Database Management Systems

T-SQL SCRIPTING

September 2019

Contents

Batch

Statements

- Script processing
- Flow control

Variables

- Scalar
- Table

Batch

Unit of parsing, binding & optimization

Scope for variables & parameters

Script can contain many batches, end with GO

GO – not a T-SQL statement, but a MS & SQLCMD command

Some statements should have their own batch, such as:

```
CREATE VIEW          CREATE PROCEDURE      CREATE FUNCTION
CREATE TRIGGER       CREATE SCHEMA
```

Statements for Script Processing

Keyword	Description
USE	Changes the database context to the specified database.
PRINT	Returns a message to the client.
DECLARE	Declares a local variable.
SET	Sets the value of a local variable or a session variable.
EXEC	Executes a dynamic SQL statement or stored procedure.

Statements for Execution Flow Control

Keyword	Description
`IF...ELSE`	Controls the flow of execution based on a condition.
`BEGIN...END`	Defines a statement block.
`WHILE`	Repeats statements while a specific condition is true.
`BREAK`	Exits the innermost WHILE loop.
`CONTINUE`	Returns to the beginning of a WHILE loop.
`TRY...CATCH`	Controls the flow of execution when an error occurs.
`GOTO`	Unconditionally changes the flow of execution.
`RETURN`	Exits unconditionally.

USE & PRINT Statements

Syntax

- USE database
- PRINT string_expression

Example

```
USE  AccountPayables;
DECLARE @TotalDue money;
SET @TotalDue = (SELECT SUM(InvoiceTotal - PaymentTotal - CreditTotal)
    FROM Invoices);
IF @TotalDue > 0
    PRINT 'Total invoices due = $' + CONVERT(varchar,@TotalDue,1);
ELSE
    PRINT 'Invoices paid in full';
```

SCALAR Variables

Variables are local (inside their batch)

Define

- DECLARE @variable_name_1 data_type [,…]

Set values

- SET @variable_name = expression
- SELECT @variable_name = expression

Example

Results

```
Maximum invoice is $46.21.
Minimum invoice is $16.33.
Maximum is 182.97% more than minimum.
Number of invoices: 6.
```

Script

```
DECLARE @MaxInvoice money, @MinInvoice money;
DECLARE @PercentDifference decimal(8,2);
DECLARE @InvoiceCount int, @VendorIDVar int;

SET @VendorIDVar = 95;
SET @MaxInvoice = (SELECT MAX(InvoiceTotal) FROM Invoices
    WHERE VendorID = @VendorIDVar);
SELECT @MinInvoice = MIN(InvoiceTotal), @InvoiceCount = COUNT(*)
FROM Invoices
WHERE VendorID = @VendorIDVar;
SET @PercentDifference = (@MaxInvoice - @MinInvoice) / @MinInvoice * 100;

PRINT 'Maximum invoice is $' + CONVERT(varchar,@MaxInvoice,1) + '.';
PRINT 'Minimum invoice is $' + CONVERT(varchar,@MinInvoice,1) + '.';
PRINT 'Maximum is ' + CONVERT(varchar,@PercentDifference) +
    '% more than minimum.';
PRINT 'Number of invoices: ' + CONVERT(varchar,@InvoiceCount) + '.';
```

TABLE Variables

Store the contents of an entire table

Defined used DECLARE, table data type

Could be used with INSERT, SELECT, UPDATE, DELETE

Could NOT be used with SELECT INTO

```
DECLARE @table_name TABLE
(column_name_1 data_type [column_attributes]
[, column_name_2 data_type [column_attributes]]...
[, table_attributes])
```

Example

```
DECLARE @BigVendors table
(VendorID int,
VendorName varchar(50));

INSERT @BigVendors
SELECT VendorID, VendorName
FROM Vendors
WHERE VendorID IN
    (SELECT VendorID FROM Invoices WHERE InvoiceTotal > 5000);

SELECT * FROM @BigVendors;
```

The result set

	VendorID	VendorName
1	72	Data Reproductions Corp
2	99	Bertelsmann Industry Svcs. Inc
3	104	Digital Dreamworks
4	110	Malloy Lithographing Inc

T-SQL Table Objects

Type	Scope
Standard table	Available within the system until explicitly deleted.
Temporary table	Available within the system while the current database session is open.
Table variable	Available within a script while the current batch is executing.
Derived table	Available within a statement while the current statement is executing.
View	Available within the system until explicitly deleted.

Temporary tables

Local (current session)

- #table_name
- Automatically deleted at the end of session

Global

- ##table_name
- DROP TABLE ##table_name

Example (local temporary table)

```
SELECT TOP 1 VendorID, AVG(InvoiceTotal) AS AvgInvoice
INTO #TopVendors
FROM Invoices
GROUP BY VendorID
ORDER BY AvgInvoice DESC;

SELECT Invoices.VendorID, MAX(InvoiceDate) AS LatestInv
FROM Invoices JOIN #TopVendors
    ON Invoices.VendorID = #TopVendors.VendorID
GROUP BY Invoices.VendorID;
```

The result set

	VendorID	LatestInv
1	110	2016-03-31 00:00:00

Example (Global temporary table)

```
CREATE TABLE ##RandomSSNs
(
  SSN_ID int      IDENTITY,
  SSN    char(9) DEFAULT
           LEFT(CAST(CAST(CEILING(RAND()*10000000000)AS bigint)AS varchar),9)
);

INSERT ##RandomSSNs VALUES (DEFAULT);
INSERT ##RandomSSNs VALUES (DEFAULT);

SELECT * FROM ##RandomSSNs;
```

The result set

	SSN_ID	SSN
1	1	419741221
2	2	327280021

Derived Table

```
WITH TopVendors AS
(
    SELECT TOP 1 VendorID, AVG(InvoiceTotal) AS AvgInvoice
    FROM Invoices
    GROUP BY VendorID
    ORDER BY AvgInvoice DESC
)
SELECT Invoices.VendorID, MAX(InvoiceDate) AS LatestInv
FROM Invoices JOIN TopVendors
    ON Invoices.VendorID = TopVendors.VendorID
GROUP BY Invoices.VendorID;
```

IF ELSE

```
IF Boolean_expression
    {statement|BEGIN...END}
[ELSE
    {statement|BEGIN...END}]
```

```
DECLARE @MinInvoiceDue money, @MaxInvoiceDue money;
DECLARE @EarliestInvoiceDue smalldatetime, @LatestInvoiceDue smalldatetime;
SELECT @MinInvoiceDue = MIN(InvoiceTotal - PaymentTotal - CreditTotal),
    @MaxInvoiceDue = MAX(InvoiceTotal - PaymentTotal - CreditTotal),
    @EarliestInvoiceDue = MIN(InvoiceDueDate),
    @LatestInvoiceDue = MAX(InvoiceDueDate)
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;
IF @EarliestInvoiceDue < GETDATE()
    BEGIN
        PRINT 'Outstanding invoices overdue!';
        PRINT 'Dated ' + CONVERT(varchar,@EarliestInvoiceDue,1) +
            ' through ' + CONVERT(varchar,@LatestInvoiceDue,1) + '.';
        PRINT 'Amounting from $' + CONVERT(varchar,@MinInvoiceDue,1) +
            ' to $' + CONVERT(varchar,@MaxInvoiceDue,1) + '.';
    END;
ELSE --@EarliestInvoiceDue >= GETDATE()
    PRINT 'No overdue invoices.';
```

The response from the system

```
Outstanding invoices overdue!
Dated 04/09/16 through 04/30/16.
Amounting from $30.75 to $19,351.18.
```

Database Object Existence Testing

SQL Server 2016 or later

- DROP OBJECT_TYPE IF EXISTS object_name
- Ex: DROP DATABASE IF EXISTS testdb;

Earlier versions

- OBJECT_ID(‘object’) function
- DB_ID(‘database’) function

Examples

Code that tests whether a database exists before it deletes it

```
USE master;
IF DB_ID('TestDB') IS NOT NULL
    DROP DATABASE TestDB;

CREATE DATABASE TestDB;
```

Code that tests for the existence of a table

```
IF OBJECT_ID('InvoiceCopy') IS NOT NULL
    DROP TABLE InvoiceCopy;
```

Another way to test for the existence of a table

```
IF EXISTS (SELECT * FROM sys.tables
           WHERE name = 'InvoiceCopy')
    DROP TABLE InvoiceCopy;
```

Code that tests for the existence of a temporary table

```
IF OBJECT_ID('tempdb..#AllUserTables') IS NOT NULL
    DROP TABLE #AllUserTables;
```

WHILE

```
WHILE expression
    {statement | BEGIN...END}
    [BREAK]
    [CONTINUE]
```

```
IF OBJECT_ID('tempdb..#InvoiceCopy') IS NOT NULL
    DROP TABLE #InvoiceCopy;

SELECT * INTO #InvoiceCopy FROM Invoices
WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0;

WHILE (SELECT SUM(InvoiceTotal - CreditTotal - PaymentTotal)
       FROM #InvoiceCopy) >= 20000
    BEGIN
        UPDATE #InvoiceCopy
        SET CreditTotal = CreditTotal + .05
        WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0;

        IF (SELECT MAX(CreditTotal) FROM #InvoiceCopy) > 3000
            BREAK;
        ELSE --(SELECT MAX(CreditTotal) FROM #InvoiceCopy) <= 3000
            CONTINUE;
    END;

SELECT InvoiceDate, InvoiceTotal, CreditTotal
FROM #InvoiceCopy;
```

Cursor

Process one row at a time

Slower

Use more server resources than standard database access

Declare a cursor

```
DECLARE cursor_name CURSOR FOR select_statement;
```

Open the cursor

```
OPEN cursor_name;
```

Get column values from the row and store them in a series of variables

```
FETCH NEXT FROM cursor_name INTO @variable1[, @variable2][, @variable3]...;
```

Close and deallocate the cursor

```
CLOSE cursor_name;
DEALLOCATE cursor_name;
```

Example

```
DECLARE @InvoiceIDVar int, @InvoiceTotalVar money, @UpdateCount int;
SET @UpdateCount = 0;

DECLARE Invoices_Cursor CURSOR
FOR
    SELECT InvoiceID, InvoiceTotal  FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;

OPEN Invoices_Cursor;

FETCH NEXT FROM Invoices_Cursor INTO @InvoiceIDVar, @InvoiceTotalVar;
WHILE @@FETCH_STATUS <> -1
    BEGIN
        IF @InvoiceTotalVar > 1000
        BEGIN
            UPDATE Invoices
            SET CreditTotal = CreditTotal + (InvoiceTotal * .1)
            WHERE InvoiceID = @InvoiceIDVar;

            SET @UpdateCount = @UpdateCount + 1;
        END;
        FETCH NEXT FROM Invoices_Cursor INTO @InvoiceIDVar, @InvoiceTotalVar;
    END;

CLOSE Invoices_Cursor;
DEALLOCATE Invoices_Cursor;

PRINT '';
PRINT CONVERT(varchar, @UpdateCount) + ' row(s) updated.';
```

Error Handling

The syntax of the TRY...CATCH statement

```
BEGIN TRY
     {sql_statement|statement_block}
END TRY
BEGIN CATCH
     {sql_statement|statement_block}
END CATCH
```

Functions you can use within a CATCH block

Function	Description
`ERROR_NUMBER()`	Returns the error number.
`ERROR_MESSAGE()`	Returns the error message.
`ERROR_SEVERITY()`	Returns the severity of the error.
`ERROR_STATE()`	Returns the state of the error.

Example

A script that uses a TRY...CATCH statement

```
BEGIN TRY
    INSERT Invoices
    VALUES (799, 'ZXK-799', '2016-05-07', 299.95, 0, 0,
            1, '2016-06-06', NULL);
    PRINT 'SUCCESS: Record was inserted.';
END TRY
BEGIN CATCH
    PRINT 'FAILURE: Record was not inserted.';
    PRINT 'Error ' + CONVERT(varchar, ERROR_NUMBER(), 1)
        + ': ' + ERROR_MESSAGE();
END CATCH;
```

The message that's displayed

```
FAILURE: Record was not inserted.
Error 547: The INSERT statement conflicted with the FOREIGN KEY constraint
"FK_Invoices_Vendors". The conflict occurred in database "AP", table
"dbo.Vendors", column 'VendorID'.
```

System Functions

Function name	Description
`@@IDENTITY`	Returns the last value generated for an identity column on the server. Returns NULL if no identity value was generated.
`IDENT_CURRENT('tablename')`	Similar to @@IDENTITY, but returns the last identity value that was generated for a specified table.
`@@ROWCOUNT`	Returns the number of rows affected by the most recent SQL statement.
`@@ERROR`	Returns the error number generated by the execution of the most recent SQL statement. Returns 0 if no error occurred.
`@@SERVERNAME`	Returns the name of the local server.
`HOST_NAME()`	Returns the name of the current workstation.
`SYSTEM_USER`	Returns the name of the current user.

Session Settings

Statement	Description
`SET DATEFORMAT format`	Sets the order of the parts of a date (month/day/year) for entering date/time data. The default is mdy, but any permutation of m, d, and y is valid.
`SET NOCOUNT {ON|OFF}`	Determines whether SQL Server returns a message indicating the number of rows that were affected by a statement. OFF is the default.
`SET ANSI_NULLS {ON|OFF}`	Determines how SQL Server handles equals (=) and not equals (<>) comparisons with null values. The default is ON, in which case "WHERE column = NULL" will always return an empty result set, even if there are null values in the column.
`SET ANSI_PADDING {ON|OFF}`	Determines how SQL Server stores char and varchar values that are smaller than the maximum size for a column or that contain trailing blanks. Only affects new column definitions. The default is ON, which causes char values to be padded with blanks. In addition, trailing blanks in varchar values are not trimmed. If this option is set to OFF, char values that don't allow nulls are padded with blanks, but blanks are trimmed from char values that allow nulls as well as from varchar values.
`SET ROWCOUNT number`	Limits the number of rows that are processed by a query. The default setting is 0, which causes all rows to be processed.