SQL Friendly Database Insert and Update Process

Disclaimer: This article if read without prior experience may end up boring and dry.

T-SQL Insert and Update are part of the Data Manipulation Language (DML).

http://technet.microsoft.com/en-us/library/ff848766.aspx

The biggest bane with T-SQL DML belongs to an issue so common that 100% of programmers will encounter it in the first place - the reason being is that we had expected the T-SQL to be more <strike>intelligent</strike> friendly, but it didn’t.

As the matter of fact, T-SQL is not that friendly and it requires one to follow specific rules.

The biggest bane is to perform DML with string data that consist of single quotes; it will result in exception because under T-SQL, single quote is considered to be a control character or reserved character. Hence, when a string data consists of single quote such as Brandon's PC, it will create confusion.

The best practice solution is to perform what is known as single quote offset.

i.e

Suppose I want to perform INSERT into a table called EmployeeTBL

```
INSERT INTO EmployeeTBL (Name,Position) VALUEs ('Brandon', 'General Manager')
```

When single quote is required to be present as part of the data, it will go like this supposedly.

```
INSERT INTO EmployeeTBL (Name,Position) VALUEs ('Michelle', 'Brandon's PA')
```

Nevertheless, the above T-SQL statement will generate exception; 'Incorrect syntax near ...'

The solution to perform single quote offset by adding an additional single quote in front of the single quote which belongs to the string data, hence:

```
INSERT INTO EmployeeTBL (Name,Position) VALUEs ('Michelle', 'Brandon''s PA')
```

Notice that now Brandon''s PA has two single quotes - this is what we mean by offsetting, it will work well for T-SQL.

Today, I just want to go further to discuss some strategies to deal with this issue on a broader perspective.

Strategy:

Assuming that every little process incurs additional CPU resources, hence the process of performing single quote offset will eventually incurring a considerable amount of CPU resources and hence slow things down.

Therefore, the strategy is not to perform single quote offset all the time, except only when necessary.

To do that, we need to identify the types of data - there are 4 types:

1.) Data expected to have single quote

2.) Data may or may not have single quote - user-defined data

3.) Data not expected to have single quote except when corrupted - unmanaged data

4.) Data that would not have single quote because it is self managed - managed data

To allocate considerations in order to minimize CPU resources during the process of offsetting single quote in the data, the strategies are:

i.) For data type 1, perform single quote offset all the time.

ii.) For data types 2,3,4, perform single quote offset whenever exception occurs.

In other words, for data types 2,3,4; perform Insert, Update using original data first, if exception occurs, then perform single quote offset on the data and retry again.

Doing this, it will ensure process optimization. Assuming that exceptions only occur 10% of the time, this would mean that the system is 90% optimal and 10% normal.

Otherwise, the system would end up 90% inefficient and 10% normal.

Ok. To cut the long story short, let me throw everything on the table first.

Other issues that are common:

1.) Insert and Update DML process must also ensure that the string data size is not bigger than the size of the column, otherwise, one my end up getting 'String or binary data would be truncated.'

Hence, a SQL friendly process must also check for the data size to ensure compatibility with the column's size.

2.) How to check for data size, perform single quote offset and ensure that the object-relational is as scalable as possible ?

To cut things shorter, I will just present a sample code in VB.NET which is self explanatory.

```
Public Class clsTrans

        Private position As String = ""

        Public Property VAR_position() As String
          Get
             VAR_position = position
          End Get
            Set(ByVal value As String)
            position = value
          End Set
        End Property


  Public Function Get_position_1(ByVal p_Length As Integer, ByVal p_OffSQuote As Integer) As String

        Try

        If position.Length > 0 Then

          If position.Length > p_Length Then

            If p_OffSQuote = 1 Then
              Return FormatSQLFriendlyStr_Glob(Microsoft.VisualBasic.Left(position,
p_Length)).ToString
            Else
                Return Microsoft.VisualBasic.Left(position, p_Length)
             End If 'If p_OffSQuote = 1

           Else

              If p_OffSQuote = 1 Then
                  Return FormatSQLFriendlyStr_Glob(position).ToString
              Else

                Return position
               End If 'If p_OffSQuote = 1

             End If 'If position.Length >= p_Length

        Else
             '--returns original ---
             Get_position_1 = position
        End If 'If position.Length > 0

        Catch ex As Exception

          '--returns original ---
          Get_position_1 = position
          call Debug.Write(ex)

        End Try

        End Function

   End Class
```

```
Module Module1

         '-- This method looks for occurrence of single quote and replace it with double single quotes
----
         Public Function FormatSQLFriendlyStr_Glob(ByVal tSource As Object) As Object

           Dim tempStr As String = ""
             Dim newStr As String = ""

             FormatSQLFriendlyStr_Glob = "" 'init

             Try

               tempStr = tSource.ToString
             'replace a single quote with a double single quotes
            newStr = tempStr.Replace(Chr(39), Chr(39) & Chr(39))
         Catch ex As Exception

             FormatSQLFriendlyStr_Glob = ""
             call Debug.Write(ex)

       Finally
```

```
            FormatSQLFriendlyStr_Glob = newStr

        End Try

        End Function

End Module
```

```
Public Class clsProcess

  Private Sub InsertData(byval objTrans as clsTrans)

    Dim objManager as new clsManager
    Dim tsql as string = ""
    Dim tsuccess as boolean

    'assuming the column size for Position column is 10 chars; the second param indicates not to
offset single quote first

    tsql="INSERT INTO EmployeeTBL (Name,Position) VALUES ('Brandon','" &
objTrans.Get_position_1(10,0) & "')

    tsuccess = objManager.InsertData(tsql)

   if not tsuccess then

     tsql="INSERT INTO EmployeeTBL (Name,Position) VALUES ('Brandon','" &
objTrans.Get_position_1(10,1) & "')

     'assuming that clsManager.InsertData is a method to connect to database and make use of the
tsql param to insert the record

     tsuccess = objManager.InsertData(tsql)

     if not tsuccess then
          msgbox "insert data failed even after single quote offset"

     else
          msgbox "insert data success"

     end if

   else
```

```
        msgbox "insert data success"

    end if

   End Sub

End Class
```

Conclusion:

1.) Usage of clsTrans.Get_position_1 to replace clsTrans.VAR_position ensures efficiency because it performs 3-in-1.

Otherwise, the following has to be done.

```
dim tempStr as string

tempStr=objTrans.VAR_position

tempStr=Microsoft.VisualBasic.Left(tempStr, 10)

tempStr=FormatSQLFriendlyStr_Glob(tempStr).toString
```

2.) Usage of clsTrans.Get_position_1 caters for flexibility because the original value for clsTrans.VAR_position is retained.

3.) Finally, usage of clsTrans.Get_position_1 caters for SQL friendly implementation.