

**Tikrit University**  
**College of Computer Science and Mathematics**  
**Department of Computer Science**

Flow Control ( if – else ), Repetition with loops ( while )

**Lec7**

**2<sup>nd</sup> Stage**

**Assistant teacher. Mustafa Latif**

## Understanding SQL Server IF statements

SQL Server IF statement provides a way to execute code blocks based on specific conditions. This control-of-flow statement allows you to handle different scenarios and make decisions within your SQL Server scripts or stored procedures. The basic syntax of the SQL Server IF statement is simple:

```
IF condition
  BEGIN
    -- code block to execute if the condition is true
  END;
```

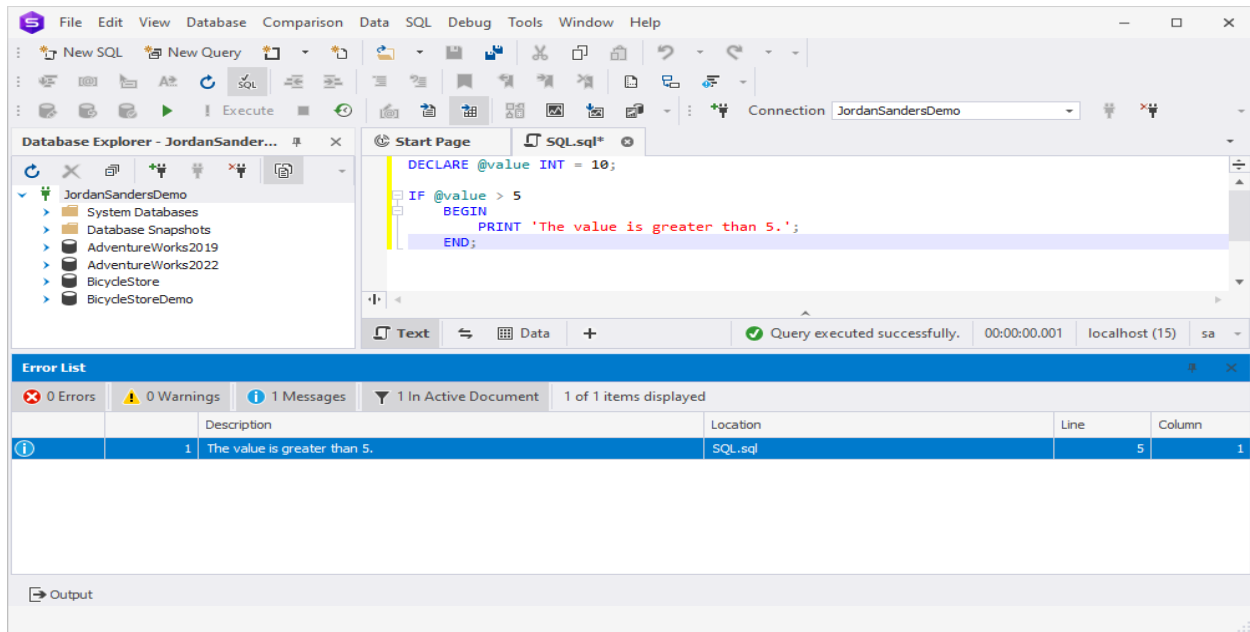
In the syntax above, the condition is an expression that evaluates to either true or false. If the condition evaluates to true, the code block within the BEGIN and END keywords will be executed.

Here is an example that demonstrates the usage of an SQL Server IF statement:

```
DECLARE @value INT = 10;

IF @value > 5
  BEGIN
    PRINT 'The value is greater than 5.';
  END;
```

In this example, the variable *@value* is assigned a value of 10. The IF statement checks if it is greater than 5. Since the condition is true, the message *'The value is greater than 5.'* is printed after the query is executed. In dbForge Studio for SQL Server, the result will be displayed in the **Messages** tab of the **Error List**:



## SQL Server IF-ELSE statements

To address the situation and obtain a response for both scenarios, where the condition holds true or false value, we will incorporate an IF-ELSE statement into our query:

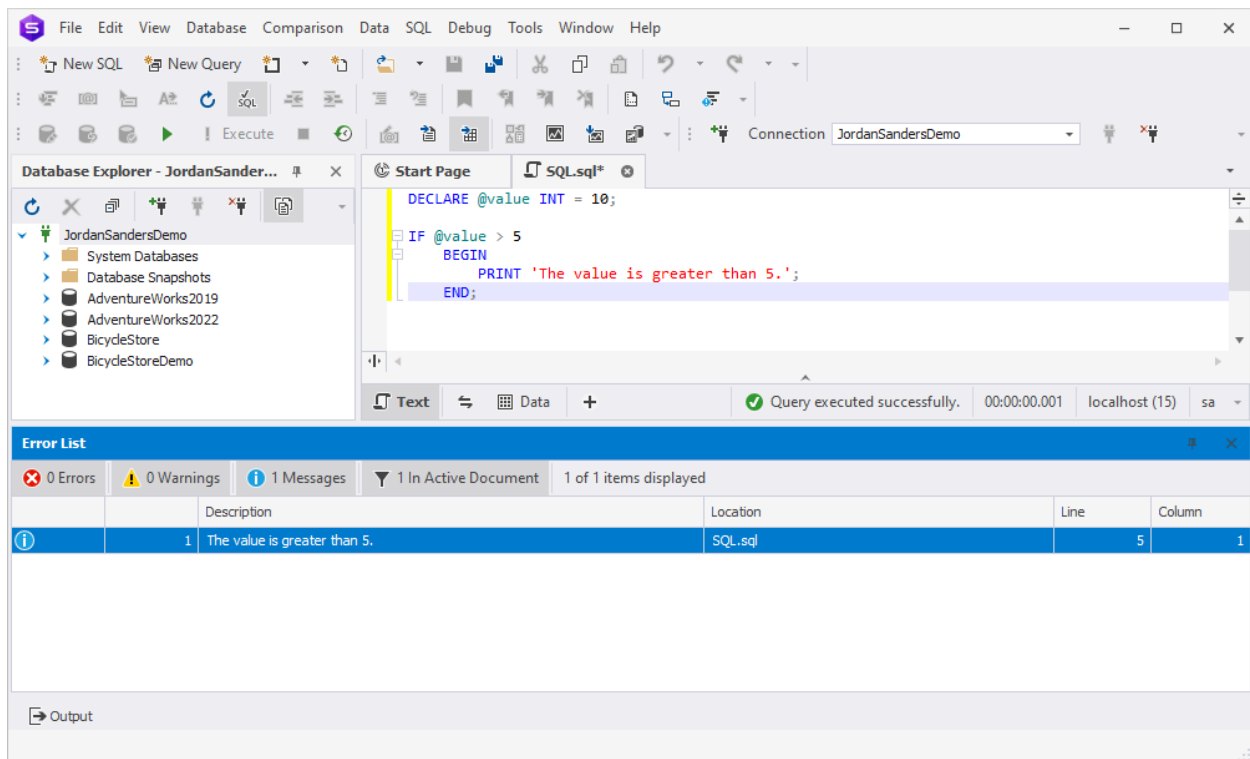
```
IF condition
BEGIN
    -- code block to execute if the condition is true
END
ELSE
BEGIN
    -- code block to execute if condition is false
END;
```

As you can see, the basic syntax looks pretty much the same, except there is one additional BEGIN-END clause that will be executed in case the condition does not end up being true.

Now, getting back to our *Product* table: let us now execute a query with a deliberately false condition:

```
WHERE Price > @Threshold
)
BEGIN
    PRINT 'There are products with prices greater than $' +
    CAST(@Threshold AS VARCHAR);
END
ELSE
BEGIN
    PRINT 'No products with prices greater than $' +
    CAST(@Threshold AS VARCHAR);
END;
```

As a result, SQL Server has no problem notifying us in both cases: when the specified condition is true and when it is false.



The screenshot displays the SQL Server Enterprise Edition interface. The main window shows a query editor with the following code:

```
DECLARE @value INT = 10;
IF @value > 5
BEGIN
    PRINT 'The value is greater than 5.';
END;
```

The status bar at the bottom of the query editor indicates "Query executed successfully. 00:00:00.001 localhost (15) sa". Below the query editor, the Error List pane is visible, showing one message:

Icon	Description	Location	Line	Column
Information	1 The value is greater than 5.	SQL.sql	5	1

The Output pane at the bottom is currently empty.

## SQL Server Loops

Now we're ready to move to SQL Server loops. We have 1 loop at our disposal, and that is the WHILE loop. You might think why we don't have others too, and the answer is that the WHILE loop will do the job. First, we'll take a look at its syntax.


```
WHILE {condition holds}
BEGIN
{...do something...}
END;
```

As you could easily conclude – while the loop conditions are true, we'll execute all statements in the **BEGIN ... END** block. Since we're talking about SQL Server loops, we have all SQL statements at our disposal, and we can use them in the WHILE loop as we like.

Let's now take a look at the first example of the WHILE loop.

```
DECLARE @i INTEGER;
SET @i = 1;

WHILE @i <= 10
BEGIN
    PRINT CONCAT ('Pass...', @i);
    SET @i = @i + 1;
END;
```



The screenshot shows a SQL Server Messages window with a zoom level of 100%. The window title is "Messages" and it contains the following output:

```
Pass...1
Pass...2
Pass...3
Pass...4
Pass...5
Pass...6
Pass...7
Pass...8
Pass...9
Pass...10
```

Two keywords – **BREAK** and **CONTINUE**, are present in most programming languages. Same stands for SQL Server loops. The idea is the following:

- When you encounter the **BREAK** keyword in the loop, you simply disregard all statements until the end of the loop (don't execute any) and exit the loop (not going to the next step, even if the loop condition holds)
- The **CONTINUE** acts similar to **BREAK** – it disregards all statements until the end of the loop, but then continues with the loop

```
DECLARE @i INTEGER;
SET @i = 1;

WHILE @i <= 10
BEGIN
    PRINT CONCAT('Pass...', @i);
    IF @i = 9 BREAK;
    SET @i = @i + 1;
END;
```



You can notice that placing a **BREAK** in the loop resulted that we exited the loop in the 9<sup>th</sup> pass.

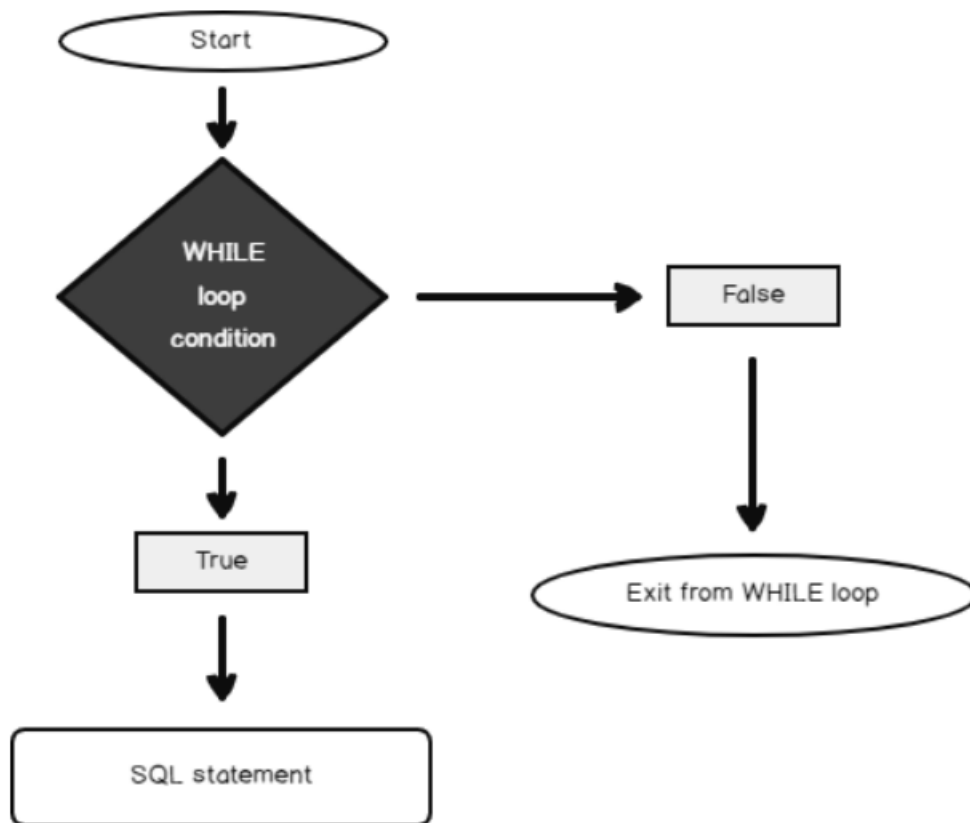


## SQL WHILE loop

SQL WHILE loop provides us with the advantage to execute the SQL statement(s) repeatedly until the specified condition result turn out to be false.

In the following sections of this article, we will use more flowcharts in order to explain the notions and examples. For this reason, firstly, we will explain what a flowchart is briefly. The flowchart is a visual geometric symbol that helps to explain algorithms visually. The flowchart is used to simply design and document the algorithms. In the flowchart, each geometric symbol specifies different meanings.

The following flowchart explains the essential structure of the WHILE loop in SQL:



As you can see, in each iteration of the loop, the defined condition is checked, and then, according to the result of the condition, the code flow is determined. If the result of the condition is true, the SQL statement will be executed. Otherwise, the code flow will exit the loop. If any SQL statement exists outside the loop, it will be executed.



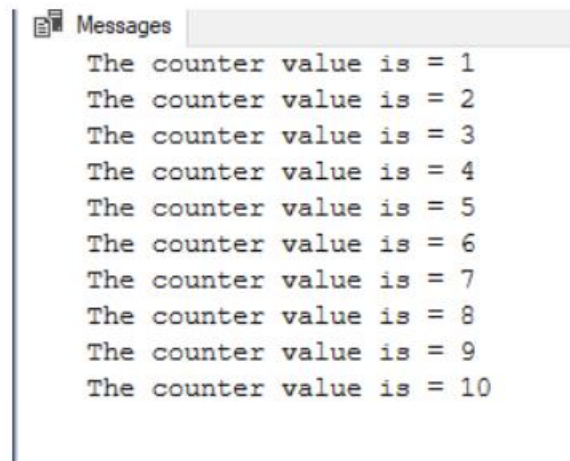
## SQL WHILE loop syntax and example

The syntax of the WHILE loop in SQL looks like as follows:

```
WHILE condition
BEGIN
  {...statements...}
END
```

After these explanations, we will give a very simple example of a WHILE loop in SQL. In the example given below, the WHILE loop example will write a value of the variable ten times, and then the loop will be completed:

```
DECLARE @Counter INT
SET @Counter=1
WHILE ( @Counter <= 10)
BEGIN
  PRINT 'The counter value is = ' + CONVERT(VARCHAR,@Counter)
  SET @Counter = @Counter + 1
END
```



The screenshot shows a window titled "Messages" with a list of ten lines of output. Each line reads "The counter value is = " followed by a number from 1 to 10. The output is displayed in a monospaced font, typical of a SQL Server command window.

Now, we will handle the WHILE loop example line by line and examine it with details.

In this part of the code, we declare a variable, and we assign an initializing value to it:

```
DECLARE @Counter INT
```

```
SET @Counter=1
```

This part of the code has a specified condition that until the variable value reaches till 10, the loop continues and executes the PRINT statement. Otherwise, the while condition will not occur, and the loop will end:

```
WHILE ( @Counter <= 10)
```

In this last part of the code, we executed the SQL statement, and then we incremented the value of the variable:

```
BEGIN
```

```
PRINT 'The counter value is = ' + CONVERT(VARCHAR,@Counter)
```

```
SET @Counter = @Counter + 1
```

```
END
```

The following flowchart illustrates this WHILE loop example visually:

