**[Control of Flow with TSQL](http://sqlpractices.wordpress.com/2008/01/11/control-of-flow-with-tsql/)**

Transact SQL statements are usually executed in sequential order. When a long stored procedure or statement is created, line 1 will run, followed by line 2, followed by line 3. In many cases, we will want to interrupt this normal flow. Special processing may be needed depending on the results just received. We may want to branch or loop for example. Like other programming languages, Transact SQL has key words for controlling this order of execution. Collectively, these key words are called a Control of Flow language.

This article examines these key words and how they are used to interrelate and connect separate statements into a traditional programming like structure.

***IF***  
One of the most often used Control of Flow key words is IF. Like other programming languages, IF is used to test a Boolean condition. When the result of the IF statement is True, the next proceeding line is executed. From inside Query Analyzer, change to the pubs database and execute: IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = ‘Smith’)  
PRINT ‘Found it’The words “Found it” should be returned.

***ELSE***  
The ELSE key word can be coupled with IF to execute logic when the IF statement returns False. Else is an optional statement.

IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = Jones)  
PRINT ‘Found it’  
ELSE  
PRINT ‘Not Found’

This statement will return “Not Found.” Unlike other languages, TSQL does not have an ELSE IF key word. The work around is to nest IF ELSE statements inside other IF ELSE statements. There are no limits to the number of times an IF ELSE statement can be nested. There is no END IF statement.

IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = ‘Jones’)  
PRINT ‘Found it Jones’  
ELSE  
IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = ‘Smith’)  
PRINT ‘Found it Smith’  
ELSE  
PRINT ‘Not Found Smith or Jones’

***BEGIN…END***  
The BEGIN and END keywords are used to group multiple lines into one Statement Block. An example of when Statement Blocks are required is in the result of an IF ELSE statement. In this example, two PRINT lines are wanted on True result. Nothing should be returned on a False result.

IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = ‘Jones’)  
PRINT ‘Found it’  
PRINT ‘Last name of Jones’

The statement will return:

Last name of Jones

Even though there is no Jones in the database. The unwanted return is generated because IF statements only return the next line or Statement Block. In our case, there is no Statement Block, because the BEGIN and END key words are not used, so only the next line of: PRINT ‘Found it’ is considered to be tied to the IF statement. The PRINT ‘Last name of Jones’ is viewed as a new command all by itself–not attached to the IF. To get the desired results, modify the statement to:

IF EXISTS(SELECT au\_fname FROM authors WHERE au\_lname = ‘Jones’)  
BEGIN  
PRINT ‘Found it’  
PRINT ‘Last name of Jones’  
END

Now the statement will come back without any of the PRINT lines being executed.

***GOTO***  
Yes, the old faithful GOTO, despised as the spaghetti code instigator in other languages, is alive and well in TSQL. GOTO simply defines a label, and lets the code jump to that label from some other point. Cursors usually make use of a GOTO statement. However, a GOTO can also be used alone. This statement defines a GOTO Label, do\_it\_again, by ending it with a colon. When the second IF is executed, GOTO sends control back to the Label if the count is less than 100.

DECLARE @Count int  
SET @Count = 0

do\_it\_again:  
IF @Count <> BEGIN  
PRINT ‘Hello World’  
SET @Count = @Count + 1  
END  
IF @Count <> GOTO do\_it\_again

A point sometimes overlooked is that any statement ending in a colon is considered a label. In addition, the code under the label is executed on the first pass. In other words, the code logically belonging to a label is executed as standard TSQL when first encountered. So the value of @Count, when it reaches:

IF @Count <> GOTO do\_it\_again

For the first time is 1, not 0.

***RETURN***  
When the RETURN key word is encountered, statement execution ends, unconditionally. Any lines following a RETURN are not executed. Optionally, the RETURN can pass an integer. If a RETURN is placed inside our Label code, only one “Hello World” will be returned.

DECLARE @Count int  
SET @Count = 0

do\_it\_again:  
IF @Count <> BEGIN  
PRINT ‘Hello World’  
RETURN  
SET @Count = @Count + 1  
END  
IF @Count <> GOTO do\_it\_again

***WAITFOR***  
WAITFOR allows statement execution to be paused for a delayed time amount, or until a specific time of day. This is similar to the PAUSE key word in other languages. The following example replaces the RETURN statement with a two second delay. Execution time will now take twenty seconds for a 10 count loop.

DECLARE @Count int  
SET @Count = 0

do\_it\_again:  
IF @Count <> BEGIN  
PRINT ‘Hello World’  
WAITFOR DELAY ’00:00:02′  
SET @Count = @Count + 1  
END  
IF @Count <> GOTO do\_it\_again

***WHILE***  
Like other languages, the TSQL WHILE creates a loop that keeps executing until a Boolean False condition is received. Like IF statements, WHILE statements can be nested inside other WHILE statements. BEGIN and END are also used in the same IF statement way.

DECLARE @Count int  
SET @Count = 0

WHILE @Count <> BEGIN  
PRINT ‘Hello World’  
SET @Count = @Count + 1  
END

***BREAK…CONTINUE***  
BREAK and CONTINUE are used to exit, or continue executing WHILE or IF statements. The above statements have been modified to show an example of their use. The statement will only return ten rows now.

DECLARE @Count int  
SET @Count = 0

WHILE @Count <> BEGIN  
PRINT ‘Hello World’  
SET @Count = @Count + 1  
IF @Count > 10  
BREAK  
ELSE  
CONTINUE  
END

Examples Of Case Statemnt

### A. Using a SELECT statement with a simple CASE expression

Within a SELECT statement, a simple CASE expression allows for only an equality check; no other comparisons are made. The following example uses the CASE expression to change the display of product line categories to make them more understandable.

USE AdventureWorks2008R2;

GO

SELECT ProductNumber, Category =

CASE ProductLine

WHEN 'R' THEN 'Road'

WHEN 'M' THEN 'Mountain'

WHEN 'T' THEN 'Touring'

WHEN 'S' THEN 'Other sale items'

ELSE 'Not for sale'

END,

Name

FROM Production.Product

ORDER BY ProductNumber;

GO

### B. Using a SELECT statement with a searched CASE expression

Within a SELECT statement, the searched CASE expression allows for values to be replaced in the result set based on comparison values. The following example displays the list price as a text comment based on the price range for a product.

USE AdventureWorks2008R2;

GO

SELECT ProductNumber, Name, 'Price Range' =

CASE

WHEN ListPrice = 0 THEN 'Mfg item - not for resale'

WHEN ListPrice < 50 THEN 'Under $50'

WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'

WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'

ELSE 'Over $1000'

END

FROM Production.Product

ORDER BY ProductNumber ;

### C. Using CASE to replace the IIf function that is used in Microsoft Access

CASE provides functionality that is similar to the IIf function in Microsoft Access. The following example shows a simple query that uses IIf to provide an output value for the TelephoneInstructions column in an Access table that is named db1.ContactInfo.

SELECT FirstName, LastName, TelephoneNumber,

IIf(IsNull(TelephoneInstructions),"Any time",

TelephoneInstructions) AS [When to Contact]

FROM db1.ContactInfo;

The following example uses CASE to provide an output value for the TelephoneSpecialInstructions column in the AdventureWorks2008R2 view Person.vAdditionalContactInfo.

USE AdventureWorks2008R2;

------------------------------------------------------------------

SELECT FirstName, LastName, TelephoneNumber, 'When to Contact' =

CASE

WHEN TelephoneSpecialInstructions IS NULL THEN 'Any time'

ELSE TelephoneSpecialInstructions

END

FROM Person.vAdditionalContactInfo;

### D. Using CASE in an ORDER BY clause

The following examples uses the CASE expression in an ORDER BY clause to determine the sort order of the rows based on a given column value. In the first example, the value in the SalariedFlag column of the HumanResources.Employee table is evaluated. Employees that have the SalariedFlag set to 1 are returned in order by the EmployeeID in descending order. Employees that have the SalariedFlag set to 0 are returned in order by the EmployeeID in ascending order. In the second example, the result set is ordered by the column TerritoryName when the column CountryRegionName is equal to 'United States' and by CountryRegionName for all other rows.

SELECT BusinessEntityID, SalariedFlag

FROM HumanResources.Employee

ORDER BY CASE SalariedFlag WHEN 1 THEN BusinessEntityID END DESC

,CASE WHEN SalariedFlag = 0 THEN BusinessEntityID END;

---------------------------------------------------------------------

SELECT BusinessEntityID, LastName, TerritoryName, CountryRegionName

FROM Sales.vSalesPerson

WHERE TerritoryName IS NOT NULL

ORDER BY CASE CountryRegionName WHEN 'United States' THEN TerritoryName

ELSE CountryRegionName END;

### E. Using CASE in an UPDATE statement

The following example uses the CASE expression in an UPDATE statement to determine the value that is set for the column VacationHours for employees with SalariedFlag set to 0. When subtracting 10 hours from VacationHours results in a negative value, VacationHours is increased by 40 hours; otherwise, VacationHours is increased by 20 hours. The OUTPUT clause is used to display the before and after vacation values.

USE AdventureWorks2008R2;

GO

UPDATE HumanResources.Employee

SET VacationHours =

( CASE

WHEN ((VacationHours - 10.00) < 0) THEN VacationHours + 40

ELSE (VacationHours + 20.00)

END

)

OUTPUT Deleted.BusinessEntityID, Deleted.VacationHours AS BeforeValue,

Inserted.VacationHours AS AfterValue

WHERE SalariedFlag = 0;

### F. Using CASE in a SET statement

The following example uses the CASE expression in a SET statement in the table-valued function dbo.GetContactInfo. In the AdventureWorks2008R2 database, all data related to people is stored in the Person.Person table. For example, the person may be an employee, vendor representative, or a customer. The function returns the first and last name of a given BusinessEntityID and the contact type for that person.The CASE expression in the SET statement determines the value to display for the column ContactType based on the existence of the BusinessEntityID column in the Employee, Vendor, or Customer tables.

USE AdventureWorks2008R2;

GO

CREATE FUNCTION dbo.GetContactInformation(@BusinessEntityID int)

RETURNS @retContactInformation TABLE

(

BusinessEntityID int NOT NULL,

FirstName nvarchar(50) NULL,

LastName nvarchar(50) NULL,

ContactType nvarchar(50) NULL,

PRIMARY KEY CLUSTERED (BusinessEntityID ASC)

)

AS

-- Returns the first name, last name and contact type for the specified contact.

BEGIN

DECLARE

@FirstName nvarchar(50),

@LastName nvarchar(50),

@ContactType nvarchar(50);

-- Get common contact information

SELECT

@BusinessEntityID = BusinessEntityID,

@FirstName = FirstName,

@LastName = LastName

FROM Person.Person

WHERE BusinessEntityID = @BusinessEntityID;

SET @ContactType =

CASE

-- Check for employee

WHEN EXISTS(SELECT \* FROM HumanResources.Employee AS e

WHERE e.BusinessEntityID = @BusinessEntityID)

THEN 'Employee'

-- Check for vendor

WHEN EXISTS(SELECT \* FROM Person.BusinessEntityContact AS bec

WHERE bec.BusinessEntityID = @BusinessEntityID)

THEN 'Vendor'

-- Check for store

WHEN EXISTS(SELECT \* FROM Purchasing.Vendor AS v

WHERE v.BusinessEntityID = @BusinessEntityID)

THEN 'Store Contact'

-- Check for individual consumer

WHEN EXISTS(SELECT \* FROM Sales.Customer AS c

WHERE c.PersonID = @BusinessEntityID)

THEN 'Consumer'

END;

-- Return the information to the caller

IF @BusinessEntityID IS NOT NULL

BEGIN

INSERT @retContactInformation

SELECT @BusinessEntityID, @FirstName, @LastName, @ContactType;

END;

RETURN;

END;

GO

--------------------------------------------------------------------

SELECT BusinessEntityID, FirstName, LastName, ContactType

FROM dbo.GetContactInformation(2200);

GO

SELECT BusinessEntityID, FirstName, LastName, ContactType

FROM dbo.GetContactInformation(5);

### G. Using CASE in a HAVING clause

The following example uses the CASE expression in a HAVING clause to restrict the rows returned by the SELECT statement. The statement returns the the maximum hourly rate for each job title in the HumanResources.Employee table. The HAVING clause restricts the titles to those that are held by men with a maximum pay rate greater than 40 dollars or women with a maximum pay rate greater than 42 dollars.

USE AdventureWorks2008R2;

GO

SELECT JobTitle, MAX(ph1.Rate)AS MaximumRate

FROM HumanResources.Employee AS e

JOIN HumanResources.EmployeePayHistory AS ph1 ON e.BusinessEntityID = ph1.BusinessEntityID

GROUP BY JobTitle

HAVING (MAX(CASE WHEN Gender = 'M'

THEN ph1.Rate

ELSE NULL END) > 40.00

OR MAX(CASE WHEN Gender = 'F'

THEN ph1.Rate

ELSE NULL END) > 42.00)

ORDER BY MaximumRate DESC;