

Module 16: Programming with T-SQL

Contents:

Module Overview

Lesson 1: T-SQL Programming Elements

Lesson 2: Controlling Program Flow

Lab: Programming with T-SQL

Module Review and Takeaways

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 sent to SQL Server as a unit for parsing, optimization, and execution
- Batches are terminated with GO by default
- Batches are boundaries for variable scope
- Some statements (for example, CREATE FUNCTION, CREATE PROCEDURE, CREATE VIEW) may not be combined with others in the same batch

```
CREATE VIEW <view_name>  
AS ...;  
GO  
CREATE PROCEDURE <procedure_name>  
AS ...;  
GO
```

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

- Batches are parsed for syntax as a unit
 - Syntax errors cause the entire batch to be rejected
 - Runtime errors may allow the batch to continue after failure, by default

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

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

- Batches can contain error-handling code

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

- Initialize a variable using the DECLARE statement

```
DECLARE @i INT = 0;
```

- Assign a single (scalar) value using the SET statement

```
SET @i = 1;
```

- Assign a value to a variable using a SELECT statement

- Be sure that the SELECT statement returns exactly one row

```
SELECT @i = COUNT(*) FROM Sales.SalesOrderHeader;
```

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

- A synonym is an alias or link to an object stored either on the same SQL Server instance or on a linked server
 - Synonyms can point to tables, views, procedures, and functions
- Synonyms can be used for referencing remote objects as though they were located locally, or for providing alternative names to other local objects
- Use the CREATE and DROP commands to manage synonyms

```

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

```

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.ssmssIn** 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](#)

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 additional language elements that control the flow of execution of T-SQL statements
 - Used in batches, stored procedures, and multistatement functions
- Control-of-flow elements allow statements to be performed in a specified order or not at all
 - The default is for statements to execute sequentially
- Includes IF...ELSE, BEGIN...END, WHILE, RETURN, and others

```
IF OBJECT_ID('dbo.t1') IS NOT NULL
    DROP TABLE dbo.t1;
GO
```

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

IF...ELSE uses a predicate to determine the flow of the code

- The code in the IF block is executed if the predicate evaluates to TRUE
- The code in the ELSE block is executed if the predicate evaluates to FALSE or UNKNOWN
- Very useful when combined with the EXISTS operator

```
IF OBJECT_ID('dbo.t1') IS NULL
    PRINT 'Object does not exist';
ELSE
    DROP TABLE dbo.t1;
GO
```

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

- WHILE enables code to execute in a loop
- Statements in the WHILE block repeat as the predicate evaluates to TRUE
- The loop ends when the predicate evaluates to FALSE or UNKNOWN
- Execution can be altered by BREAK or CONTINUE

```

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;

```

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

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 ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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 ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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 ▼

Detailed Steps ▼

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 ▼

Detailed Steps ▼

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

Check Your Knowledge

Discovery

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

Show solution Reset

Check Your Knowledge

Discovery

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

Show solution Reset