UNIVERSIDADE ABERTA

Projeto de Engenharia Informática

t-SQL STRING FUNCTIONS

Relatório Final

Responsável: Professora Gracinda Carvalho
Orientador: Professor Luís Cavique

Pedro Miguel Almeida Costa
Nº 1001237

Email: 1001237@estudante.uab.pt
Tel: 913795500

Índice

Proposta

Para a realização do Projeto de Engenharia Informática proponho a criação de uma função STRING e de uma função FIND em transact-SQL com a seguinte abordagem:

Notas Introdutórias

As funções deverão ser desenvolvidas para execução em qualquer versão de SQL Server a partir da versão SQL Server 2008.

Em cada versão de SQL dever-se-á tirar partido das melhores ferramentas disponíveis que permitam a execução da função com a melhor performance e controlo possíveis, recorrendo, por exemplo a

Try_Convert/Try_Cast	(SQL Server 2012)
String_Split	(SQL Server 2016)

No entanto, deverá ser claro que o objetivo das funções não será a performance, uma vez que a sua implementação será inevitavelmente baseada nas funções de SQL existentes, mas sim a simplicidade de leitura e facilidade de utilização correspondente na implementação de pesquisas e operações sobre strings.

Outras funções e/ou operações sobre as funções a desenvolver que se considerem interessantes poderão também ser implementadas.

Como exemplos da simplicidade de leitura e versatilidade de utilização que se pretende atingir junto 3 exemplos, entre as muitas opções, de operações a implementar:

Exemplo 1:

Pretendendo selecionar do campo TEXTO o terceiro carater temos:

Opção atual:

```
SUBSTRING(TEXTO, 3, 1)
```

Opção proposta:

```
STRING(TEXTO, '[3]')
```

Exemplo 2:

Pretendendo selecionar do campo TEXTO os caracteres 3 a 5, o 7, e os restantes a partir do 9:

Opção atual:

```
SUBSTRING(TEXTO, 3, 3) + SUBSTRING(TEXTO, 7, 1) + Case When LEN(TEXTO) >= 9 Then RIGHT(TEXTO, LEN(TEXTO) - 8) Else '' End
```

Opção proposta:

```
STRING(TEXTO, '[3:5,7,9:]')
```

Exemplo 3:

Pretendendo selecionar de um nome o Primeiro e Último com o formato Apelido, Nome:

Opção atual:

```
Select
  SUBSTRING(Name, LEN(Name) - CHARINDEX(' ', REVERSE(Name)) + 2, LEN(Name) - (LEN(Name) -
  CHARINDEX(' ', REVERSE(Name))) - 1) + ', ' + SUBSTRING(Name, 1, CHARINDEX(' ', Name) - 1)
From
  Clients
```

Opção proposta:

```
Select
  STRING(Name, '[' + FIND(Name, ' ', null, -1) + ':]',', ','[:' + FIND(Name, ' ') + ']')
From
  Clients
```

SQL – Função STRING

Contexto

Em linguagem de programação sobre SQL (Transact-SQL) é sempre complicado trabalhar com operações sobre strings, tendo frequentemente que se recorrer a funções como SubString, CharIndex, Stuff, Right, Left e afins, que tornam a vida de um programador nesta linguagem difícil.

Objetivo

Criação de uma função STRING que permita a aplicação fácil e intuitiva de um ou vários operadores sobre uma string à semelhança do que existe em linguagens orientadas a objetos e nas novas linguagens interpretadas como Python e R.

Definição

STRING (Transact-SQL)

Applies to: SQL Server (starting with 2008)

Returns a new string by applying operations over a string parameter.

Syntax

STRING (expression [, operator])

Arguments

expression

Is a character, text or ntext expression.

For expression of different data types, converts to character.

operator

Character optional argument that defines the operations patern to be applied to the expression

If the *operator* is not provided, the original expression is returned

Operator Types

[i]	Returns the character on the ith position
	i < 0: returns the character position of |i| counting backwards from the end
[i_1:i_2]	Returns the characters between the i_1 and i_2 positions
	i_1 empty: returns all the characters up until i_2
	i_2 empty: returns all the characters starting from i_1
	i_1 and i_2 empty: returns all the characters
	i_1 or i_2 < 0: counts the position of |i| backwards from the end
	i_1 > i_2: returns the characters in reverse from right to left
[o_1,o_2]	Returns the concatenation of the results from indexing operator o_1 and o_2
o_1,o_2	Returns the concatenation of the results from operator o_1 and o_2
s	Returns the string s

Return Types

Returns character data as:

Specified expression	Return Type
char/varchar/text	varchar
nchar/nvarchar/ntext	nvarchar

Remarks

If the expression is null or empty or the result of applying the operators results in error, returns empty.

To include the ‘,’ as text, the comma should be preceeded by the \ (backslash) character as ‘\,’.

The ith operator are int values. If numeric they are rounded to int. If non numeric the operation is not considered.

The ith positions are 1 based.

Examples

A. Returning the 3rd and 5th character from expression

```
STRING(expression, '[3,5]')
```

B. Returning the first 2 and last 2 characters from expression

```
STRING(expression, '[:2],[-2:]')
```

C. Returning the last 3 characters in reverse order from expression

```
STRING(expression, '[:-3]')
```

D. Returning formated phone numbers as +351 ## ### ## ## from clients with format #########

```
Select
  STRING(PhoneNumber, '+351,[:2], ,[3:5], ,[6:7], ,[8:9]')
From
  Clients
```

E. Returning the last name from clients name

```
Select
  STRING(Name, '[' + FIND(Name, ' ', null, -1) + ':]' )
From
  Clients
```

See also

FIND (Transact-SQL)

Estrutura

Parâmetros

Expressão
Operador

Lógica

Expressão Não Nula
Operador Não Nulo

Controla ‘,’ como texto em *Operador*

Separa *Operador* por ‘,’

Para cada operador (item)
Identifica se Operador de Indexação

Se operador de indexação

Se contém ‘:’

Determina operador1 : operador2

```
                        Se Leitura Normal (operador1 <= operador2)

                              Se operador1 ou operador2 Negativo
                                    operador1/2 = Conta posição no sentido inverso a partir do
                                    fim

                              Se Leitura de Intermeio (operador1 <= operador2)
                                    resultado junta parte de Expressão desde operador1 até
                                    operador2

                              Se Leitura de Início e Fim
                                    resultado junta Concatenação da parte de Expressão desde
                                    operador 1 até final com a parte de Expressão desde início
                                    até operador2

                        Se Leitura Inversa (operador1 > operador2)

                              Se operador1 ou operador2 Negativo
                                    operador1/2 = |operador1/2|

                              Se Leitura de Intermeio (operador1 <= operador2)
                                    resultado junta parte de Inverso de Expressão desde
                                    operador1 até operador2

                              Se Leitura de Início e Fim
                                    resultado junta Concatenação da parte de Inverso de
                                    Expressão desde operador 1 até final com a parte de Inverso
                                    de Expressão desde início até operador2

                  Se Não contém ':'

                        Determina operador1

                        Se operador1 Negativo
                              operador1 = Conta posição no sentido inverso a partir do fim

                        resultado junta Caracter da posição operador1 de Expressão

            Se operador não é de indexação (texto livre)

                  resultado junta operador

Controla ',' em resultado

Retorna resultado
```

Especificação

Parâmetros

```
@pExpression Varchar(max)
@pOperator Varchar(max)
```

Algoritmo

```
@pExpression Não Nulo
@pOperator Não Nulo

Controla ',' como texto em @pOperator
```

```
  Separa @pOperator por ‘,’

       Para cada operador (item)
              Se começa em ‘[‘ é início de operador de indexação
              Se termina em ‘]’ é fim de operador de indexação

              Se operador de indexação

                     Se contém ‘:’

                            operador1 = Number(operador até ‘:’)
                            operador2 = Number(operador de ‘:’ até final)

                            operador1 defaults to 1
                            operador2 defaults to Length(@pExpression)

                            Se operador1 <= operador2

                                   Se operador1 < 0
                                          operador1 = Length(@pExpression) + operador1 + 1

                                   Se operador2 < 0
                                          operador2 = Length(@pExpression) + operador2 + 1

                                   Se operador1 <= operador2
                                          resultado += @pExpression desde operador1 até operador2
                                   Senão
                                          resultado += @pExpression desde operador 1 até final e
                                          desde início até operador2

                            Senão

                                   Se operador1 < 0
                                          operador1 = |operador1|

                                   Se operador2 < 0
                                          operador2 = |operador2|

                                   Se operador1 <= operador2
                                          resultado += Inverso de @pExpression desde operador1 até
                                          operador2
                                   Senão
                                          resultado += Inverso de @pExpression desde operador 1 até
                                          final e desde início até operador2

                     Se Não contém ‘:’

                            operador1 = Number(operador)

                            Se operador1 < 0
                                   operador1 = Length(@pExpression) + operador1 + 1

                            resultado += @pExpression[operador1]


              Se operador não é de indexação

                     resultado += operador

Controla ‘,’ em resultado

Retorna resultado
```

Implementação

```
@pExpression Null ? -> return ''
@pOperator Null ? -> return ''

@pOperator.Contains('\,')
     @CtrlComa = true
     @pOperator.Replace('\,', '\#;#')

ForEach( operator = @pOperator.Split(',') )

     operator[1] = '[' ?
            startIndex

     operator[end] = ']' ?
            endIndex

     Index ? (Between startIndex and endIndex)

            operator.Contains(':') ?
                   operator1 = Number( operator[1:':'] )
                   operator2 = Number( operator[':':end] )

                   operator1 Null ? -> Not a Number
                   operator2 Null ? -> Not a Number

                   operator1 = 0 ? <- 1
                   operator2 = 0 ? <- Len( @pExpression )

                   operator1 <= operator2 ?
                   {
                          operator1 < 0 ? <- Len( @pExpression ) + operator1 + 1
                          operator2 < 0 ? <- Len( @pExpression ) + operator2 + 1

                          operator1 <= operator2 ?
                                 result += Substring( @pExpression, operator1, operator2 -
                                        operator1 + 1 )
                          else
                                 result += Substring( @pExpression, operator1, Len( @pExpression )
                                        - operator1 + 1 )
                                 result += Substring( @pExpression, 1, operator2 )
                   }
                   else
                   {
                          operator1 < 0 ? <- Abs( operator1 )
                          operator2 < 0 ? <- Abs( operator2 )

                          operator1 <= operator2 ?
                                 result += Substring( Reverse( @pExpression ), operator1, operator2
                                        - operator1 + 1 )
                          else
                                 result += Substring( Reverse( @pExpression ), operator1, Len(
                                        @pExpression ) - operator1 + 1 )
                                 result += Substring( Reverse( @pExpression ), 1, operator2 )
                   }

            operator Not Null
                   operator1 = Number( operator )
```

```
                    operator1 Null ? -> Not a Number

                    operator1 < 0 ? <- Len( @pExpression ) + operator1 + 1

                    result += Substring( @pExpression, operator1, 1 )

      Not Index ?

            result += operator

  @CtrlComa = true ?
      result.Replace('\#;#', ',')

  return result
```

Código - SQL Server (starting from 2012)

```
-- ================================================================================================
-- Description: STRING SQL Server Function
-- Author: Pedro Costa
-- Create date: 2018-06-24
-- Version: SQL Server (starting with 2012)
-- ================================================================================================
Create Function dbo.STRING
(
  @pExpression Varchar(max),
  @pOperator   Varchar(max)
)
Returns Nvarchar( max )
As
Begin
  Declare @result Varchar(max) = ''

  If IsNull( @pExpression, '' ) = ''
    Return @result

  If IsNull( @pOperator, '' ) = ''
    Return @pExpression

  Declare @revExpression Varchar(max) = Reverse( @pExpression )
  -- Assures The Len Expression Value Without the Automatic Trim
  Declare @lenExpression Bigint = Len( Replace(@pExpression, ' ', '.'))

  Declare @CtrlComa Bit = 0
  If CharIndex( '\,', @pOperator ) > 0
  Begin
    Set @CtrlComa = 1
    Set @pOperator = Replace(@pOperator, '\,', '\#;#')
  End

  Declare @operator Varchar(max) = @pOperator + ','

  Declare @i Int
  Set @i = Charindex( ',', @operator )

  Declare @index Bit = 0

  Declare @j Int
  Declare @o Varchar(max)
  Declare @o1 Varchar(max),
          @o2 Varchar(max)
  Declare @j1 Int,
          @j2 Int

  While @i > 0
  Begin
    Set @o = Substring( @operator, 1, @i - 1 )

    If Substring( @o, 1, 1 ) = '['
    Begin
      Set @index = 1
      Set @o = Substring( @o, 2, Len( @o ) - 1 )
    End

    If @index = 1
```

```
    Begin
      If Substring( @o, Len( @o ), 1 ) = ']'
      Begin
        Set @index = 0
        Set @o = Substring( @o, 1, Len( @o ) - 1 )
      End

      Set @j = Charindex( ':', @o )
      If @j > 0
      Begin
        Set @o1 = Substring( @o, 1, @j - 1 )
        Set @o2 = Substring( @o, @j + 1, @i - @j - 1 )

        Set @j1 = Try_Cast( @o1 As Int )
        Set @j2 = Try_Cast( @o2 As Int )

        If @j1 Is Null
          Set @j1 = Round( Try_Cast( @o1 As Float ), 0 )

        If @j2 Is Null
          Set @j2 = Round( Try_Cast( @o2 As Float ), 0 )

        If @j2 = 0
          Set @j2 = @lenExpression

        If Not( @j1 > @j2 )
        Begin
          If @j1 < 0
            Set @j1 = @lenExpression + @j1 + 1

          If @j2 < 0
            Set @j2 = @lenExpression + @j2 + 1

          If Not( @j1 > @j2 )
            Set @result += Substring( @pExpression, @j1, @j2 - @j1 + 1 )
          Else
          Begin
            Set @result += Substring( @pExpression, @j1, @lenExpression - @j1 + 1 )
            Set @result += Substring( @pExpression, 1, @j2 )
          End
        End
        Else
        Begin
          If @j1 < 0
            Set @j1 = Abs( @j1 )

          If @j2 < 0
            Set @j2 = Abs( @j2 )

          If Not( @j1 > @j2 )
            Set @result += Substring( @revExpression, @j1, @j2 - @j1 + 1 )
          Else
          Begin
            Set @result += Substring( @revExpression, @j1, @lenExpression - @j1 + 1 )
            Set @result += Substring( @revExpression, 1, @j2 )
          End
        End
      End
      Else
      Begin
        If IsNull( @o, '' ) != ''
        Begin
```

```
            Set @j1 = Try_Cast( @o As Int )

            If @j1 Is Null
              Set @j1 = Round( Try_Cast( @o As Float ), 0 )

            If @j1 Is Not Null
            Begin
              If @j1 < 0
                Set @j1 = @lenExpression + @j1 + 1

              Set @result += Substring( @pExpression, @j1, 1 )
            End
          End
        End
      End
      Else
      Begin
        Set @result += @o
      End

      Set @operator = Substring( @operator, @i + 1, Len( @operator ) - @i )
      Set @i = Charindex( ',', @operator )
    End

    If @CtrlComa = 1
      Set @result = Replace(@result, '\#;#', ',')

    Return @result
  End
```

Código - SQL Server (starting from 2016)

```
-- ================================================================================================
-- Description: STRING SQL Server Function
-- Author: Pedro Costa
-- Create date: 2018-06-24
-- Version: SQL Server (starting with 2016)
-- ================================================================================================
Create Function dbo.STRING
(
  @pExpression Varchar(max),
  @pOperator   Varchar(max)
)
Returns Nvarchar( max )
As
Begin
  Declare @result Varchar(max) = ''

  If IsNull( @pExpression, '' ) = ''
    Return @result

  If IsNull( @pOperator, '' ) = ''
    Return @pExpression

  Declare @revExpression Varchar(max) = Reverse( @pExpression )
  -- Assures The Len Expression Value Without the Automatic Trim
  Declare @lenExpression Bigint = Len( Replace(@pExpression, ' ', '.'))

  Declare @CtrlComa Bit = 0
  If CharIndex( '\,', @pOperator ) > 0
  Begin
    Set @CtrlComa = 1
    Set @pOperator = Replace(@pOperator, '\,', '\#;#')
  End

  Declare @id int = 0
  Declare @index Bit = 0

  Declare @j Int
  Declare @o Varchar(max)
  Declare @o1 Varchar(max),
          @o2 Varchar(max)
  Declare @j1 Int,
          @j2 Int

  Declare cur Cursor Local Forward_only Static Read_only For
    Select [value] From STRING_SPLIT(@pOperator, ',')

  Open cur
  Fetch Next From cur Into @o
  While @@fetch_status = 0
  Begin

    If Substring( @o, 1, 1 ) = '['
    Begin
      Set @index = 1
      Set @o = Substring( @o, 2, Len( @o ) - 1 )
    End

    If @index = 1
```

```
Begin
  If Substring( @o, Len( @o ), 1 ) = ']'
  Begin
    Set @index = 0
    Set @o = Substring( @o, 1, Len( @o ) - 1 )
  End

  Set @j = Charindex( ':', @o )
  If @j > 0
  Begin
    Set @o1 = Substring( @o, 1, @j - 1 )
    Set @o2 = Substring( @o, @j + 1, len(@o) - @j )

    Set @j1 = Try_Cast( @o1 As Int )
    Set @j2 = Try_Cast( @o2 As Int )

    If @j1 Is Null
      Set @j1 = Round( Try_Cast( @o1 As Float ), 0 )

    If @j2 Is Null
      Set @j2 = Round( Try_Cast( @o2 As Float ), 0 )

    If @j2 = 0
      Set @j2 = @lenExpression

    If Not( @j1 > @j2 )
    Begin
      If @j1 < 0
        Set @j1 = @lenExpression + @j1 + 1

      If @j2 < 0
        Set @j2 = @lenExpression + @j2 + 1

      If Not( @j1 > @j2 )
        Set @result += Substring( @pExpression, @j1, @j2 - @j1 + 1 )
      Else
      Begin
        Set @result += Substring( @pExpression, @j1, @lenExpression - @j1 + 1 )
        Set @result += Substring( @pExpression, 1, @j2 )
      End
    End
    Else
    Begin
      If @j1 < 0
        Set @j1 = Abs( @j1 )

      If @j2 < 0
        Set @j2 = Abs( @j2 )

      If Not( @j1 > @j2 )
        Set @result += Substring( @revExpression, @j1, @j2 - @j1 + 1 )
      Else
      Begin
        Set @result += Substring( @revExpression, @j1, @lenExpression - @j1 + 1 )
        Set @result += Substring( @revExpression, 1, @j2 )
      End
    End
  End
  Else
  Begin
    If IsNull( @o, '' ) != ''
    Begin
```

```
            Set @j1 = Try_Cast( @o As Int )

            If @j1 Is Null
              Set @j1 = Round( Try_Cast( @o As Float ), 0 )

            If @j1 Is Not Null
            Begin
              If @j1 < 0
                Set @j1 = @lenExpression + @j1 + 1

              Set @result += Substring( @pExpression, @j1, 1 )
            End
          End
        End
      End
      Else
      Begin
        Set @result += @o
      End

      Fetch Next From cur Into @o
    End
    Close cur
    Deallocate cur

    If @CtrlComa = 1
      Set @result = Replace(@result, '\#;#', ',')

    Return @result
  End
```

Exemplos de Execução

```
-- A. Returning the 3rd and 5th character from ProductNumber
Select ProductNumber, dbo.STRING(ProductNumber, '[3,5]') As Result From Production.Product
```

Results Messages

	ProductNumber	Result
1	AR-5381	-3
2	BA-8327	-3
3	BB-7421	-4
4	BB-8107	-1
5	BB-9108	-1
6	BC-M005	-0
7	BC-R205	-2
8	BE-2349	-3
9	BE-2908	-9
10	BK-M18B-40	-1
11	BK-M18B-42	-1
12	BK-M18B-44	-1

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 504 rows

Fig1: Exemplo A. STRING – A.jpg

```
-- B. Returning the first 2 and last 2 characters from ProductNumber
Select ProductNumber, dbo.STRING(ProductNumber, '[:2],[-2:]') As Result From Production.Product
```

Results Messages

	ProductNumber	Result
1	AR-5381	AR81
2	BA-8327	BA27
3	BB-7421	BB21
4	BB-8107	BB07
5	BB-9108	BB08
6	BC-M005	BC05
7	BC-R205	BC05
8	BE-2349	BE49
9	BE-2908	BE08
10	BK-M18B-40	BK40
11	BK-M18B-42	BK42
12	BK-M18B-44	BK44
13	BK-M18B-48	BK48

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 504 rows

Fig2: Exemplo B. STRING – B.jpg

```
-- C. Returning the last 3 characters in reverse order from Product Number
Select ProductNumber, dbo.STRING(ProductNumber, '[:-3]') As Result From Production.Product
```

Results | Messages

	ProductNumber	Result
1	AR-5381	183
2	BA-8327	723
3	BB-7421	124
4	BB-8107	701
5	BB-9108	801
6	BC-M005	500
7	BC-R205	502
8	BE-2349	943
9	BE-2908	809
10	BK-M18B-40	04-
11	BK-M18B-42	24-
12	BK-M18B-44	44-

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 504 rows

Fig3: Exemplo C. STRING – C.jpg

```
-- D. Returning formated Credit Card Number as ### ### ### ##-### from clients with format
###############
Select CardNumber, dbo.STRING(CardNumber, '[:3], ,[4:6], ,[7:9], ,[10,11],-,[12:14]') As Result From
Sales.CreditCard
```

Results | Messages

	CardNumber	Result
1	11111000471254	111 110 004 71-254
2	11111002034157	111 110 020 34-157
3	11111005230447	111 110 052 30-447
4	11111007955171	111 110 079 55-171
5	11111009772675	111 110 097 72-675
6	11111016029803	111 110 160 29-803
7	11111016740421	111 110 167 40-421
8	11111017091860	111 110 170 91-860
9	11111017448906	111 110 174 48-906
10	11111018232698	111 110 182 32-698
11	11111019008951	111 110 190 08-951
12	11111024201256	111 110 242 01-256
13	11111026083356	111 110 260 83-356
14	11111028981641	111 110 289 81-641
15	11111029667845	111 110 296 67-845

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:03 | 19118 rows

Fig4: Exemplo D. STRING – D.jpg

SQL – Função FIND

Contexto

Em linguagem de programação sobre SQL (Transact-SQL) não é versátil o trabalhar com pesquisas sobre strings, tendo que se recorrer à função CharIndex ou equivalentes.

Objetivo

Criação de uma função FIND que permita versatilidade e facilidade na pesquisa da existência e posição de uma expressão numa string.

Definição

FIND (Transact-SQL)

Applies to: SQL Server (starting with 2008)

Searches an expression for another expression and returns its starting position if found.

Syntax

FIND (expression, expressionToFind [, startLocation] [, ocorrenceNumber])

Arguments

expression

Is a character, text or ntext expression to be searched.

For expression of different data types, converts to character.

expressionToFind

Sequence to be found

startLocation

int expression at which the search starts. If negative or zero, the search starts at the beginning of expressionToSearch

ocorrenceNumber

int expression defining the number of the ocorrence to be found.

If ocorrenceNumber < 0 counts the ocorrence to be found backwards from the end of the expression.

Return Types

Returns int/bigint

Remarks

If the expression or expressionToFind is null, returns null.

If expressionToFind is not found within expression or the solicited number of ocorrence is greater then the number of ocorrences found, returns 0.

The startingLocation and ocorrenceNumber are 1 based.

Examples

A. Returning the starting position of '.' in the expression

```
FIND(expression, '.')
```

B. Returning the position of the second ocorrence of '.' in the expression, starting from position 10

```
FIND(expression, '.', 10, 2)
```

C. Returning the last position of '.' in the expression

```
FIND(expression, '.', null, -1)
```

Especificação

Parâmetros

```
@pExpressionToFind Varchar(max)
@pExpressionToSearch Varchar(max)
@pStartLocation Bigint
@pOcorrenceNumber Int
```

Default Values

```
@pStartLocation = 0
@pOcorrenceNumber = 1
```

Algoritmo

```
@pExpressionToFind Não Nulo
@pExpressionToSearch Não Nulo

Posição = @pStartLocation

Se @pOcorrenceNumber Negativo
     Inverte @pExpressionToFind
     Inverte @pExpressionToSearch

Enquanto Não @pOcorrenceNumber vezes
     Procura @pExpressionToFind em @pExpressionToSearch partindo de Posição
     Se encontrou
          Incrementa Posição pela Posição encontrada
     Se não encontrou
          Termina sem encontrar

Se @pOcorrenceNumber Negativo
     Retorna Length(@pExpressionToSearch) – Posição – Length(@pExpressionToFind)

Retorna Posição
```

Implementação

```
@pExpressionToFind Null ? -> return Null
@pExpressionToSearch Null ? -> return Null

@pStartLocation < 0 ? <- 0
Location <- @pStartLocation

@pOcorrenceNumber < 0 ?
     @pExpressionToFind <- Reverse( @pExpressionToFind )
     @pExpressionToSearch <- Reverse( @pExpressionToSearch )

For(int i = 0; i < Abs( @pOcorrenceNumber ); i++)
{
     location += Charindex( @pExpressionToFind, @pExpressionToSearch, location )
     location = 0 ? return 0
}

@pOcorrenceNumber < 0 ?
     location <- Len( @pExpressionToSearch ) - location + 1 - Len( @pExpressionToFind ) + 1

return location
```

Código - SQL Server (starting from 2012)

```
-- ================================================================================================
-- Description: FIND SQL Server Function
-- Author: Pedro Costa
-- Create date: 2018-06-24
-- Version: SQL Server (starting with 2012)
-- ================================================================================================
Create Function dbo.FIND
(
  @pExpressionToFind   Varchar(max),
  @pExpressionToSearch Varchar(max),
  @pStartLocation      Bigint = 0,
  @pOcorrenceNumber    Int = 1
)
Returns Bigint
As
Begin
  Declare @result Bigint

  If @pExpressionToFind Is Null
      Or IsNull( @pExpressionToSearch, '' ) = ''
    Return 0

  If @pStartLocation Is Null
      Or IsNull( @pStartLocation, 0 ) < 0
    Set @pStartLocation = 0

  If IsNull( @pOcorrenceNumber, 0 ) = 0
    Set @pOcorrenceNumber = 1

  If @pOcorrenceNumber < 0
  Begin
    Set @pExpressionToSearch = Reverse( @pExpressionToSearch )
    Set @pExpressionToFind = Reverse( @pExpressionToFind )
  End

  Declare @i Int = 0
  Set @result = @pStartLocation
  Declare @location Bigint = @pStartLocation
  Declare @expressionToSearch Varchar(max) = Substring( @pExpressionToSearch, @location + 1, Len(
@pExpressionToSearch ) - @location )

  While @i < Abs( @pOcorrenceNumber )
  Begin
    Set @location = Charindex( @pExpressionToFind, @expressionToSearch )

    If @location = 0
      Return 0

    Set @result += @location

    Set @expressionToSearch = Substring( @expressionToSearch, @location + 1, Len(
@expressionToSearch ) - @location )

    Set @i += 1
  End

  If @pOcorrenceNumber < 0
    Set @result = Len( @pExpressionToSearch ) - @result + 1 - Len( @pExpressionToFind ) + 1
```

```
    Return @result
  End
```

Exemplos de Execução

```
-- A. Returning the starting position of '-' in the Phone Number
Select Distinct PhoneNumber, dbo.FIND('-', PhoneNumber, null, null) As Result From
Person.PersonPhone
```

Results | Messages

	PhoneNumber	Result
1	193-555-0125	4
2	355-555-0191	4
3	446-555-0170	4
4	786-555-0191	4
5	561-555-0179	4
6	426-555-0142	4
7	682-555-0116	4
8	727-555-0159	4
9	100-555-0155	4
10	139-555-0150	4
11	272-555-0141	4
12	293-555-0189	4
13	1 (11) 500 555-0158	15
14	187-555-0152	4

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 9938 rows

Fig5: Exemplo A. FIND – A.jpg

```
-- B. Returning the position of '.' in the Document Summary
Select DocumentSummary, dbo.FIND('.', DocumentSummary, null, 1) As Result From Production.Document
Where DocumentSummary Is Not Null
```

Results | Messages

	DocumentSummary	Result
1	It is important that you maintain your bicycle and keep it in good repair. Detailed repair an...	74
2	Guidelines and recommendations for lubricating the required components of your Advent...	110
3	Reflectors are vital safety components of your bicycle. Always ensure your front and bac...	55
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement ped...	90
5	Worn or damaged seats can be easily replaced following these simple instructions. Instru...	81

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 5 rows

Fig6: Exemplo B. FIND – B.jpg

```
-- B.2 Returning the position of '.' in the Document Summary, starting from position 60
Select DocumentSummary, dbo.FIND('.', DocumentSummary, 60, 1) As Result From Production.Document
Where DocumentSummary Is Not Null
```

Results Messages

	DocumentSummary	Result
1	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guideline...	74
2	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bi...	110
3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clea...	130
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are...	90
5	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicab...	81

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 5 rows

Fig7: Exemplo B2. FIND – B2.jpg

```
-- C. Returning the last position of ‘ ’ in the Reviewer Name
Select ReviewerName, dbo.FIND(' ', ReviewerName, null, -1) From Production.ProductReview
```

Results Messages

	ReviewerName	(No column name)
1	John Smith	6
2	David	0
3	Jill	0
4	Laura Norman	7

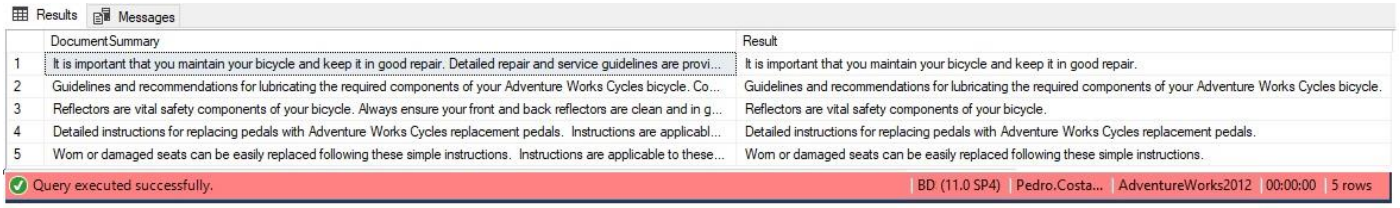
Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 4 rows

Fig8: Exemplo C. FIND – C.jpg

SQL – Funções STRING & FIND

Exemplos de Execução

```
-- E.1 Returning the First Sentence of Document Summary
Select DocumentSummary, dbo.STRING(DocumentSummary, '[:' + CAST(dbo.FIND('.', DocumentSummary, null,
1) as varchar) + ']') As Result From Production.Document Where DocumentSummary Is Not Null
```

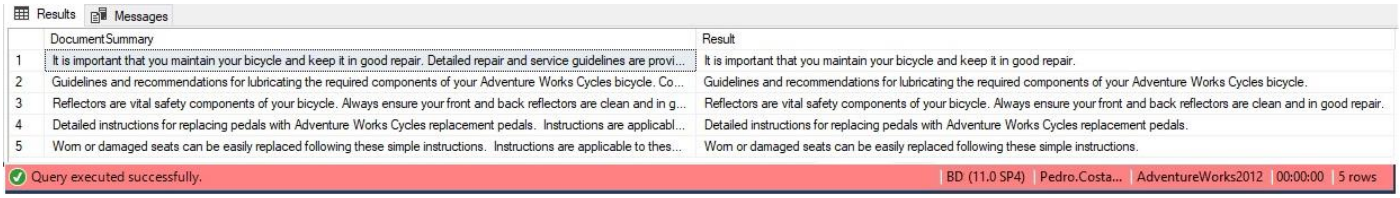


	DocumentSummary	Result
1	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provi...	It is important that you maintain your bicycle and keep it in good repair.
2	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Co...	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle.
3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in g...	Reflectors are vital safety components of your bicycle.
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicabl...	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals.
5	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these...	Worn or damaged seats can be easily replaced following these simple instructions.

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 5 rows

Fig9: Exemplo E1. STRING & FIND –E1.jpg

```
-- E.1.2 Returning the First Sentence of Document Summary, starting from position 60
Select DocumentSummary, dbo.STRING(DocumentSummary, '[:' + CAST(dbo.FIND('.', DocumentSummary, 60,
1) as varchar) + ']') As Result From Production.Document Where DocumentSummary Is Not Null
```



	DocumentSummary	Result
1	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provi...	It is important that you maintain your bicycle and keep it in good repair.
2	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Co...	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle.
3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in g...	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair.
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicabl...	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals.
5	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to thes...	Worn or damaged seats can be easily replaced following these simple instructions.

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 5 rows

Fig10: Exemplo E1.2. STRING & FIND –E1.2.jpg

```
-- E.2 Returning the LoginID from Employees without the Domain
Select LoginID, dbo.STRING(LoginID, '[' + CAST(dbo.FIND('\', LoginID, null, null) as varchar) +
':]') As Result From HumanResources.Employee
```

Results | Messages

	LoginID	Result
1	adventure-works\alan0	\alan0
2	adventure-works\alejandro0	\alejandro0
3	adventure-works\alex0	\alex0
4	adventure-works\alice0	\alice0
5	adventure-works\amy0	\amy0
6	adventure-works\andreas0	\andreas0
7	adventure-works\andrew0	\andrew0
8	adventure-works\andrew1	\andrew1
9	adventure-works\andy0	\andy0
10	adventure-works\angela0	\angela0
11	adventure-works\anibal0	\anibal0
12	adventure-works\annette0	\annette0

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 290 rows

Fig11: Exemplo E2. STRING & FIND –E2.jpg

```
-- E.3 Returning the last name from Stores Name
Select [Name], dbo.FIND(' ', [Name], null, -1) As Find, dbo.STRING([Name], '[' + CAST(dbo.FIND(' ',
[Name], null, -1) as varchar) + ':]') As Result From Sales.Store
```

Results | Messages

	Name	Find	Result
1	Next-Door Bike Store	16	Store
2	Professional Sales and Service	24	Service
3	Riders Company	8	Company
4	The Bike Mechanics	10	Mechanics
5	Nationwide Supply	12	Supply
6	Area Bike Accessories	11	Accessories
7	Bicycle Accessories and Kits	25	Kits
8	Clamps & Brackets Co.	19	Co.
9	Valley Bicycle Specialists	16	Specialists
10	New Bikes Company	11	Company
11	Vinyl and Plastic Goods Corporation	25	Corporation
12	Top of the Line Bikes	17	Bikes

Query executed successfully. | BD (11.0 SP4) | Pedro.Costa... | AdventureWorks2012 | 00:00:00 | 701 rows

Fig12: Exemplo E3. STRING & FIND –E3.jpg

```
-- E.4 Returning the Reviewer Name from Product Review As LastName, First Name
Select
  ReviewerName,
  Case
    When dbo.FIND(' ', ReviewerName, null, -1) != 0
      Then dbo.STRING(ReviewerName, '[' + CAST(dbo.FIND(' ', ReviewerName, null, -1) as varchar) +
':],\, ,[:' + CAST(dbo.FIND(' ', ReviewerName, null, null) - 1 as varchar) + ']')
    Else
      ReviewerName
  End As Result
From Production.ProductReview
```

Fig13: Exemplo E4. STRING & FIND –E4.jpg

Testes

Testes Unitários

Foram elaborados testes unitários com base no script abaixo com o objetivo de automatizar a realização dessa tarefa.

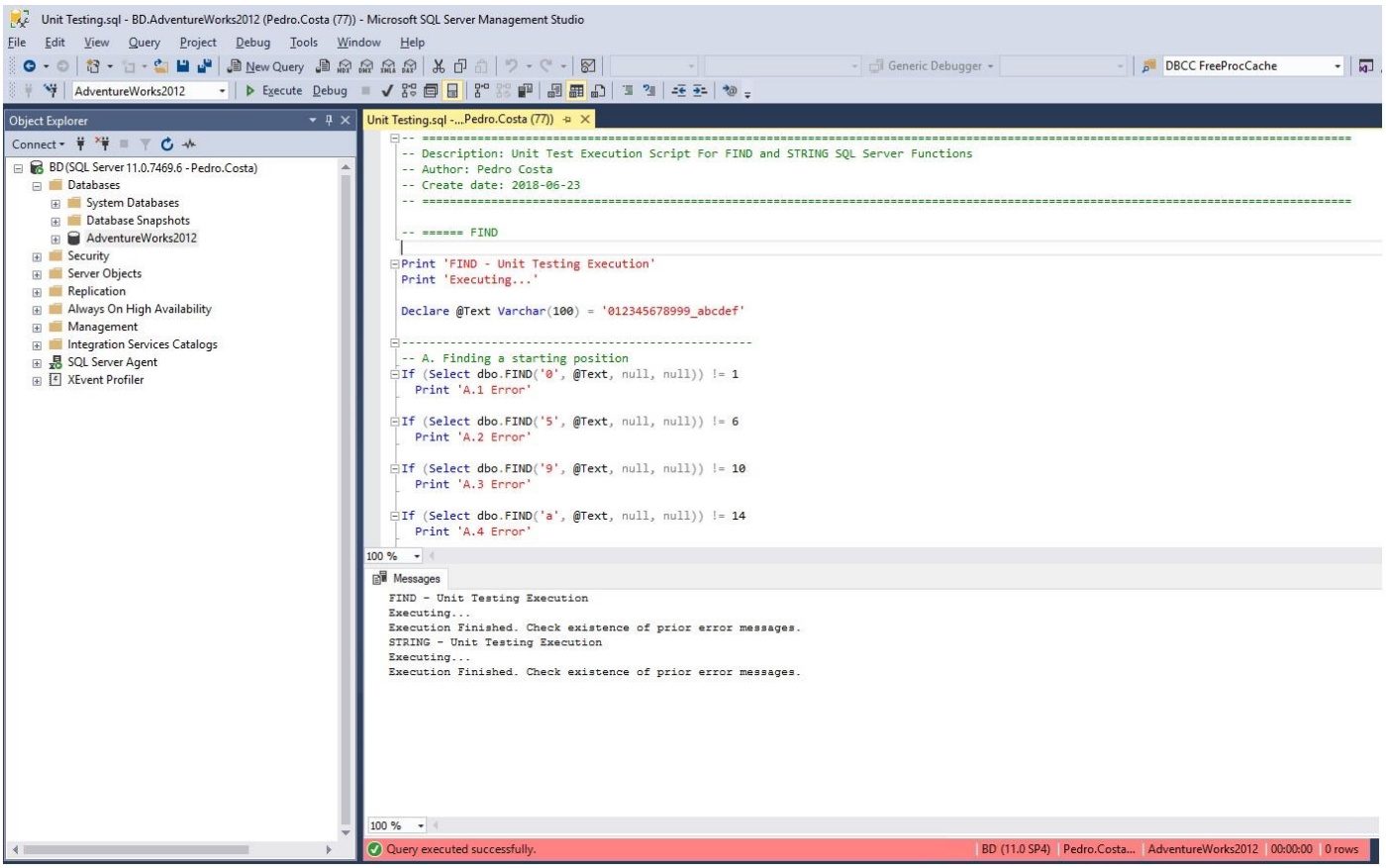


Fig14: Testes Unitários. Unit Testing.jpg

Os exemplos foram desenvolvidos para incluir as seguintes situações

FIND

```
-- A. Finding a starting position
-- B. Finding a starting position starting from
-- C. Find a starting position (first ocurrence)
-- D. Find a starting position (first ocurrence), starting from
-- E. Find a recurring position
-- F. Returnig last positions
-- G. Non Existant Positions
```

STRING

```
-- A. Returning chars positions
-- B. Returning ranges
```

```
-- C. Returning chars and ranges
-- D. Returning negative chars
-- E. Returning negative ranges
-- F. Returning Text
-- G. Retuning formated expressions
-- H. Non Existant Returns
```

A execução do script sem reporte de erros indica a correta execução das funções para os exemplos elaborados.

Testes de Performance

Foram elaborados testes de performance com base no script abaixo, para os mesmos queries usados como exemplos de execução, para as funções FIND, STRING e FIND & STRING, com o objetivo de aferir os custos de execução de cada exemplo assim como uma medida da execução global.

Para cada exemplo foi executado o query base (retorno de informação direta) e o mesmo query recorrendo à função FIND/STRING e à alternativa em funções de sistema de SQL existentes CHARINDEX/SUBSTRING/LEN/RIGHT/REVERSE.

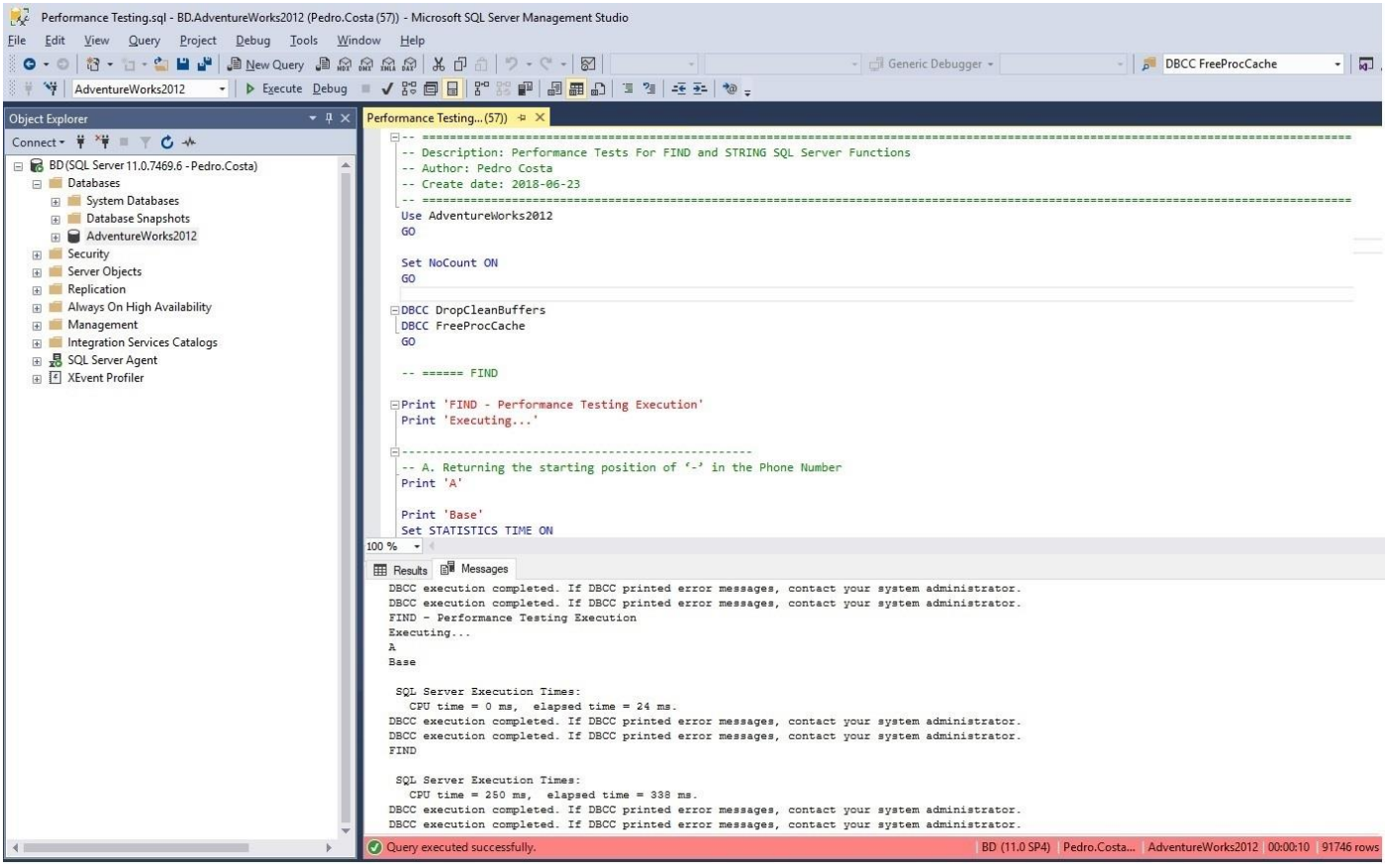


Fig15: Testes de Performance. Performance Testing.jpg

Tempos de execução:

FIND	ms	Base	FIND	CHARINDEX
	A	30	330	16
	B	2	8	0
	B.2	0	1	0
	C	4	16	4
	Total	36	355	20
	Cost	1	10	1

Em termos de performance de execução a função FIND desenvolvida revelou um custo cerca de 10 vezes superior à alternativa usando a função de sistema CHARINDEX.

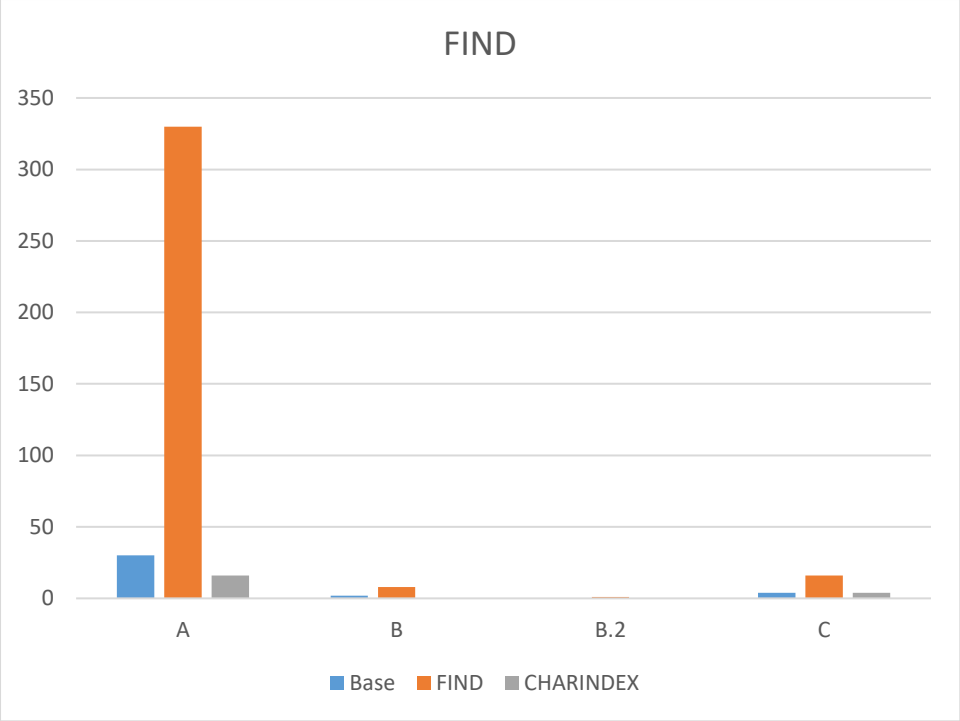


Fig16: FIND Performance Analysis. Performance Testing Output Analisys.xlsx

Tempos de execução:

STRING	ms	Base	STRING	SUBSTRING
	A	0	44	0
	B	0	57	0
	C	0	40	0
	D	27	4979	44
	Total	27	5120	44
	Cost	1	190	2

Em termos de performance de execução a função STRING desenvolvida revelou um custo cerca de 95 (190/2) vezes superior à alternativa usando a função de sistema SUBSTRING.

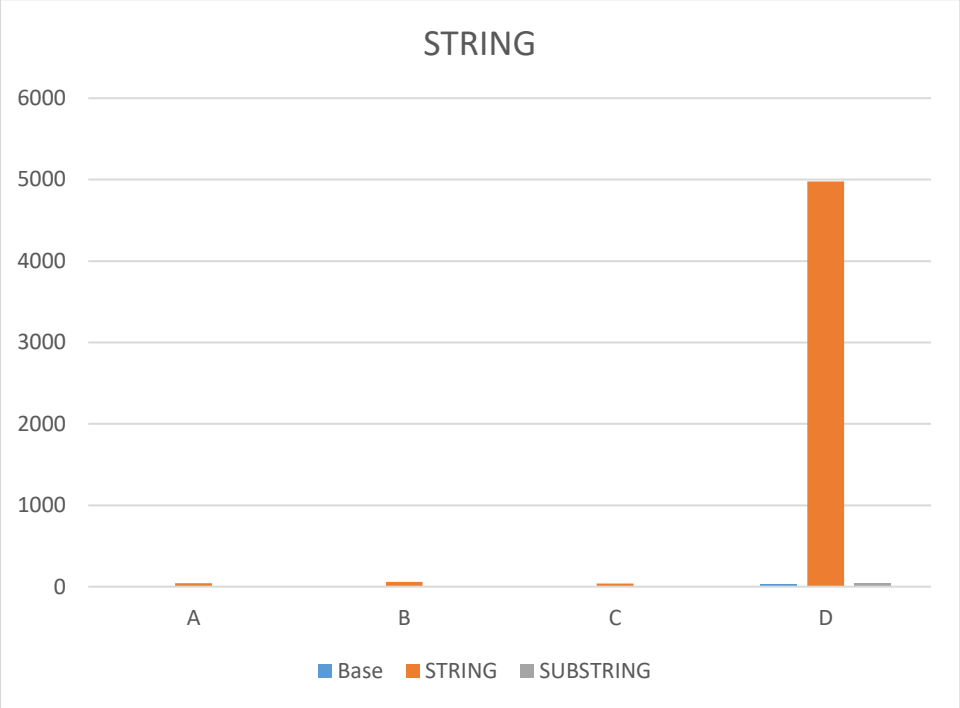


Fig17: STRING Performance Analysis. Performance Testing Output Analisys.xlsx

O custo incidiu sobretudo no exemplo D, reflexo do maior número de operadores de transformação envolvidos nesse caso.

SQL Server 2016

Foram elaborados testes de performance com base no script abaixo, para os mesmos queries usados como exemplos de execução, neste caso com a função STRING desenvolvida para SQL Server 2016, recorrendo em particular à nova função STRING_SPLIT disponibilizada e que permitiu selecionar diretamente o conjunto de operadores enviados em parâmetro separados por ‘,’. O objetivo foi aferir os custos de execução de cada exemplo assim como uma medida da execução global e comparativa com a mesma função com código SQL Server 2012.

Para cada exemplo foi executado o query base (retorno de informação direta) e o mesmo query recorrendo à função FIND/STRING e à alternativa em funções de sistema de SQL existentes CHARINDEX/SUBSTRING/LEN/RIGHT/REVERSE.

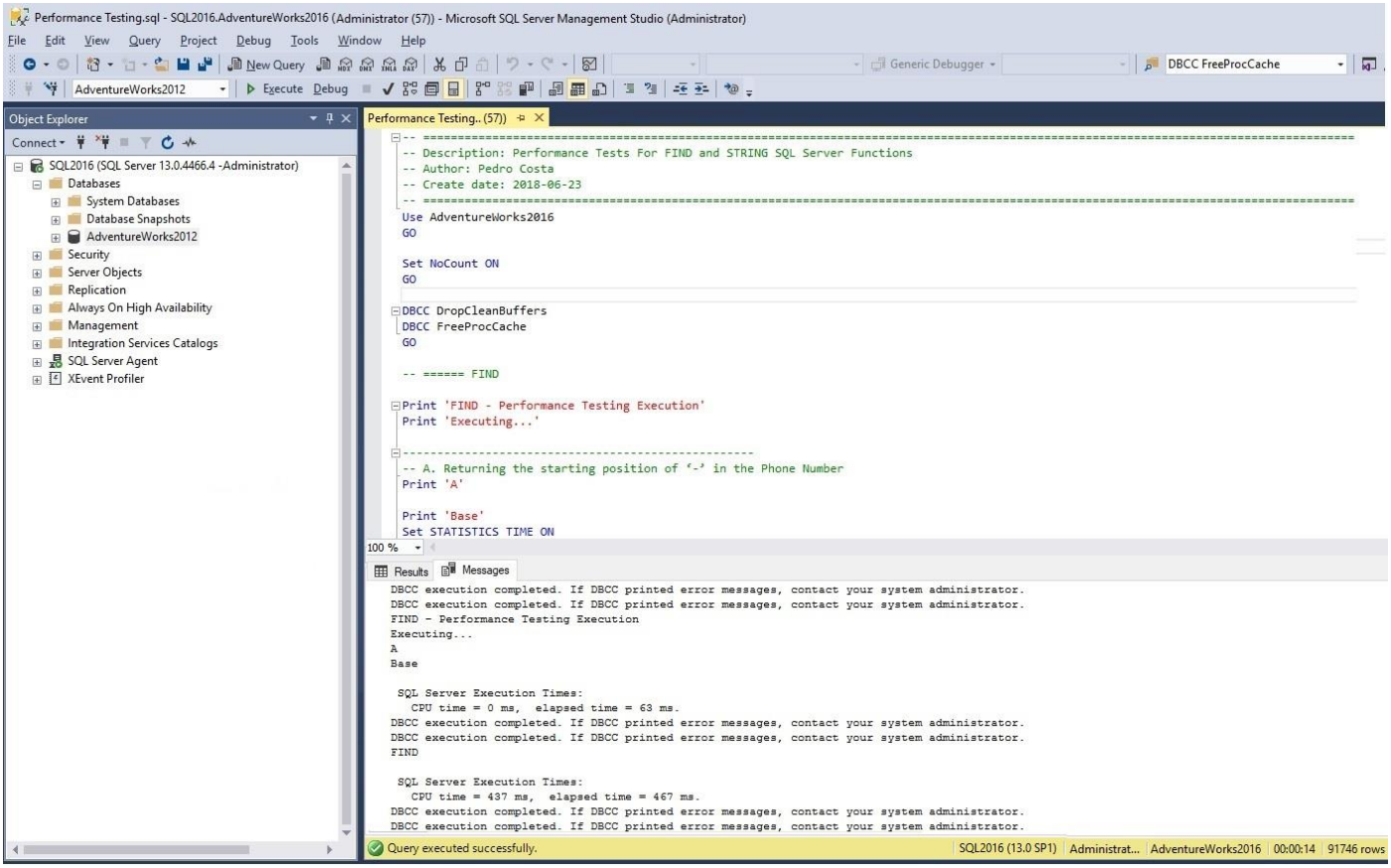


Fig18: Testes de Performance SQL Server 2016. Performance Testing.jpg

Tempos de execução:

FIND	ms	Base	FIND	CHARINDEX
	A	148	597	151
	B	6	9	0
	B.2	0	2	0
	C	7	5	1
	Total	161	613	152
	Cost	1	4	1

Em termos de performance de execução a função FIND desenvolvida, executada em SQL 2016 revelou um custo cerca de 4 vezes superior à alternativa usando a função de sistema CHARINDEX. O custo ficou comparativamente muito abaixo do custo de execução em SQL Server 2012 que foi cerca de 10 vezes superior, o que poderá indicar uma melhoria de eficiência de execução nesta mais recente versão de SQL.

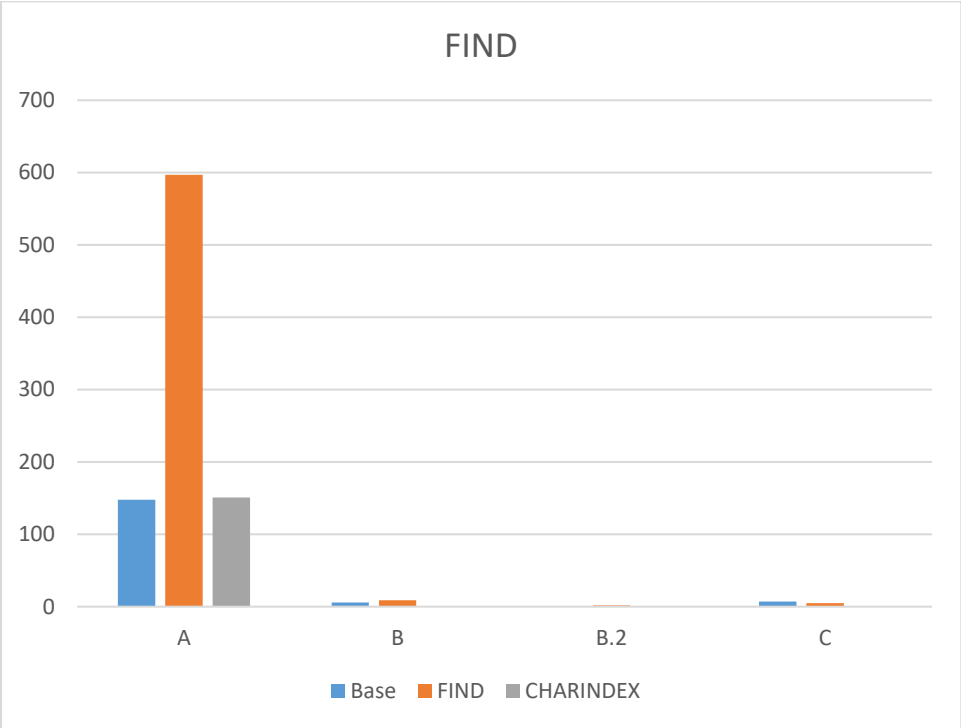


Fig19: FIND Performance Analysis SQL Server 2016. Performance Testing Output Analisys.xlsx

Tempos de execução:

STRING	ms	Base	STRING	SUBSTRING
	A	0	112	23
	B	1	77	80
	C	0	116	21
	D	252	6560	237
	Total	253	6865	361
	Cost	1	27	1

Em termos de performance de execução a função STRING desenvolvida para SQL 2016 revelou um custo cerca de 27 vezes superior à alternativa usando a função de sistema SUBSTRING. O custo ficou comparativamente muito abaixo do custo de execução em SQL Server 2012 que foi cerca de 80 vezes superior, o que, neste caso, poderá indicar uma melhor adequação do código criado para esta mais recente versão de SQL.

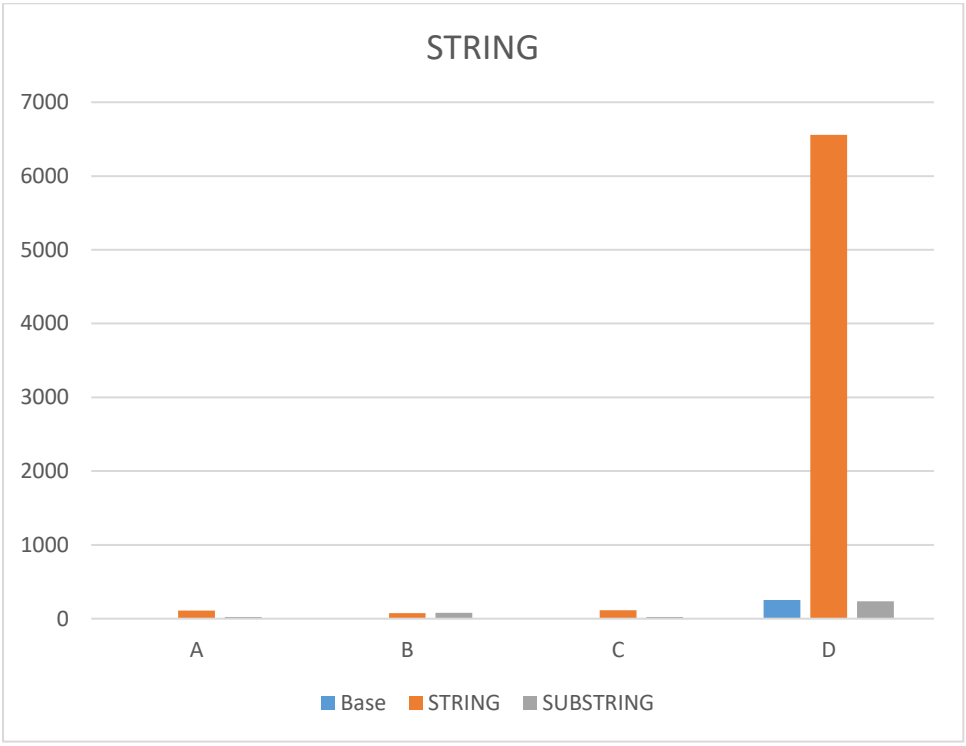


Fig20: STRING Performance Analysis SQL Server 2016. Performance Testing Output Analisys.xlsx

Mais uma vez o custo incidiu sobretudo no exemplo D, reflexo do maior número de operadores de transformação envolvidos nesse caso.

Testes de Carga

Foram elaborados testes de carga com base no script abaixo, para os mesmos queries usados como exemplos de execução, para as funções FIND e STRING. O objetivo foi aferir os custos de execução de cada exemplo em medida da carga envolvida.

Para cada exemplo foi executado o query sobre um universo de mil, dez mil e cem mil registos.

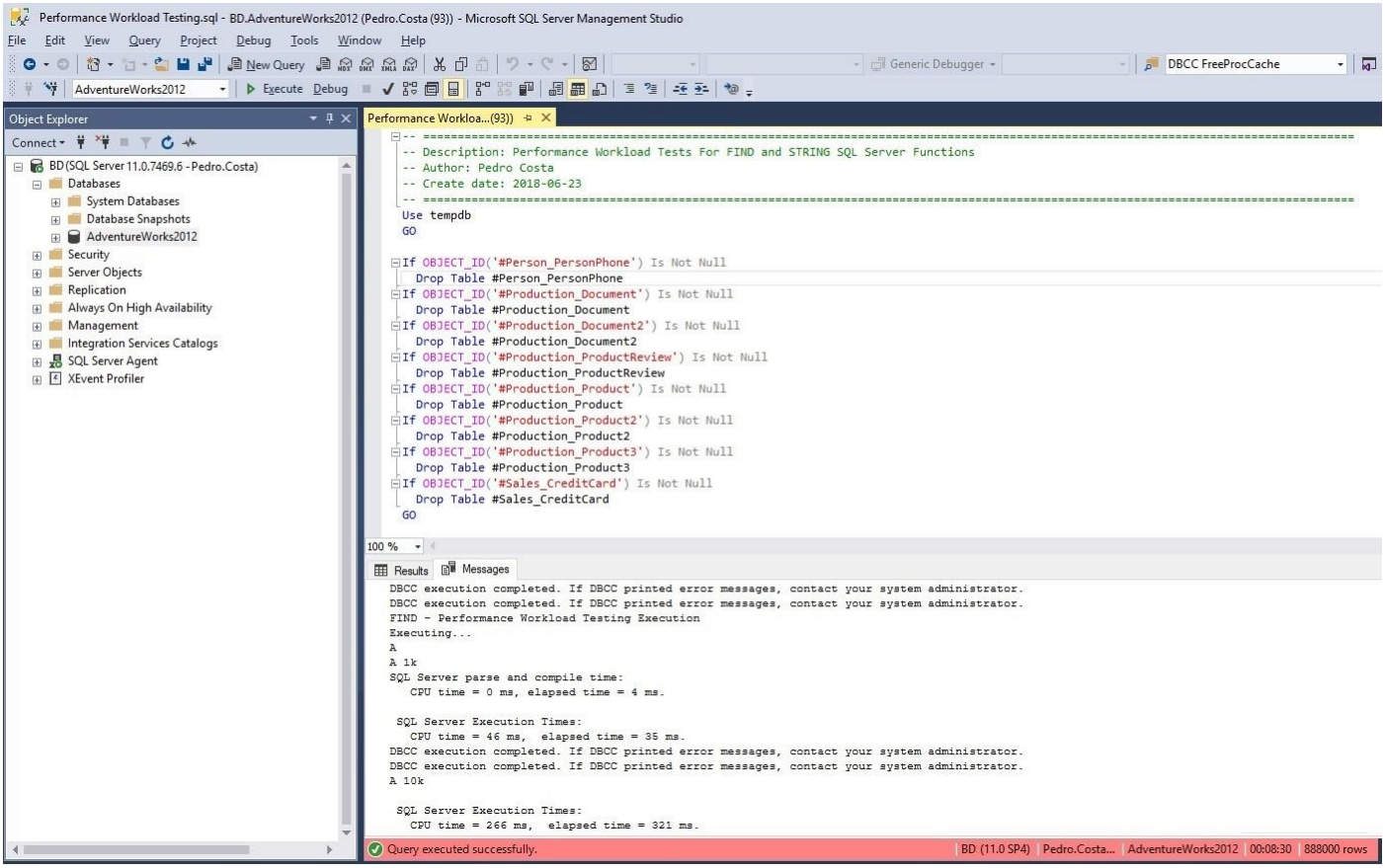


Fig21: Testes de Carga. Performance Worload Testing.jpg

Tempos de execução:

FIND	ms	A	B	B.2	C
	1k	37	42	41	8
	10k	325	398	492	344
	100k	3208	41468	37880	3378

Os testes de carga efetuados para a função FIND revelaram uma performance aceitável para a ordem dos 10 mil registos mas francamente má para cargas da ordem dos 100 mil registos.

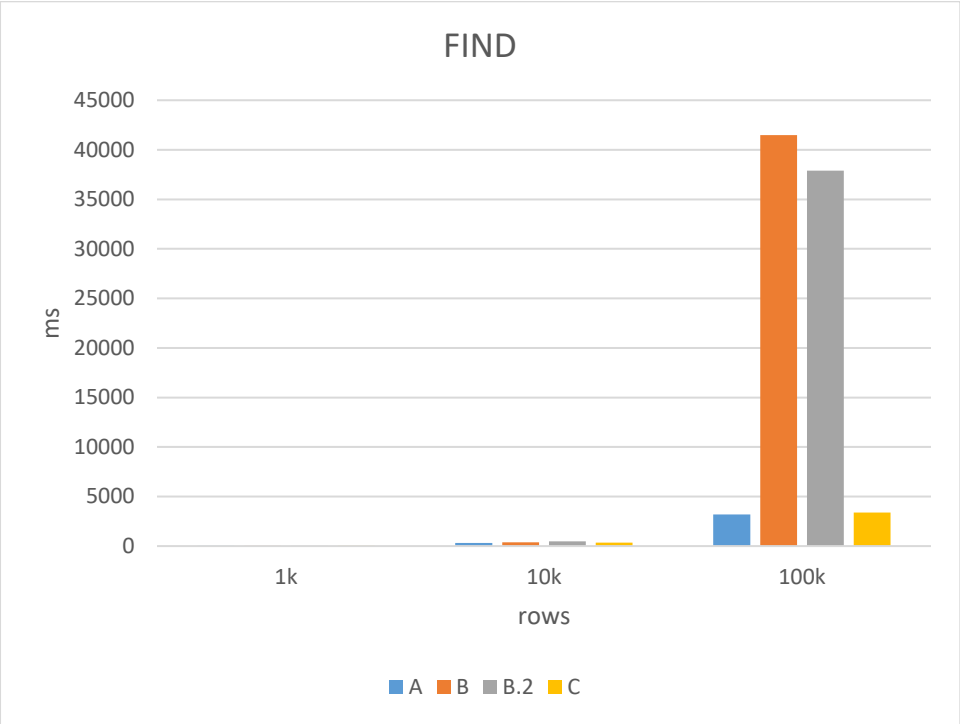


Fig22: FIND Workload Performance Analysis. [Performance Workload Testing Output Analisys.xlsx]

Custos de execução:

COST					
FIND	ms/1k	A	B	B.2	C
	1k	1.0	1.0	1.0	1.0
	10k	0.9	0.9	1.2	4.3
	100k	0.9	9.9	9.2	4.2

Constatou-se pela análise dos custos de execução que estes são do tipo O(n) com n = 1, ou seja, o custo é linear e equivalente ao número de registos tratados para os exemplos A, B e B2 até 10 mil registos. Para os exemplos B e B2 com 100 mil registos o custo revelou-se n = 9. E para o exemplo C temos n = 4.

Tempos de execução:

STRING	ms	A	B	C	D
	1k	90	113	81	262
	10k	837	1086	753	2573
	100k	8308	10750	7485	25803

Os testes de carga efetuados para a função STRING revelaram uma performance ainda aceitável para a ordem dos 10 mil registos mas má para cargas da ordem dos 100 mil registos.

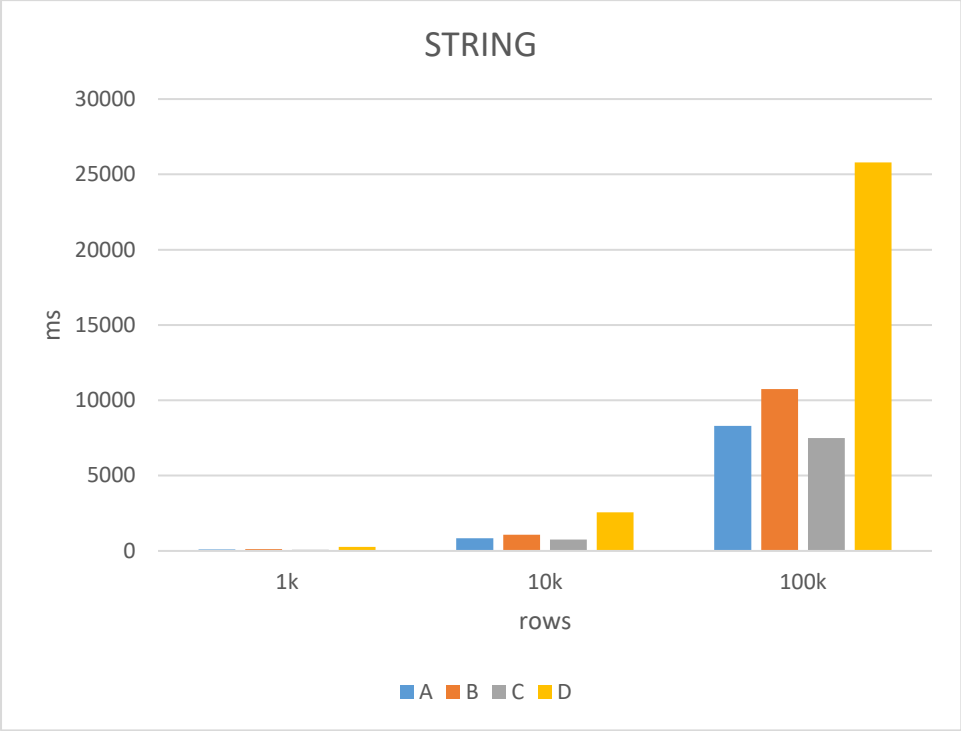


Fig23: STRING Workload Performance Analysis. Performance Workload Testing Output Analisys.xlsx

Custos de execução:

COST STRING	ms/1k	A	B	B.2	C
	1k	1.0	1.0	1.0	1.0
	10k	0.9	1.0	0.9	1.0
	100k	0.9	1.0	0.9	1.0

De igual modo também se constatou pela análise dos custos de execução que estes são do tipo O(n) com n = 1, ou seja, o custo é linear e equivalente ao número de registos tratados.

SQL Server 2016

Foram elaborados testes de carga com base no script abaixo, para os mesmos queries usados como exemplos de execução, para a função STRING desenvolvida para SQL Server 2016. O objetivo foi aferir os custos de execução de cada exemplo em medida da carga envolvida assim como uma medida da execução global e comparativa com a mesma função com código SQL Server 2012.

Para cada exemplo foi executado o query sobre um universo de mil, dez mil e cem mil registos.

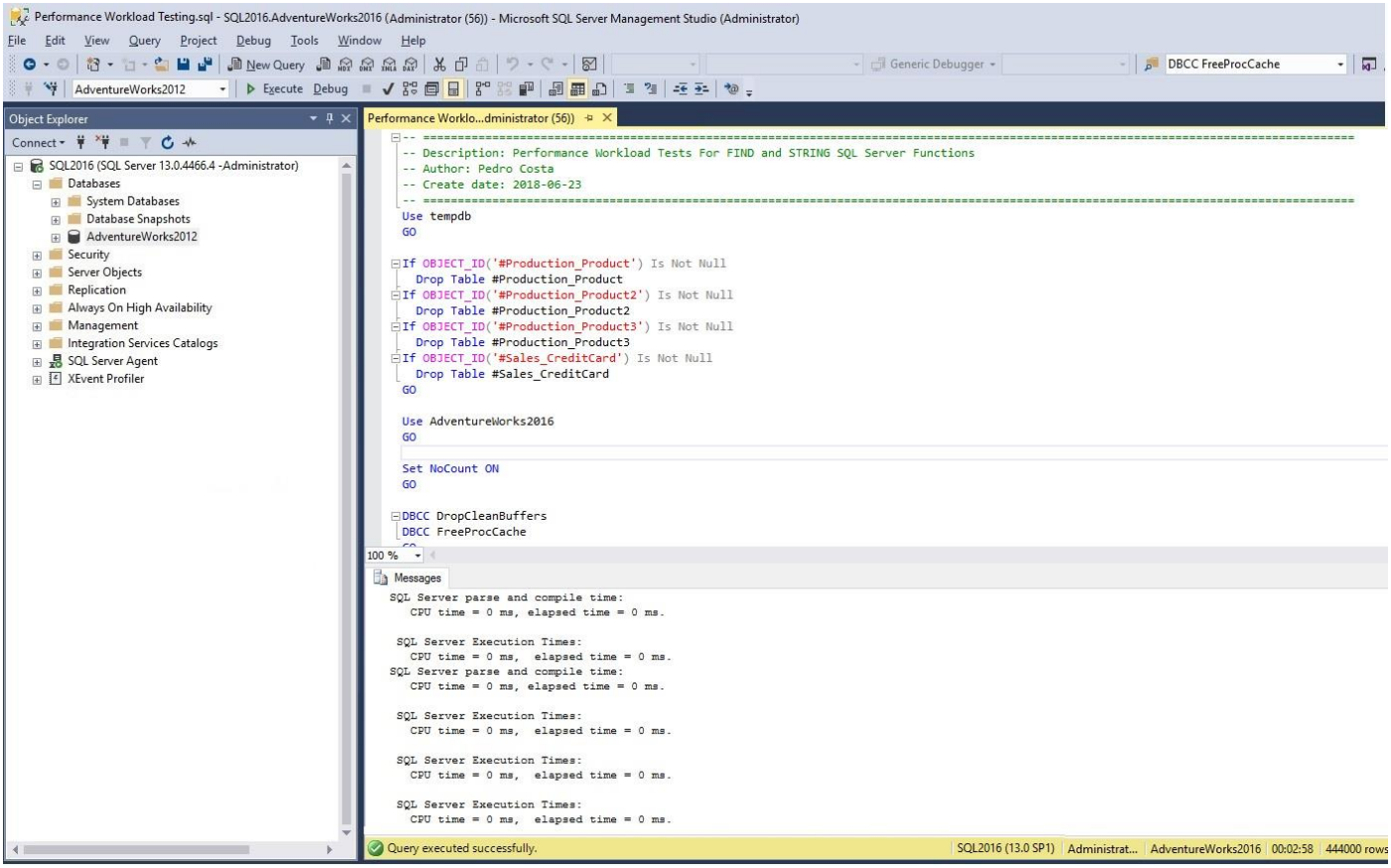


Fig24: Testes de Carga SQL Server 2016. Performance Worload Testing.jpg

Tempos de execução:

STRING	ms	A	B	C	D
	1k	707	712	521	1661
	10k	6636	6875	5045	16733
	100k	65916	68820	49917	165870

Os testes de carga efetuados para a função STRING em SQL Server 2016 revelaram uma performance má para cargas logo da ordem dos 10 mil registos.

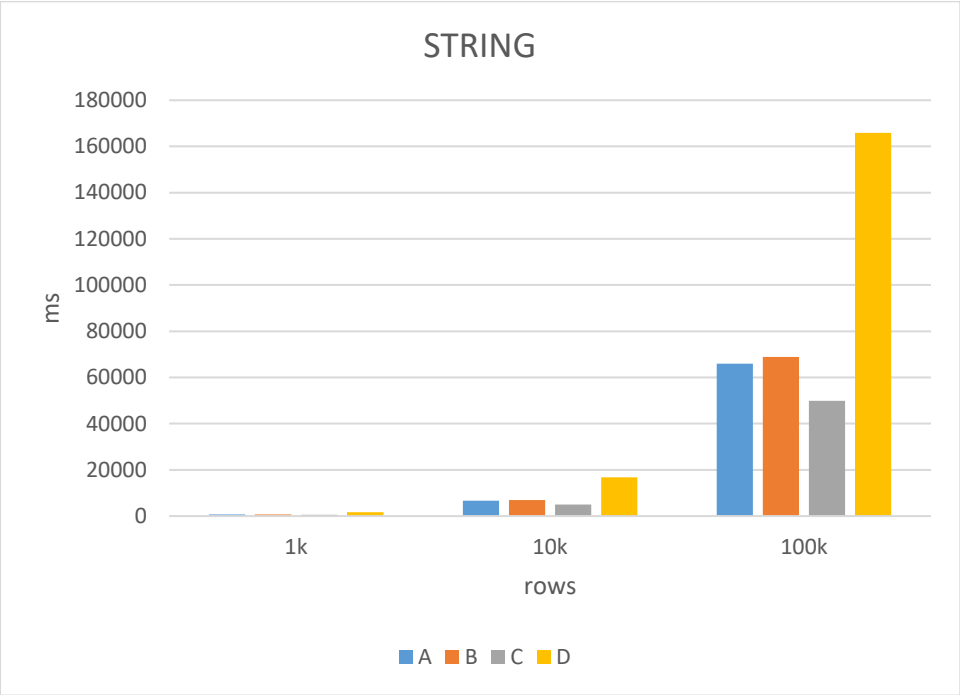


Fig25: STRING Workload Performance Analysis SQL Server 2016. Performance Workload Testing Output Analisys.xlsx

Custos de execução:

COST STRING	ms/1k	A	B	B.2	C
	1k	1.0	1.0	1.0	1.0
	10k	0.9	1.0	1.0	1.0
	100k	0.9	1.0	1.0	1.0

Mas de igual modo também se constatou pela análise dos custos de execução que estes são do tipo O(n) com n = 1, ou seja, o custo é linear e equivalente ao número de registos tratados.

SQL Server 2012

Foram elaborados também testes de carga em SQL Server 2016 para a função STRING desenvolvida para SQL Server 2012. O objetivo foi aferir os custos de execução de cada exemplo em medida da carga envolvida assim como uma medida comparativa da execução em SQL Server 2016 da mesma função com código SQL Server 2012.

Para cada exemplo foi executado o query sobre um universo de mil, dez mil e cem mil registos.

Tempos de execução:

STRING	ms	A	B	C	D
	1k	159	219	111	417
	10k	1264	1525	1100	3453
	100k	11419	14805	10438	34514

Os testes de carga efetuados para a função STRING (SQL Server 2012) em SQL Server 2016 revelaram uma performance aceitável para os vários exemplos e cargas envolvidas. E a performance revelou-se substancialmente melhor que a mesma execução da função desenvolvida especificamente para o SQL Server 2016 mas não melhor que a execução em SQL Server 2012.

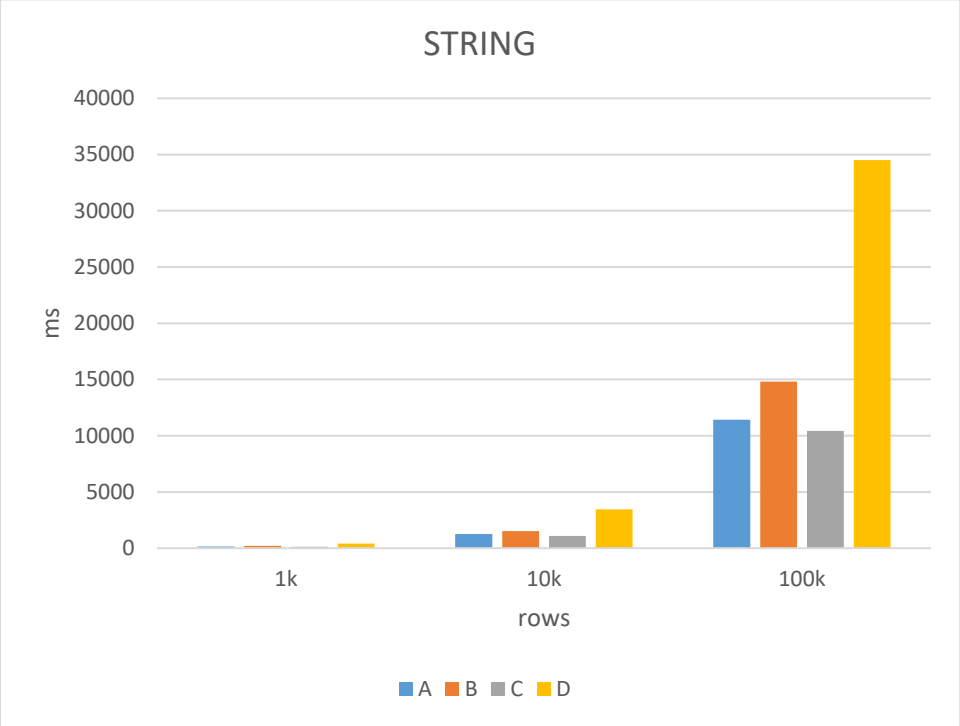


Fig26: STRING Workload Performance Analysis SQL 2012 on SQL 2016. Performance Workload Testing Output Analisys.xlsx

Custos de execução:

COST STRING	ms/1k	A	B	B.2	C
	1k	1.0	1.0	1.0	1.0
	10k	0.8	0.7	1.0	0.8
	100k	0.7	0.7	0.9	0.8

Mas de igual modo também se constatou pela análise dos custos de execução que estes são do tipo O(n) com n = 1, ou seja, o custo é linear e equivalente ao número de registos tratados.

Conclusões

As funções desenvolvidas para SQL Server 2012 garantem uma melhor performance que a equivalente STRING desenvolvida para SQL Server 2016. No entanto, e apesar de um custo relativamente unitário em função do número de registos, a performance dessas funções fica sempre aquém das execuções de exemplo equivalentes recorrendo às funções de sistema de SQL.

Notas Finais

Com este trabalho procurei, mais do que criar as funções com objetivos de performance, o que de partida ficou logo assente que não seria o objetivo, e como, de facto, se veio a revelar não serem, mas criar uma abordagem mais simples e standard para tratamento de strings em SQL e que facilite a interpretação e gestão do código desenvolvido.

E, no futuro, talvez possamos esperar a inclusão destas funções, ou lógica equivalente, como funções de sistema, em alguma próxima versão de SQL Server, seguindo a abordagem já iniciada na versão 2016 de tentar disponibilizar um conjunto alargado e simples de funções para tratamento e manipulação de strings como são o STRING_SPLIT, STRING_ESCAPE ou CONCAT_WS, beneficiando certamente nesse caso de uma possível e substancial melhoria de performance.

UCs Usadas para a Realização do Projeto

As Unidades Curriculares que contribuiram com as bases para a realização do projeto são:

21053 – Fundamentos de Bases de Dados

- Sistemas de Gestão de Bases de Dados (SGBD)
- Linguagem Structured Query Language e Álgebra Relacional

21078 – Linguagens e Computação

- Expressões e linguagens regulares
- Gramáticas e linguagens independentes do contexto

21018 - Compilação

- Processamento de linguagens
- Interpretadores
- Análise léxica
- Análise sintática
- Geração e Otimização de código

File Index

1. FIND SQL Server Function (starting with 2012)
   Find.sql
2. STRING SQL Server Function (starting with 2012)
   String.sql

3. Unit Test Execution Script for FIND and STRING SQL Server Functions
   Unit Testing.sql

4. Examples for FIND and STRING SQL Server Functions
   Examples.sql

5. Performance Tests for FIND and STRING SQL Server Functions
   Performance Testing.sql
6. Performance Tests Results (SQL Server 2012)
   Performance Testing Output.txt
7. Performance Tests Results Analysis (SQL Server 2012)
   Performance Testing Output Analysis.xlsx
8. Performance Workload Tests for FIND and STRING SQL Server Functions
   Performance Workload Testing.sql
9. Performance Workload Tests Results (SQL Server 2012)
   Performance Workload Testing Output.txt
10. Performance Workload Tests Results (SQL Server 2012)
    Performance Workload Testing Output Analysis.xlsx

11. SQL Server 2016
    - 11.1. FIND SQL Server Function (starting with 2012)
      Find.sql
    - 11.2. STRING SQL Server Function (starting with 2016)
      String.sql

    - 11.3. Unit Test Execution Script for FIND and STRING SQL Server Functions
      Unit Testing.sql

    - 11.4. Performance Tests for FIND and STRING SQL Server Functions
      Performance Testing.sql
    - 11.5. Performance Tests Results (SQL Server 2016)
      Performance Testing Output.txt
    - 11.6. Performance Tests Result Analysis (SQL Server 2016)
      Performance Testing Output Analysis.xlsx
    - 11.7. Performance Workload Tests for FIND and STRING SQL Server Functions
      Performance Workload Testing.sql
    - 11.8. Performance Workload Tests Results (SQL Server 2016)
      Performance Workload Testing Output.txt

11.9. Performance Workload Tests Results (SQL Server 2016)
Performance Workload Testing Output Analysis.xlsx

11.10. SQL Server 2012

11.10.1. STRING SQL Server Function (starting with 2012)
String.sql

11.10.2. Performance Workload Tests for FIND and STRING SQL Server Functions
Performance Workload Testing.sql

11.10.3. Performance Workload Tests Results (SQL Server 2016)
Performance Workload Testing Output.txt

11.10.4. Performance Workload Tests Results (SQL Server 2016)
Performance Workload Testing Output Analysis.xlsx

Bibliografia

Livros

Abraham Silberschatz, Henry F. Korth, S. Sudarshan. *Database System Concepts*. McGraw-Hill.

Adam Drozdek. *Data Structures and Algorithms in C++*. Fourth Edition, Cengage Learning.

Alfred Aho, Lam, Setti, Ullman. (2007). *Compilers: principles, techniques and tools*. Second Ed. Pearson.

Coelho, José. (2010) *Conceitos e exercícios de programação, utilizando Linguagem C*. Universidade Aberta. Lisboa.

Hopcroft, Motwani, Ullman. *Introduction to Automata and Language Theory*. Pearson.

URLs

Microsoft (2016). String Functions (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/string-functions-transact-sql?view=sql-server-2017

Microsoft (2017). CharIndex (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/charindex-transact-sql?view=sql-server-2017

Microsoft (2017). Replace (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/replace-transact-sql?view=sql-server-2017

Microsoft (2017). Right (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/right-transact-sql?view=sql-server-2017

Microsoft (2016). SubString (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/substring-transact-sql?view=sql-server-2017

Microsoft (2016). String_Split (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/string-split-transact-sql?view=sql-server-2017

Microsoft (2018). Concat_WS (Transact-SQL). Acedido em https://docs.microsoft.com/en-us/sql/t-sql/functions/concat-ws-transact-sql?view=sql-server-2017