SQL Scripts

Variables

Variable names must begin with an @. For the remainder of the name, standard SQL server identifier naming rules apply:

- The variable name must be 128 characters or less
- It must start with a letter (A-Z, a-z) immediately after the @, and subsequent characters can be letters (A-Z, a-z), numbers (0-9), underscore, @ signs, or # signs.
- It cannot contain embedded spaces or other illegal characters
- It cannot be a SQL or SQL Server reserved word.

Variables are declared as follows:

```
DECLARE @varname datatype{, …n}
```

where varname is the variable name, and datatype is the name of a system or user defined datatype.

Examples:

```
DECLARE @x int
```

```
DECLARE @y int, @z smallint
```

To use the variable value in an expression, simply insert the variable name into the expression. To set the value of the variable, use the following syntax:

```
SELECT @varname = expression
```

Examples:

```
SELECT @x= @x * 3.2
```

Variables can also be used within select statements and subqueries **both** to provide values and to extract values, as the following example illustrate.

```
SELECT @x=LastName FROM Employees WHERE EmpID=@y
```

The above performs the select (matching all employees whose id equals the value currently stored in @y) and sets @x to the matching employees last name field value. If no records are returned, the value of @x will be unchanged. If multiple records are returned, @x will be set to the value of last name for the last record in the recordset.

The scope of a variable is from its point of declaration to the end of the script or stored procedure.

Structures & Flow Control

Like other programming languages, SQL has structures and statements for controlling program flow and looping.

IF...ELSE structure

The IF..ELSE structure is used for conditionally controlling program flow. Conditions can contain subqueries but must ultimately reduce to a TRUE or FALSE value. IF statements can be nested. The following syntax are all permissible:

```
IF boolean_condition
      sql statement
```

or

```
IF boolean_condition
      sql statement
ELSE
      sql statement
```

If it is desired to have multiple SQL statements within the IF or ELSE part, use the BEGIN and END statements to block them, for example:

```
IF condition
      BEGIN
            sql statements
      END
```

Example:

```
IF (Select Avg([Value]) FROM Books) > 50
      PRINT 'Average Too High'
ELSE
BEGIN
      SELECT @X=4.3*Avg([Value]) FROM Books
      PRINT @x          -- print returns a printed message
            END
```

CASE structure

The CASE structure is actually an expression that can only be used inside other statements, rather than as a statement itself. However, it is introduced here as it provides a select case-like construct for processing. It can be used in a SELECT fieldlist, GROUP BY clause, ORDER BY clause, WHERE CLAUSE, the SET clause of an UPDATE query, or anywhere SQL allows an expression to be used. Syntax:

```
CASE
```

```
        WHEN boolean_condition THEN result_expression
        {…n}
        {ELSE else_result_expression}
END
```

Or alternatively:

```
CASE input_expression
        WHEN expression THEN result_expression
        {…n}
        {ELSE else_result_expression}
END
```

Examples:

```
/* Note alternative method for aliasing a column (‘name’=field_or_expression)
*/
SELECT title, [Value],
CASE
        WHEN [Value] < 30.00 THEN 'Cheap!'
        WHEN [Value] BETWEEN 30.00 AND 75 THEN 'Moderately Priced!'
        ELSE 'Expensive!'
        END
AS [CommentField]
FROM Books

UPDATE Employee
SET Salary=
CASE
        WHEN AnnualReview='A' THEN Salary*2.0
        WHEN AnnualReview='B' THEN Salary*1.5
        WHEN AnnualReview='C' THEN Salary
        ELSE  Salary*0.5
END

        /* same as above with alternate case syntax */
UPDATE Employee
SET Salary=
CASE AnnualReview
        WHEN 'A' THEN Salary*2.0
        WHEN 'B' THEN Salary*1.5
        WHEN 'C' THEN Salary
        ELSE Salary*0.5
END
```

WHILE structure

The `WHILE` structure is used for repeatedly executing a statement or block of statements subject to a condition. If there are multiple statements within the loop, contain them within a `BEGIN..END` block. `WHILE` loops can be nested. Syntax:

```
WHILE condition
      sql statement

WHILE condition
      BEGIN
            sql statements
      END
```

The key words CONTINUE and BREAK can be used for finer control. CONTINUE causes control to jump immediately to the top of the loop, skipping any further statements within the loop. BREAK causes execution to immediately exit the loop, and start with the next statement following the loop. For example:

```
SET @X=1
WHILE (SELECT Max( [Value]) FROM Books) > @X
      BEGIN
      SET @X=@X+1
      If @X>50
            CONTINUE
      ELSE
            BREAK
      PRINT 'Going…'
      END
PRINT 'Gone!'
```