Module 16: Programming with T-SQL

Contents:

Module Overview

In addition to the data retrieval and manipulation statements you have learned about in this course,

T-SQL provides some basic programming features, such as variables, control-of-flow elements, and conditional execution. In this module, you will learn how to enhance your T-SQL code with programming elements.

Objectives

After completing this module, you will be able to:

- Describe the language elements of T-SQL used for simple programming tasks.
- Describe batches and how they are handled by SQL Server.
- Declare and assign variables and synonyms.
- Use IF and WHILE blocks to control program flow.

Lesson 1: T-SQL Programming Elements

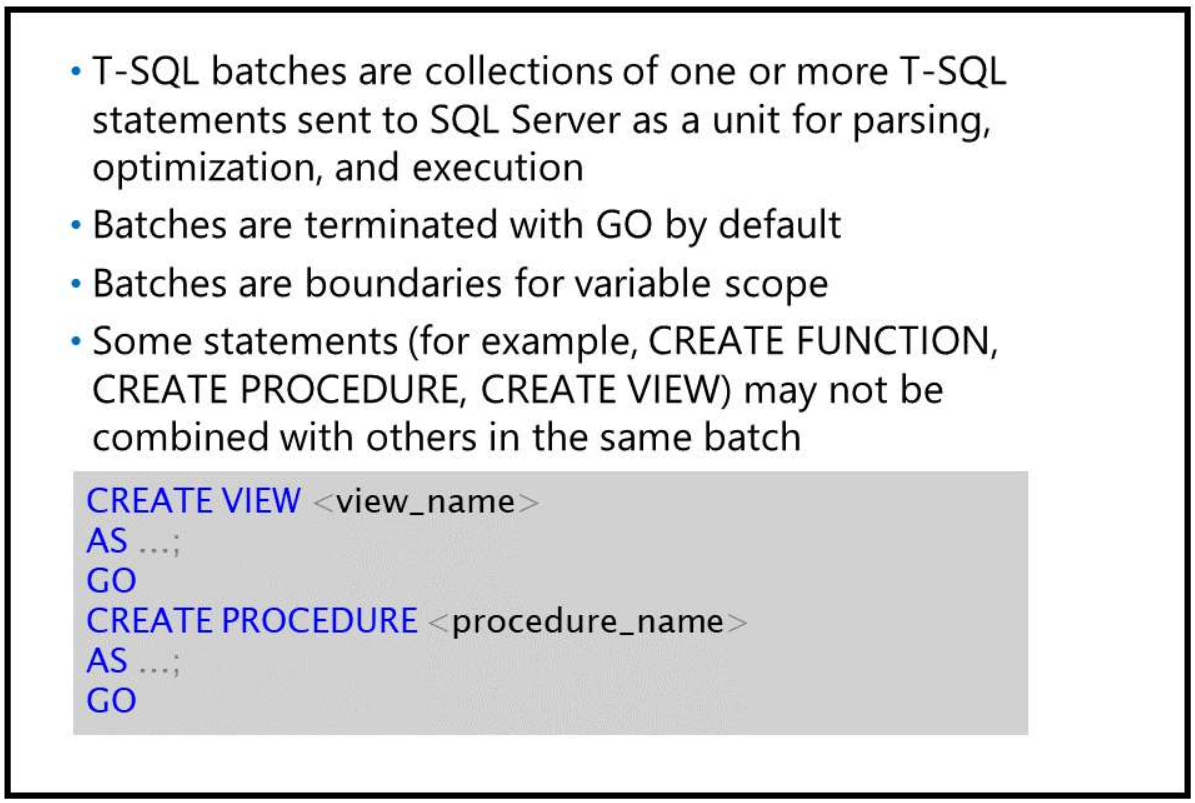
With a few exceptions, most of your work with T-SQL in this course so far has focused on single-statement structures, such as SELECT statements. As you move from executing code objects to creating them, you will need to understand how multiple statements interact with the server on execution. You will also need to be able to temporarily store values. For example, you might need to temporarily store values that will be used as parameters in stored procedures. Finally, you might want to create aliases, or pointers, to objects so that you can reference them by a different name or from a different location than where they are defined. This lesson will cover each of these topics.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how Microsoft® SQL Server® treats collections of statements as batches.
- Create and submit batches of T-SQL code for execution by SQL Server.
- Describe how SQL Server stores temporary objects as variables.
- Write code that declares and assigns variables.
- Create and invoke synonyms.

Introducing T-SQL Batches



T-SQL batches are collections of one or more T-SQL statements that are submitted to SQL Server by a client as a single unit. SQL Server operates on all the statements in a batch at the same time when parsing, optimizing, and executing the code.

If you are a report writer tasked primarily with writing SELECT statements and not procedures, it is still important to understand batch boundaries, because they will affect your work with variables and parameters in stored procedures and other routines. As you will see, you must declare a variable in the same batch in which it is referenced. It is important, therefore, to recognize what is contained in a batch.

Batches are delimited by the client application—how you mark the end of a batch will depend on the settings of your client. For example, the default batch terminator in SQL Server Management Studio (SSMS) is the keyword GO. This is not a T-SQL keyword, but is one recognized by SSMS to indicate the end of a batch.

When working with T-SQL batches, there are two important considerations to keep in mind:

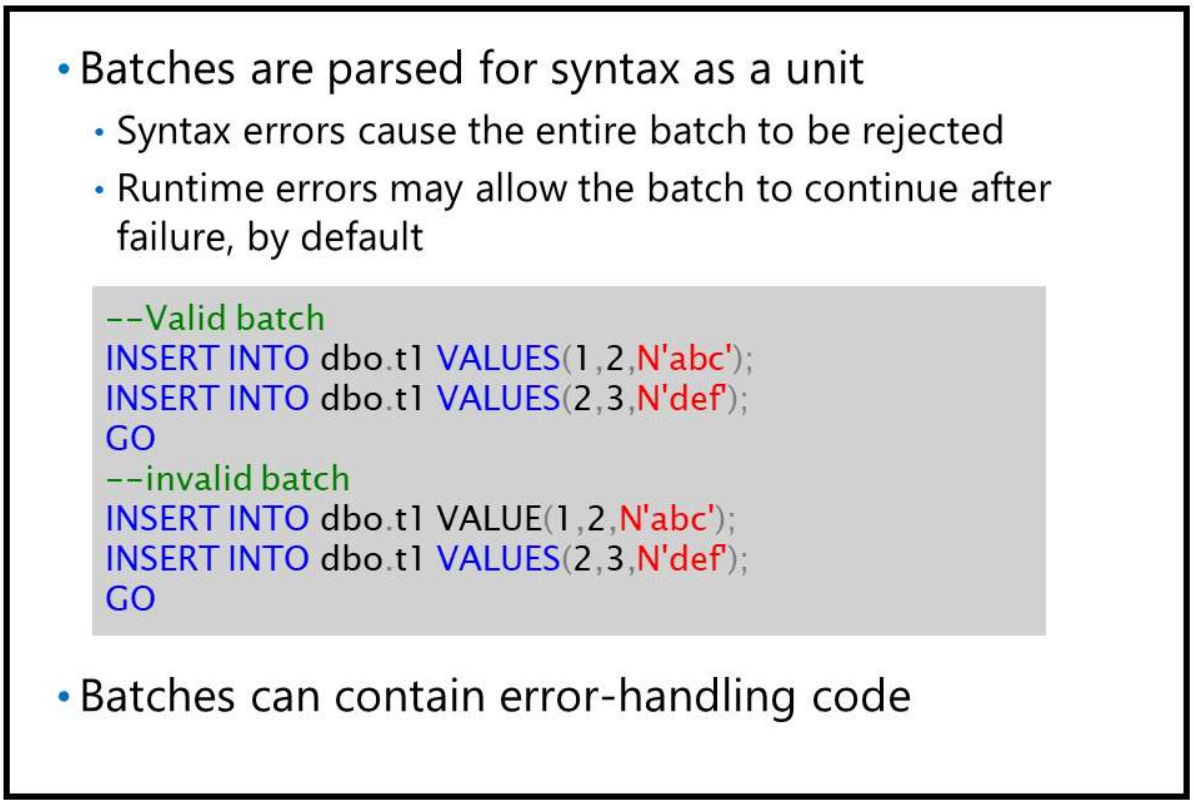
- Batches are boundaries for variable scope, which means that a variable defined in one batch may only be referenced by other code in the same batch.

- Some statements, typically data definition statements such as CREATE VIEW, may not be combined with others in the same batch. There is a complete list in the SQL Server Technical Documentation, where you will also find additional reading.

Batches

http://go.microsoft.com/fwlink/?LinkID=402796

Working with Batches



As you have seen, batches are collections of T-SQL statements submitted as a unit to SQL Server for parsing, optimization, and execution. Understanding how batches are parsed will be useful in identifying error messages and behavior.

When a batch is submitted by a client (such as when you press the Execute button in SSMS), the batch is parsed for syntax errors by the SQL Server engine. Any errors found will cause the entire batch to be rejected; there will be no partial execution of statements within the batch.

If the batch passes the syntax check, then SQL Server proceeds with additional steps—resolving object names, checking permissions, and optimizing the code for execution. Once this process completes and execution begins, statements succeed or fail individually. This is an important contrast to syntax checking. If a runtime error occurs on one line, the next line may be executed, unless you've added error handling to the code.

> **Note:** Error handling will be covered in a later module.

Batch With Error

```
INSERT INTO dbo.t1 VALUE(1,2,N'abc');
INSERT INTO dbo.t1 VALUES(2,3,N'def');
GO
```

Upon submitting the batch, the following error is returned:

```
Msg 102, Level 15, State 1, Line 1
Incorrect syntax near 'VALUE'.
```

The error occurred in line 1, but the entire batch is rejected, and execution does not continue with line 2. Even if the lines were reversed and the syntax error occurred in the second line, the first line would not be executed because the entire batch would be rejected.

Introducing T-SQL Variables

- Variables are objects that allow storage of a value for use later in the same batch
- Variables are defined with the DECLARE keyword
  - In SQL Server 2008 and later, variables can be declared and initialized in the same statement
- Variables are always local to the batch in which they're declared and go out of scope when the batch ends

```
--Declare and initialize variables
DECLARE @numrows INT = 3, @catid INT = 2;
--Use variables to pass parameters to procedure
EXEC Production.ProdsByCategory
        @numrows = @numrows, @catid = @catid;
GO
```

In T-SQL, as with other programming languages, variables are objects that allow temporary storage of a value for later use. You have already encountered variables in this course, using them to pass parameter values to stored procedures and functions.

In T-SQL, variables must be declared before they can be used. They may be assigned a value, or initialized, when they are declared. Declaring a variable includes providing a name and a data type, as shown below.

As you have previously learned, variables must be declared in the same batch in which they are referenced. In other words, all T-SQL variables are local in scope to the batch, both in visibility and lifetime. Only other statements in the same batch can see a variable declared in the batch. A variable is automatically destroyed when the batch ends.

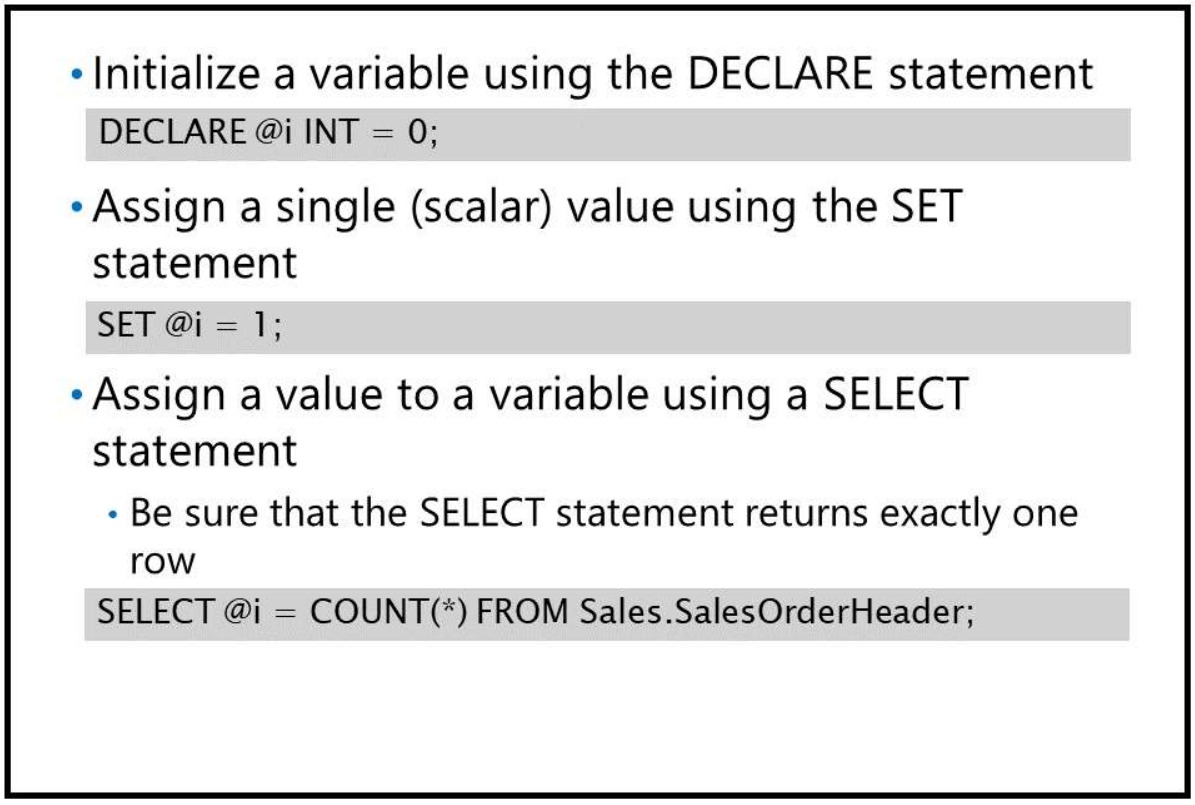
Using Variables

```
--Declare and initialize the variables.
DECLARE @numrows INT = 3, @catid INT = 2;
--Use variables to pass the parameters to the procedure.
EXEC Production.ProdsByCategory
        @numrows = @numrows, @catid = @catid;
GO
```

Variables (Transact-SQL)

http://aka.ms/Jub7kl

Working with Variables



Once you have declared a variable, you must initialize it, or assign it a value. You can do that in three ways:

- In SQL Server 2008 or later, you may initialize a variable using the DECLARE statement.
- In any version of SQL Server, you may assign a single (scalar) value using the SET statement.
- In any version of SQL Server, you can assign a value to a variable using a SELECT statement. Be sure that the

SELECT statement returns exactly one row. An empty result will leave the variable with its original value; more than one result will cause an error.

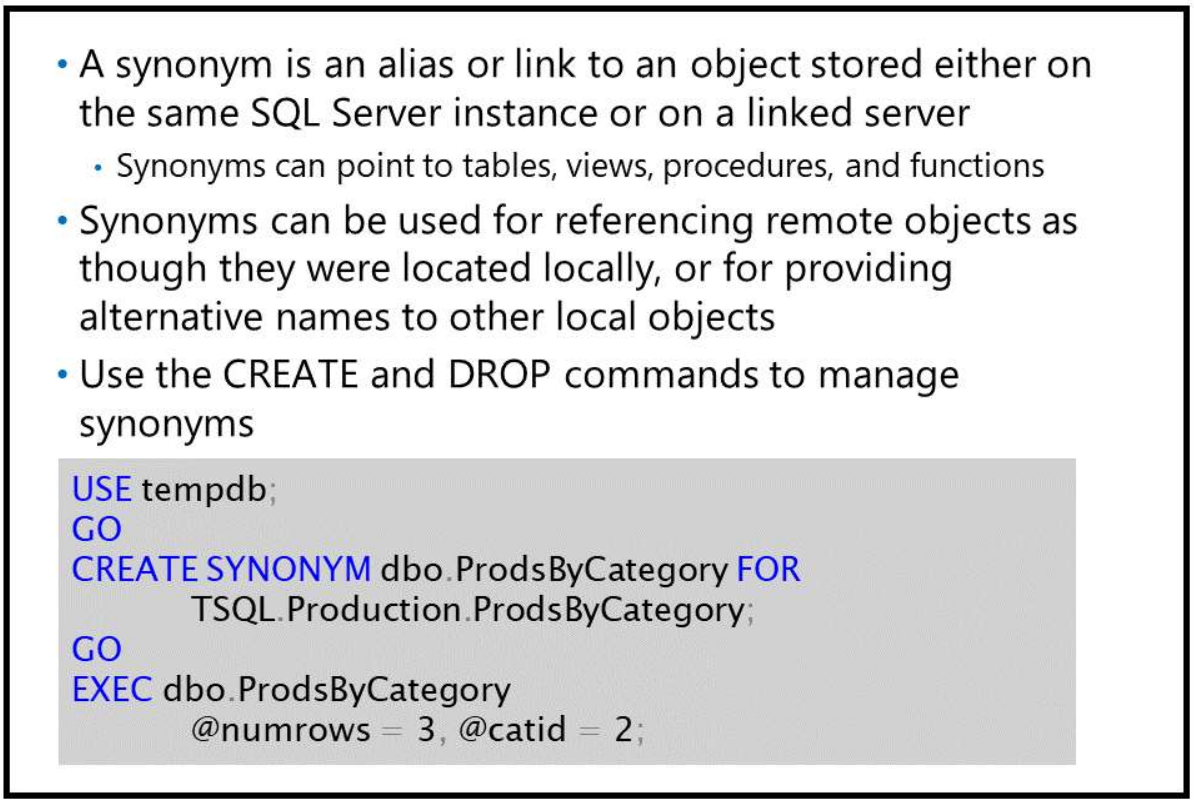
Declaring and Assigning Values to Variables

```
DECLARE @var1 AS INT = 99;
DECLARE @var2 AS NVARCHAR(255);
SET @var2 = N'string';
DECLARE @var3 AS NVARCHAR(20);
SELECT @var3 = lastname FROM HR.Employees WHERE empid=1;
SELECT @var1 AS var1, @var2 AS var2, @var3 AS var3;
GO
```

The results are:

```
var1 var2   var3
---- ------ ----
99   string Davis
```

Working with Synonyms



In SQL Server, synonyms provide a method for creating a link, or alias, to an object stored in the same database or even on another instance of SQL Server. Objects that might have synonyms defined for them include tables, views,

stored procedures, and user-defined functions.

Synonyms can be used to make a remote object appear local or to provide an alternative name for a local object. For example, synonyms can be used to provide an abstraction layer between client code and the actual database objects used by the code. The code references objects by their aliases, regardless of the object's actual name.

> **Note:** You can create a synonym which points to an object that does not yet exist. This is called deferred name resolution. The SQL Server engine will not check for the existence of the actual object until the synonym is used at runtime.

Managing Synonyms

```
CREATE SYNONYM dbo.ProdsByCategory FOR TSQL.Production.ProdsByCategory;
GO
EXEC dbo.ProdsByCategory @numrows = 3, @catid = 2;
```

To create a synonym, you must have CREATE SYNONYM permission as well as permission to alter the schema in which the synonym will be stored.

For more information, see *Using Synonyms (Database Engine)* in the SQL Server Technical Documentation:

Using Synonyms (Database Engine)

http://go.microsoft.com/fwlink/?LinkID=402798

Demonstration: T-SQL Programming Elements

In this demonstration, you will see how to control batch execution and variable usage.

Demonstration Steps

Control Batch Execution and Variable Usage

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod16\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, and then press Enter.
5. When the script completes, press any key to continue.
6. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod16\Demo** folder.
8. In Solution Explorer, expand **Queries**, and then double-click the query **11 - Demonstration A.sql** script file.

9. Select the code under the comment **Step 1**, and then click **Execute**.
10. Select the code under the comment **Step 2**, and then click **Execute**.
11. Select the code under the comment **-- Create a proc to search for category**, and then click **Execute**.
12. Select the code under the comment **-- Set up table for batch demos**, and then click **Execute**.
13. Select the code under the comment **Step 3**, and then click **Execute**.
14. Select the code under the comment **-- Show that the batch was successful**, and then click **Execute**.
15. Select the code under the comment **Step 4**, and then click **Execute**.
16. Select the code under the comment **Step 5**, and then click **Execute**. Note the error message.
17. Select the code under the comment **--Show that no rows were inserted**, and then click **Execute**.
18. Select the code under the comment **Step 6**, and then click **Execute**.
19. Select the code under the comment **--Run the following batch in its entirety to show the choices**, and then click **Execute**.
20. Select the code under the comment **Step 7**, and then click **Execute**.
21. Select the code under the comment **-- Declare a parameter to search for category**, and then click **Execute**.
22. Select the code under the comment **-- Test it locally**, and then click **Execute**.
23. Select the code under the comment **Step 8**, and then click **Execute**.
24. Select the code under the comment **Step 9**, and then click **Execute**.
25. Select the code under the comment **Step 10**, and then click **Execute**.
26. Select the code under the comment **Step 11**, and then click **Execute.**
27. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have the following T-SQL script:

```
INSERT INTO HumanResources.PossibleSkills (SkillName, Category, Credit)
  VALUES('Database Administration', 'IT Professional', 5);
INSERT INTO HumanResources.PossibleSkills (SkillName, Category, Credit)
  VALUES('C#.NET', 'Developer', 4);
INSERT INTO HumanResources.PossibleSkills (SkillName, Category, Credit)
  VALUES('Project Management', 'Management', 'Two');
GO
```

The script generates an error on the third INSERT statement. How many new rows do you expect to find in the PossibleSkills table after this error?

Show solution Reset

Two new rows. There is no syntax error in this batch so SQL can begin to execute statements in the batch. Two correct INSERT statements are executed before the error arises.

Lesson 2: Controlling Program Flow

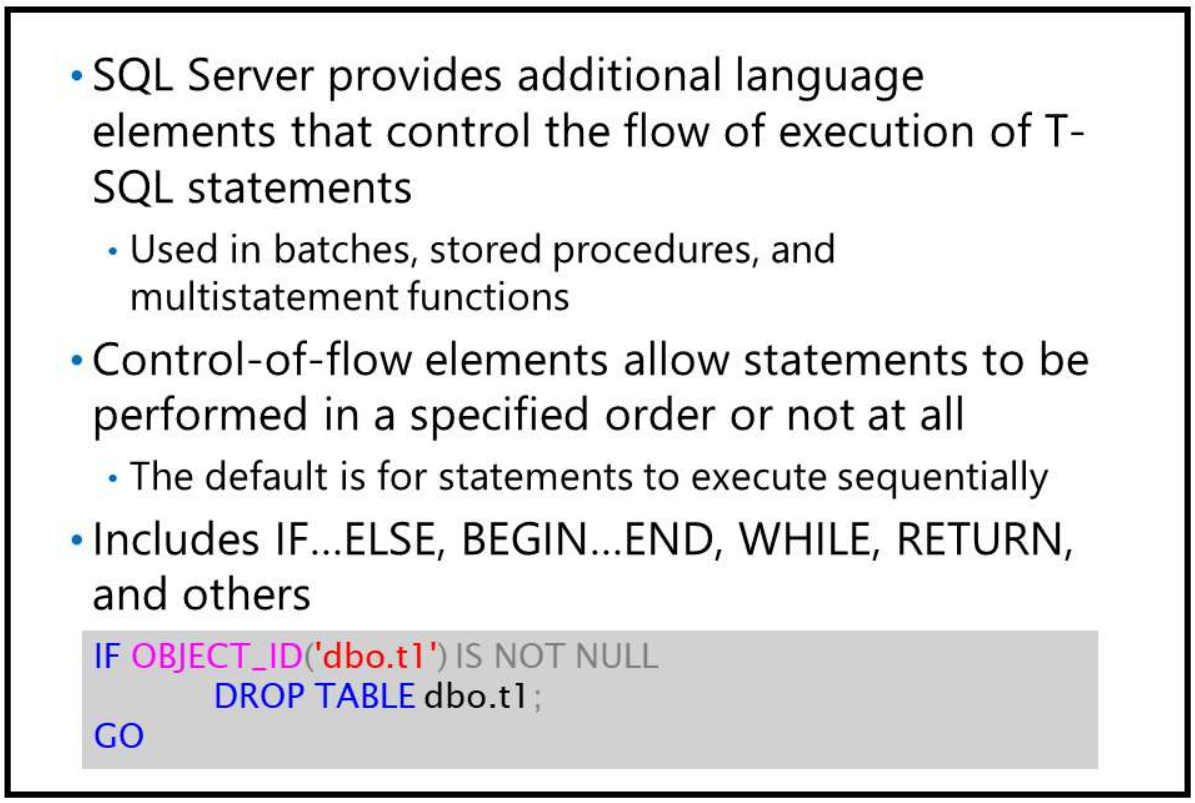
All programming languages include elements that help you to determine the flow of the program, or the order in which statements are executed. While not as fully featured as languages like C#, T-SQL provides a set of control-of-flow keywords you can use to perform logic tests and create loops containing your T-SQL data manipulation statements. In this lesson, you will learn how to use the T-SQL IF and WHILE keywords.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the control-of-flow elements in T-SQL.
- Write T-SQL code using IF...ELSE blocks.
- Write T-SQL code that uses WHILE.

Understanding T-SQL Control-of-Flow Language



SQL Server provides language elements that control the flow of program execution within T-SQL batches, stored procedures, and multistatement user-defined functions. These control-of-flow elements mean you can programmatically determine whether or not to execute statements and programmatically determine the order of those statements that should be executed.

These elements include, but are not limited to:

- IF...ELSE, which executes code based on a Boolean expression.
- WHILE, which creates a loop that executes providing a condition is true.
- BEGIN…END, which defines a series of T-SQL statements that should be executed together.
- Other keywords (for example, BREAK, CONTINUE, WAITFOR, and RETURN), which are used to support T-SQL control-of-flow operations.

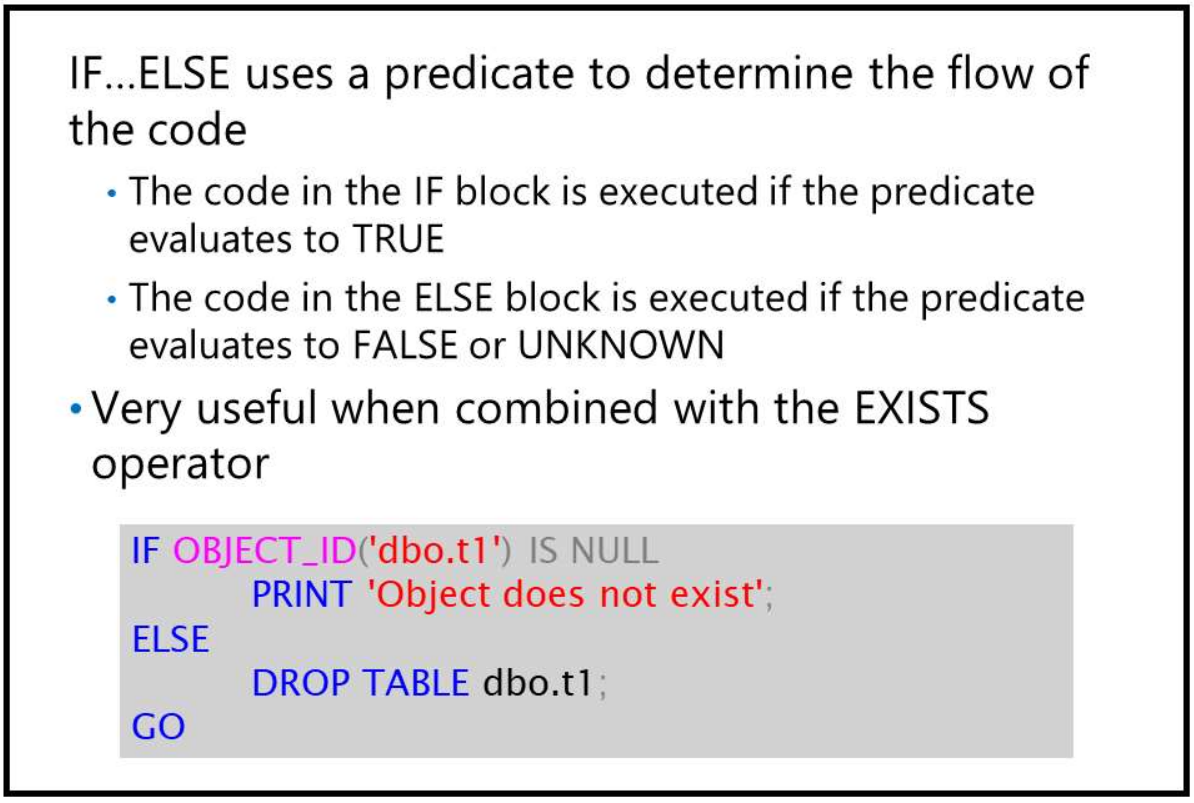
You will learn how to use some of these elements in the next lesson.

For more information, see the SQL Server Technical Documentation:

Control-of-Flow Language (Transact-SQL)

http://aka.ms/Pvihnn

Working with IF…ELSE



The IF...ELSE structure is used in T-SQL to conditionally execute a block of code based on a predicate. The IF statement determines whether or not the following statement or block (if BEGIN...END is used) executes. If the predicate evaluates to TRUE, the code in the block is executed. If the predicate evaluates to FALSE or UNKNOWN, the block is not executed, unless the optional ELSE keyword is used to identify another block of code.

IF Example

```
USE TSQL;
GO
IF OBJECT_ID('HR.Employees') IS NULL --this object does exist in the sample database
BEGIN
        PRINT 'The specified object does not exist';
END;
```

IF...ELSE Example

```
IF OBJECT_ID('HR.Employees') IS NULL
BEGIN
        PRINT 'The specified object does not exist';
END
ELSE
BEGIN
        PRINT 'The specified object exists';
END;
```

Existence Check

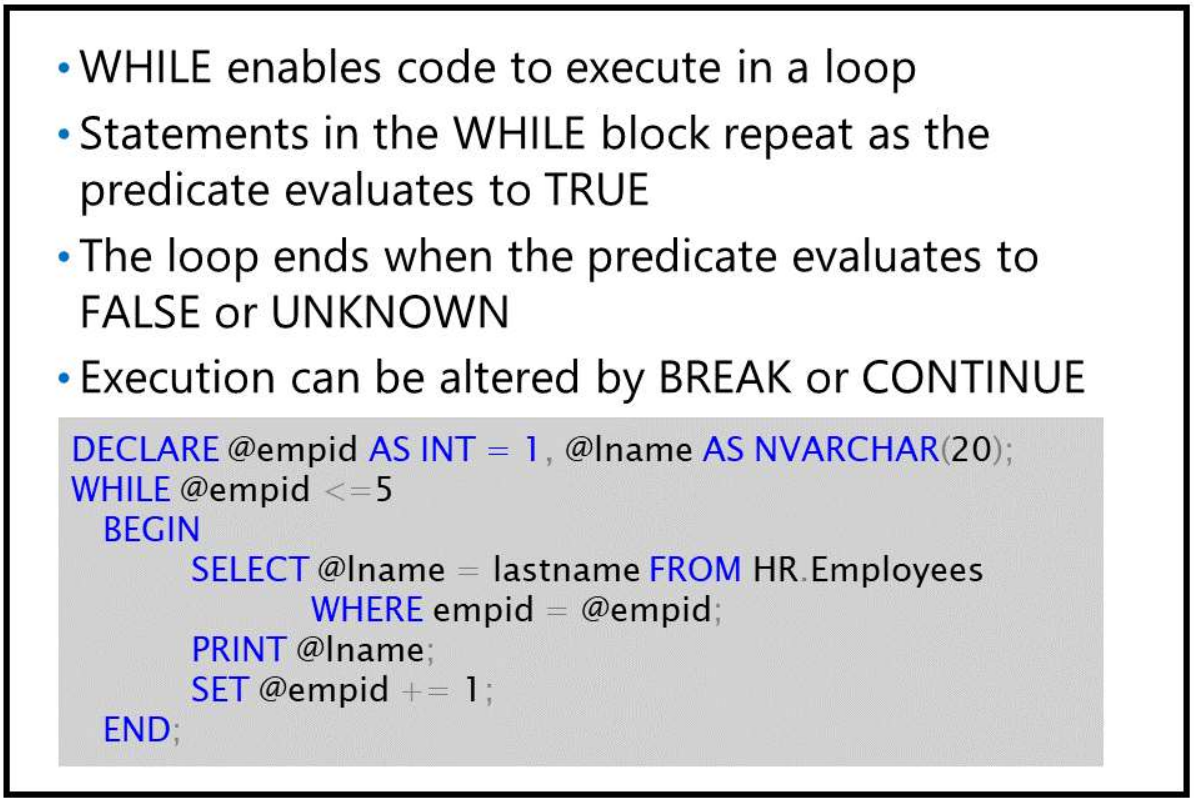
```
IF EXISTS (SELECT * FROM Sales.EmpOrders WHERE empid =5)
        BEGIN
                PRINT 'Employee has associated orders';
        END;
```

For more information, see the SQL Server Technical Documentation:

IF...ELSE (Transact-SQL)

http://aka.ms/mvl2f5

Working with WHILE



The WHILE statement is used to execute code in a loop based on a predicate. Like the IF statement, the WHILE statement determines whether the following statement or block (if BEGIN...END is used) executes. The loop ends when the predicate evaluates to FALSE or UNKNOWN. Typically, you control the loop with a variable tested by the predicate and manipulated in the body of the loop itself.

WHILE Example

```
DECLARE @empid AS INT = 1, @lname AS NVARCHAR(20);
WHILE @empid <=5
   BEGIN
        SELECT @lname = lastname FROM HR.Employees
                WHERE empid = @empid;
        PRINT @lname;
        SET @empid += 1;
   END;
```

Note: Remember that if SELECT returns UNKNOWN, the variable retains its current value. If there is no employee with an ID equal to @empid, the variable doesn't change from one iteration to another. This would lead to an infinite loop.

The result is:

```
Davis
Funk
```

```
Lew
Peled
Buck
```

For additional options within a WHILE loop, you can use the CONTINUE and BREAK keywords to control the flow. For more information about these options, see the SQL Server Technical Documentation:

WHILE (Transact-SQL)

http://aka.ms/Beqqci

Demonstration: Controlling Program Flow

In this demonstration, you will see how to control the flow of execution.

Demonstration Steps

Control the Flow of Execution

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

Check Your Knowledge

Select the best answer

You want to populate a table by creating 15 new rows. Before you create the rows, you need to check that the table exists. From the following T-SQL keywords, choose the one that you will NOT need to use.

IF

WHILE

BEGIN

END

INSERT

Check answer Show solution Reset

Check that the table exists by using an IF statement. Populate the table by using INSERT statements. Because you want to execute 15 INSERT statements, enclose them in BEGIN and END keywords so that the IF condition applies to them all.

Lab: Programming with T-SQL

Scenario

As a junior database developer for Adventure Works, you have so far focused on writing reports using corporate databases stored in SQL Server. To prepare for upcoming tasks, you will be working with some basic T-SQL programming objects.

Objectives

After completing this lab, you will be able to:

- Declare variables and delimit batches.
- Use control of flow elements.
- Use variables with a dynamic SQL statement.
- Use synonyms.

Lab Setup

Estimated Time: 45 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Declaring Variables and Delimiting Batches

Scenario

You will practice how to declare variables, retrieve their values, and use them in a SELECT statement to return specific employee information.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Declare a Variable and Retrieve the Value
3. Set the Variable Value Using a SELECT Statement
4. Use a Variable in the WHERE Clause

5. Use Variables with Batches

Detailed Steps ▲

Task 1: Prepare the Lab Environment

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab16\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Declare a Variable and Retrieve the Value

1. In SQL Server Management Studio, open the project file **D:\Labfiles\Lab16\Starter\Project\Project.ssmssln** and the T-SQL script **51 - Lab Exercise 1.sql**. Ensure that you are connected to the TSQL database.
2. Write T-SQL code that will create a variable called @num as an int data type. Set the value of the variable to 5 and display it using the alias mynumber. Execute the T-SQL code.
3. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab16\Solution\52 - Lab Exercise 1 - Task 1_1 Result.txt.
4. Write the batch delimiter GO after the written T-SQL code. In addition, write new T-SQL code that defines two variables, @num1 and @num2, both as an int data type. Set the values to 4 and 6 respectively. Write a SELECT statement to retrieve the sum of both variables using the alias totalnum. Execute the T-SQL code.
5. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab16\Solution\53 - Lab Exercise 1 - Task 1_2 Result.txt.

Detailed Steps ▲

Task 3: Set the Variable Value Using a SELECT Statement

1. Write T-SQL code that defines the variable @empname as an nvarchar(30) data type.
2. Set the value by executing a SELECT statement against the HR.Employees table. Compute a value that concatenates the firstname and lastname column values. Add a space between the two column values and filter the results to return the employee whose empid value is equal to 1.
3. Return the @empname variable's value using the alias employee.

4. Execute the T-SQL code.
5. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab16\Solution\54 - Lab Exercise 1 - Task 2Result.txt.
6. What would happen if the SELECT statement was returning more than one row?

Detailed Steps ▲

Task 4: Use a Variable in the WHERE Clause

1. Copy the T-SQL code from task 2 and modify it by defining an additional variable named @empid with an int data type. Set the variable's value to 5. In the WHERE clause, modify the SELECT statement to use the newly-created variable as a value for the column empid.
2. Execute the modified T-SQL code.
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.

Detailed Steps ▲

Task 5: Use Variables with Batches

1. Copy the T-SQL code from task 3 and modify it by adding the batch delimiter GO before the statement:

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

2. Execute the modified T-SQL code.
3. What happened? What is the error message? Can you explain why the batch delimiter caused an error?

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

Exercise 2: Using Control-of-Flow Elements

Scenario

You would like to include conditional logic in your T-SQL code to control the flow of elements by setting different values to a variable using the IF statement.

The main tasks for this exercise are as follows:

1. Write Basic Conditional Logic
2. Check the Employee Birthdate
3. Create and Execute a Stored Procedure
4. Execute a Loop Using the WHILE Statement
5. Remove the Stored Procedure

Detailed Steps ▲

Task 1: Write Basic Conditional Logic

1. Open the SQL script **61 - Lab Exercise 2.sql**. Ensure that you are connected to the TSQL database.
2. Write T-SQL code that defines the variable @result as an nvarchar(20) data type and the variable @i as an int data type. Set the value of the @i variable to 8. Write an IF statement that implements the following logic:
   - For @i variable values less than 5, set the value of the @result variable to “Less than 5”.
   - For @i variable values between 5 and 10, set the value of the @result variable to “Between 5 and 10”.
   - For all @i variable values over 10, set the value of the @result variable to “More than 10”.
   - For other @i variable values, set the value of the @result variable to “Unknown”.
3. At the end of the T-SQL code, write a SELECT statement to retrieve the value of the @result variable using the alias result. Highlight the complete T-SQL code and execute it.
4. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab16\Solution\62 - Lab Exercise 2 - Task 1 Result.txt.
5. Copy the T-SQL code and modify it by replacing the IF statement with a CASE expression to get the same result.

Detailed Steps ▲

Task 2: Check the Employee Birthdate

1. Write T-SQL code that declares two variables: @birthdate (data type date) and @cmpdate (data type date).

2. Set the value of the @birthdate variable by writing a SELECT statement against the HR.Employees table and retrieving the column birthdate. Filter the results to include only the employee with an empid equal to 5.

3. Set the @cmpdate variable to the value January 1, 1970.

4. Write an IF conditional statement by comparing the @birthdate and @cmpdate variable values. If @birthdate is less than @cmpdate, use the PRINT statement to print the message “The person selected was born before January 1, 1970”. Otherwise, print the message “The person selected was born on or after January 1, 1970”.

5. Execute the T-SQL code.

6. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\63 - Lab Exercise 2 - Task 2 Result.txt. This is a simple example for the purpose of this exercise. Typically, a different statement block would execute in each case.

Detailed Steps ▲

Task 3: Create and Execute a Stored Procedure

1. The IT department has provided T-SQL code that encapsulates the previous task in 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. Execute the provided T-SQL code:

```
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. Write an EXECUTE statement to invoke the Sales.CheckPersonBirthDate stored procedure using the parameters of 3 for @empid and January 1, 1990, for @cmpdate. Execute the T-SQL code.

3. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\64 - Lab Exercise 2 - Task 3 Result.txt.

Detailed Steps ▲

Task 4: Execute a Loop Using the WHILE Statement

1. Write T-SQL code to loop 10 times, displaying the current loop information on each occasion.
2. Define the @i variable as an int data type. Write a WHILE statement to execute while the @i variable value is less than or equal to 10. Inside the loop statement, write a PRINT statement to display the value of the @i variable using the alias loopid. Add T-SQL code to increment the @i variable value by 1.
3. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\65 - Lab Exercise 2 - Task 4 Result.txt.

Detailed Steps ▲

Task 5: Remove the Stored Procedure

- Execute the provided T-SQL code under the task 5 description to remove the created stored procedure.

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

Scenario

You will practice how to invoke dynamic SQL code and how to pass variables to it.

The main tasks for this exercise are as follows:

1. Write a Dynamic SQL Statement That Does Not Use a Parameter
2. Write a Dynamic SQL Statement That Uses a Parameter

Detailed Steps ▲

Task 1: Write a Dynamic SQL Statement That Does Not Use a Parameter

1. Open the T-SQL script **71 - Lab Exercise 3.sql**. Ensure that you are connected to the TSQL database.
2. Write T-SQL code that defines the variable @SQLstr as nvarchar(200) data type. Set the value of the variable to a SELECT statement that retrieves the empid, firstname, and lastname columns in the HR.Employees table.
3. Write an EXECUTE statement to invoke the written dynamic SQL statement inside the @SQLstr variable. Execute the T-SQL code.
4. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\72 - Lab Exercise 3 - Task 1 Result.txt.

Detailed Steps ▲

Task 2: Write a Dynamic SQL Statement That Uses a Parameter

1. Copy the previous T-SQL code and modify it to include in the dynamic batch stored in @SQLstr, a filter in which empid is equal to a parameter named @empid. In the calling batch, define a variable named @SQLparam as nvarchar(100). This variable will hold the definition of the @empid parameter. This means setting the value of the @SQLparam variable to @empid int.
2. Write an EXECUTE statement that uses sp_executesql to invoke the code in the @SQLstr variable, passing the parameter definition stored in the @SQLparam variable to sp_executesql. Assign the value 5 to the @empid parameter in the current execution.
3. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\73 - Lab Exercise 3 - Task 2 Result.txt.

Result: After this exercise, you should have a basic knowledge of generating and invoking dynamic SQL statements.

Exercise 4: Using Synonyms

Scenario

You will practice how to create a synonym for a table inside the AdventureWorks2008R2 database and how to write a query against it.

The main tasks for this exercise are as follows:

1. Create and Use a Synonym for a Table
2. Drop the Synonym

Detailed Steps ▲

Task 1: Create and Use a Synonym for a Table

1. Open the T-SQL script **81 - Lab Exercise 4.sql**. Ensure that you are connected to the TSQL database.
2. Write T-SQL code to create a synonym named dbo.Person for the Person.Person table in the AdventureWorks database. Execute the written statement.
3. Write a SELECT statement against the dbo.Person synonym and retrieve the FirstName and LastName columns. Execute the SELECT statement.
4. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab16\Solution\82 - Lab Exercise 4 - Task 1 Result.txt.

Detailed Steps ▲

Task 2: Drop the Synonym

- Execute the provided T-SQL code under the task 2 description to remove the synonym.

Result: After this exercise, you should know how to create and use a synonym.

Module Review and Takeaways

In this module, you have learned how to:

- Describe the language elements of T-SQL used for simple programming tasks.
- Describe batches and how they are handled by SQL Server.
- Declare and assign variables and synonyms.
- Use IF and WHILE blocks to control program flow.

Review Question(s)

Check Your Knowledge

Discovery

Can you declare a variable in one batch and reference it in multiple batches?

Show solution Reset

No, variables are local to the batch in which they are declared.

Check Your Knowledge

Discovery

Can you create a synonym that references an object that does not yet exist?

Show solution Reset

Yes, resolution doesn't occur until the synonym is used.

Check Your Knowledge

Discovery

Will a WHILE loop exit when the predicate evaluates to NULL?

Show solution Reset

Yes.