SQL Variables: Basics and usage

November 18, 2019 by Esat Erkec

In this article, we will learn the notions and usage details of the SQL variable. In SQL Server, local variables are used to store data during the batch execution period. The local variables can be created for different data types and can also be assigned values. Additionally, variable assigned values can be changed during the execution period. The life cycle of the variable starts from the point where it is declared and has to end at the end of the batch. On the other hand, If a variable is being used in a stored procedure, the scope of the variable is limited to the current stored procedure. In the next sections, we will reinforce this theoretical information with various examples

Note: In this article examples, the sample AdventureWorks database is used.

SQL Variable declaration

The following syntax defines how to declare a variable:

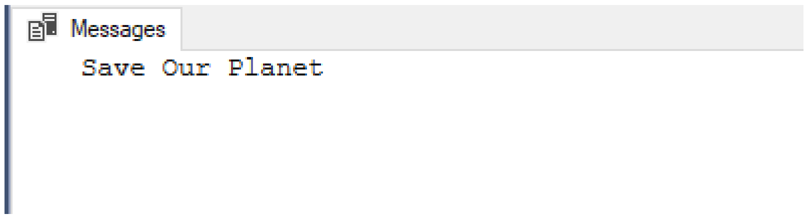
```
DECLARE { @LOCAL_VARIABLE data_type [ = value ] }
```

Now, let's interpret the above syntax.

Firstly, if we want to use a variable in SQL Server, we have to declare it. The DECLARE statement is used to declare a variable in SQL Server. In the second step, we have to specify the name of the variable. Local variable names have to start with an at (@) sign because this rule is a syntax necessity. Finally, we defined the data type of the variable. The value argument which is indicated in the syntax is an optional parameter that helps to assign an initial value to a variable during the declaration. On the other hand, we can assign or replace the value of the variable on the next steps of the batch. If we don't make any initial value assigned to a variable, it is initialized as NULL.

The following example will declare a variable whose name will be @VarValue and the data type will be varchar. At the same time, we will assign an initial value which is 'Save Our Planet':

```
DECLARE @TestVariable AS VARCHAR(100)='Save Our Planet'
PRINT @TestVariable
```

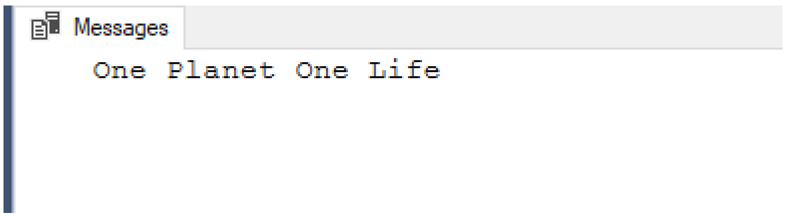

Messages

```
Save Our Planet
```

Assigning a value to SQL Variable

SQL Server offers two different methods to assign values into variables except for initial value assignment. The first option is to use the SET statement and the second one is to use the SELECT statement. In the following example, we will declare a variable and then assign a value with the help of the SET statement:

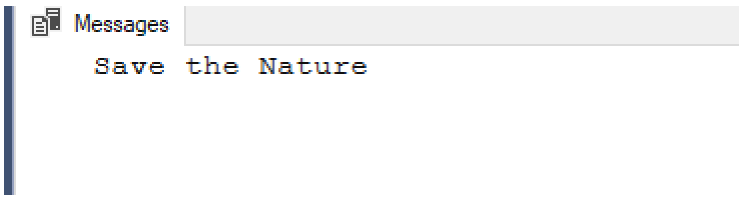
```
DECLARE @TestVariable AS VARCHAR(100)
SET @TestVariable = 'One Planet One Life'
PRINT @TestVariable
```


Messages

```
One Planet One Life
```

In the following example, we will use the SELECT statement in order to assign a value to a variable:

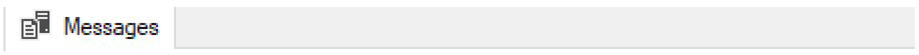
```
DECLARE @TestVariable AS VARCHAR(100)
SELECT @TestVariable = 'Save the Nature'
PRINT @TestVariable
```

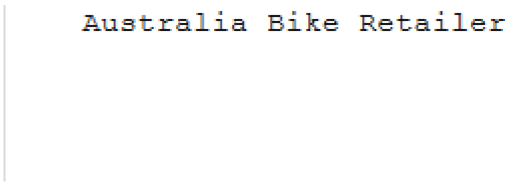

Messages

```
Save the Nature
```

Additionally, the SELECT statement can be used to assign a value to a variable from table, view or scalar-valued functions. Now, we will take a glance at this usage concept through the following example:

```
DECLARE @PurchaseName AS NVARCHAR(50)
SELECT @PurchaseName = [Name]
FROM [Purchasing].[Vendor]
WHERE BusinessEntityID = 1492
PRINT @PurchaseName
```


Messages

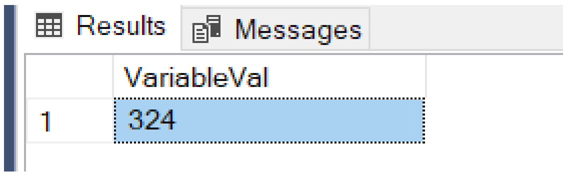


```
Australia Bike Retailer
```

As can be seen, the @PurchaseName value has been assigned from the Vendor table.

Now, we will assign a value to variable from a scalar-valued function:

```
DECLARE @StockVal AS INT
SELECT @StockVal=dbo.ufnGetStock(1)
SELECT @StockVal AS [VariableVal]
```

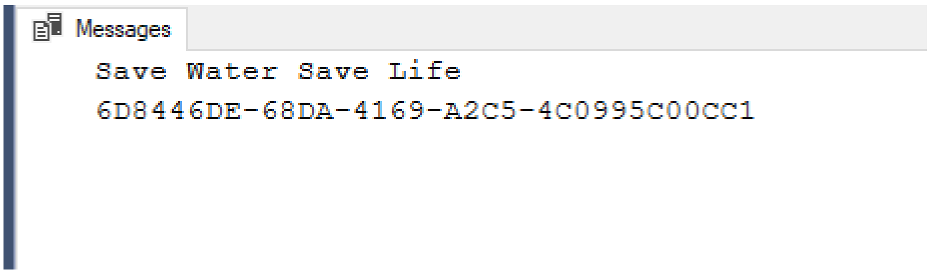


	VariableVal
1	324

Multiple SQL Variables

For different cases, we may need to declare more than one variable. In fact, we can do this by declaring each variable individually and assigned a value for every parameter:

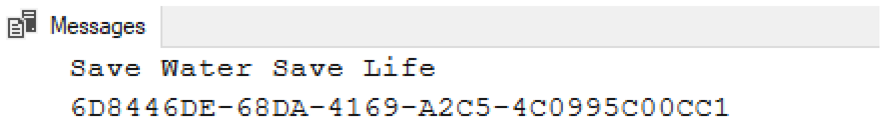
```
DECLARE @Variable1 AS VARCHAR(100)
DECLARE @Variable2 AS UNIQUEIDENTIFIER
SET @Variable1 = 'Save Water Save Life'
SET @Variable2= '6D8446DE-68DA-4169-A2C5-4C0995C00CC1'
PRINT @Variable1
PRINT @Variable2
```



```
Save Water Save Life
6D8446DE-68DA-4169-A2C5-4C0995C00CC1
```

This way is tedious and inconvenient. However, we have a more efficient way to declare multiple variables in one statement. We can use the DECLARE statement in the following form so that we can assign values to these variables in one SELECT statement:

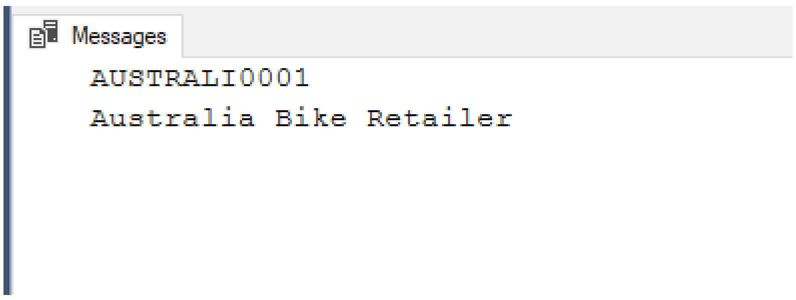
```
DECLARE @Variable1 AS VARCHAR(100), @Variable2 AS UNIQUEIDENTIFIER
SELECT @Variable1 = 'Save Water Save Life' , @Variable2= '6D8446DE-68DA-4169-A2
C5-4C0995C00CC1'
PRINT @Variable1
PRINT @Variable2
```



```
Save Water Save Life
6D8446DE-68DA-4169-A2C5-4C0995C00CC1
```

Also, we can use a SELECT statement in order to assign values from tables to multiple variables:

```
DECLARE @VarAccountNumber AS NVARCHAR(15)
,@VariableName AS NVARCHAR(50)
SELECT @VarAccountNumber=AccountNumber , @VariableName=Name
FROM [Purchasing].[Vendor]
WHERE BusinessEntityID = 1492
PRINT @VarAccountNumber
PRINT @VariableName
```

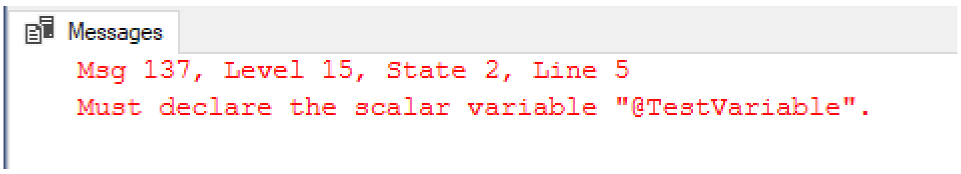


```
AUSTRALI0001
Australia Bike Retailer
```

Useful tips about the SQL Variables

Tip 1: As we mentioned before, the local variable scope expires at the end of the batch. Now, we will analyze the following example of this issue:

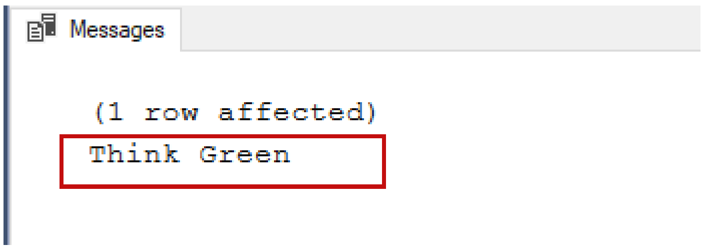
```
DECLARE @TestVariable AS VARCHAR(100)
SET @TestVariable = 'Think Green'
GO
PRINT @TestVariable
```



```
Msg 137, Level 15, State 2, Line 5
Must declare the scalar variable "@TestVariable".
```

The above script generated an error because of the GO statement. GO statement determines the end of the batch in SQL Server thus @TestVariable lifecycle ends with GO statement line. The variable which is declared above the GO statement line can not be accessed under the GO statement. However, we can overcome this issue by carrying the variable value with the help of the temporary tables:

```
IF OBJECT_ID('tempdb..#TempTbl') IS NOT NULL DROP TABLE #TempTbl
DECLARE @TestVariable AS VARCHAR(100)
SET @TestVariable = 'Hello World'
SELECT @TestVariable AS VarVal INTO #TempTbl
GO
DECLARE @TestVariable AS VARCHAR(100)
SELECT @TestVariable = VarVal FROM #TempTbl
PRINT @TestVariable
```

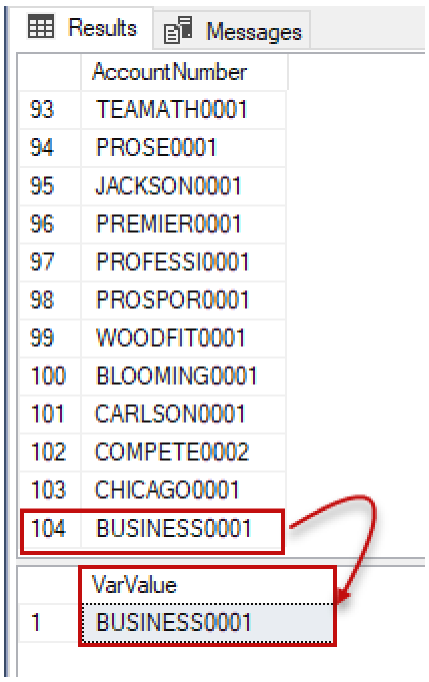


```
(1 row affected)
Think Green
```

Tip 2: Assume that, we assigned a value from table to a variable and the result set of the SELECT statement returns more than one row. The main issue at this point will be which row value is assigned to the variable. In this circumstance, the assigned value to the variable will be the last row of the resultset. In the following example, the last row of the resultset will be assigned to the variable:

```
SELECT AccountNumber
FROM [Purchasing].[Vendor]
ORDER BY BusinessEntityID

DECLARE @VarAccountNumber AS NVARCHAR(15)
SELECT @VarAccountNumber=AccountNumber
FROM [Purchasing].[Vendor]
order by BusinessEntityID
SELECT @VarAccountNumber AS VarValue
```



	AccountNumber
93	TEAMATH0001
94	PROSE0001
95	JACKSON0001
96	PREMIER0001
97	PROFESSI0001
98	PROSPOR0001
99	WOODFIT0001
100	BLOOMING0001
101	CARLSON0001
102	COMPETE0002
103	CHICAGO0001
104	BUSINESS0001

	VarValue
1	BUSINESS0001

Tip 3: If the variable declared data types and assigned value data types are not matched, SQL Server makes an implicit conversion in the value assignment process, if it is possible. The lower precedence data type is converted to the higher precedence data type by the SQL Server but this operation may lead to data loss. For the following example, we will assign a float value to the variable but this variable data type has declared as an integer:

```
DECLARE @FloatVar AS FLOAT = 12312.1232
DECLARE @IntVar AS INT
SET @IntVar=@FloatVar
PRINT  @IntVar
```