

Module 16

Programming with T-SQL

Module Overview

- T-SQL Programming Elements
- Controlling Program Flow

Lesson 1: T-SQL Programming Elements

- Introducing T-SQL Batches
- Working with Batches
- Introducing T-SQL Variables
- Working with Variables
- Working with Synonyms
- Demonstration: T-SQL Programming Elements

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
```

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

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
```

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;
```

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;
```


Demonstration: T-SQL Programming Elements

In this demonstration, you will see how to:

- Control batch execution and variable usage
- Demonstration A

Lesson 2: Controlling Program Flow

- Understanding T-SQL Control-of-Flow Language
- Working with IF...ELSE
- Working with WHILE
- Demonstration: Controlling Program Flow

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
```

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
```

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;
```

Demonstration: Controlling Program Flow

In this demonstration, you will see how to:

- Control the flow of execution
- Demonstration B

Module Review and Takeaways

- Review Question(s)
- **Question 1**

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

- **Question 2**

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

- **Question 3**

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

- **Answer 1**

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

- **Answer 2**

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

- **Answer 3**

Yes.