan.

3.

In the Results pane, click the Execution plan tab. Hover your mouse pointer over the Table Scan operator and look at the properties displayed in the yellow tooltip box.

4.

Position your mouse pointer over the arrow between the SELECT operator and the Table Scan operator in the execution plan. You should see three properties: Estimated Number of Rows, Estimated Data Size, and Estimated Row Size.

5.

Right-click the SELECT operator and click Properties in the context menu.

6.

Click the SELECT operator.

7.

On the toolbar, click Include Actual Execution Plan.

8.

Highlight the written query and click Execute.

9.

In the Results pane, click the Execution plan tab and observe the actual execution plan.

Task 4: Analyze the execution plan of another SELECT statement
1.

Highlight the previous query in task 2. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste.

3.

In the query pane, alter the copied query to look like this:

```
SELECT TOP (1) orderid, custid, orderdate
FROM Sales.TempOrders;
```
4.

Highlight the altered query and click Display Estimated Execution Plan.

5.

Compare this task's execution plan with the one for the previous task. Which operator is new? The TOP operator is new.

Task 5: Graphically compare two execution plans
1.

Highlight the query in task 2. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste.

3.

Highlight the query in task 3. On the toolbar, click Edit and then Copy.

4.

In the query window, click the line after the copied SELECT statement. On the toolbar, click Edit and then Paste.

5.

Highlight both SELECT statements and click Display Estimated Execution Plan.

6.

In the toolbar, click Include Actual Execution Plan.

7.

Highlight both SELECT statements and click Execute.

Result: After this exercise, you should be able to display estimated and actual execution plans.

Exercise 2: Viewing Index Usage and Using SET STATISTICS Statements
Task 1: Create a clustered index and write a SELECT statement
1.

In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.

When the query window opens, highlight the statement USE TSQL; and click Execute.

3.

Highlight the provided T-SQL code after the task 1 description and click Execute.

4.

In the query pane, type the following query after the provided T-SQL code:

```
SELECT orderid, custid, orderdate
FROM Sales.TempOrders
WHERE YEAR(orderdate) = 2007 AND MONTH(orderdate) = 6;
```
5.

Highlight the written query and click Execute.

6.

Highlight the written query and click Display Estimated Execution Plan.

Task 2: Enable I/O statistics to observe the number of needed reads
1.

In the query pane, type the following T-SQL statement after the task 2 description:

```
SET STATISTICS IO ON;
```

2.

Highlight the written statement and click Execute.

3.

Highlight the query in task 1. On the toolbar, click Edit and then Copy.

4.

In the query window, click the line after the written T-SQL statement. On the toolbar, click Edit and then Paste.

5.

Highlight the copied SELECT statement and click Execute.

6.

In the Results pane, click the Messages tab and observe the number of logical reads.

Task 3: Modify the SELECT statement to use a search argument in the WHERE clause

1.

Highlight the SELECT statement in task 1. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 3 description. On the toolbar, click Edit and then Paste.

3.

Modify the SELECT statement to look like this:

```
SELECT orderid, custid, orderdate
FROM Sales.TempOrders
WHERE orderdate >= '20070601' AND orderdate < '20070701';
```

4.

Highlight the modified query and click Execute.

5.

Highlight the modified query and click Display Estimated Execution Plan.

Task 4: Compare both SELECT statements
1.

Highlight the SELECT statement in task 1. On the toolbar, click Edit and then Copy.

2.

In the query window, click the line after the task 4 description. On the toolbar, click Edit and then Paste.

3.

Highlight the query in task 3. On the toolbar, click Edit and then Copy.

4.

In the query window, click the line after the copied SELECT statement. On the toolbar, click Edit and then Paste.

5.

Highlight both SELECT statements and click Execute.

6.

Highlight both SELECT statements and click Display Estimated Execution Plan.

7.

Compare the execution plans for the two queries. Why is the SELECT statement from task 3 so much faster? This SELECT statement efficiently uses the created clustered index and does a clustered index seek operation. The SELECT statement from task 1 does a clustered index scan (that is, a table scan).

Task 5: Remove the created table and disable I/O statistics
1.
Highlight the provided T-SQL code after the Task 5 description and click Execute.

Result: After this exercise, you should have a basic understanding of how to enable SET STATISTICS options. Remember to invest time in understanding indexes so that you can write efficient queries.
--end Mod 19

Lab Answer Key: Module 20: Appendix B Querying SQL Server Metadata

Lab: Querying SQL Server Metadata
Exercise 1: Querying System Catalog Views
Task 1: Prepare the Lab Environment

1.
Ensure that the 20461C-MIA-DC and 20461C-MIA-SQL virtual machines are both running, and then log on to 20461C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.

2.
In the D:\Labfiles\Lab20\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 databases
1.
Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.

2.
On the File menu, click Open and click Project/Solution.

3.
In the Open Project window, open the project D:\Labfiles\Lab20\Starter\Project\Project.ssmssln.

4.
In Solution Explorer, expand Queries and double-click the query 51 - Lab Exercise 1.sql. (If Solution Explorer is not visible, select Solution Explorer on the View menu or press Ctrl+Alt+L on the keyboard.)

5.
When the query window opens, highlight the statement USE TSQL; and click Execute on the toolbar (or press F5 on the keyboard).

6.
In the query pane, type the following query after the task 1 description:

```
SELECT name, dbid, crdate
FROM sys.sysdatabases;
```

7.
Highlight the written query and click Execute. Observe that the query retrieves a row for each database.

Task 3: Write a SELECT statement to retrieve all user-defined tables in the TSQL database

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

```
SELECT
object_id, name, schema_id, type, type_desc, create_date, modify_date
FROM sys.objects;
```

2.
Highlight the written query and click Execute.

3.
Highlight the previous query. On the toolbar, click Edit and then Copy.

4.
In the query window, click the line after the written T-SQL statement. On the toolbar, click Edit and then Paste.

5.
Modify the T-SQL statement to retrieve all distinct values for the columns type and type_desc. The query should look like this:

```
SELECT DISTINCT
type, type_desc
FROM sys.objects
ORDER BY type_desc;
```

6.
Highlight the written query and click Execute.

7.
Highlight the first query. On the toolbar, click Edit and then Copy.

8.
In the query window, click the line after the written T-SQL statement. On the toolbar, click Edit and then Paste.

9.
Modify the T-SQL statement to filter only user-defined tables. The query should look like this:

```
SELECT
object_id, name, schema_id, type, type_desc, create_date, modify_date
FROM sys.objects
```

WHERE type = N'U';

10.
Highlight the written query and click Execute.

Task 4: Use a different approach to retrieve all user-defined tables in the TSQL database
1.
In the query pane, type the following query after the task 3 description:

SELECT
object_id, name, SCHEMA_NAME(schema_id) AS schemaname, type, type_desc, create_date, modify_date
FROM sys.tables;
2.
Highlight the written query and click Execute.

3.
In the query pane, type the following query after the previous one:

SELECT
object_id, name, SCHEMA_NAME(schema_id) AS schemaname, type, type_desc, create_date, modify_date
FROM sys.views;

4.
Highlight the written query and click Execute.

Task 5: Write a SELECT statement to retrieve all columns from the Sales.Customers table

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

SELECT
c.name AS columnname, c.column_id, c.system_type_id, c.max_length, c.precision, c.scale, c.collation_name
FROM sys.columns AS c
WHERE object_id = OBJECT_ID('Sales.Customers')
ORDER BY c.column_id;

2.
Highlight the written query and click Execute.

Result: After this exercise, you should be able to retrieve some system information from the system catalog views.

Exercise 2: Querying System Functions
Task 1: Write a SELECT statement to retrieve the current database name

1.
In Solution Explorer, double-click the query 61 - Lab Exercise 2.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

```
SELECT
DB_ID() AS databaseid,
DB_NAME(DB_ID()) AS databasename,
USER_NAME() as currusername;
```

4.
Highlight the written query and click Execute.

Task 2: Write a SELECT statement to retrieve the object name and schema name

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

```
SELECT
name,
OBJECT_NAME(object_id) AS tablename,
OBJECT_SCHEMA_NAME(object_id) AS schemaname
FROM sys.columns;
```
2.
Highlight the written query and click Execute.

Task 3: Write a SELECT statement to retrieve all the columns from the user-defined tables that contain the word "name" in the column name
1.
In the query pane, type the following query after the task 3 description:

```
SELECT
c.name AS columnname,
OBJECT_NAME (c.object_id) AS tablename,
OBJECT_SCHEMA_NAME(c.object_id) AS schemaname
FROM sys.columns AS c
```

WHERE
c.name LIKE N'%name%'
AND OBJECTPROPERTY(c.object_id, N'IsUserTable') = 1;
2.
Highlight the written query and click Execute.

Task 4: Retrieve the view definition
1.
In the query pane, type the following query after the task 4 description:

SELECT OBJECT_DEFINITION(OBJECT_ID(N'Sales.CustOrders'));
2.
Highlight the written query and click Execute.

Result: After this exercise, you should know how to use different system functions.

Exercise 3: Querying System Dynamic Management Views
Task 1: Write a SELECT statement to return all current sessions
1.
In Solution Explorer, double-click the query 71 - Lab Exercise 3.sql.

2.
When the query window opens, highlight the statement USE TSQL; and click Execute.

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

SELECT
session_id, login_time, host_name, language, date_format
FROM
sys.dm_exec_sessions;
4.
Highlight the written query and click Execute.

Task 2: Execute the provided T-SQL statement
1.
Highlight the following T-SQL code under the task 2 description:

SELECT
cpu_count AS 'Logical CPU Count',
hyperthread_ratio AS 'Hyperthread Ratio',
cpu_count/hyperthread_ratio As 'Physical CPU Count',
physical_memory_kb/1024 AS 'Physical Memory (MB)',
SQL Server _start_time AS 'Last SQL Start'

FROM sys.dm_os_sys_info;
2.

Click Execute.

Task 3: Write a SELECT statement to retrieve the current memory information
1.
In the query pane, type the following query after the task 3 description:

```
SELECT
total_physical_memory_kb,
available_physical_memory_kb,
total_page_file_kb,
available_page_file_kb,
system_memory_state_desc
FROM sys.dm_os_sys_memory;
```

2.
Highlight the written query and click Execute.
Result: After this exercise, you should have an understanding of how to write queries against the system DMVs.
--end Mod 20