Contents

AccessControlKind.Allow

AccessControlKind.Deny

Access.Database

ActiveDirectory.Domains

AdobeAnalytics.Cubes

AdoDotNet.DataSource

AdoDotNet.Query

AnalysisServices.Database

AnalysisServices.Databases

AzureStorage.BlobContents

AzureStorage.Blobs

AzureStorage.DataLake

AzureStorage.DataLakeContents

AzureStorage.Tables

Cdm.Contents

Csv.Document

CsvStyle.QuoteAfterDelimiter

CsvStyle.QuoteAlways

Cube.AddAndExpandDimensionColumn

Cube.AddMeasureColumn

Cube.ApplyParameter

Cube.AttributeMemberId

Cube.AttributeMemberProperty

Cube.CollapseAndRemoveColumns

Cube.Dimensions

Cube.DisplayFolders

Cube.MeasureProperties

Cube.MeasureProperty

Cube.Measures

Cube.Parameters

Cube.Properties

Cube.PropertyKey

Cube.ReplaceDimensions

Cube.Transform

DB2.Database

Essbase.Cubes

Excel.CurrentWorkbook

Excel.Workbook

Exchange.Contents

File.Contents

Folder.Contents

Folder.Files

GoogleAnalytics.Accounts

Hdfs.Contents

Hdfs.Files

HdInsight.Containers

HdInsight.Contents

HdInsight.Files

Html.Table

Identity.From

Identity.IsMemberOf

IdentityProvider.Default

Informix.Database

Json.Document

Json.FromValue

MySQL.Database

OData.Feed

ODataOmitValues.Nulls

Odbc.DataSource

Odbc.InferOptions

Odbc.Query

OleDb.DataSource

OleDb.Query

Oracle.Database

Parquet.Document

Pdf.Tables

PostgreSQL.Database

RData.FromBinary

Salesforce.Data

Salesforce.Reports

SapBusinessWarehouse.Cubes

SapBusinessWarehouseExecutionMode.DataStream

SapBusinessWarehouseExecutionMode.BasXml

SapBusinessWarehouseExecutionMode.BasXmlGzip

SapHana.Database

SapHanaDistribution.All

SapHanaDistribution.Connection

SapHanaDistribution.Off

SapHanaDistribution.Statement

SapHanaRangeOperator.Equals

SapHanaRangeOperator.GreaterThan

SapHanaRangeOperator.GreaterThanOrEquals

SapHanaRangeOperator.LessThan

SapHanaRangeOperator.LessThanOrEquals

SapHanaRangeOperator.NotEquals

SharePoint.Contents

SharePoint.Files

SharePoint.Tables

Soda.Feed

Sql.Database

Sql.Databases

Sybase.Database

Teradata.Database

WebAction.Request

Web.BrowserContents

Web.Contents

Date.IsInCurrentMonth

Date.IsInCurrentQuarter

Date.IsInCurrentWeek

Date.IsInCurrentYear

Date.IsInNextDay

Date.IsInNextMonth

Date.IsInNextNDays

Date.IsInNextNMonths

Date.IsInNextNQuarters

Date.IsInNextNWeeks

Date.IsInNextNYears

Date.IsInNextQuarter

Date.IsInNextWeek

Date.IsInNextYear

Date.IsInPreviousDay

Date.IsInPreviousMonth

Date.IsInPreviousNDays

Date.IsInPreviousNMonths

Date.IsInPreviousNQuarters

Date.IsInPreviousNWeeks

Date.IsInPreviousNYears

Date.IsInPreviousQuarter

Date.IsInPreviousWeek

Date.IsInPreviousYear

Date.IsInYearToDate

Date.IsLeapYear

Date.Month

Date.MonthName

Date.QuarterOfYear

Date.StartOfDay

Date.StartOfMonth

Date.StartOfQuarter

List.Generate

List.InsertRange

List.Intersect

List.IsDistinct

List.IsEmpty

List.Last

List.LastN

List.MatchesAll

List.MatchesAny

List.Max

List.MaxN

List.Median

List.Min

List.MinN

List.Mode

List.Modes

List.NonNullCount

List.Numbers

List.PositionOf

List.PositionOfAny

List.Positions

List.Product

List.Random

List.Range

List.RemoveFirstN

List.RemoveItems

List.RemoveLastN

List.RemoveMatchingItems

List.RemoveNulls

List.RemoveRange

List.Repeat

List.ReplaceMatchingItems

Number.Acos

Number.Asin

Number.Atan

Number.Atan2

Number.BitwiseAnd

Number.BitwiseNot

Number.BitwiseOr

Number.BitwiseShiftLeft

Number.BitwiseShiftRight

Number.BitwiseXor

Number.Combinations

Number.Cos

Number.Cosh

Number.E

Number.Epsilon

Number.Exp

Number.Factorial

Number.From

Number.FromText

Number.IntegerDivide

Number.IsEven

Number.IsNaN

Number.IsOdd

Number.Ln

Number.Log

Number.Log10

Number.Mod

Number.NaN

Number.NegativeInfinity

Number.Permutations

Number.PI

Number.PositiveInfinity

Order.Descending

RowExpression.Column

RowExpression.From

RowExpression.Row

Table.AddColumn

Table.AddFuzzyClusterColumn

Table.AddIndexColumn

Table.AddJoinColumn

Table.AddKey

Table.AggregateTableColumn

Table.AlternateRows

Table.Buffer

Table.Column

Table.ColumnCount

Table.ColumnNames

Table.ColumnsOfType

Table.Combine

Table.CombineColumns

Table.CombineColumnsToRecord

Table.ConformToPageReader

Table.Contains

Table.ContainsAll

Table.ContainsAny

Table.DemoteHeaders

Table.Distinct

Table.DuplicateColumn

Table.ExpandListColumn

Table.ExpandRecordColumn

Table.ExpandTableColumn

Table.FillDown

Table.FillUp

Table.FilterWithDataTable

Table.FindText

Table.First

Table.FirstN

Table.FirstValue

Table.FromColumns

Table.FromList

Table.FromPartitions

Table.FromRecords

Table.FromRows

Table.FromValue

Table.FuzzyGroup

Table.FuzzyJoin

Table.FuzzyNestedJoin

Table.Group

Table.HasColumns

Table.InsertRows

Table.IsDistinct

Table.IsEmpty

Table.Join

Table.Keys

Table.Last

Table.LastN

Table.MatchesAllRows

Table.MatchesAnyRows

Table.Max

Table.MaxN

Table.Min

Table.MinN

Table.NestedJoin

Table.Partition

Table.PartitionValues

Table.Pivot

Table.PositionOf

Table.PositionOfAny

Table.PrefixColumns

Table.Profile

Table.PromoteHeaders

Table.Range

Table.RemoveColumns

Table.RemoveFirstN

Table.RemoveLastN

Table.RemoveMatchingRows

Table.RemoveRows

Table.RemoveRowsWithErrors

Table.RenameColumns

Table.ReorderColumns

Table.Repeat

Table.ReplaceErrorValues

Table.ReplaceKeys

Table.ReplaceMatchingRows

Table.ReplaceRelationshipIdentity

Table.ReplaceRows

Table.ReplaceValue

Table.Reverse

Table.ReverseRows

Table.RowCount

Table.Schema

Table.SelectColumns

Table.SelectRows

Table.SelectRowsWithErrors

Table.SingleRow

Table.Skip

Table.Sort

Table.Split

Text.EndsWith

Text.Format

Text.From

Text.FromBinary

Text.InferNumberType

Text.Insert

Text.Length

Text.Lower

Text.Middle

Text.NewGuid

Text.PadEnd

Text.PadStart

Text.PositionOf

Text.PositionOfAny

Text.Proper

Text.Range

Text.Remove

Text.RemoveRange

Text.Repeat

Text.Replace

Text.ReplaceRange

Text.Reverse

Text.Select

Text.Split

Text.SplitAny

Text.Start

Text.StartsWith

Text.ToBinary

Text.ToList

Text.Trim

Text.TrimEnd

Text.TrimStart

Value.Firewall

Value.FromText

Value.Is

Value.Lineage

Value.Metadata

Value.Multiply

Value.NativeQuery

Value.NullableEquals

Value.Optimize

Value.RemoveMetadata

Value.ReplaceMetadata

Value.ReplaceType

Value.Subtract

Value.Traits

Value.Type

Variable.Value

Tour rápido pela linguagem de fórmula do Power Query M

09/05/2020 • 4 minutes to read

Esta apresentação rápida descreve como criar consultas da linguagem de fórmula Power Query M.

> **NOTE**
> M é uma linguagem que diferencia maiúsculas de minúsculas.

Criar uma consulta com o Editor de Consultas

Para criar uma consulta avançada, use o **Editor de Consulta**. Uma consulta de mashup é composta por variáveis, expressões e valores encapsulados por uma expressão **let**. Uma variável pode conter espaços usando o identificador # com o nome entre aspas como em #"Nome da variável".

Uma expressão **let** segue esta estrutura:

```
let
   Variablename = expression,
   #"Variable name" = expression2
in
   Variablename
```

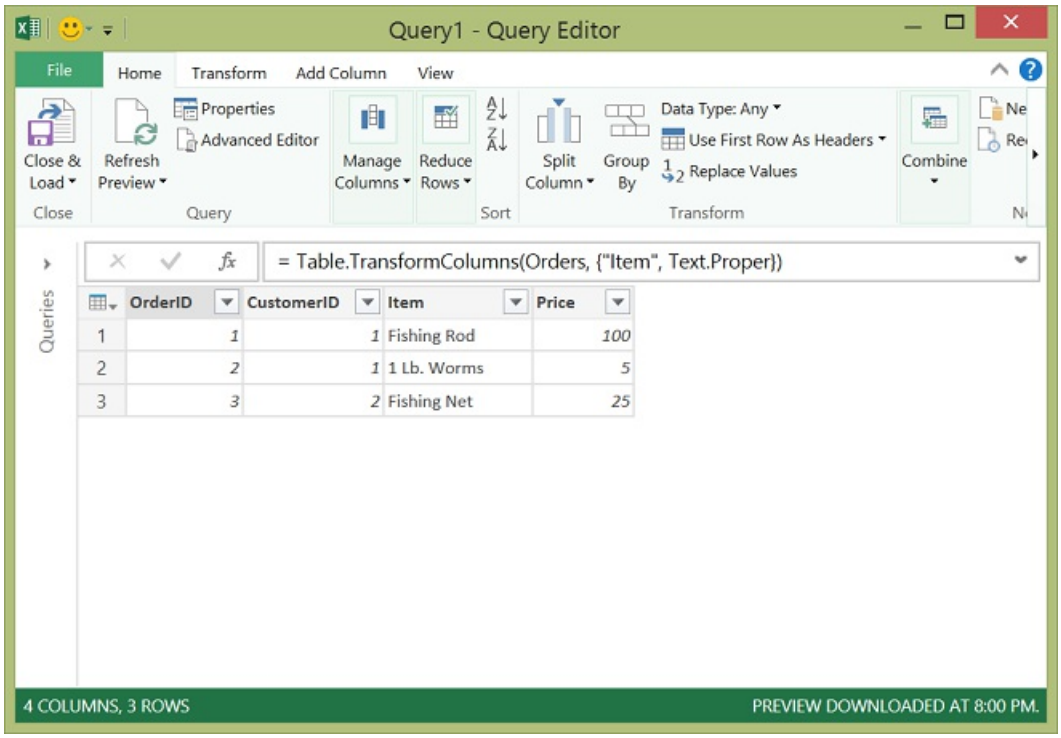
Para criar uma consulta M no **Editor de Consulta**, siga este processo básico:

- Crie uma série de etapas de fórmula de consulta que comece com a instrução **let**. Cada etapa é definida por um nome de variável de etapa. Uma **variável** M pode incluir espaços usando o caractere # como #"Nome da Etapa". Uma etapa da fórmula pode ser uma fórmula personalizada. Observe que a Linguagem de Fórmula do Power Query diferencia maiúsculas de minúsculas.
- Cada etapa da fórmula de consulta se baseia em uma etapa anterior referindo-se a uma etapa pelo seu nome de variável.
- Saída de uma etapa de fórmula de consulta usando a instrução **in**. Em geral, a última etapa de consulta é usada como o resultado do conjunto de dados final.

Para saber mais sobre expressões e valores, confira Expressões, valores e expressão let.

Etapas simples da fórmula do Power Query M

Vamos supor que você tenha criado a transformação a seguir no **Editor de Consulta** para converter nomes de produtos nas maiúsculas e minúsculas adequadas.



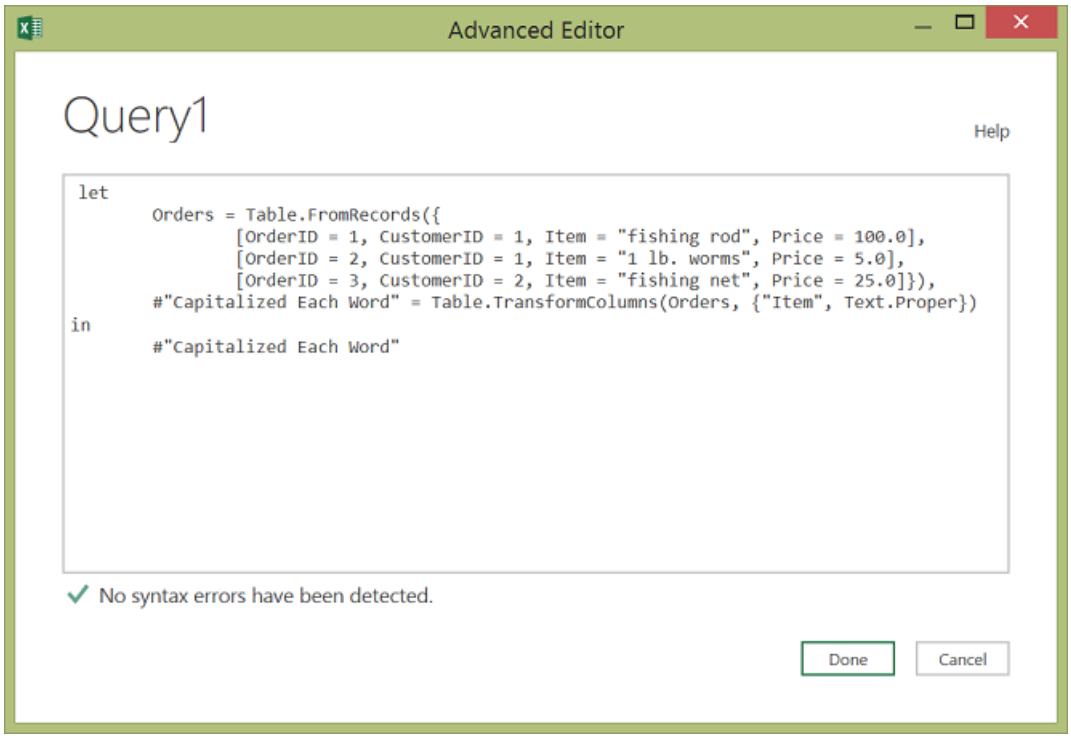
Você tem uma tabela parecida com esta:

ORDERID	CUSTOMERID	ITEM	PREÇO
1	1	vara de pescar	100
2	1	1 lb de minhocas	5
3	2	rede de pesca	25

Você deseja colocar cada palavra na coluna item em maiúscula para produzir a seguinte tabela:

ORDERID	CUSTOMERID	ITEM	PREÇO
1	1	Vara de pescar	100
2	1	1 lb Minhocas	5
3	2	Rede de pesca	25

As etapas da fórmula M para projetar a tabela original na tabela de resultados são como segue:



Este é o código que você pode colar no **Editor de Consulta**:

```
let Orders = Table.FromRecords({
    [OrderID = 1, CustomerID = 1, Item = "fishing rod", Price = 100.0],
    [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
    [OrderID = 3, CustomerID = 2, Item = "fishing net", Price = 25.0]}),
    #"Capitalized Each Word" = Table.TransformColumns(Orders, {"Item", Text.Proper})
in
    #"Capitalized Each Word"
```

Vamos examinar cada etapa da de fórmula.

1. **Pedidos** – criar uma tabela [Table](#_Table_value) com os dados para Pedidos.
2. **# "Cada Palavra em Maiúsculas"** – para colocar cada palavra em maiúscula, use Table.TransformColumns().
3. **em #"Cada Palavra em Maiúscula"** – saída da tabela com cada palavra em maiúscula.

Consulte também

Expressões, valores e expressão let
Operadores
Conversão de tipo

Especificação da linguagem do Power Query M

08/05/2020 • 2 minutes to read

A especificação descreve os valores, as expressões, os ambientes e variáveis, os identificadores e o modelo de avaliação que formam os conceitos básicos da linguagem Power Query M.

A especificação está contida nos tópicos a seguir.

Introdução

08/05/2020 • 22 minutes to read

Visão geral

O Microsoft Power Query fornece uma experiência avançada de obtenção de dados que abrange vários recursos. Uma das principais funcionalidades do Power Query é filtrar e combinar, ou seja, realizar o mashup de dados de uma ou mais coleções avançadas de fontes de dados compatíveis. Qualquer mashup de dados desse tipo é expresso usando a Linguagem de fórmula do Power Query (informalmente conhecida como "M"). Power Query incorpora documentos do M em pastas de trabalho do Excel e do Power BI para habilitar o mashup repetível de dados.

Este documento fornece a especificação para M. Após uma breve introdução que visa criar uma primeira intuição e familiaridade com a linguagem, o documento aborda a linguagem de maneira mais precisa, em várias etapas progressivas:

1. A *estrutura lexical* define o conjunto de textos que são lexicalmente válidos.
2. Os valores, as expressões, os ambientes, as variáveis, os identificadores e o modelo de avaliação que formam os *conceitos básicos* da linguagem.
3. A especificação detalhada dos *valores*, tanto primitivos quanto estruturados, define o domínio de destino da linguagem.
4. Os valores têm *tipos*, que são um tipo especial de valor, caracterizam os tipos fundamentais de valores e contêm metadados adicionais que são específicos para as formas de valores estruturados.
5. O conjunto de *operadores* no M define que tipos de expressões podem ser formados.
6. As *funções*, outro tipo de valores especiais, fornecem a base para uma biblioteca padrão avançada para o M e permitem a adição de novas abstrações.
7. Os *erros* podem ocorrer ao se aplicar operadores ou funções durante uma avaliação de expressão. Embora os erros não sejam valores, há maneiras de *tratar erros* que mapeiam erros de volta para valores.
8. As *expressões let* permitem a introdução de definições auxiliares usadas para criar expressões complexas em etapas menores.
9. as *expressões if* dão suporte à avaliação condicional.
10. As *seções* fornecem um mecanismo de modularidade simples. (O Power Query ainda não é capaz de tirar proveito de funções.)
11. Por fim, uma *gramática consolidada* coleta os fragmentos de gramática de todas as outras seções deste documento em uma definição completa.

Para teóricos de linguagem de computador: a linguagem de fórmula especificada neste documento é uma linguagem funcional essencialmente pura, de ordem mais alta, dinamicamente tipada e parcialmente lenta.

Expressões e valores

O constructo central no M é a *expressão*. Uma expressão pode ser avaliada (computada), produzindo um *valor*.

Embora muitos valores possam ser escritos literalmente como uma expressão, um valor não é uma expressão. Por exemplo, a expressão `1` é avaliada como o valor *1*; a expressão `1+1` é avaliada como o valor *2*. Essa distinção é

sutil, mas importante. As expressões são receitas para avaliação; os valores são os resultados da avaliação.

Os exemplos a seguir ilustram os diferentes tipos de valores disponíveis em M. Como uma convenção, um valor é escrito usando o formato literal em que eles apareceriam em uma expressão que é avaliada apenas para esse valor. (Observe que as `//` indicam o início de um comentário que continua até o final da linha.)

- Um valor *primitivo* é um valor de parte única, como um número, lógico, texto ou nulo. Um valor nulo pode ser usado para indicar a ausência de dados.

  ```
  123                 // A number
  true                // A logical
  "abc"               // A text
  null                // null value
  ```

- Um valor de *lista* é uma sequência ordenada de valores. O M dá suporte a listas infinitas, mas, se escritas como um literal, as listas têm um comprimento fixo. Os caracteres de chave `{` e `}` denotam o início e o fim de uma lista.

  ```
  {123, true, "A"}     // list containing a number, a logical, and
                        //     a text
  {1, 2, 3}            // list of three numbers
  ```

- Um *registro* é um conjunto de *campos*. Um campo é um par de nome/valor em que o nome é um valor de texto exclusivo dentro do registro do campo. A sintaxe literal para valores de registro permite que os nomes sejam gravados sem aspas, um formato também conhecido como *identificadores*. O exemplo a seguir mostra um registro que contém três campos denominados "`A`", "`B`" e "`C`", que têm os valores `1`, `2` e `3`.

  ```
  [
        A = 1,
        B = 2,
        C = 3
  ]
  ```

- Uma *tabela* é um conjunto de valores organizados em colunas (que são identificadas por nome) e linhas. Não há nenhuma sintaxe literal para criação de uma tabela, mas há várias funções padrão que podem ser usadas para criar tabelas com base em listas ou registros.

  Por exemplo:

  ```
  #table( {"A", "B"}, { {1, 2}, {3, 4} } )
  ```

  Isso cria uma tabela com o seguinte formato:

A	B
1	2
3	4

- Uma *função* é um valor que, quando invocado com argumentos, produz um novo valor. As funções são

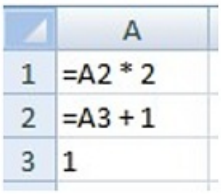
gravadas pela listagem dos *parâmetros* da função entre parênteses, seguidos pelo símbolo de ir para `=>`, seguido pela expressão que define a função. Essa expressão normalmente se refere aos parâmetros (por nome).

```
(x, y) => (x + y) / 2`
```

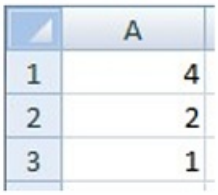
Avaliação

O modelo de avaliação da linguagem M é modelado após o modelo de avaliação normalmente encontrado em planilhas, em que a ordem dos cálculos pode ser determinada com base nas dependências entre as fórmulas nas células.

Se você tiver escrito fórmulas em uma planilha como o Excel, poderá reconhecer que as fórmulas à esquerda resultarão nos valores à direita quando calculadas:



	A
1	=A2 * 2
2	=A3 + 1
3	1



	A
1	4
2	2
3	1

No M, uma expressão pode referenciar outras partes da expressão por nome, e o processo de avaliação determinará automaticamente a ordem na qual as expressões referenciadas serão calculadas.

Podemos usar um registro para produzir uma expressão que seja equivalente ao exemplo da planilha acima. Ao inicializar o valor de um campo, podemos fazer referência a outros campos dentro do registro pelo uso do nome do campo, da seguinte maneira:

```
[
    A1 = A2 * 2,
    A2 = A3 + 1,
    A3 = 1
]
```

A expressão acima é equivalente à seguinte (no sentido de que ambas são avaliadas para valores iguais):

```
[
    A1 = 4,
    A2 = 2,
    A3 = 1
]
```

Os registros podem estar contidos ou *aninhados* em outros registros. É possível usar o *operador de pesquisa* (`[]`) para acessar os campos de um registro por nome. Por exemplo, o seguinte registro tem um campo chamado `Sales` com um registro e um campo chamado `Total` que acessa os campos `FirstHalf` e `SecondHalf` do registro `Sales`:

```
[
    Sales = [ FirstHalf = 1000, SecondHalf = 1100 ],
    Total = Sales[FirstHalf] + Sales[SecondHalf]
]
```

A expressão acima é equivalente à seguinte, quando avaliada:

```
[
    Sales = [ FirstHalf = 1000, SecondHalf = 1100 ],
    Total = 2100
]
```

Os registros também podem estar contidos em listas. É possível usar o *operador de índice posicional* (`{}`) para acessar um item em uma lista pelo respectivo índice numérico. Os valores dentro de uma lista são referenciados usando um índice de base zero desde o início da lista. Por exemplo, os índices `0` e `1` são usados para referenciar o primeiro e o segundo itens na lista abaixo:

```
[
    Sales =
        {
            [
                Year = 2007,
                FirstHalf = 1000,
                SecondHalf = 1100,
                Total = FirstHalf + SecondHalf // 2100
            ],
            [
                Year = 2008,
                FirstHalf = 1200,
                SecondHalf = 1300,
                Total = FirstHalf + SecondHalf // 2500
            ]
        },
    TotalSales = Sales{0}[Total] + Sales{1}[Total] // 4600
]
```

As expressões de membro de lista e de registro (bem como as expressões let, apresentadas mais adiante) são avaliadas usando *avaliação lenta*, o que significa que são avaliadas somente conforme necessário. Todas as outras expressões são avaliadas usando a *avaliação rápida*, ou seja, imediatamente, quando encontradas durante o processo de avaliação. Uma boa maneira de pensar sobre nisso é lembrar de que a avaliação de uma expressão de lista ou registro retornará um valor de lista ou registro que se lembra de como seus itens de lista ou campos de registro precisam ser computados, quando solicitado (por operadores de pesquisa ou índice).

Funções

No M, uma *função* é um mapeamento de um conjunto de valores de entrada para um valor de saída. Para escrever uma função, primeiro é preciso nomear o conjunto necessário de valores de entrada (os parâmetros para a função) e, em seguida, fornecer uma expressão que calcule o resultado da função usando esses valores de entrada (o corpo da função) após o símbolo de ir para (`=>`). Por exemplo:

```
(x) => x + 1                    // function that adds one to a value
(x, y) =>  x + y                // function that adds two values
```

Uma função é um valor, assim como um valor de número ou de texto. O exemplo a seguir mostra uma função que é o valor de um campo Add, que é *invocada* ou executada de vários outros campos. Quando uma função é chamada, um conjunto de valores é especificado, que são substituídos logicamente pelo conjunto necessário de valores de entrada dentro da expressão do corpo da função.

```
[
    Add = (x, y) => x + y,
    OnePlusOne = Add(1, 1),     // 2
    OnePlusTwo = Add(1, 2)      // 3
]
```

Biblioteca

O M inclui um conjunto comum de definições disponíveis para uso por meio de uma expressão chamada de *biblioteca padrão* ou simplesmente biblioteca. Essas definições consistem em um conjunto de valores nomeados. Os nomes de valores fornecidos por uma biblioteca estão disponíveis para uso em uma expressão sem terem sido definidos explicitamente pela expressão. Por exemplo:

```
Number.E                        // Euler's number e (2.7182...)
Text.PositionOf("Hello", "ll")  // 2
```

Operadores

O M inclui um conjunto de operadores que podem ser usados em expressões. Os *operadores* são aplicados a *operandos* para formar expressões simbólicas. Por exemplo, na expressão `1 + 2`, os números `1` e `2` são operandos e o operador é o operador de adição (`+`).

O significado de um operador pode variar dependendo do tipo de valores dos seus operandos. Por exemplo, o operador de adição pode ser usado com outros tipos de valores que não sejam números:

```
1 + 2                   // numeric addition: 3
#time(12,23,0) + #duration(0,0,2,0)
                        // time arithmetic: #time(12,25,0)
```

Outro exemplo de um operador com significado dependente do operando é o operador de combinação (`&`):

```
"A" & "BC"              // text concatenation: "ABC"
{1} & {2, 3}            // list concatenation: {1, 2, 3}
[ a = 1 ] & [ b = 2 ]   // record merge: [ a = 1, b = 2 ]
```

Observe que nem todas as combinações de valores são compatíveis com um operador. Por exemplo:

```
1 + "2"  // error: adding number and text is not supported
```

As expressões que, quando avaliadas, encontram condições de operador indefinidas são avaliadas como erros. Mais informações sobre erros no M serão fornecidas posteriormente.

Metadados

Os *metadados* são informações sobre um valor que está associado a um valor. Os metadados são representados como um valor de registro, chamado de *registro de metadados*. Os campos de um registro de metadados podem ser usados para armazenar os metadados de um valor.

Cada valor tem um registro de metadados. Se o valor do registro de metadados não tiver sido especificado, o registro de metadados estará vazio (não terá nenhum campo).

Os registros de metadados fornecem uma forma de associar informações adicionais a qualquer tipo de valor de maneira discreta. A associação de um registro de metadados com um valor não altera o valor nem o comportamento dele.

Um valor de registro de metadados `y` é associado a um valor `x` existente usando a sintaxe `x meta y`. Por exemplo, este código associa um registro de metadados aos campos `Rating` e `Tags` com o valor de texto `"Mozart"`:

```
"Mozart" meta [ Rating = 5, Tags = {"Classical"} ]
```

Para valores que já contêm um registro de metadados não vazio, o resultado de aplicar o operador meta é que a mesclagem do registro de metadados novo e do existente é computada. Por exemplo, as seguintes duas expressões são equivalentes uma à outra e à expressão anterior:

```
("Mozart" meta [ Rating = 5 ]) meta [ Tags = {"Classical"} ]
"Mozart" meta ([ Rating = 5 ] & [ Tags = {"Classical"} ])
```

Um registro de metadados pode ser acessado para um determinado valor usando a função *Value.Metadata*. No exemplo a seguir, a expressão no campo `ComposerRating` acessa o registro de metadados do valor no campo `Composer` e, em seguida, acessa o campo `Rating` do registro de metadados.

```
[
    Composer = "Mozart" meta [ Rating = 5, Tags = {"Classical"} ],
    ComposerRating = Value.Metadata(Composer)[Rating] // 5
]
```

Expressão let

Muitos dos exemplos mostrados até agora incluíam todos os valores literais da expressão no resultado da expressão. A expressão *let* permite que um conjunto de valores sejam computados, que nomes sejam atribuídos a eles e, depois, que esses valores sejam usados em uma expressão subsequente que segue a instrução *in*. Por exemplo, em nosso exemplo de dados de vendas, poderíamos fazer:

```
let
    Sales2007 =
        [
            Year = 2007,
            FirstHalf = 1000,
            SecondHalf = 1100,
            Total = FirstHalf + SecondHalf // 2100
        ],
    Sales2008 =
        [
            Year = 2008,
            FirstHalf = 1200,
            SecondHalf = 1300,
            Total = FirstHalf + SecondHalf // 2500
        ]
  in Sales2007[Total] + Sales2008[Total] // 4600
```

O resultado da expressão acima é um valor numérico (`4600`) que foi computado com base nos valores associados aos nomes `Sales2007` e `Sales2008`.

Expressão if

A expressão `if` seleciona entre duas expressões com base em uma condição lógica. Por exemplo:

```
if 2 > 1 then
    2 + 2
else
    1 + 1
```

A primeira expressão (`2 + 2`) será selecionada se a expressão lógica (`2 > 1`) for verdadeira e a segunda expressão (`1 + 1`) será selecionada se ela for falsa. A expressão selecionada (nesse caso, `2 + 2`) é avaliada e torna-se o resultado da expressão `if` (`4`).

Errors

Um *erro* é uma indicação de que o processo de avaliação de uma expressão não pôde produzir um valor.

Os erros são gerados por operadores e funções que encontram condições de erro ou pelo uso da expressão error. Os erros são tratados usando a expressão try. Quando um erro é gerado, é especificado um valor que pode ser usado para indicar por que o erro ocorreu.

```
let Sales =
    [
        Revenue = 2000,
        Units = 1000,
        UnitPrice = if Units = 0 then error "No Units"
                    else Revenue / Units
    ],
    UnitPrice = try Number.ToText(Sales[UnitPrice])
in "Unit Price: " &
    (if UnitPrice[HasError] then UnitPrice[Error][Message]
    else UnitPrice[Value])
```

O exemplo acima acessa o campo `Sales[UnitPrice]` e formata o valor que produz o resultado:

```
"Unit Price: 2"
```

Se o campo `Units` tivesse sido zero, o campo `UnitPrice` teria gerado um erro que teria sido tratado pela `try` . O valor resultante teria sido:

```
"No Units"
```

Uma expressão `try` converte valores e erros adequados em um valor de registro que indica se a expressão try tratou um erro ou não, bem como o valor apropriado ou o registro de erro que ela extraiu ao tratar o erro. Por exemplo, considere a seguinte expressão que gera um erro e o trata imediatamente:

```
try error "negative unit count"
```

Essa expressão avalia valor de registro aninhado a seguir, explicando as pesquisas de campo `[HasError]` , `[Error]` e `[Message]` no exemplo de preço unitário anterior.

```
[
    HasError = true,
    Error =
        [
            Reason = "Expression.Error",
            Message = "negative unit count",
            Detail = null
        ]
]
```

Um caso comum é substituir erros por valores padrão. A expressão `try` também pode ser usada com uma cláusula `otherwise` opcional para obter apenas isso em um formato compacto:

```
try error "negative unit count" otherwise 42
// 42
```

Estrutura lexical

26/05/2020 • 19 minutes to read

Documentos

Um *documento* do M é uma sequência ordenada de caracteres Unicode. O M permite o uso de diferentes classes de caracteres Unicode em diferentes partes de um documento do M. Para obter informações sobre classes de caracteres Unicode, confira *O Padrão Unicode, Versão 3.0*, seção 4.5.

Um documento consiste em exatamente uma *expressão* ou em grupos de *definições* organizadas em *seções*. As seções são descritas em detalhes no Capítulo 10. Em termos conceituais, as seguintes etapas são usadas para ler uma expressão de um documento:

1. O documento é decodificado de acordo com o respectivo esquema de codificação de caracteres em uma sequência de caracteres Unicode.
2. A análise lexical é executada, convertendo assim o fluxo de caracteres Unicode em um fluxo de tokens. As subseções restantes desta seção abordam a análise lexical.
3. A análise sintática é executada, convertendo assim o fluxo de tokens em um formulário que pode ser avaliado. Esse processo é abordado nas seções subsequentes.

Convenções gramaticais

As gramáticas lexical e sintática são apresentadas usando *produções gramaticais*. Cada produção de gramática define um símbolo não terminal e as possíveis expansões desse símbolo não terminal em sequências de símbolos terminais ou não terminais. Em produções de gramática, os símbolos não terminais são mostrados em itálico e os símbolos *terminais* são mostrados em uma fonte de largura fixa.

A primeira linha de uma produção de gramática é o nome do símbolo não terminal que está sendo definido, seguido por dois-pontos. Cada linha recuada sucessiva contém uma possível expansão do símbolo não terminal fornecida como uma sequência de símbolos terminais ou não terminais. Por exemplo, a produção:

expressão-if:
    `if` *condição-if* `then` *expressão-true* `else` *expressão-false*

define uma *expressão-if* consistindo do token `if`, seguido por uma *condição-if*, seguida pelo token `then`, seguido por uma *expressão-true*, seguida pelo token `else`, seguido por uma *expressão-false*.

Quando há mais de uma possível expansão de um símbolo não terminal, as alternativas são listadas em linhas separadas. Por exemplo, a produção:

lista-de-variáveis:
    variável
    lista-de-variáveis `,` *variável*

define uma *lista-de-variáveis* para consistir de uma *variável* ou consistir de uma *lista-de-variáveis* seguida por uma *variável*. Em outras palavras, a definição é recursiva e especifica que uma lista de variáveis consiste em uma ou mais variáveis, separadas por vírgulas.

Um sufixo subscrito "$_{opt}$" é usado para indicar um símbolo opcional. A produção:

especificação-do-campo:
    `optional`$_{opt}$ *nome-do-campo* `=` *tipo-do-campo*

é uma versão abreviada de:

especificação-de-campo:
    nome-do-campo `=` *tipo-do-campo*
    `optional` *nome-do-campo* `=` *tipo-do-campo*

e define uma *especificação-do-campo* para, opcionalmente, começar com o símbolo do terminal `optional` seguido por *nome-do-campo*, o símbolo do terminal `=` e um *tipo-de-campo*.

As alternativas normalmente são listadas em linhas separadas, porém, nos casos em que há muitas alternativas, a frase "one of" pode preceder uma lista de expansões dadas em uma única linha. Isso é simplesmente uma forma mais prática do que listar cada uma das alternativas em uma linha separada. Por exemplo, a produção:

dígito-decimal: one of
    `0 1 2 3 4 5 6 7 8 9`

é uma versão abreviada de:

dígito-decimal:
    `0`
    `1`
    `2`
    `3`
    `4`
    `5`
    `6`
    `7`
    `8`
    `9`

Análise lexical

A produção *unidade-lexical* define a gramática lexical para um documento do M. Todos os documentos do M válidos estão em conformidade com essa gramática.

unidade-lexical:
    elementos-lexicais$_{opt}$
elementos-lexicais:
    elemento-lexical
    elemento-lexical
    elementos-lexicais
elemento-lexical:
    espaço em branco
    token comentário

No nível lexical, um documento do M consiste em um fluxo de elementos de *espaço em branco*, *comentário* e *token*. Cada uma dessas produções é abordada nas seções a seguir. Somente os elementos *token* são significativos na gramática sintática.

Espaço em branco

O espaço em branco é usado para separar comentários e tokens em um documento do M. O espaço em branco inclui o caractere de espaço (que faz parte da classe Unicode ZS), bem como a tabulação horizontal e a vertical, o feed de formulário e as sequências de caracteres de nova linha. As sequências de caracteres de nova linha incluem retorno de carro, alimentação de linha, retorno de carro seguido de alimentação de linha, linha seguinte e

caracteres separadores de parágrafo.

espaço em branco:
    Qualquer caractere com classe Unicode ZS
    Caractere de tabulação horizontal (`U+0009`)
    Caractere de tabulação vertical (`U+000B`)
    Caractere de feed de formulário (`U+000C`)
    Caractere de retorno de carro (`U+000D`) seguido pelo caractere de alimentação de linha (`U+000A`)
    caractere-de-nova-linha
caractere-de-nova-linha:
    Caractere de retorno de carro (`U+000D`)
    Caractere de feed de linha (`U+000A`)
    Caractere de próxima linha (`U+0085`)
    Caractere separador de linha (`U+2028`)
    Caractere separador de parágrafo (`U+2029`)

Para compatibilidade com ferramentas de edição de código-fonte que adicionam marcadores de final de arquivo e para permitir que um documento seja exibido como uma sequência de linhas encerradas corretamente, as seguintes transformações são aplicadas, em ordem, a um documento do M:

- Se o último caractere do documento for um caractere de Controle Z (`U+001A`), esse caractere será excluído.
- Um caractere de retorno de carro (`U+000D`) será adicionado ao final do documento se esse documento não estiver vazio e se o último caractere do documento não for um retorno de carro (`U+000D`), uma alimentação de linha (`U+000A`), um separador de linha (`U+2028`) nem um separador de parágrafo (`U+2029`).

Comentários

Dois formulários de comentários são compatíveis: comentários de linha única e comentários delimitados. Os *comentários de linha única* começam com os caracteres `//` e se estendem até o final da linha de origem. Os *comentários delimitados* começam com os caracteres `/*` e terminam com os caracteres `*/`.

Os comentários delimitados podem abranger várias linhas.

comentário:
    comentário-de-linha-única
    comentário-delimitado
comentário-de-linha-única:
    `//` *caracteres-de-comentário-de-linha-única*$_{opt}$
caracteres-de-comentário-de-linha-única:
    caracteres-de-comentário-de-linha-única caractere-de-comentário-de-linha-única$_{opt}$
caractere-de-comentário-de-linha-única:
    Qualquer caractere Unicode, exceto um comentário-delimitado por *caractere-de-nova-linha*
:
    `/*` *texto-de-comentário-delimitado*$_{opt}$ *asteriscos* `/`
texto-de-comentário-delimitado:
    seção-de-comentário-delimitado texto-de-comentário-delimitado$_{opt}$
seção-de-comentário-delimitado:
    `/`
    asteriscos$_{opt}$ *caracteres-diferentes-de-barra-e-asterisco*
asteriscos:
    `*` *asteriscos*$_{opt}$
caracteres-diferentes-de-barra-e-asterisco asteriscos:
    Qualquer caractere Unicode, exceto `*` ou `/`

Os comentários não podem ser aninhados. As sequências de caracteres `/*` e `*/` não têm nenhum significado especial dentro de um comentário de linha única, e as sequências de caracteres `//` e `/*` não têm nenhum significado especial dentro de um comentário delimitado.

Os comentários não são processados em literais de texto. O exemplo

```
/* Hello, world
*/
    "Hello, world"
```

inclui um comentário delimitado.

O exemplo

```
// Hello, world
//
"Hello, world" // This is an example of a text literal
```

mostra vários comentários de linha única.

Tokens

Um *token* é um identificador, uma palavra-chave, um literal, um operador ou um pontuador. O espaço em branco e os comentários são usados para separar tokens, mas não são considerados tokens.

token:
identificador
palavra-chave
literal
operador-ou-pontuador

Sequências de escape de caractere

Os valores de texto do M podem conter caracteres Unicode arbitrários. Os literais de texto, por sua vez, são limitados a caracteres gráficos e exigem o uso de *sequências de escape* para caracteres não gráficos. Por exemplo, para incluir um retorno de carro, avanço de linha ou caractere de tabulação em um literal de texto, é possível usar as sequências de escape `#(cr)`, `#(lf)` e `#(tab)`, respectivamente. Para inserir os caracteres iniciais de sequência de escape `#(` em um literal de texto, o `#` propriamente dito precisa ser escapado:

```
#(#)(
```

As sequências de escape também podem conter valores curtos (quatro dígitos hexadecimais) ou longos (oito dígitos hexadecimais) de ponto de código Unicode. Portanto, estas três sequências de escape são equivalentes:

```
#(000D)     // short Unicode hexadecimal value
#(0000000D) // long Unicode hexadecimal value
#(cr)       // compact escape shorthand for carriage return
```

Vários códigos de escape podem ser incluídos em uma única sequência de escape, separados por vírgulas. As seguintes duas sequências são, portanto, equivalentes:

```
#(cr,lf)
#(cr)#(lf)
```

A descrição a seguir representa o mecanismo padrão de saída de caracteres em um documento do M.

sequência-de-escape-de-caracteres:
`#(` *lista-de-sequências-de-escape* `)`
lista-de-sequências-de-escape:
sequência-de-escape-única
sequência-de-escape-única `,` *lista-de-sequência-de-escape*
sequência-de-escape-única:
sequência-de-escape-unicode-longa
sequência-de-escape-unicode-curta
sequência-de-escape-de-caracteres-de-controle
escape-escape
sequência-de-escape-unicode-longa:
dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal
sequência-de-escape-unicode-curta:
dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal
sequência-de-escape-de-caracteres-de-controle:
caractere-de-controle
caractere-de-controle:
`cr`
`lf`
`tab`
escape-escape:
`#`

Literais

Um *literal* é uma representação de código-fonte de um valor.

literal:
literal-lógico
literal-de-número
literal-de-texto
literal-nulo
literal-textual

Literais nulos

O literal nulo é usado para escrever o valor de `null`. O valor de `null` representa um valor ausente.

literal-nulo:
`null`

Literais lógicos

Um literal lógico é usado para gravar os valores `true` e `false` e produz um valor lógico.

literal-lógico:
`true`
`false`

Literais de número

Um literal de número é usado para escrever um valor de número e produz um valor de número.

literal-de-número:
literal-de-número-decimal
literal-de-número-hexadecimal
literal-de-número-decimal:

dígitos-decimais `.` *dígitos-decimais parte-do-expoente*$_{opt}$

`.` *dígitos-decimais parte-do-expoente*$_{opt}$

dígitos-decimais parte-do-expoente$_{opt}$

dígitos-decimais:

dígito-decimal dígitos-decimais$_{opt}$

dígito-decimal: um de

`0 1 2 3 4 5 6 7 8 9`

parte-do-expoente:

`e` *sinal*$_{opt}$ *dígitos-decimais*

`E` *sinal*$_{opt}$ *dígitos-decimais*

sinal: one of

`+ -`

literal-de-número-hexadecimal:

`0x` *dígitos-hexadecimais*

`0X` *dígitos-hexadecimais*

dígitos-hexadecimais:

dígito-hexadecimal dígitos-hexadecimais$_{opt}$

dígito-hexadecimal: um entre

`0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f`

Um número pode ser especificado em formato hexadecimal colocando-se os caracteres `0x` antes dos *dígitos-hexadecimais*. Por exemplo:

```
0xff // 255
```

Observe que, se um ponto decimal for incluído em um número literal, pelo menos um dígito precisará existir depois dele. Por exemplo, `1.3` é um número literal, mas `1.` e `1.e3` não são.

Literais de texto

Um literal de texto é usado para escrever uma sequência de caracteres Unicode e produz um valor de texto.

literal-de-texto:

`"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

caracteres-de-literal-de-texto:

caractere-de-literal-de-texto caracteres-de-literal-de-texto$_{opt}$

caractere-de-literal-de-texto:

caractere-de-texto-único

sequência-de-escape-de-caracteres

sequência-de-escape-com-aspas-duplas

caractere-de-texto-único:

Qualquer caractere, exceto `"` (`U+0022`) ou `#` (`U+0023`) seguido por `(` (`U+0028`)

sequência-de-escape-com-aspas-duplas:

`""` (`U+0022`, `U+0022`)

Para incluir aspas em um valor de texto, a marca de aspas é repetida, da seguinte maneira:

```
"The ""quoted"" text" // The "quoted" text
```

A produção *sequência-de-escape-de-caracteres* pode ser usada para gravar caracteres em valores de texto sem precisar codificá-los diretamente como caracteres Unicode no documento. Por exemplo, um retorno de carro e uma alimentação de linha podem ser escritos em um valor de texto como:

```
"Hello world#(cr,lf)"
```

Literais textuais

Um literal textual é usado para armazenar uma sequência de caracteres Unicode que foram inseridos por um usuário como código, mas que não podem ser analisados corretamente como código. No tempo de execução, ele produz um valor de erro.

literal-textual:
`#!"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

Identificadores

Um *identificador* é um nome usado para fazer referência a um valor. Os identificadores podem ser identificadores comuns ou identificadores entre aspas.

identificador:
identificador-comum
identificador-entre-aspas
identificador-comum:
identificador-disponível
identificador-disponível caractere-de-ponto identificador-comum
identificador-disponível:
Uma *palavra-chave-ou-identificador* que não é uma *palavra-chave*
palavra-chave-ou-identificador:
caractere-inicial-do-identificador caracteres-da-parte-do-identificador$_{opt}$
caractere-inicial-do-identificador:
caractere-de-letra
caractere-de-sublinhado
caracteres-da-parte-do-identificador:
caractere-da-parte-do-identificador caracteres-da-parte-do-identificador$_{opt}$
caractere-da-parte-do-identificador:
caractere-de-letra
caractere-de-dígito-decimal
caractere-de-sublinhado
caractere-de-conexão
caractere-de-combinação
caractere-de-formatação
caractere-de-ponto:
`.` (`U+002E`)
caractere-de-sublinhado:
`_` (`U+005F`)
caractere-de-letra:
Um caractere Unicode de uma das classes Lu, Ll, Lt, Lm, Lo ou Nl
caractere-de-combinação:
Um caractere Unicode de uma das classes Mn ou Mc
caractere-de-dígito-decimal:
Um caractere Unicode da classe Nd
caractere-de-conexão:
Um caractere Unicode da classe Pc
caractere-de-formatação:
Um caractere Unicode da classe Cf

Um *identificador-entre-aspas* pode ser usado para permitir que qualquer sequência de zero ou mais caracteres

Unicode seja usada como um identificador, incluindo palavras-chave, espaço em branco, comentários, operadores e pontuadores.

identificador-entre-aspas:
`#"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

Observe que as sequências de escape e as aspas duplas para as aspas de escape podem ser usadas em um *identificador entre aspas*, assim como em um *literal-de-texto*.

O seguinte exemplo usa aspas em identificadores naqueles nomes que contêm um caractere de espaço:

```
[
    #"1998 Sales" = 1000,
    #"1999 Sales" = 1100,
    #"Total Sales" = #"1998 Sales" + #"1999 Sales"
]
```

O seguinte exemplo usa aspas em identificadores a fim de incluir o operador `+` em um identificador:

```
[
    #"A + B" = A + B,
    A = 1,
    B = 2
]
```

Identificadores generalizados

Há dois lugares no M em que nenhuma ambiguidade é introduzida por identificadores que contêm espaços em branco ou que são palavras-chave ou literais de número. Esses locais são os nomes dos campos de registro em um literal de registro e em um operador de acesso de campo (`[ ]`). Ali, o M permite o uso desses identificadores sem necessidade de usar de identificadores entre aspas.

```
[
    Data = [ Base Line = 100, Rate = 1.8 ],
    Progression = Data[Base Line] * Data[Rate]
]
```

Os identificadores usados para nomear e acessar campos são chamados de *identificadores generalizados* e são definidos da seguinte maneira:

identificador-generalizado:
    parte-do-identificador-generalizado
    identificador-generalizado separado somente por espaços em branco (`U+0020`)
parte-do-identificador-generalizado
parte-do-identificador-generalizado:
    segmento-do-identificador-generalizado
    caractere-de-dígito-decimal segmento-do-identificador-generalizado
segmento-do-identificador-generalizado:
    palavra-chave-ou-identificador
    palavra-chave-ou-identificador caractere-de-ponto palavra-chave-ou-identificador

Palavras-chave

Uma *palavra-chave* é uma sequência de caracteres semelhante à de um identificador que é reservada e não pode ser usada como um identificador, exceto ao usar o mecanismo de colocação de aspas em identificadores ou nos casos em que um identificador genérico é permitido.

palavra-chave: um entre

`and as each else error false if in is let meta not null or otherwise`
`section shared then true try type #binary #date #datetime`
`#datetimezone #duration #infinity #nan #sections #shared #table #time`

Operadores e pontuadores

Há vários tipos de operadores e pontuadores. Os operadores são usados em expressões para descrever as operações que envolvem um ou mais operandos. Por exemplo, a expressão `a + b` usa o operador `+` para adicionar os dois operandos `a` e `b`. Os pontuadores servem para agrupamento e separação.

operador-ou-pontuador: um entre
`, ; = < <= > >= <> + - * / & ( ) [ ] { } @ ! ? => .. ...`

Conceitos básicos

08/05/2020 • 15 minutes to read

Esta seção aborda os conceitos básicos que aparecem nas seções posteriores.

Valores

Apenas um dado é chamado de *valor*. Falando de maneira mais ampla, há duas categorias gerais de valores: *valores primitivos*, que são atômicos, e *valores estruturados*, que são construídos com valores primitivos e outros valores estruturados. Por exemplo, os valores \

```
1
true
3.14159
"abc"
```

são primitivos, pois não são compostos por outros valores. Por outro lado, os valores

```
{1, 2, 3}
[ A = {1}, B = {2}, C = {3} ]
```

são construídos usando valores primitivos e, no caso do registro, outros valores estruturados.

Expressões

Uma *expressão* é uma fórmula usada para construir valores. Uma expressão pode ser formada usando diversas construções sintáticas. Veja a seguir alguns exemplos de expressões. Cada linha é uma expressão separada.

```
"Hello World"            // a text value
123                      // a number
1 + 2                    // sum of two numbers
{1, 2, 3}                // a list of three numbers
[ x = 1, y = 2 + 3 ]     // a record containing two fields:
                         //        x and y
(x, y) => x + y          // a function that computes a sum
if 2 > 1 then 2 else 1   // a conditional expression
let x = 1 + 1  in x * 2  // a let expression
error "A"                // error with message "A"
```

A forma de expressão mais simples, como visto acima, é um literal que representa um valor.

Expressões mais complexas são criadas com base em outras expressões, chamadas de *subexpressões*. Por exemplo:

```
1 + 2
```

Na verdade, a expressão acima é composta por três expressões. Os literais `1` e `2` são subexpressões da expressão pai `1 + 2`.

Executar o algoritmo definido pelas construções sintáticas usadas em uma expressão é chamado de *avaliar* a expressão. Cada tipo de expressão tem regras de como é avaliada. Por exemplo, uma expressão literal como `1` produzirá um valor constante, enquanto a expressão `a + b` usará os valores resultantes produzidos pela avaliação

de duas outras expressões (`a` e `b`) e as adicionará de acordo com algum conjunto de regras.

Ambientes e variáveis

As expressões são avaliadas em um determinado ambiente. Um *ambiente* é um conjunto de valores nomeados, chamados *variáveis*. Cada variável em um ambiente tem um nome exclusivo dentro do ambiente, chamado de *identificador*.

Uma expressão de nível superior (ou *raiz*) é avaliada dentro do *ambiente global*. O ambiente global é fornecido pelo avaliador da expressão, em vez de ser determinado com base no conteúdo da expressão que está sendo avaliada. O conteúdo do ambiente global inclui as definições de biblioteca padrão e pode ser afetado por exportações de seções de um conjunto de documentos. (Para simplificar, os exemplos nesta seção presumirão que o ambiente global está vazio. Ou seja, presume-se que não há uma biblioteca padrão nem outras definições baseadas em seção.)

O ambiente usado para avaliar uma subexpressão é determinado pela expressão pai. A maioria dos tipos de expressão pai avalia uma subexpressão dentro do mesmo ambiente em que foi avaliada, mas alguns usam um ambiente diferente. O ambiente global é o *ambiente pai* no qual a expressão global é avaliada.

Por exemplo, a *expressão-inicializadora-de-registro* avalia a subexpressão de cada campo com um ambiente modificado. O ambiente modificado inclui uma variável para cada um dos campos do registro, exceto pelo que está sendo inicializado. Incluir os outros campos do registro permite que os campos dependam dos valores dos campos. Por exemplo:

```
[
    x = 1,          // environment: y, z
    y = 2,          // environment: x, z
    z = x + y       // environment: x, y
]
```

Da mesma forma, a *expressão-let* avalia a subexpressão de cada variável com um ambiente que contém cada uma das variáveis de ler, exceto pela que está sendo inicializada. A *expressão-let* avalia a expressão que segue in com um ambiente que contém todas as variáveis:

```
let

    x = 1,          // environment: y, z
    y = 2,          // environment: x, z
    z = x + y       // environment: x, y
in
    x + y + z       // environment: x, y, z
```

(Acontece que tanto a *expressão-inicializadora-de-registro* quanto a *expressão-let* na verdade definem *dois* ambientes, um dos quais inclui a variável que está sendo inicializada). Isso é útil para definições recursivas avançadas e é abordado em Referências de identificador.

Para formar os ambientes para as subexpressões, as novas variáveis são "mescladas" com as variáveis do ambiente pai. O exemplo a seguir mostra os ambientes para registros aninhados:

```
[
    a =
    [

        x = 1,      // environment: b, y, z
        y = 2,      // environment: b, x, z
        z = x + y   // environment: b, x, y
    ],
    b = 3           // environment: a
]
```

O exemplo a seguir mostra os ambientes para um registro aninhado em um let:

```
Let
    a =
    [
        x = 1,       // environment: b, y, z
        y = 2,       // environment: b, x, z
        z = x + y    // environment: b, x, y
    ],
    b = 3            // environment: a
in
    a[z] + b         // environment: a, b
```

Mesclar variáveis com um ambiente pode introduzir um conflito entre variáveis (uma vez que cada variável em um ambiente precisa ter um nome exclusivo). O conflito é resolvido da seguinte maneira: se o nome de uma nova variável que está sendo mesclada for igual ao de uma variável existente no ambiente pai, a nova variável terá precedência no novo ambiente. No exemplo a seguir, a variável interna (aninhada mais profundamente) `x` terá precedência sobre a variável externa `x`.

```
[
    a =
    [
        x = 1,       // environment: b, x (outer), y, z
        y = 2,       // environment: b, x (inner), z
        z = x + y    // environment: b, x (inner), y
    ],
    b = 3,           // environment: a, x (outer)
    x = 4            // environment: a, b
]
```

Referências de identificador

Uma *referência-de-identificador* é usada para fazer referência a uma variável em um ambiente.

expressão-de-identificador:
    referência-de-identificador
referência-de-identificador:
    referência-de-identificador-exclusiva
    referência-de-identificador-inclusiva

A forma mais simples de referência de identificador é uma *referência-de-identificador-exclusiva*:

referência-de-identificador-exclusiva:
    identificador

É um erro uma *referência-de-identificador-exclusiva* se referir a uma variável que não faz parte do ambiente da expressão em que o identificador aparece ou se referir a um identificador que está sendo inicializado.

Uma *referência-de-identificador-inclusiva* pode ser usada para ter acesso ao ambiente que inclui o identificador sendo inicializado. Se for usada em um contexto em que não há um identificador sendo inicializado, ela será equivalente a uma *referência-de-identificador-exclusiva*.

referência-de-identificador-inclusiva:
`@` *identificador*

Isso é útil ao definir funções recursivas, pois o nome da função normalmente não estaria no escopo.

```
[
    Factorial = (n) =>
        if n <= 1 then
            1
        else
            n * @Factorial(n - 1),  // @ is scoping operator

    x = Factorial(5)
]
```

Assim como ocorre com uma *expressão-inicializadora-de-registro*, uma *referência-de-identificador-inclusiva* pode ser usada dentro de uma *expressão-let* para acessar o ambiente que inclui o identificador que está sendo inicializado.

Ordem de avaliação

Considere a seguinte expressão que inicializa um registro:

```
[
    C = A + B,
    A = 1 + 1,
    B = 2 + 2
]
```

Quando avaliada, essa expressão produz o seguinte valor de registro:

```
[
    C = 6,
    A = 2,
    B = 4
]
```

A expressão declara que, para executar o cálculo `A + B` para o campo `C`, os valores dos campos `A` e `B` devem ser conhecidos. Este é um exemplo da *ordenação de dependências* de cálculos fornecida por uma expressão. O avaliador de M segue a ordenação de dependências fornecida pelas expressões, mas é livre para executar os cálculos restantes em qualquer ordem escolhida. Por exemplo, a ordem de computação poderia ser:

```
A = 1 + 1
B = 2 + 2
C = A + B
```

Ou poderia ser:

```
B = 2 + 2
A = 1 + 1
C = A + B
```

Ou, como não dependem uns dos outros, `A` e `B` podem ser computados simultaneamente:

`B = 2 + 2` *simultaneamente com* `A = 1 + 1`
`C = A + B`

Efeitos colaterais

Permitir que um avaliador de expressão compute automaticamente a ordem dos cálculos para casos em que não há dependências explícitas declaradas pela expressão é um modelo de computação simples e poderoso.

No entanto, ele depende da capacidade de reordenar cálculos. Como as expressões podem chamar funções, e essas funções podem observar o estado externo à expressão emitindo consultas externas, é possível construir um cenário no qual a ordem de cálculo é importante, mas não é capturada na ordem parcial da expressão. Por exemplo, uma função pode ler o conteúdo de um arquivo. Se essa função for chamada repetidamente, as alterações externas nesse arquivo poderão ser observadas e, portanto, a reordenação poderá causar diferenças observáveis no comportamento do programa. Dependendo da ordem de avaliação observada, para que haja exatidão de uma expressão M, há uma dependência de determinadas opções de implementação que podem variar de um avaliador para outro ou pode até mesmo variar no mesmo avaliador em diferentes circunstâncias.

Imutabilidade

Depois que um valor é calculado, ele é *imutável*, o que significa que não pode mais ser alterado. Isso simplifica o modelo para avaliar uma expressão e facilita a interpretação do resultado, uma vez que não é possível alterar um valor após ele ser usado para avaliar uma parte posterior da expressão. Por exemplo, um campo de registro só é computado quando necessário. No entanto, após computado, ele permanece fixo durante o tempo de vida do registro. Mesmo que a tentativa de computar o campo tenha gerado um erro, esse erro será gerado novamente a cada tentativa de acessar esse campo do registro.

Uma exceção importante à regra de imutabilidade após o cálculo se aplica aos valores de lista e tabela. Ambas têm *semântica de streaming*. Ou seja, a enumeração repetida dos itens em uma lista ou das linhas em uma tabela pode produzir resultados variados. A semântica de streaming permite a construção de expressões em M que transformam conjuntos de dados que não caberiam na memória de uma só vez.

Além disso, observe que a aplicação da função *não* é o mesmo que a construção do valor. As funções de biblioteca podem expor o estado externo (como a hora atual ou os resultados de uma consulta em um banco de dados que evolui ao longo do tempo), tornando-as *não determinísticas*. Embora dessa maneira as funções definidas em M não exponham nenhum comportamento não determinístico desse tipo, elas poderiam se fossem definidas para invocar outras funções não determinísticas.

Uma última fonte de não determinismo em M são os *erros*. Os erros param as avaliações quando ocorrem (até o nível em que são tratados por uma expressão try). Normalmente, não é observável se `a + b` causou a avaliação de `a` antes de `b` ou de `b` antes de `a` (ignorando a simultaneidade aqui para simplificar). No entanto, se a subexpressão que foi avaliada primeiro gerar um erro, será possível determinar qual das duas expressões foi avaliada primeiro.

Valores

26/05/2020 • 31 minutes to read

Um valor são dados produzidos pela avaliação de uma expressão. Esta seção descreve os tipos de valores na linguagem M. Cada tipo de valor é associado a uma sintaxe literal, um conjunto de valores que são desse tipo, um conjunto de operadores definidos sobre esse conjunto de valores e um tipo intrínseco atribuído a valores recém-criados.

TIPO	LITERAL
Nulo	`null`
Lógico	`true` `false`
Número	`0` `1` `-1` `1.5` `2.3e-5`
Hora	`#time(09,15,00)`
Data	`#date(2013,02,26)`
DateTime	`#datetime(2013,02,26, 09,15,00)`
DateTimeZone	`#datetimezone(2013,02,26, 09,15,00, 09,00)`
Duration	`#duration(0,1,30,0)`
Texto	`"hello"`
Binary	`#binary("AQID")`
Lista	`{1, 2, 3}`
Registro	`[ A = 1, B = 2 ]`
Tabela	`#table({"X","Y"},{{0,1},{1,0}})`
Função	`(x) => x + 1`
Tipo	`type { number }` `type table [ A = any, B = text ]`

As seções a seguir abordam cada tipo de valor em detalhes. Os tipos e a atribuição de tipo são definidos formalmente em Tipos. Os valores de função são definidos em Funções. As seções a seguir listam os operadores definidos para cada tipo de valor e fornecem exemplos. A definição completa de semântica de operadores é fornecida a seguir, em Operadores.

Nulo

Um *valor nulo* é usado para representar a ausência de um valor ou um valor de um estado indeterminado ou desconhecido. Um valor nulo é gravado usando o literal `null`. Os seguintes operadores são definidos para valores nulos:

OPERADOR	RESULTADO
`x > y`	Maior que
`x >= y`	Maior ou igual
`x < y`	Menor que
`x <= y`	Inferior ou igual
`x = y`	Igual
`x <> y`	Diferente

O tipo nativo do valor `null` é o tipo intrínseco `null`.

Logical

Um *valor lógico* é usado para operações boolianas e tem o valor true ou false. Um valor lógico é escrito usando os literais `true` e `false`. Os seguintes operadores são definidos para valores lógicos:

OPERADOR	RESULTADO
`x > y`	Maior que
`x >= y`	Maior ou igual
`x < y`	Menor que
`x <= y`	Inferior ou igual
`x = y`	Igual
`x <> y`	Diferente
`x or y`	OR lógico condicional
`x and y`	AND lógico condicional
`not x`	NOT lógico

O tipo nativo de ambos os valores lógicos (`true` e `false`) é o tipo intrínseco `logical`.

Número

Um *valor de número* é usado para operações numéricas e aritméticas. Estes são exemplos de literais de número:

```
3.14  // Fractional number
-1.5  // Fractional number
1.0e3 // Fractional number with exponent
123   // Whole number
1e3   // Whole number with exponent
0xff  // Whole number in hex (255)
```

Um número é representado com, no mínimo, a precisão de um *Double* (mas pode reter mais precisão). A representação de *Double* é congruente com o padrão de precisão dupla de 64 bits do IEEE para aritmética de ponto flutuante binário definido em [IEEE 754-2008]. (A representação de *Double* tem um intervalo dinâmico aproximado de 5,0 x 10^{324} para 1,7 x 10^{308} com uma precisão de 15-16 dígitos.)

Os seguintes valores especiais também são considerados valores do tipo *number*:

- Zero positivo e zero negativo. Na maioria das situações, zero positivo e zero negativo se comportam de maneira idêntica ao valor zero simples, mas certas operações distinguem entre os dois.
- Infinito positivo (`#infinity`) e infinito negativo (`-#infinity`). Os infinitos são produzidos por operações como a divisão de um número diferente de zero por zero. Por exemplo, `1.0 / 0.0` produz infinito positivo e `-1.0 / 0.0` produz infinito negativo.
- O valor *não é um número* (`#nan`), geralmente abreviado como NaN. NaNs são produzidos por operações de ponto flutuante inválidas, como a divisão de zero por zero.

As operações matemáticas binárias são executadas usando uma *precisão*. A precisão determina o domínio no qual os operandos são arredondados e o domínio no qual a operação é executada. Na ausência de uma precisão especificada explicitamente, essas operações são executadas usando *precisão dupla*.

- Se o resultado de uma operação matemática for muito pequeno para o formato de destino, o resultado da operação se tornará zero positivo ou zero negativo.
- Se o resultado de uma operação matemática for muito grande para o formato de destino, o resultado da operação se tornará infinito positivo ou infinito negativo.
- Se uma operação matemática for inválida, o resultado da operação se tornará NaN.
- Se um ou ambos os operandos de uma operação de ponto flutuante forem NaN, o resultado da operação se tornará NaN.

Os seguintes operadores são definidos para valores de número:

OPERADOR	RESULTADO
`x > y`	Maior que
`x >= y`	Maior ou igual
`x < y`	Menor que
`x <= y`	Inferior ou igual
`x = y`	Igual
`x <> y`	Diferente
`x + y`	Soma

OPERADOR	RESULTADO
`x - y`	Diferença
`x * y`	Produto
`x / y`	Quociente
`+x`	Adição de unário
`-x`	Negação

O tipo nativo de valores de número é o tipo intrínseco `number`.

Hora

Um *valor temporal* armazena uma representação opaca de hora do dia. Uma hora é codificada como o número de *tiques desde a meia-noite*, que conta o número de tiques de 100 nanossegundos decorridos em um relógio de 24 horas. O número máximo de *tiques desde a meia-noite* corresponde a 23:59:59,9999999 horas.

Os valores temporais podem ser construídos usando o tipo intrínseco #time.

```
#time(hour, minute, second)
```

As seguintes condições precisarão ser compridas ou um erro com o código de motivo `Expression.Error` será gerado:

0 ≤ hora ≤ 24
0 ≤ minuto ≤ 59
0 ≤ segundo ≤ 59

Além disso, se hora = 24, então minuto e segundo precisam ser zero.

Os seguintes operadores são definidos para valores de tempo:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

Os seguintes operadores permitem que um ou ambos os operandos sejam uma data:

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
`x + y`	`time`	`duration`	Deslocamento de Date por duração
`x + y`	`duration`	`time`	Deslocamento de Date por duração
`x - y`	`time`	`duration`	Deslocamento de data por duração negada
`x - y`	`time`	`time`	Duração entre datas
`x & y`	`date`	`time`	Data/hora mescladas

O tipo nativo de valores de tempo é o tipo intrínseco `time`.

Data

Um *valor de data* armazena uma representação opaca de um dia específico. Uma data é codificada como um número de *dias desde a época*, a partir de 1º de janeiro de 0001 da Era Comum no calendário gregoriano. O número máximo de dias desde a época é de 3652058, correspondendo a 31 de dezembro de 9999.

Os valores de data podem ser construídos usando o tipo intrínseco `#date`.

```
#date(year, month, day)
```

As seguintes condições precisarão ser compridas ou um erro com o código de motivo `Expression.Error` será gerado:

1 ≤ ano ≤ 9999
1 ≤ mês ≤ 12
1 ≤ dia ≤ 31

Além disso, o dia deve ser válido para o mês e o ano escolhidos.

Os seguintes operadores são definidos para valores de data:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

Os seguintes operadores permitem que um ou ambos os operandos sejam uma data:

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
`x + y`	`date`	`duration`	Deslocamento de Date por duração
`x + y`	`duration`	`date`	Deslocamento de Date por duração
`x - y`	`date`	`duration`	Deslocamento de data por duração negada
`x - y`	`date`	`date`	Duração entre datas
`x & y`	`date`	`time`	Data/hora mescladas

O tipo nativo de valores de data é o tipo intrínseco `date`.

DateTime

Um *valor datetime* contém uma data e uma hora.

Os valores datetime podem ser construídos usando o tipo intrínseco `#datetime`.

```
#datetime(year, month, day, hour, minute, second)
```

As seguintes condições precisarão ser compridas ou um erro com o código de motivo Expression.Error será gerado: 1 ≤ ano ≤ 9999
1 ≤ mês ≤ 12
1 ≤ dia ≤ 31
0 ≤ hora ≤ 23
0 ≤ minuto ≤ 59
0 ≤ segundo ≤ 59

Além disso, o dia deve ser válido para o mês e o ano escolhidos.

Os seguintes operadores são definidos para valores datetime:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

OPERADOR	RESULTADO

Os seguintes operadores permitem que um ou ambos os operandos sejam um datetime:

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
`x + y`	`datetime`	`duration`	Deslocamento de data/hora por duração
`x + y`	`duration`	`datetime`	Deslocamento de data/hora por duração
`x - y`	`datetime`	`duration`	Deslocamento de Datetime por duração negada
`x - y`	`datetime`	`datetime`	Duração entre datetimes

O tipo nativo de valores datetime é o tipo intrínseco `datetime`.

DateTimeZone

Um valor de *datetimezone* contém um datetime e um timezone. Um *timezone* é codificado como um número de *minutos de diferença UTC*, que conta o número de minutos que *datetime* deverá ter de diferença em relação ao UTC (horário coordenado universal). O número mínimo de *minutos de diferença UTC* é -840, representando uma diferença UTC de -14:00 ou quatorze horas antes do UTC. O número máximo de *minutos de diferença UTC* é 840, correspondendo a uma diferença UTC de 14:00.

Os valores datetimezone podem ser construídos usando o tipo intrínseco `#datetimezone`.

```
#datetimezone(
       year, month, day,
       hour, minute, second,
       offset-hours, offset-minutes)
```

As seguintes condições precisarão ser compridas ou um erro com o código de motivo `Expression.Error` será gerado:

1 ≤ ano ≤ 9999
1 ≤ mês ≤ 12
1 ≤ dia ≤ 31
0 ≤ hora ≤ 23
0 ≤ minuto ≤ 59
0 ≤ segundo ≤ 59
-14 ≤ horas de diferença ≤ 14
-59 ≤ minutos de diferença ≤ 59

Além disso, o dia precisa ser válido para o mês e o ano escolhidos e, se horas-de-diferença = 14, então minutos-de-diferença < = 0 e, se horas-de-diferença =-14, então minutos-de-diferença > = 0.

Os seguintes operadores são definidos para valores datetimezone:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

Os seguintes operadores permitem que um ou ambos os operandos sejam um datetimezone:

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
`x + y`	`datetimezone`	`duration`	Deslocamento DateTimeZone por duração
`x + y`	`duration`	`datetimezone`	Deslocamento DateTimeZone por duração
`x - y`	`datetimezone`	`duration`	Deslocamento de datetimezone por duração negada
`x - y`	`datetimezone`	`datetimezone`	Duração entre datetimezones

O tipo nativo de valores datetimezone é o tipo intrínseco `datetimezone`.

Duração

Um *valor de duração* armazena uma representação opaca da distância entre dois pontos em uma linha do tempo medida em tiques de 100 nanossegundos. A magnitude de uma *duração* pode ser positiva ou negativa, em que valores positivos indicam avanço no tempo e valores negativos indicam retrocesso no tempo. O valor mínimo que pode ser armazenado em uma *duração* é de -9.223.372.036.854.775.808 tiques ou 10.675.199 dias, 2 horas, 48 minutos e 5,4775808 segundos recuando no tempo. O valor máximo que pode ser armazenado em uma *duração* é de -9.223.372.036.854.775.807 tiques ou 10.675.199 dias, 2 horas, 48 minutos e 5,4775807 segundos avançando no tempo.

Os valores de duração podem ser construídos usando a função intrínseca `#duration`:

```
#duration(0, 0, 0, 5.5)          // 5.5 seconds
#duration(0, 0, 0, -5.5)         // -5.5 seconds
#duration(0, 0, 5, 30)           // 5.5 minutes
#duration(0, 0, 5, -30)          // 4.5 minutes
#duration(0, 24, 0, 0)           // 1 day
#duration(1, 0, 0, 0)            // 1 day
```

Os seguintes operadores são definidos em valores de duração:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

Além disso, os seguintes operadores permitem que um ou ambos os operandos sejam um valor de duração:

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
`x + y`	`datetime`	`duration`	Deslocamento de data/hora por duração
`x + y`	`duration`	`datetime`	Deslocamento de data/hora por duração
`x + y`	`duration`	`duration`	Soma das durações
`x - y`	`datetime`	`duration`	Deslocamento de Datetime por duração negada
`x - y`	`datetime`	`datetime`	Duração entre datetimes
`x - y`	`duration`	`duration`	Diferença de durações
`x * y`	`duration`	`number`	N vezes uma duração
`x * y`	`number`	`duration`	N vezes uma duração
`x / y`	`duration`	`number`	Fração de uma duração

O tipo nativo de valores de duração é o tipo intrínseco `duration`.

Texto

Um valor de *texto* representa uma sequência de caracteres Unicode. Os valores de texto têm um formato literal em conformidade com a seguinte gramática:

_literal-de-texto:
    `"` caracteres-de-literal-de-texto$_{opt}$ `"`
caracteres-de-literal-de-texto:

caractere-de-literal-de-texto caracteres-de-literal-de-texto$_{opt}$

caractere-de-literal-de-texto:

caractere-de-texto-único

sequência-de-escape-de-caracteres

sequência-de-escape-com-aspas-duplas

caractere-de-texto-único:

Qualquer caractere, exceto `"` (`U+0022`) ou `#` (`U+0023`) seguido por `(` (`U+0028`)

sequência-de-escape-com-aspas-duplas:

`""` (`U+0022`, `U+0022`)

Segue um exemplo de um valor de *texto*:

```
"ABC" // the text value ABC
```

Os seguintes operadores são definidos em valores de *texto*:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual
`x & y`	Concatenação

O tipo nativo de valores de texto é o tipo intrínseco `text`.

Binário

Um *valor binário* representa uma sequência de bytes. Não há nenhum formato literal. Várias funções de biblioteca padrão são fornecidas para construir valores binários. Por exemplo, `#binary` pode ser usado para construir um valor binário com base em uma lista de bytes:

```
#binary( {0x00, 0x01, 0x02, 0x03} )
```

Os seguintes operadores são definidos em valores *binários*:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente

OPERADOR	RESULTADO
`x >= y`	Maior ou igual
`x > y`	Maior que
`x < y`	Menor que
`x <= y`	Inferior ou igual

O tipo nativo de valores binários é o tipo intrínseco *binary*.

Lista

Um *valor de lista* é um valor que produz uma sequência de valores quando enumerado. Um valor produzido por uma lista pode conter qualquer tipo de valor, incluindo uma lista. As listas podem ser construídas usando a sintaxe de inicialização, da seguinte maneira:

expressão-de-lista:
    { *lista-de-itens*$_{opt}$ }
lista-de-itens:
    item
    item `,` *lista-de-itens*
item:
    expressão
    expressão `..` *expressão*

Veja o seguinte exemplo, que mostra uma *expressão-de-lista* que define uma lista com três valores de texto: `"A"`, `"B"` e `"C"`.

```
{"A", "B", "C"}
```

O valor `"A"` é o primeiro item na lista e o valor `"C"` é o último item da lista.

- Os itens de uma lista não são avaliados até que sejam acessados.
- Embora os valores de lista construídos usando a sintaxe de lista produzam itens na ordem em que eles aparecem na *lista-de-itens*, em geral, as listas retornadas de funções de biblioteca podem produzir um conjunto diferente ou um número diferente de valores sempre que são enumeradas.

Para incluir uma sequência de números inteiros em uma lista, o formato `a..b` pode ser usado:

```
{ 1, 5..9, 11 }     // { 1, 5, 6, 7, 8, 9, 11 }
```

O número de itens em uma lista, conhecido como a *contagem da lista*, pode ser determinado usando a função `List.Count`.

```
List.Count({true, false})  // 2
List.Count({})             // 0
```

Uma lista pode ter efetivamente um número infinito de itens; a `List.Count` para essas listas é indefinida e pode gerar um erro ou não ter fim.

Se uma lista não contiver nenhum item, ela será chamada de *lista vazia*. Uma lista vazia é escrita como:

```
{}  // empty list
```

Os seguintes operadores são definidos para listas:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x & y`	Concatenate

Por exemplo:

```
{1, 2} & {3, 4, 5}   // {1, 2, 3, 4, 5}
{1, 2} = {1, 2}      // true
{2, 1} <> {1, 2}     // true
```

O tipo nativo de valores de lista é o tipo intrínseco `list`, que especifica um tipo de item de `any`.

Record

Um *valor de registro* é uma sequência ordenada de campos. Um *campo* consiste em um *nome do campo* (um valor de texto que identifica o campo de maneira exclusiva dentro do registro) e em um *valor do campo*. O valor do campo pode ser qualquer tipo de valor, incluindo um registro. Registros podem ser construídos usando sintaxe de inicialização, da seguinte maneira:

expressão-de-registro:
    `[` *lista-de-campos*$_{opt}$ `]`
lista-de-campos:
    campo
    campo `,` *lista-de-campos*
campo:
    nome-do-campo `=` *expressão*
nome-do-campo:
    identificador-generalizado
    identificador-entre-aspas

O exemplo a seguir constrói um registro com um campo chamado `x` com o valor `1` e um campo chamado `y` com o valor `2`.

```
[ x = 1, y = 2 ]
```

O exemplo a seguir constrói um registro com um campo `a` chamado a com um valor de registro aninhado. O registro aninhado tem um campo chamado `b` com o valor `2`.

```
[ a = [ b = 2 ] ]
```

Os seguintes preceitos são válidos ao avaliar uma expressão de registro:

- A expressão atribuída a cada nome de campo é usada para determinar o valor do campo associado.
- Se a expressão atribuída a um nome de campo produzir um valor quando avaliada, esse se tornará o valor do campo do registro resultante.
- Se a expressão atribuída a um nome de campo gerar um erro quando for avaliada, o fato de um erro ter sido gerado será registrado com o campo junto com o valor de erro que tiver sido gerado. O acesso subsequente a esse campo fará com que um erro seja recriado com o valor de erro registrado.
- A expressão é avaliada em um ambiente como o ambiente pai somente com variáveis mescladas, no sentido de que correspondem ao valor de todos os campos do registro, exceto por aquele que está sendo inicializado.
- Um valor em um registro não é avaliado até que o campo correspondente seja acessado.
- Um valor em um registro é avaliado no máximo uma vez.
- O resultado da expressão é um valor de registro com um registro de metadados vazio.
- A ordem dos campos dentro do registro é definida pela ordem em que eles aparecem na *expressão-inicializadora-do-registro*.
- Cada nome de campo especificado precisa ser exclusivo dentro do registro, caso contrário resulta em um erro. Os nomes são comparados pelo uso de uma comparação ordinal.

```
    [ x = 1, x = 2 ] // error: field names must be unique
    [ X = 1, x = 2 ] // OK
```

Um registro sem campos é chamado de *registro vazio* e é escrito da seguinte maneira:

```
[] // empty record
```

Embora a ordem dos campos de um registro não seja significativa ao acessar um campo ou comparar dois registros, ela é significativa em outros contextos, como quando os campos de um registro são enumerados.

Os mesmos dois registros produzem resultados diferentes quando os campos são obtidos:

```
Record.FieldNames([ x = 1, y = 2 ]) // [ "x", "y" ]
Record.FieldNames([ y = 1, x = 2 ]) // [ "y", "x" ]
```

O número de campos em um registro pode ser determinado usando a função `Record.FieldCount`. Por exemplo:

```
Record.FieldCount([ x = 1, y = 2 })  // 2
Record.FieldCount([])                // 0
```

Além de usar a sintaxe de inicialização de registro `[ ]`, os registros podem ser construídos com base em uma lista de valores e uma lista de nomes de campo ou um tipo de registro. Por exemplo:

```
Record.FromList({1, 2}, {"a", "b"})
```

O exemplo acima é equivalente a:

```
[ a = 1, b = 2 ]
```

Os seguintes operadores são definidos para valores de registro:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente
`x & y`	Mesclar

Os exemplos a seguir ilustram o uso dos operadores acima. Observe que a mesclagem de registros usa os campos do operando direito para substituir campos do operando esquerdo, caso haja uma sobreposição em nomes de campo.

```
[ a = 1, b = 2 ] & [ c = 3 ]    // [ a = 1, b = 2, c = 3 ]
[ a = 1, b = 2 ] & [ a = 3 ]    // [ a = 3, b = 2 ]
[ a = 1, b = 2 ] = [ b = 2, a = 1 ]         // true
[ a = 1, b = 2, c = 3 ] <> [ a = 1, b = 2 ] // true
```

O tipo nativo de valores de registro é o tipo intrínseco `record`, que especifica uma lista de campos vazia aberta.

Tabela

Um *valor de tabela* é uma sequência ordenada de linhas. Uma *linha* é uma sequência ordenada de valores. O tipo da tabela determina o comprimento de todas as linhas na tabela, os nomes das colunas da tabela, os tipos das colunas da tabela e a estrutura das chaves da tabela (se houver).

Não há sintaxe literal para tabelas. Várias funções de biblioteca padrão são fornecidas para construir valores binários. Por exemplo, `#table` pode ser usado para construir uma tabela com base em uma lista de listas de linhas e uma lista de nomes de cabeçalho:

```
#table({"x", "x^2"}, {{1,1}, {2,4}, {3,9}})
```

O exemplo acima constrói uma tabela com duas colunas, sendo que as duas são do `type any`.

`#table` também pode ser usado para especificar um tipo de tabela completa:

```
#table(
    type table [Digit = number, Name = text],
    {{1,"one"}, {2,"two"}, {3,"three"}}
    )
```

Aqui, o novo valor de tabela tem um tipo de tabela que especifica os nomes de coluna e os tipos de coluna.

Os seguintes operadores são definidos para valores de tabela:

OPERADOR	RESULTADO
`x = y`	Igual

OPERADOR	RESULTADO
`x <> y`	Diferente
`x & y`	Concatenação

A concatenação de tabelas alinha colunas com nomes semelhantes e preenche `null` para colunas que aparecem em apenas uma das tabelas de operandos. Este exemplo ilustra a concatenação de tabelas:

```
  #table({"A","B"}, {{1,2}})
& #table({"B","C"}, {{3,4}})
```

A	B	C
`1`	`2`	`null`
`null`	`3`	`4`

O tipo nativo de valores de tabela é um tipo de tabela personalizado (derivado do tipo intrínseco `table`) que lista os nomes de coluna, especifica todos os tipos de coluna como any e não tem nenhuma chave. (Confira Tipos de tabela para obter detalhes sobre os tipos de tabela.)

Função

Um *valor de função* é um valor que mapeia um conjunto de argumentos para apenas um valor. Os detalhes dos valores de *função* são descritos em Funções.

Tipo

Um *valor de tipo* é um valor que classifica outros valores. Os detalhes dos valores de *tipo* são descritos em Tipos.

Tipos

26/05/2020 • 23 minutes to read

Um *valor de tipo* é um valor que *classifica* outros valores. Um valor classificado por um tipo *obedece* a esse tipo. O sistema de tipos de M é composto pelas seguintes categorias de tipos:

- Tipos primitivos, que classificam valores primitivos (`binary` , `date` , `datetime` , `datetimezone` , `duration` , `list` , `logical` , `null` , `number` , `record` , `text` , `time` , `type`) e incluem alguns tipos abstratos (`function` , `table` , `any` e `none`)
- Tipos de registro, que classificam valores de registro com base em nomes de campo e em tipos de valor
- Tipos de lista, que classificam listas usando apenas um tipo de base de item
- Tipos de função, que classificam valores de função com base nos tipos de seus parâmetros e valores de retorno
- Tipos de tabela, que classificam valores de tabela com base em nomes de coluna, tipos de coluna e chaves
- Tipos anuláveis, que classificam o valor nulo além de todos os valores classificados por um tipo base
- Tipos de tipo, que classificam valores que são tipos

O conjunto de *tipos primitivos* inclui os tipos de valores primitivos de alguns *tipos abstratos*, que são tipos que não classificam os valores de maneira exclusiva: `function` , `table` , `any` e `none` . Todos os valores de função obedecem o tipo abstrato `function` ; todos os valores de tabela, o tipo abstrato `table` ; todos os valores, o tipo abstrato `any` e nenhum valor, o tipo abstrato `none` . Uma expressão do tipo `none` deve gerar um erro ou falhar ao ser encerrada, pois não é possível produzir nenhum valor que obedeça o tipo `none` . Observe que os tipos primitivos `function` e `table` são abstratos, porque nenhuma função ou tabela é diretamente desses tipos, respectivamente. Os tipos primitivos `record` e `list` não são abstratos porque representam um registro aberto sem campos definidos e uma lista do tipo any, respectivamente.

Todos os tipos que não são membros do conjunto fechado de tipos primitivos são chamados coletivamente de *tipos personalizados*. Tipos personalizados podem ser escritos usando um `type-expression` :

expressão-de-tipo:
    expressão-primária
    `type` *tipo-primário*
tipo:
    expressão-entre-parênteses
    tipo-primário
tipo-primário:
    tipo-primitivo
    tipo-de-registro
    tipo-de-lista
    tipo-de-função
    tipo-de-tabela
    tipo-anulável
tipo-primitivo: um entre
    `any binary date datetime datetimezone duration function list logical`
    `none null number record table text time type`

Os nomes de *tipo-primitivo* são *palavras-chave contextuais* reconhecidas somente em um contexto de *tipo*. O uso

de parênteses em um contexto de *tipo* muda a gramática de volta para o contexto de expressão regular, exigindo o uso da palavra-chave type para voltar para o contexto de tipo. Por exemplo, para invocar uma função em um contexto de *tipo*, os parênteses podem ser usados:

```
type nullable ( Type.ForList({type number}) )
// type nullable {number}
```

Os parênteses também podem ser usados para acessar uma variável cujo nome colide com um nome de *tipo-primitivo*:

```
let  record = type [ A = any ]  in  type {(record)}
// type {[ A = any ]}
```

O exemplo a seguir define um tipo que classifica uma lista de números:

```
type { number }
```

De maneira semelhante, o exemplo a seguir define um tipo personalizado que classifica os registros com campos obrigatórios chamados `X` e `Y`, cujos valores são números:

```
type [ X = number, Y = number ]
```

O tipo atribuído de um valor é obtido usando a função de biblioteca padrão `Value.Type`, conforme mostrado nos exemplos a seguir:

```
Value.Type( 2 )                // type number
Value.Type( {2} )              // type list
Value.Type( [ X = 1, Y = 2 ] )  // type record
```

O operador `is` é usado para determinar se o tipo de um valor é compatível com um determinado tipo, conforme mostrado nos exemplos a seguir:

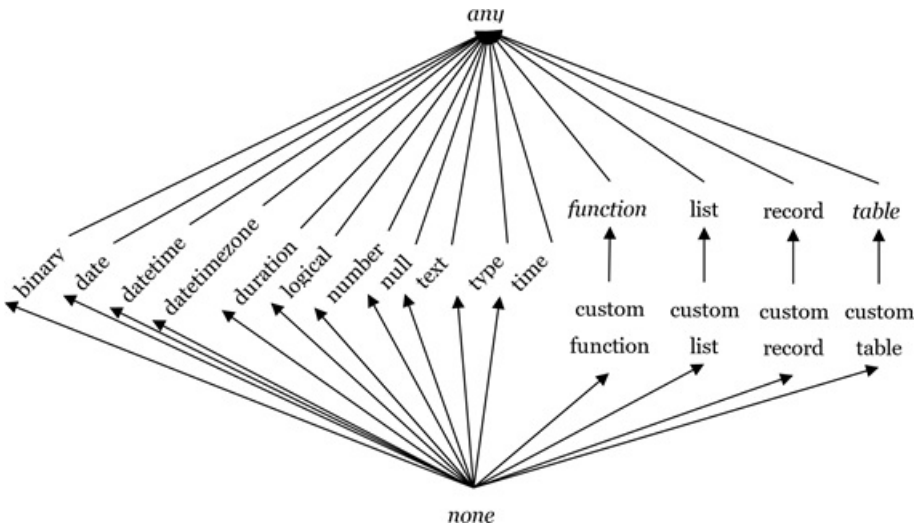
```
1 is number          // true
1 is text            // false
{2} is list          // true
```

O operador `as` verifica se o valor é compatível com o tipo determinado e gera um erro se não for. Caso contrário, ele retorna o valor original.

```
Value.Type( 1 as number )   // type number
{2} as text                 // error, type mismatch
```

Observe que os operadores `is` e `as` só aceitam tipos primitivos como o operando à direita. A linguagem M não fornece um meio de verificar os valores quanto à conformidade com tipos personalizados.

Um tipo `X` é *compatível* com um tipo `Y` se, e somente se, todos os valores que obedecem `X` também obedecem `Y`. Todos os tipos são compatíveis com o tipo `any` e nenhum tipo (exceto pelo próprio `none`) é compatível com o tipo `none`. O gráfico a seguir mostra a relação de compatibilidade. (A compatibilidade do tipo é reflexiva e transitiva. Ele forma um malha com o tipo `any` como o valor superior e o tipo `none` como o inferior.) Os nomes dos tipos abstratos estão em *itálico*.



Os seguintes operadores são definidos para os valores de tipo:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente

O tipo nativo de valores de tipo é o tipo intrínseco `type`.

Tipos primitivos

Os tipos na linguagem M formam uma hierarquia não contígua enraizada no tipo `any`, que é o tipo que classifica todos os valores. Qualquer valor de M obedece exatamente um subtipo primitivo de `any`. O conjunto fechado de tipos primitivos derivados do tipo `any` é o seguinte:

- `type null`, que classifica o valor nulo.
- `type logical`, que classifica os valores true e false.
- `type number`, que classifica valores numéricos.
- `type time`, que classifica valores de tempo.
- `type date`, que classifica valores de data.
- `type datetime`, que classifica valores de datetime.
- `type datetimezone`, que classifica valores de datetimezone.
- `type duration`, que classifica valores de duração.
- `type text`, que classifica valores de texto.
- `type binary`, que classifica valores binários.
- `type type`, que classifica valores de tipo.
- `type list`, que classifica valores de lista.
- `type record`, que classifica valores de registro.
- `type table`, que classifica valores de tabela.
- `type function`, que classifica valores de função.
- `type anynonnull`, que classifica todos os valores, exceto por nulo. O tipo intrínseco `none` não classifica nenhum valor.

Tipo any

O tipo `any` é abstrato, ele classifica todos os valores em M e todos os tipos em M são compatíveis com `any`. Variáveis do tipo `any` podem ser associadas a todos os valores possíveis. Como `any` é abstrato, ele não pode ser atribuído a valores—ou seja, nenhum valor tem diretamente o tipo `any`.

Tipos de lista

Qualquer valor que seja uma lista obedece o tipo intrínseco `list`, que não estabelece nenhuma restrição sobre os itens dentro de um valor de lista.

tipo-de-lista:
    `{` *tipo-de-item* `}`
tipo-de-item:
    tipo

O resultado da avaliação de um *tipo-de-lista* é um *valor de tipo de lista* cujo tipo base é `list`.

Os exemplos a seguir ilustram a sintaxe para declarar tipos de lista homogêneos:

```
type { number }        // list of numbers type
     { record }        // list of records type
     {{ text }}        // list of lists of text values
```

Um valor obedece um tipo de lista se o valor é uma lista e cada item nesse valor de lista obedece o tipo de item do tipo de lista.

O tipo de item de um tipo de lista indica uma associação: todos os itens de uma lista compatível obedecem o tipo de item.

Tipos de registro

Qualquer valor que seja um registro obedece o tipo intrínseco record, que não estabelece nenhuma restrição sobre os itens dentro de um valor de registro. Um *valor-de-tipo-de-registro* é usado para restringir o conjunto de nomes válidos, bem como os tipos de valores que têm permissão para ser associados a esses nomes.

tipo-de-registro:
    `[` *marcador-de-registro-aberto* `]`
    `[` *lista-de-especificação-de-campos*$_{opt}$ `]`
    `[` *lista-de-especificações-de-campo, marcador-de-registro-aberto* `]`
lista-de-especificações-de-campo:
    especificação-de-campo
    especificação-de-campo `,` *lista-de-especificações-de-campo*
especificação-de-campo:
    `optional`$_{opt}$ *nome-do-campo especificação-de-tipo-de-campo*$_{opt}$
especificação-de-tipo-de-campo:
    `=` *tipo-de-campo*
tipo-de-campo:
    tipo
marcador-de-registro-aberto:
    `...`

O resultado da avaliação de um *tipo-de-registro* é um valor de tipo cujo tipo base é `record`.

Os exemplos a seguir ilustram a sintaxe para declarar tipos de registro:

```
type [ X = number, Y = number]
type [ Name = text, Age = number ]
type [ Title = text, optional Description = text ]
type [ Name = text, ... ]
```

Os tipos de registro são *fechados* por padrão, o que significa que campos adicionais que não estão presentes na *lista-de-especificações-de-campo* não podem estar presentes em valores compatíveis. Incluir o *marcador-de-registro-aberto* no tipo de registro declara que o tipo é *aberto*, o que permite campos não presentes na lista de especificações de campo. As duas expressões a seguir são equivalentes:

```
type record   // primitive type classifying all records
type [ ... ]  // custom type classifying all records
```

Um valor obedecerá um tipo de registro se o valor for um registro e cada especificação de campo no tipo de registro for satisfeita. Uma especificação de campo será atendida se qualquer uma das seguintes opções for verdadeira:

- Um nome de campo correspondente ao identificador da especificação existe no registro e o valor associado obedece o tipo da especificação
- A especificação é marcada como opcional e nenhum nome de campo correspondente é encontrado no registro

Um valor compatível pode conter nomes de campo não listados na lista especificações de campo se, e somente se, o tipo de registro é aberto.

Tipos de função

Qualquer valor de função obedece o tipo primitivo `function`, que não estabelece nenhuma restrição sobre os tipos dos parâmetros formais da função nem sobre o valor retornado da função. Um *valor de tipo-de-função* personalizado é usado para estabelecer restrições de tipo sobre assinaturas dos valores de função compatíveis.

tipo-de-função:
    `function (` *lista-de-especificações-de-parâmetro*$_{opt}$ `)` *tipo-retornado-da-função*
lista-de-especificações-de-parâmetro:
    lista-de-especificações-de-parâmetro-obrigatórias
    lista-de-especificações-de-parâmetro-obrigatórias `,` *lista-de-especificações-de-parâmetro-opcionais*
    lista-de-especificações-de-parâmetro-opcionais
lista-de-especificações-de-parâmetro-obrigatórias:
    especificação-de-parâmetro-obrigatória
    especificação-de-parâmetro-obrigatória `,` *lista-de-especificações-de-parâmetro-obrigatória*
especificação-de-parâmetro-obrigatória:
    especificação-de-parâmetro
lista-de-especificações-de-parâmetro-obrigatórias:
    especificação-de-parâmetro-opcional
    especificação-de-parâmetro-opcional `,` *lista-de-especificações-de-parâmetro-opcionais*
especificação-de-parâmetro-opcional:
    `optional` *especificação-de-parâmetro*
especificação-de-parâmetro:
    nome-do-parâmetro tipo-de-parâmetro
tipo-retornado-da-função:
    asserção
asserção:

`as` *tipo-primitivo-que-permite-valor-nulo*

O resultado da avaliação de um *tipo-de-função* é um valor de tipo cujo tipo base é `function`.

Os exemplos a seguir ilustram a sintaxe para declarar tipos de função:

```
type function (x as text) as number
type function (y as number, optional z as text) as any
```

Um valor de função obedece um tipo de função se o tipo retornado do valor da função é compatível com o tipo retornado do tipo de função e cada especificação de parâmetro do tipo de função é compatível com o parâmetro formal correspondente em termos de posição da função. Uma especificação de parâmetro é compatível com um parâmetro formal se o *tipo-de-parâmetro* especificado é compatível com o tipo do parâmetro formal e a especificação de parâmetro é opcional se o parâmetro formal é opcional.

Os nomes de parâmetro formais são ignorados para fins de determinação da conformidade do tipo de função.

Tipos de tabela

Um *valor de tipo-de-tabela* é usado para definir a estrutura de um valor de tabela.

tipo-de-tabela:
`table` *tipo-de-linha*
tipo-de-linha:
`[` *lista-de-especificações-de-campo* `]`

O resultado da avaliação de um *tipo-de-tabela* é um valor de tipo cujo tipo base é `table`.

O *tipo de linha* de uma tabela especifica os nomes de coluna e os tipos de coluna da tabela como um tipo de registro fechado. Para que todos os valores de tabela obedeçam o tipo `table`, seu tipo de linha é tipo `record` (o tipo de registro aberto vazio). Portanto, o tipo de tabela é abstrato, uma vez que nenhum valor de tabela pode ter o tipo de linha do tipo `table` (mas todos os valores de tabela têm um tipo de linha compatível com o tipo de linha do tipo `table`). O exemplo a seguir mostra a construção de um tipo de tabela:

```
type table [A = text, B = number, C = binary]
// a table type with three columns named A, B, and C
// of column types text, number, and binary, respectively
```

Um valor de tipo de tabela também contém a definição das *chaves* de um valor de tabela. Uma chave é um conjunto de nomes de coluna. No máximo uma chave pode ser designada como a *chave primária* da tabela. (Na linguagem M, as chaves de tabela não têm significado semântico. No entanto, é comum que fontes de dados externas, como bancos de dados ou feeds OData, definam chaves em tabelas. O Power Query usa informações de chave para aprimorar o desempenho das funcionalidades avançadas, como operações de junção entre fontes.)

As funções de biblioteca padrão `Type.TableKeys`, `Type.AddTableKey` e `Type.ReplaceTableKeys` podem ser usadas para obter as chaves de um tipo de tabela, adicionar uma chave a um tipo de tabela e substituir todas as chaves de um tipo de tabela, respectivamente.

```
Type.AddTableKey(tableType, {"A", "B"}, false)
// add a non-primary key that combines values from columns A and B
Type.ReplaceTableKeys(tableType, {})
// returns type value with all keys removed
```

Tipos anuláveis

Para qualquer `type T`, uma variante anulável pode ser derivada usando o *tipo-anulável*:

tipo-anulável:
`nullable` *tipo*

O resultado é um tipo abstrato que permite valores do tipo *T* ou o valor `null`.

```
42 is nullable number            // true null is
nullable number                  // true
```

A atribuição de `type nullable` *T* reduz a atribuição `type null` ou `type` *T*. (Lembre-se de que os tipos anuláveis são abstratos e nenhum valor pode ter diretamente um tipo abstrato.)

```
Value.Type(42 as nullable number)       // type number
Value.Type(null as nullable number)     // type null
```

As funções de biblioteca padrão `Type.IsNullable` e `Type.NonNullable` podem ser usadas para testar um tipo quanto à nulidade e remover a nulidade de um tipo.

O seguinte é válido (para qualquer `type T`):

- `type T` é compatível com `type nullable T`
- `Type.NonNullable(type T)` é compatível com `type T`

Os seguintes são equivalentes emparelhados (para qualquer `type T`):

`type nullable any`
`any`

`Type.NonNullable(type any)`
`type anynonnull`

`type nullable none`
`type null`

`Type.NonNullable(type null)`
`type none`

`type nullable nullable T`
`type nullable T`

`Type.NonNullable(Type.NonNullable(type T))`
`Type.NonNullable(type T)`

`Type.NonNullable(type nullable T)`
`Type.NonNullable(type T)`

`type nullable (Type.NonNullable(type T))`
`type nullable T`

Tipo atribuído de um valor

O *tipo atribuído* de um valor é o tipo a que é *declarado* que um valor obedece. Quando um tipo é atribuído a um valor, apenas uma verificação de conformidade limitada ocorre. *A linguagem M não executa verificações de conformidade além do tipo primitivo anulável. Os autores de programas em M que optam por atribuir aos valores definições de tipo mais complexas do que um tipo primitivo anulável devem garantir que esses valores*

obedeçam esses tipos.

É possível atribuir um tipo a um valor usando a função de biblioteca `Value.ReplaceType`. A função retorna um novo valor com o tipo atribuído ou gera um erro quando o novo tipo é incompatível com o tipo primitivo nativo do valor. Em particular, a função gera um erro quando é feita uma tentativa de atribuir um tipo abstrato, como `any`.

As funções de biblioteca podem optar por computar e atribuir tipos complexos para a resultados com base nos tipos atribuídos dos valores de entrada.

O tipo atribuído de um valor pode ser obtido usando a função de biblioteca `Value.Type`. Por exemplo:

```
Value.Type( Value.ReplaceType( {1}, type {number} )
// type {number}
```

Equivalência e compatibilidade de tipos

A equivalência de tipos não é definida em M. Quaisquer dois valores de tipo comparados quanto à igualdade podem ou não retornar `true`. No entanto, a relação entre esses dois tipos (se `true` ou `false`) será sempre a mesma.

A compatibilidade entre um determinado tipo e um tipo primitivo anulável pode ser determinada usando a função de biblioteca `Type.Is`, que aceita um valor de tipo arbitrário como o primeiro e um valor de tipo primitivo anulável como o segundo argumento:

```
Type.Is(type text, type nullable text)  // true
Type.Is(type nullable text, type text)  // false
Type.Is(type number, type text)         // false
Type.Is(type [a=any], type record)      // true
Type.Is(type [a=any], type list)        // false
```

Não há suporte em M para determinar a compatibilidade de um determinado tipo com um tipo personalizado.

A biblioteca padrão inclui uma coleção de funções para extrair as características de definição de um tipo personalizado, de modo que testes de compatibilidade específicos podem ser implementados como expressões em M. Veja abaixo alguns exemplos; consulte a especificação da biblioteca de M para ver os detalhes completos.

```
Type.ListItem( type {number} )
  // type number
Type.NonNullable( type nullable text )
  // type text
Type.RecordFields( type [A=text, B=time] )
  // [ A = [Type = type text, Optional = false],
  //   B = [Type = type time, Optional = false] ]
Type.TableRow( type table [X=number, Y=date] )
  // type [X = number, Y = date]
Type.FunctionParameters(
        type function (x as number, optional y as text) as number)
  // [ x = type number, y = type nullable text ]
Type.FunctionRequiredParameters(
        type function (x as number, optional y as text) as number)
  // 1
Type.FunctionReturn(
        type function (x as number, optional y as text) as number)
  // type number
```

Operadores

26/05/2020 • 70 minutes to read

Esta seção define o comportamento dos vários operadores do M.

Precedência do operador

Quando uma expressão contém vários operadores, a *precedência* dos operadores controla a ordem na qual os operadores individuais são avaliados. Por exemplo, a expressão `x + y * z` é avaliada como `x + (y * z)` porque o operador `*` tem precedência maior do que o operador binário `+`. A precedência de um operador é estabelecida pela definição da sua produção de gramática associada. Por exemplo, uma *expressão-aditiva* consiste em uma sequência de produções de *expressão-multiplicativa* separadas por operadores `+` ou `-`, permitindo assim que os operadores `+` e `-` tenham mais precedências do que os operadores `*` e `/`.

A produção *expressão-entre-parênteses* pode ser usada para alterar a ordenação de precedência padrão.

expressão-entre-parênteses:
`(` *expressão* `)`

Por exemplo:

```
1 + 2 * 3       // 7
(1 + 2) * 3     // 9
```

A tabela a seguir resume os operadores M, listando as categorias de operador em ordem de precedência, da mais alta para a mais baixa. Os operadores em uma mesma categoria têm precedência igual.

CATEGORIA	EXPRESSÃO	DESCRIÇÃO
Primária	*i* @*i*	Expressão de identificador
	(*x*)	Expressão entre parênteses
	x[*i*]	Lookup
	x{*y*}	Acesso ao item
	x(...)	Invocação de função
	{*x*, *y*, ...}	Inicialização de lista
	[*i* = *x*, ...]	Inicialização de registro
	...	Não implementado
Unário	+*x*	Identidade
	-*x*	Negação

	`not` *x*	Negação lógica
Metadados	*x* `meta` *y*	Associar metadados
Multiplicativo	*x* * *y*	Multiplicação
	x / *y*	Divisão
Aditiva	*x* + *y*	Adição
	x - *y*	Subtração
Relacional	*x* < *y*	Menor que
	x > *y*	Maior que
	x <= *y*	Inferior ou igual
	x >= *y*	Maior ou igual
Igualitário	*x* = *y*	Igual
	x <> *y*	Diferente
Asserção de tipo	*x* `as` *y*	É um tipo primitivo que permite valor nulo ou então resulta em erro
Conformidade de tipo	*x* `is` *y*	Testar se o tipo primitivo que permite valor nulo é compatível
AND lógico	*x* `and` *y*	Conjunção de curto-circuito
OR lógico	*x* `or` *y*	Disjunção de curto-circuito

Operadores e metadados

Cada valor tem um valor de registro associado que pode conter informações adicionais sobre o valor. Esse registro é conhecido como o *registro de metadados* para um valor. Um registro de metadados pode ser associado a qualquer tipo de valor, mesmo `null`. O resultado dessa associação é um novo valor com os metadados fornecidos.

Um registro de metadados é apenas um registro comum e pode conter os mesmos tipos de campos e valores que um registro regular, além de conter um registro de metadados. Associar um registro de metadados com um valor é uma ação "não intrusiva". Essa associação não altera o comportamento do valor em avaliações, exceto aqueles que inspecionam explicitamente os registros de metadados.

Cada valor tem um registro de metadados padrão, mesmo que um não tenha sido especificado. O registro de metadados padrão é vazio. Os exemplos a seguir mostram como acessar o registro de metadados de um valor de texto usando a função de biblioteca padrão `Value.Metadata`:

```
Value.Metadata( "Mozart" )   // []
```

Os registros de metadados geralmente *não são preservados* quando um valor é usado com um operador ou função que cria um novo valor. Por exemplo, se dois valores de texto forem concatenados usando o operador `&`, os metadados do valor de texto resultante serão o registro vazio `[]`. As seguintes expressões são equivalentes:

```
"Amadeus " & ("Mozart" meta [ Rating = 5 ])
"Amadeus " & "Mozart"
```

As funções de biblioteca padrão `Value.RemoveMetadata` e `Value.ReplaceMetadata` podem ser usadas para remover todos os metadados de um valor e para substituir os metadados de um valor (em vez de mesclar os metadados em metadados possivelmente existentes).

O único operador que retorna os resultados que contêm metadados é o operador meta.

Operadores recursivos estruturalmente

Os valores podem ser *cíclicos*. Por exemplo:

```
let l = {0, @l} in l
// {0, {0, {0, ... }}}
[A={B}, B={A}]
// [A = {{ ... }}, B = {{ ... }}]
```

O M manipula valores cíclicos mantendo o processo de construção de registros, listas e tabelas lento. Uma tentativa de criar um valor cíclico que não se beneficie de valores estruturados lentos que tenham sofrido intervenção gera um erro:

```
[A=B, B=A]
// [A = Error.Record("Expression.Error",
//         "A cyclic reference was encountered during evaluation"),
//  B = Error.Record("Expression.Error",
//         "A cyclic reference was encountered during evaluation"),
// ]
```

Alguns operadores no M são definidos pela recursão estrutural. Por exemplo, a igualdade de listas e registros é definida pela igualdade conjunta de listas de itens e campos de registro correspondentes, respectivamente.

Para valores não cíclicos, a aplicação da recursão estrutural produz uma *expansão finita* do valor: valores aninhados compartilhados são percorridos repetidamente, mas o processo de recursão sempre é encerrado.

Um valor cíclico tem uma *expansão infinita* ao aplicar a recursão estrutural. A semântica do M não faz acomodações especiais para tais expansões infinitas—. Uma tentativa de comparar valores cíclicos para igualdade, por exemplo, normalmente ficará sem recursos e terminará de maneira excepcional.

Operadores de seleção e de projeção

Os operadores de seleção e projeção permitem que os dados sejam extraídos dos valores de lista e de registro.

Acesso ao item

Um valor pode ser selecionado de uma lista ou tabela com base na respectiva posição de base zero nessa lista ou tabela usando uma *expressão-de-acesso-a-item*.

expressão-de-acesso-a-item:
    seleção-de-item
    seleção-de-item-opcional
seleção-de-item:

expressão-primária `{` *seletor-de-item* `}`

seleção-de-item-opcional:

expressão-primária `{` *seletor-de-item* `} ?`

seletor-de-item:

expressão

A *expressão-de-acesso-a-item* `x{y}` retorna:

- Para obter uma lista `x` e um número `y`, o item da lista `x` na posição `y`. O primeiro item de uma lista é considerado como tendo um índice ordinal igual a zero. Se a posição solicitada não existir na lista, um erro será gerado.
- Para uma tabela `x` e um número `y`, a linha da tabela `x` na posição `y`. A primeira linha de uma tabela é considerada como tendo um índice ordinal igual a zero. Se a posição solicitada não existir na tabela, um erro será gerado.
- Para uma tabela `x` e um registro `y`, a linha da tabela `x` que corresponde aos valores de campo do registro `y` para campos com nomes de campo equivalentes aos nomes de coluna de tabela correspondentes. Se não houver nenhuma linha correspondente exclusiva na tabela, um erro será gerado.

Por exemplo:

```
{"a","b","c"}{0}                      // "a"
{1, [A=2], 3}{1}                      // [A=2]
{true, false}{2}                      // error
#table({"A","B"},{{0,1},{2,1}}){0}    // [A=0,B=1]
#table({"A","B"},{{0,1},{2,1}}){[A=2]}  // [A=2,B=1]
#table({"A","B"},{{0,1},{2,1}}){[B=3]}  // error
#table({"A","B"},{{0,1},{2,1}}){[B=1]}  // error
```

A *expressão-de-acesso-a-item* também dá suporte ao formato `x{y}?`, que retorna `null` quando a posição (ou correspondência) `y` não existe na lista ou na tabela `x`. Se houver várias correspondências para `y`, um erro ainda será gerado.

Por exemplo:

```
{"a","b","c"}{0}?                     // "a"
{1, [A=2], 3}{1}?                     // [A=2]
{true, false}{2}?                     // null
#table({"A","B"},{{0,1},{2,1}}){0}    // [A=0,B=1]
#table({"A","B"},{{0,1},{2,1}}){[A=2]}  // [A=2,B=1]
#table({"A","B"},{{0,1},{2,1}}){[B=3]}  // null
#table({"A","B"},{{0,1},{2,1}}){[B=1]}  // error
```

O acesso ao item não força a avaliação de itens de lista ou de tabela além daquela que está sendo acessada. Por exemplo:

```
{ error "a", 1, error "c"}{1}  // 1
{ error "a", error "b"}{1}     // error "b"
```

Os seguintes preceitos são válidos quando o operador `x{y}` de acesso ao item é avaliado:

- Erros gerados durante a avaliação das expressões `x` ou `y` são propagados.
- A expressão `x` produz uma lista ou um valor de tabela.
- A expressão `y` produz um valor de registro se `x` produz um valor de tabela e, caso contrário, produz um

valor numérico.

- Se `y` produzir um valor numérico e o valor de `y` for negativo, um erro com o código de motivo `"Expression.Error"` será gerado.
- Se `y` produzir um valor numérico e o valor de `y` for maior ou igual à contagem de `x`, um erro com o código de motivo `"Expression.Error"` será gerado, a menos que o formato de operador opcional `x{y}?` seja usado; nesse caso, o valor `null` será retornado.
- Se `x` produz um valor de tabela e `y` produz um valor de registro e não há correspondências para `y` em `x`, um erro com o código de motivo `"Expression.Error"` é gerado, a menos que o formato de operador opcional `x{y}?` seja usado; nesse caso, o valor `null` é retornado.
- Se `x` produz um valor de tabela e `y` produz um valor de registro e há várias correspondências para `y` em `x`, um erro com o código de motivo `"Expression.Error"` é gerado.

Nenhum item em `x` diferente daquele na posição `y` é avaliado durante o processo de seleção de itens. (Para listas de streaming ou tabelas, os itens ou linhas que antecedem isso na posição `y` são ignorados, o que pode fazer com que eles sejam avaliados, dependendo da origem da lista ou da tabela.)

Acesso ao campo

A *expressão-de-acesso-ao-campo* é usada para *selecionar* um valor de um registro ou para *projetar* um registro ou uma tabela para um com menos campos ou colunas, respectivamente.

expressão-de-acesso-ao-campo:
    seleção-de-campo
    seleção-de-campo-de-destino-implícita
    projeção
    projeção-de-destino-implícita
seleção-de-campo:
    expressão-primária seletor-de-campo
seletor-de-campo:
    seletor-de-campo-obrigatório
    seletor-de-campo-opcional
seletor-de-campo-obrigatório:
    `[` *nome-do-campo* `]`
seletor-de-campo-opcional:
    `[` *nome-do-campo* `] ?`
nome-do-campo:
    identificador-generalizado
    identificador-entre-aspas
seleção-de-campo-de-destino-implícita:
    seletor-de-campo
projeção:
    expressão-primária projeção-obrigatória
    expressão-primária projeção-opcional
projeção-obrigatória:
    `[` *lista-de-seletores-obrigatórios* `]`
projeção-opcional:
    `[` *lista-de-seletores-obrigatórios* `] ?`
lista-de-seletores-obrigatórios:
    lista-de-seletores-obrigatórios
    lista-de-seletores-obrigatórios `,` *lista-de-seletores-obrigatórios*
projeção-de-destino-implícita:

projeção-obrigatória
projeção-opcional

A forma mais simples de acesso ao campo é a *seleção de campo obrigatório*. Ele usa o operador `x[y]` para procurar um campo em um registro por nome de campo. Se o campo `y` não existir em `x`, um erro será gerado. O formato `x[y]?` é usado para executar seleção de campo *opcional* e retorna `null` se o campo solicitado não existe no registro.

Por exemplo:

```
[A=1,B=2][B]       // 2
[A=1,B=2][C]       // error
[A=1,B=2][C]?      // null
```

O acesso coletivo de vários campos é compatível com os operadores para *projeção de registro obrigatória* e *projeção de registro opcional*. O operador `x[[y1],[y2],...]` projeta o registro em um novo registro com menos campos (selecionados por `y1`, `y2` e `...`). Se um campo selecionado não existir, um erro será gerado. O operador `x[[y1],[y2],...]` projeta o registro para um novo registro com os campos selecionados por `y1`, `y2` e `...`; se um campo estiver ausente, `null` será usado em seu lugar. Por exemplo:

```
[A=1,B=2][[B]]          // [B=2]
[A=1,B=2][[C]]          // error
[A=1,B=2][[B],[C]]?     // [B=2,C=null]
```

Os formatos `[y]` e `[y]?` são compatíveis como uma referência *abreviada* ao identificador `_` (sublinhado). As duas expressões a seguir são equivalentes:

```
[A]
_[A]
```

O seguinte exemplo ilustra a forma abreviada de acesso ao campo:

```
let _ = [A=1,B=2] in [A] //1
```

Os formatos `[[y1],[y2],...]` e `[[y1],[y2],...]?` também são compatíveis como uma abreviação e as duas expressões a seguir são, pela mesma lógica, equivalentes:

```
[[A],[B]]
_[[A],[B]]
```

A forma abreviada é particularmente útil em combinação com a abreviação `each`, uma forma de introduzir uma função de apenas um parâmetro chamado `_` (para obter detalhes, confira Declarações simplificadas). Juntas, as duas abreviações simplificam as expressões funcionais comuns de ordem superior:

```
List.Select( {[a=1, b=1], [a=2, b=4]}, each [a] = [b])
// {[a=1, b=1]}
```

A expressão acima é equivalente à seguinte versão mais longa, que também parece de mais difícil compreensão:

```
List.Select( {[a=1, b=1], [a=2, b=4]}, (_) => _[a] = _[b])
// {[a=1, b=1]}
```

O acesso ao campo não força a avaliação de campos além daqueles que estão sendo acessados. Por exemplo:

```
[A=error "a", B=1, C=error "c"][B]  // 1
[A=error "a", B=error "b"][B]       // error "b"
```

Os seguintes preceitos são válidos quando um operador `x[y]`, `x[y]?`, `x[[y]]` ou `x[[y]]?` de acesso ao campo é avaliado:

- Erros gerados durante a avaliação da expressão `x` são propagados.
- Os erros gerados durante a avaliação do campo `y` são permanentemente associados ao campo `y` e, em seguida, propagados. Qualquer acesso futuro ao campo `y` vai gerar o erro idêntico.
- A expressão `x` produz um valor de registro ou de tabela ou resulta na geração de um erro.
- Se o identificador `y` nomeia um campo que não existe em `x`, um erro com o código de motivo `"Expression.Error"` é gerado, a menos que o formato de operador opcional `...?` seja usado; nesse caso, o valor `null` é retornado.

Nenhum campo de `x` exceto aquele nomeado por `y` é avaliado durante o processo de acesso ao campo.

Operador de metadados

O registro de metadados para um valor é alterado usando o *operador meta* (`x meta y`).

expressão-de-metadados:
expressão-unária
expressão-unária `meta` *expressão-unária*

O exemplo a seguir constrói um valor de texto com um registro de metadados usando o operador `meta` e, em seguida, acessa o registro de metadados do valor resultante usando `Value.Metadata`:

```
Value.Metadata( "Mozart" meta [ Rating = 5 ] )
// [Rating = 5 ]
Value.Metadata( "Mozart" meta [ Rating = 5 ] )[Rating]
// 5
```

Os seguintes preceitos são válidos ao aplicar o operador de combinação de metadados `x meta y`:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- A expressão `y` precisa ser um registro, caso contrário, um erro com o código de motivo `"Expression.Error"` é gerado.
- O registro de metadados resultante é o registro de metadados de `x` mesclado com `y`. (Para obter a semântica de mesclagem de registros, confira Mesclagem de registros.)
- O valor resultante é o valor da expressão `x`, sem os respectivos metadados, com o registro de metadados recém-computado anexado.

As funções de biblioteca padrão `Value.RemoveMetadata` e `Value.ReplaceMetadata` podem ser usadas para remover todos os metadados de um valor e para substituir os metadados de um valor (em vez de mesclar os metadados em metadados possivelmente existentes). As seguintes expressões são equivalentes:

```
x meta y
Value.ReplaceMetadata(x, Value.Metadata(x) & y)
Value.RemoveMetadata(x) meta (Value.Metadata(x) & y)
```

Operadores de igualdade

O *operador de igualdade* `=` é usado para determinar se dois valores são iguais. O *operador de desigualdade* `<>` é usado para determinar se dois valores não são iguais.

expressão-de-igualdade:
expressão-relacional
expressão-relacional `=` *expressão-de-igualdade*
expressão-relacional `<>` *expressão-de-igualdade*

Por exemplo:

```
1 = 1           // true
1 = 2           // false
1 <> 1          // false
1 <> 2          // true
null = true     // false
null = null     // true
```

Os metadados não fazem parte da comparação de igualdade ou desigualdade. Por exemplo:

```
(1 meta [ a = 1 ]) = (1 meta [ a = 2 ]) // true
(1 meta [ a = 1 ]) = 1                  // true
```

Os seguintes preceitos são válidos ao aplicar os operadores de igualdade `x = y` e `x <> y`:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- O operador `=` terá um resultado de `true` se os valores forem iguais; caso contrário, o resultado será `false`.
- O operador `<>` terá um resultado de `false` se os valores forem iguais; caso contrário, o resultado será `true`.
- Os registros de metadados não são incluídos na comparação.
- Se os valores produzidos pela avaliação das expressões `x` e `y` não forem do mesmo tipo de valor, os valores não serão iguais.
- Se os valores produzidos pela avaliação da expressão `x` e `y` forem do mesmo tipo de valor, haverá regras específicas para determinar se eles são iguais, conforme definido abaixo.
- Os seguintes preceitos são sempre verdadeiros:

```
    (x = y) = not (x <> y)
```

Os operadores de igualdade são definidos para os seguintes tipos:

- O valor `null` é igual apenas a ele mesmo.

```
null = null    // true
null = true    // false
null = false   // false
```

- Os valores lógicos `true` e `false` são apenas iguais a eles mesmos. Por exemplo:

```
true = true      // true
false = false    // true
true = false     // false
true = 1         // false
```

- Os números são comparados usando a precisão especificada:
  - Se um dos dois números for `#nan`, os números não serão iguais.
  - Quando nenhum dos números é `#nan`, eles são comparados usando uma comparação bit-a-bit do valor numérico.
  - `#nan` é o único valor que não é igual a si mesmo.

    Por exemplo:

```
1 = 1,              // true
1.0 = 1             // true
2 = 1               // false
#nan = #nan         // false
#nan <> #nan        // true
```

- Duas durações serão iguais se elas representarem o mesmo número de tiques de 100 nanossegundos.
- Duas horas serão iguais se as magnitudes de suas partes (hora, minuto, segundo) forem iguais.
- Duas datas serão iguais se as magnitudes de suas partes (ano, mês, dia) forem iguais.
- Dois datetimes serão iguais se as magnitudes de suas partes (ano, mês, dia, hora, minuto, segundo) forem iguais.
- Dois datetimezones serão iguais se os datetimes UTC correspondentes forem iguais. Para chegar ao datetime UTC correspondente, a diferença de horas/minutos é subtraída do componente datetime do datetimezone.
- Dois valores de texto serão iguais se, ao usar uma comparação ordinal, com diferenciação de maiúsculas de minúsculas e sem diferenciação de cultura, eles tiverem o mesmo comprimento e caracteres iguais em posições correspondentes.
- Dois valores de lista serão iguais se todas as condições a seguir forem verdadeiras:
  - As duas listas contêm o mesmo número de itens.
  - Os valores de cada item posicionalmente correspondente nas listas são iguais. Isso significa que não apenas as listas precisam conter itens iguais, os itens também precisam estar na mesma ordem.

    Por exemplo:

```
{1, 2} = {1, 2}     // true
{2, 1} = {1, 2}     // false
{1, 2, 3} = {1, 2}  // false
```

- Dois registros serão iguais se todas as seguintes condições forem verdadeiras:
  - O número de campos é o mesmo.
  - Cada nome de campo de um registro também está presente no outro registro.
  - O valor de cada campo de um registro é igual ao campo com o mesmo nome presente no outro registro.

    Por exemplo:

```
[ A = 1, B = 2 ] = [ A = 1, B = 2 ]        // true
[ B = 2, A = 1 ] = [ A = 1, B = 2 ]        // true
[ A = 1, B = 2, C = 3 ] = [ A = 1, B = 2 ] // false
[ A = 1 ] = [ A = 1, B = 2 ]               // false
```

- Duas tabelas serão iguais se todas as condições a seguir forem verdadeiras:
  - O número de colunas é o mesmo.
  - Cada nome de coluna em uma tabela também está presente na outra tabela.
  - O número de linhas é o mesmo.
  - Cada linha tem valores iguais em células correspondentes.

    Por exemplo:

```
#table({"A","B"},{{1,2}}) = #table({"A","B"},{{1,2}}) // true
#table({"A","B"},{{1,2}}) = #table({"X","Y"},{{1,2}}) // false
#table({"A","B"},{{1,2}}) = #table({"B","A"},{{2,1}}) // true
```

- Um valor de função é igual a si mesmo, mas pode ou não ser igual a outro valor de função. Se dois valores de função forem considerados iguais, eles se comportarão de maneira idêntica quando invocados.

  Dois valores de função fornecidos sempre terão a mesma relação de igualdade.

- Um valor de tipo é igual a si mesmo, mas pode ou não ser igual a outro valor de tipo. Se dois valores de tipo forem considerados iguais, eles se comportarão de maneira idêntica quando consultados quanto à conformidade.

  Dois valores de tipo fornecidos sempre terão a mesma relação de igualdade.

Operadores relacionais

Os operadores `<`, `>`, `<=` e `>=` são chamados de *operadores relacionais*.

expressão-relacional:
  expressão-aditiva
  expressão-aditiva `<` *expressão-relacional*
  expressão-aditiva `>` *expressão-relacional*
  expressão-aditiva `<=` _expressão-relacional
  expressão-aditiva `>=` *expressão-relacional*

Esses operadores são usados para determinar a relação de ordenação relativa entre dois valores, conforme mostrado na tabela a seguir:

OPERAÇÃO	RESULTADO
`x < y`	`true` se `x` for menor que `y`, caso contrário, `false`
`x > y`	`true` se `x` for maior que `y`, caso contrário, `false`
`x <= y`	`true` se `x` for menor ou igual a `y`, caso contrário, `false`
`x >= y`	`true` se `x` for maior ou igual a `y`, caso contrário, `false`

Por exemplo:

```
0 <= 1            // true
null < 1          // null
null <= null      // null
"ab" < "abc"      // true
#nan >= #nan      // false
#nan <= #nan      // false
```

Os seguintes preceitos são válidos ao avaliar uma expressão que contém os operadores relacionais:

- Erros gerados ao avaliar as expressões de operando `x` ou `y` são propagados.
- Os valores produzidos pela avaliação das expressões `x` e `y` devem ser um valor de data, número, datetime, datetimezone, duração, lógico, nulo ou hora. Caso contrário, um erro com o código de motivo `"Expression.Error"` é gerado.
- Se um ou ambos os operandos forem `null`, o resultado será o valor `null`.
- Se ambos os operandos forem lógicos, o valor `true` será considerado maior que `false`.
- Se ambos os operandos forem durações, os valores serão comparados de acordo com o número total de tiques de 100 nanossegundos que eles representarem.
- Para comparar duas horas, devemos comparar suas partes de hora; se forem iguais, suas partes de minuto e, se forem iguais, suas partes de segundo.
- Para comparar duas datas, devemos comparar suas partes de ano; se forem iguais, suas partes de mês e, se forem iguais, suas partes de dia.
- Para comparar dois datetimes, devemos comparar suas partes de ano; se forem iguais, suas partes de mês; se forem iguais, suas partes de dia; se forem iguais, suas partes de hora; se forem iguais, suas partes de minuto e, se forem iguais, suas partes de segundo.
- Para comparar dois datetimezones, devemos normalizá-los para UTC, subtraindo a diferença de hora/minuto e, em seguida, comparando seus componentes datetime.
- Dois números `x` e `y` são comparados de acordo com as regras do padrão IEEE 754:
  - Se um dos operandos for `#nan`, o resultado será `false` para todos os operadores relacionais.
  - Quando nenhum operando é `#nan`, os operadores comparam os valores dos dois operandos de ponto flutuante em relação à ordenação,

    ```
    <code>-&#8734; < -max < ... < -min < -0.0 = +0.0 < +min < ... < +max < +&#8734;</code>
    ```

em que min e max são, respectivamente, o menor e o maior valores finitos positivos que podem ser representados. Os nomes para `-∞` e `+∞` no M são `-#infinity` e `#infinity`.

Os efeitos notáveis dessa ordenação são:

- Zeros negativos e positivos são considerados iguais.
- Um valor `-#infinity` é considerado menor que todos os outros valores numéricos, mas é igual a outro `-#infinity`.
- Um valor `#infinity` é considerado maior que todos os outros valores numéricos, mas é igual a outro `#infinity`.

Operadores lógicos condicionais

Os operadores `and` e `or` são chamados de operadores lógicos condicionais.

expressão-or-lógica:
    expressão-and-lógica
    expressão-and-lógica `or` *expressão-or-lógica*
expressão-and-lógica:
    expressão-is
    expressão-is `and` *expressão-and-lógica*

O operador `or` retorna `true` quando pelo menos um de seus operandos é `true`. O operando direito será avaliado se e somente se o operando esquerdo não for `true`.

O operador `and` retorna `false` quando pelo menos um de seus operandos é `false`. O operando direito será avaliado se e somente se o operando esquerdo não for `false`.

As tabelas da verdade para os operadores `or` e `and` são mostradas abaixo, com o resultado da avaliação da expressão de operando esquerdo no eixo vertical e o resultado da avaliação da expressão do operando direito no eixo horizontal.

`and`	`true`	`false`	`null`	`error`
`true`	`true`	`false`	`null`	`error`
`false`	`false`	`false`	`false`	`false`
`null`	`null`	`false`	`null`	`error`
`error`	`error`	`error`	`error`	`error`

`or`	`true`	`false`	`null`	`error`
`true`	`true`	`true`	`true`	`true`
`false`	`true`	`false`	`null`	`error`

`null`	`true`	`null`	`null`	`error`
`error`	`error`	`error`	`error`	`error`

Os seguintes preceitos são válidos ao avaliar uma expressão que contém operadores lógicos condicionais:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- Os operadores lógicos condicionais são definidos sobre os tipos `logical` e `null`. Se os valores de operando não forem desses tipos, um erro com o código de motivo `"Expression.Error"` será gerado.
- O resultado é um valor lógico.
- Na expressão `x` ou `y`, a expressão `y` será avaliada se e somente se `x` não for avaliada como `true`.
- Na expressão `x` e `y`, a expressão `y` será avaliada se e somente se `x` não for avaliada como `false`.

As duas últimas propriedades dão aos operadores lógicos condicionais a respectiva qualificação "condicional". Essas propriedades também são conhecidas como "curto-circuito". Essas propriedades são úteis para escrever *predicados compactos protegidos*. Por exemplo, as seguintes expressões são equivalentes:

```
d <> 0 and n/d > 1 if d <> 0 then n/d > 1 else false
```

Operadores aritméticos

Os operadores `+`, `-`, `*` e `/` são os *operadores aritméticos*.

expressão-aditiva:
    expressão-multiplicativa
    expressão-aditiva `+` *expressão-multiplicativa*
    expressão-aditiva `-` *expressão-multiplicativa*
expressão-multiplicativa:
    expressão-de-metadados
    expressão-multiplicativa `*` *expressão-de-metadados*
    expressão-multiplicativa `/` *expressão-de-metadados*

Precisão

Os números em M são armazenados usando uma variedade de representações para manter o máximo possível de informações sobre números provenientes de uma variedade de fontes. Os números são convertidos apenas de uma representação para outra, conforme requerido pelos operadores aplicados a eles. Duas precisões são compatíveis com o M:

PRECISÃO	SEMÂNTICA
`Precision.Decimal`	Representação decimal de 128 bits com um intervalo de ±1.0 x 10-28 to ±7.9 x 1028 e 28-29 dígitos significativos.
`Precision.Double`	Representação científica usando mantissa e expoente; está em conformidade com o padrão aritmético IEEE 754 de precisão dupla binário de 64 bits IEEE 754-2008.

As operações aritméticas são executadas escolhendo uma precisão, convertendo ambos os operandos para essa precisão (se necessário), executando a operação propriamente dita e, finalmente, retornando um número na precisão escolhida.

Os operadores aritméticos internos (`+` , `-` , `*` e `/`) usam precisão dupla. As funções de biblioteca padrão (`Value.Add` , `Value.Subtract` , `Value.Multiply` e `Value.Divide`) podem ser usadas para solicitar essas operações usando um modelo de precisão específico.

- Nenhum estouro numérico é possível: `#infinity` ou `-#infinity` representam valores de magnitudes grandes demais para serem representados.
- Nenhum fluxo negativo numérico é possível: `0` e `-0` representam valores de magnitudes pequenas demais para serem representados.
- O valor especial do IEEE 754 `#nan` (NaN—não é um número) é usado para cobrir casos que não são matematicamente válidos, como uma divisão de zero por zero.
- A conversão de precisão decimal para dupla é executada por meio do arredondamento de números decimais para o valor duplo equivalente mais próximo.
- A conversão da precisão de dupla para decimal é executada por meio do arredondamento de números duplos para o valor decimal equivalente mais próximo e, se necessário, estourando para valores `#infinity` ou `-#infinity` .

Operador de adição

A interpretação do operador de adição (`x + y`) depende do tipo de valor das expressões x e y avaliadas, da seguinte maneira:

X	A	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	`type number`	Soma numérica
`type number`	`null`	`null`	
`null`	`type number`	`null`	
`type duration`	`type duration`	`type duration`	Soma numérica de magnitudes
`type duration`	`null`	`null`	
`null`	`type duration`	`null`	
`type` *datetime*	`type duration`	`type` *datetime*	Deslocamento de data/hora por duração
`type duration`	`type` *datetime*	`type` *datetime*	
`type` *datetime*	`null`	`null`	
`null`	`type` *datetime*	`null`	

Na tabela, `type` *datetime* representa qualquer um entre `type date` , `type datetime` , `type datetimezone` ou `type time` . Ao adicionar uma duração e um valor de algum tipo de *datetime*, o valor resultante é desse mesmo

tipo.

Para outras combinações de valores além daquelas listadas na tabela, um erro com o código de motivo `"Expression.Error"` é gerado. Cada uma das combinações é abordada nas seções a seguir.

Os erros gerados ao avaliar qualquer um dos dois operandos são propagados.

Soma numérica

A soma de dois números é computada usando o *operador de adição*, produzindo um número.

Por exemplo:

```
1 + 1            // 2
#nan + #infinity  // #nan
```

O operador de adição `+` usado em números usa precisão dupla. A função de biblioteca padrão `Value.Add` pode ser usada para especificar a precisão decimal. Os seguintes preceitos são válidos ao computar uma soma de números:

- A soma em precisão dupla é computada de acordo com as regras de aritmética de IEEE 754 de precisão dupla binária de 64 bits, IEEE 754-2008. A tabela a seguir lista os resultados de todas as combinações possíveis de valores finitos diferentes de zero, zeros, infinitos e NaNs. Na tabela, `x` e `y` são valores finitos diferentes de zero e `z` é o resultado de `x + y`. Se `x` e `y` tiverem a mesma magnitude, mas sinais opostos, `z` será zero positivo. Se `x + y` for grande demais para ser representado no tipo de destino, `z` será um infinito com o mesmo sinal que `x + y`.

+	A	+0	-0	+°	-°	NAN
x	z	x	x	+°	-°	NaN
+0	a	+0	+0	+°	-°	NaN
-0	a	+0	-0	+°	-°	NaN
+°	+°	+°	+°	+°	NaN	NaN
-°	-°	-°	-°	NaN	-°	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN

- A soma em precisão decimal é computada sem perda de precisão. A escala do resultado é maior que as escalas dos dois operandos.

Soma das durações

A soma de duas durações é a duração que representa a soma do número de tiques de 100 nanossegundos representada pelas durações. Por exemplo:

```
#duration(2,1,0,15.1) + #duration(0,1,30,45.3)
// #duration(2, 2, 31, 0.4)
```

Deslocamento de data/hora por duração

Um *datetime* `x` e uma duração `y` podem ser adicionados usando `x + y` para computar um novo *datetime* cuja distância de `x` em uma linha do tempo linear é exatamente a magnitude de `y`. Aqui, *datetime* representa

qualquer uma das opções `Date`, `DateTime`, `DateTimeZone` e `Time`; assim, um resultado não nulo será do mesmo tipo. A diferença de datetime por duração pode ser calculado da seguinte maneira:

- Se o valor de dias desde o início da época de datetime for especificado, crie um datetime com os seguintes elementos de informação:
  - Calcule um novo equivalente de dias desde o início da época, a fim de dividir a magnitude de y pelo número de tiques de 100 nanossegundos em um período de 24 horas, truncando a parte decimal do resultado e adicionando esse valor aos dias desde o início da época de x.
  - Calcule uma nova contagem de tiques desde a meia-noite equivalente à adição da magnitude de y aos tiques desde a meia-noite de x, módulo o número de tiques de 100 nanossegundos em um período de 24 horas. Se x não especificar um valor para tiques desde a meia-noite, um valor de 0 será assumido.
  - Copie o valor da diferença em minutos em relação ao UTC, inalterado, de x.
- Se o valor de dias desde o início da época de datetime não for especificado, crie um datetime com os seguintes elementos de informação especificados:
  - Calcule uma nova contagem de tiques desde a meia-noite equivalente à adição da magnitude de y aos tiques desde a meia-noite de x, módulo o número de tiques de 100 nanossegundos em um período de 24 horas. Se x não especificar um valor para tiques desde a meia-noite, um valor de 0 será assumido.
  - Copie os valores de dias desde o início da época e a diferença em minutos em relação ao UTC, inalterados, de x.

Os exemplos a seguir mostram o cálculo da soma temporal absoluta quando o datetime especifica os *dias desde o início da época*:

```
#date(2010,05,20) + #duration(0,8,0,0)
    //#datetime( 2010, 5, 20, 8, 0, 0 )
    //2010-05-20T08:00:00

#date(2010,01,31) + #duration(30,08,0,0)
    //#datetime(2010, 3, 2, 8, 0, 0)
    //2010-03-02T08:00:00

#datetime(2010,05,20,12,00,00,-08) + #duration(0,04,30,00)
    //#datetime(2010, 5, 20, 16, 30, 0, -8, 0)
    //2010-05-20T16:30:00-08:00

#datetime(2010,10,10,0,0,0,0) + #duration(1,0,0,0)
   //#datetime(2010, 10, 11, 0, 0, 0, 0, 0)
   //2010-10-11T00:00:00+00:00
```

O exemplo a seguir mostra o cálculo da diferença de datetime por duração para uma determinada hora:

```
#time(8,0,0) + #duration(30,5,0,0)
   //#time(13, 0, 0)
   //13:00:00
```

Operador de subtração

A interpretação do operador de subtração (`x - y`) depende do tipo de valor das expressões `x` e `y` avaliadas, da seguinte maneira:

X	Y	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	`type number`	Diferença numérica
`type number`	`null`	`null`	
`null`	`type number`	`null`	
`type duration`	`type duration`	`type duration`	Diferença numérica de magnitudes
`type duration`	`null`	`null`	
`null`	`type duration`	`null`	
`type` *datetime*	`type` *datetime*	`type duration`	Duração entre datetimes
`type` *datetime*	`type duration`	`type` *datetime*	Deslocamento de Datetime por duração negada
`type` *datetime*	`null`	`null`	
`null`	`type` *datetime*	`null`	

Na tabela, `type` *datetime* representa qualquer um entre `type date`, `type datetime`, `type datetimezone` ou `type time`. Ao subtrair uma duração de um valor de algum tipo de *datetime*, o valor resultante é desse mesmo tipo.

Para outras combinações de valores além daquelas listadas na tabela, um erro com o código de motivo `"Expression.Error"` é gerado. Cada uma das combinações é abordada nas seções a seguir.

Os erros gerados ao avaliar qualquer um dos dois operandos são propagados.

Diferença numérica

A diferença entre dois números é computada usando o *operador de subtração*, produzindo um número. Por exemplo:

```
1 - 1              // 0
#nan - #infinity   // #nan
```

O operador de subtração `-` usado em números usa precisão dupla. A função de biblioteca padrão `Value.Subtract` pode ser usada para especificar a precisão decimal. Os seguintes preceitos são válidos ao computar uma diferença de números:

- A diferença em precisão dupla é computada de acordo com as regras de aritmética de IEEE 754 de precisão dupla binária de 64 bits, IEEE 754-2008. A tabela a seguir lista os resultados de todas as combinações possíveis de valores finitos diferentes de zero, zeros, infinitos e NaNs. Na tabela, `x` e `y` são valores finitos diferentes de zero e `z` é o resultado de `x - y`. Se `x` e `y` forem iguais, `z` será zero positivo. Se `x - y` for grande demais para ser representado no tipo de destino, `z` será um infinito com o mesmo sinal que `x - y`.

-	A	+0	-0	+°	-°	NAN
x	z	x	x	-°	+°	NaN
+0	-y	+0	+0	-°	+°	NaN
-0	-y	-0	+0	-°	+°	NaN
+°	+°	+°	+°	NaN	+°	NaN
-°	-°	-°	-°	-°	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN

- A diferença em precisão decimal é computada sem perda de precisão. A escala do resultado é maior que as escalas dos dois operandos.

Diferença de durações

A diferença entre duas durações é a duração que representa a diferença entre o número de tiques de 100 nanossegundos representada pelas durações. Por exemplo:

```
#duration(1,2,30,0) - #duration(0,0,0,30.45)
// #duration(1, 2, 29, 29.55)
```

Deslocamento de Datetime por duração negada

Um *datetime* `x` e uma duração `y` podem ser subtraídos usando `x - y` para computar um novo *datetime*. Aqui, *datetime* representa qualquer um entre `date`, `datetime`, `datetimezone` ou `time`. O *datetime* resultante tem uma distância de `x` em uma linha do tempo linear, que é exatamente a magnitude de `y`, na direção oposta ao sinal de `y`. A subtração de durações positivas gera resultados que são regressivos no tempo em relação a `x`, enquanto a subtração de valores negativos gera resultados que são progressivos no tempo.

```
#date(2010,05,20) - #duration(00,08,00,00)
   //#datetime(2010, 5, 19, 16, 0, 0)
   //2010-05-19T16:00:00
#date(2010,01,31) - #duration( 30,08,00,00)
   //#datetime(2009, 12, 31, 16, 0, 0)
   //2009-12-31T16:00:00
```

Duração entre dois datetimes

Dois *datetimes* `t` e `u` podem ser subtraídos usando `t - u` para computar a duração entre eles. Aqui, *datetime* representa qualquer um entre `date`, `datetime`, `datetimezone` ou `time`. A duração produzida subtraindo `u` de `t` precisa resultar `t` quando adicionada a `u`.

```
#date(2010,01,31) - #date(2010,01,15)
// #duration(16,00,00,00)
// 16.00:00:00

#date(2010,01,15)- #date(2010,01,31)
// #duration(-16,00,00,00)
// -16.00:00:00

#datetime(2010,05,20,16,06,00,-08,00) -
#datetime(2008,12,15,04,19,19,03,00)
// #duration(521,22,46,41)
// 521.22:46:41
```

A subtração `t - u` quando `u > t` resulta em uma duração negativa:

```
#time(01,30,00) - #time(08,00,00)
// #duration(0, -6, -30, 0)
```

Os seguintes preceitos são válidos ao subtrair dois *datetimes* usando `t - u`:

- u + (t - u) = t

Operador de multiplicação

A interpretação do operador de multiplicação (`x * y`) depende do tipo de valor das expressões x e y avaliadas, da seguinte maneira:

X	Y	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	`type number`	Produto numérico
`type number`	`null`	`null`	
`null`	`type number`	`null`	
`type duration`	`type number`	`type duration`	Múltiplo da duração
`type number`	`type duration`	`type duration`	Múltiplo da duração
`type duration`	`null`	`null`	
`null`	`type duration`	`null`	

Para outras combinações de valores além daquelas listadas na tabela, um erro com o código de motivo `"Expression.Error"` é gerado. Cada uma das combinações é abordada nas seções a seguir.

Os erros gerados ao avaliar qualquer um dos dois operandos são propagados.

Produto numérico

O produto de dois números é computado usando o *operador de multiplicação*, produzindo um número. Por exemplo:

```
2 * 4               // 8
6 * null            // null
#nan * #infinity    // #nan
```

O operador de multiplicação `*` usado em números usa precisão dupla. A função de biblioteca padrão `Value.Multiply` pode ser usada para especificar a precisão decimal. Os seguintes preceitos são válidos ao computar um produto de números:

- O produto em precisão dupla é computado de acordo com as regras de aritmética de IEEE 754 de precisão dupla binária de 64 bits, IEEE 754-2008. A tabela a seguir lista os resultados de todas as combinações possíveis de valores finitos diferentes de zero, zeros, infinitos e NaNs. Na tabela, `x` e `y` são valores positivos finitos. `z` é o resultado de `x * y`. Se o resultado é grande demais para o tipo de destino, `z` é infinito. Se o resultado é pequeno demais para o tipo de destino, `z` é zero.

*	+Y	-Y	+0	-0	+°	-°	NAN
+x	+z	-Z	+0	-0	+°	-°	NaN
-x	-Z	+z	-0	+0	-°	+°	NaN
+0	+0	-0	+0	-0	NaN	NaN	NaN
-0	-0	+0	-0	+0	NaN	NaN	NaN
+°	+°	-°	NaN	NaN	+°	-°	NaN
-°	-°	+°	NaN	NaN	-°	+°	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- O produto em precisão decimal é computado sem perda de precisão. A escala do resultado é maior que as escalas dos dois operandos.

Múltiplos de durações

O produto de uma duração e um número é a duração que representa o número de tiques de 100 nanossegundos representado pela multiplicação do operando de duração pelo operando de número. Por exemplo:

```
#duration(2,1,0,15.1) * 2
// #duration(4, 2, 0, 30.2)
```

Operador de divisão

A interpretação do operador de divisão (`x / y`) depende do tipo de valor das expressões `x` e `y` avaliadas, da seguinte maneira:

X	Y	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	`type number`	Quociente numérico
`type number`	`null`	`null`	
`null`	`type number`	`null`	

X	Y	RESULTADO	INTERPRETAÇÃO
`type duration`	`type number`	`type duration`	Fração de uma duração
`type duration`	`type duration`	`type duration`	Quociente numérico de durações
`type duration`	`null`	`null`	
`null`	`type duration`	`null`	

Para outras combinações de valores além daquelas listadas na tabela, um erro com o código de motivo `"Expression.Error"` é gerado. Cada uma das combinações é abordada nas seções a seguir.

Os erros gerados ao avaliar qualquer um dos dois operandos são propagados.

Quociente numérico

O quociente de dois números é computado usando o *operador de divisão*, produzindo um número. Por exemplo:

```
8 / 2              // 4
8 / 0              // #infinity
0 / 0              // #nan
0 / null           // null
#nan / #infinity   // #nan
```

O operador de divisão `/` usado em números usa precisão dupla. A função de biblioteca padrão `Value.Divide` pode ser usada para especificar a precisão decimal. Os seguintes preceitos são válidos ao calcular um quociente de números:

- O quociente em precisão dupla é computado de acordo com as regras de aritmética de IEEE 754 de precisão dupla binária de 64 bits, IEEE 754-2008. A tabela a seguir lista os resultados de todas as combinações possíveis de valores finitos diferentes de zero, zeros, infinitos e NaNs. Na tabela, `x` e `y` são valores positivos finitos. `z` é o resultado de `x / y`. Se o resultado é grande demais para o tipo de destino, `z` é infinito. Se o resultado é pequeno demais para o tipo de destino, `z` é zero.

/	+Y	-Y	+0	-0	+°	-°	NAN
+x	+z	-Z	+°	-°	+0	-0	NaN
-x	-Z	+z	-°	+°	-0	+0	NaN
+0	+0	-0	NaN	NaN	+0	-0	NaN
-0	-0	+0	NaN	NaN	-0	+0	NaN
+°	+°	-°	+°	-°	NaN	NaN	NaN
-°	-°	+°	-°	+°	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- A soma em precisão decimal é computada sem perda de precisão. A escala do resultado é maior que as

escalas dos dois operandos.

Quociente de durações

O quociente de duas durações é o número que representa o quociente do número de tiques de 100 nanossegundos representada pelas durações. Por exemplo:

```
#duration(2,0,0,0) / #duration(0,1,30,0)
// 32
```

Durações dimensionadas

O quociente de uma duração `x` e um número `y` é a duração que representa o quociente do número de tiques de 100 nanossegundos representados pela duração `x` e o número `y`. Por exemplo:

```
#duration(2,0,0,0) / 32
// #duration(0,1,30,0)
```

Combinação de estrutura

O operador de combinação (`x & y`) é definido para os seguintes tipos de valores:

X	Y	RESULTADO	INTERPRETAÇÃO
`type text`	`type text`	`type text`	Concatenação
`type text`	`null`	`null`	
`null`	`type text`	`null`	
`type date`	`type time`	`type datetime`	Mesclar
`type date`	`null`	`null`	
`null`	`type time`	`null`	
`type list`	`type list`	`type list`	Concatenação
`type record`	`type record`	`type record`	Mesclar
`type table`	`type table`	`type table`	Concatenação

Concatenação

Dois textos, duas listas ou dois valores de tabela podem ser concatenados usando `x & y`.

Este exemplo ilustra a concatenação de valores de texto:

```
"AB" & "CDE"     // "ABCDE"
```

Este exemplo ilustra a concatenação de listas:

```
{1, 2} & {3}     // {1, 2, 3}
```

Os seguintes preceitos são válidos ao concatenar dois valores usando `x & y`:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- Nenhum erro será propagado se um item de `x` ou de `y` contiver um erro.
- O resultado da concatenação de dois valores de texto é um valor de texto que contém o valor de x seguido imediatamente por y. Se um dos operandos for nulo e o outro for um valor de texto, o resultado será nulo.
- O resultado da concatenação de duas listas é uma lista que contém todos os itens de `x` seguidos por todos os itens de `y`.
- O resultado da concatenação de duas tabelas é uma tabela contendo a união das colunas das tabelas dos dois operandos. A ordenação de colunas de `x` é preservada, seguida pelas colunas que aparecem apenas em `y`, preservando a ordenação relativa delas. Para colunas que aparecem apenas em um dos operandos, `null` é usado para preencher valores de célula para o outro operando.

Mesclar

Mesclagem de registros

Dois registros podem ser mesclados usando `x & y`, produzindo um registro que inclui campos de `x` e `y`.

Os exemplos a seguir ilustram a mesclagem de registros:

```
[ x = 1 ] & [ y = 2 ]               // [ x = 1, y = 2 ]
[ x = 1, y = 2 ] & [ x = 3, z = 4 ] // [ x = 3, y = 2, z = 4 ]
```

Os seguintes preceitos são válidos ao mesclar dois registros usando `x + y`:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- Se um campo aparecer tanto em `x` quanto em `y`, o valor de `y` será usado.
- A ordem dos campos no registro resultante é a de `x`, seguida pelos campos em `y` que não fazem parte de `x`, na mesma ordem em que aparecem em `y`.
- A mesclagem de registros não causa a avaliação dos valores.
- Nenhum erro é gerado devido a um campo conter um erro.
- O resultado é um registro.

Mesclagem de data e hora

Uma data `x` pode ser mesclada com uma hora `y` usando `x & y`, produzindo um datetime que combina as partes de `x` e `y`.

O exemplo a seguir ilustra a mesclagem de uma data e uma hora:

```
#date(2013,02,26) & #time(09,17,00)
// #datetime(2013,02,26,09,17,00)
```

Os seguintes preceitos são válidos ao mesclar dois registros usando `x + y`:

- Erros gerados ao avaliar as expressões `x` ou `y` são propagados.
- O resultado é um datetime.

Operadores unários

Os operadores `+`, `-` e `not` são operadores unários.

expressão-unária:
expressão-de-tipo
`+` *expressão unária*
`-` *expressão unária*
`not` *expressão unária*

Operador de adição de unário

O operador de adição de unário (`+x`) é definido para os seguintes tipos de valores:

X	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	Adição de unário
`type duration`	`type duration`	Adição de unário
`null`	nulo	

Para outros valores, um erro com o código de motivo `"Expression.Error"` é gerado.

O operador de adição de unário permite que um sinal de `+` seja aplicado a um valor numérico, datetime ou nulo. O resultado é esse mesmo valor. Por exemplo:

```
+ - 1                // -1
+ + 1                // 1
+ #nan               // #nan
+ #duration(0,1,30,0) // #duration(0,1,30,0)
```

Os seguintes preceitos são válidos ao avaliar o operador de adição de unário `+x`:

- Os erros gerados ao avaliar `x` são propagados.
- Se o resultado da avaliação de `x` não for um valor numérico, um erro com o código de motivo `"Expression.Error"` será gerado.

Operador de subtração de unário

O operador de subtração de unário (`-x`) é definido para os seguintes tipos de valores:

X	RESULTADO	INTERPRETAÇÃO
`type number`	`type number`	Negação
`type duration`	`type duration`	Negação
`null`	`null`	

Para outros valores, um erro com o código de motivo `"Expression.Error"` é gerado.

O operador de subtração de unário é usado para alterar o sinal de um número ou duração. Por exemplo:

```
- (1 + 1)       // -2
- - 1           // 1
- - - 1         // -1
- #nan          // #nan
- #infinity     // -#infinity
- #duration(1,0,0,0)  // #duration(-1,0,0,0)
- #duration(0,1,30,0) // #duration(0,-1,-30,0)
```

Os seguintes preceitos são válidos ao avaliar o operador de subtração unário `-x`:

- Os erros gerados ao avaliar `x` são propagados.
- Se a expressão for um número, o resultado será o valor numérico da expressão `x` com seu sinal alterado. Se o valor for um NaN, o resultado também será um NaN.

Operador de negação lógica

O operador de negação lógica (`not`) é definido para os seguintes tipos de valores:

X	RESULTADO	INTERPRETAÇÃO
`type logical`	`type logical`	Negação
`null`	`null`	

Esse operador computa a operação `not` lógica em um determinado valor lógico. Por exemplo:

```
not true            // false
not false           // true
not (true and true)  // false
```

Os seguintes preceitos são válidos ao avaliar o operador lógico de negação `not x`:

- Os erros gerados ao avaliar `x` são propagados.
- O valor produzido da avaliação da expressão x precisa ser um valor lógico, caso contrário, um erro com o código de motivo `"Expression.Error"` precisará ser gerado. Se o valor for `true`, o resultado será `false`. Se o operando for `false`, o resultado será `true`.

O resultado é um valor lógico.

Operadores de tipo

Os operadores `is` e `as` são conhecidos como operadores de tipo.

Operador de compatibilidade de tipo

O operador de compatibilidade de tipo `x is y` é definido para os seguintes tipos de valores:

X	Y	RESULTADO
`type any`	*tipo-primitivo-que-permite-valor-nulo*	`type logical`

A expressão `x is y` retornará `true` se o tipo atribuído de `x` for compatível com `y` e retornará `false` se o tipo atribuído de `x` for incompatível com `y`. `y` precisa ser um *tipo-primitivo-que-permite-valor-nulo*.

expressão-is:
    expressão-as
    expressão-is `is` *tipo-primitivo-que-permite-valor-nulo*
tipo-primitivo-que-permite-valor-nulo:
    `nullable`$_{opt}$ *tipo-primitivo*

A compatibilidade de tipo, conforme compatível com o operador `is`, é um subconjunto de compatibilidade de tipo geral e é definida usando as seguintes regras:

- Se `x` for nulo, ele será compatível se y for um tipo que permite valor nulo ou o tipo `any`.
- Se `x` for não nulo, ele será compatível se o tipo primitivo de `x` for o mesmo que `y`.

Os seguintes preceitos são válidos ao avaliar a expressão `x is y`:

- Um erro gerado ao avaliar a expressão `x` é propagado.

Operador de asserção de tipo

O operador de asserção de tipo `x as y` é definido para os seguintes tipos de valores:

X	Y	RESULTADO
`type any`	*tipo-primitivo-que-permite-valor-nulo*	`type any`

A expressão `x as y` declara que o valor `x` é compatível com `y`, conforme o operador `is`. Se não for compatível, um erro será gerado. `y` precisa ser um *tipo-primitivo-que-permite-valor-nulo*.

expressão-as:
    expressão-de-igualdade
    expressão-as `as` *tipo-primitivo-que-permite-valor-nulo*

A expressão `x as y` é avaliada desta maneira:

- Uma verificação de compatibilidade de tipo `x is y` é executada e a asserção retorna `x` inalterado se o teste é executado com sucesso.
- Se a verificação de compatibilidade falhar, um erro com o código de motivo `"Expression.Error"` será gerado.

Exemplos:

```
1 as number               // 1
"A" as number             // error
null as nullable number   // null
```

Os seguintes preceitos são válidos ao avaliar a expressão `x as y`:

- Um erro gerado ao avaliar a expressão `x` é propagado.

Permitir

08/05/2020 • 2 minutes to read

Expressão let

Uma expressão let pode ser usada para capturar um valor de um cálculo intermediário em uma variável.

expressão-let:
    `let` *lista-de-variáveis* `in` *expressão*
lista-de-variáveis:
    variável
    variável `,` *lista-de-variáveis*
variável:
    nome-da-variável `=` *expressão*
nome-da-variável:
    identificador

O seguinte exemplo mostra os resultados intermediários que estão sendo calculados e armazenados em variáveis `x`, `y` e `z` que, em seguida, são usados em um cálculo subsequente, `x + y + z`:

```
let     x = 1 + 1,
        y = 2 + 2,
        z = y + 1
in
        x + y + z
```

O resultado dessa expressão é:

```
11  // (1 + 1) + (2 + 2) + (2 + 2 + 1)
```

Os seguintes preceitos são válidos ao avaliar expressões dentro da *expressão-let*:

- As expressões na lista de variáveis definem um novo escopo que contém os identificadores da produção *lista-de-variáveis* e precisa estar presente ao se avaliar as expressões dentro das produções de *lista-de-variáveis*. As expressões dentro de *lista-de-variáveis* podem se referir uma à outra.
- As expressões dentro da *lista-de-variáveis* precisam ser avaliadas antes que a expressão na *expressão-let* seja avaliada.
- A menos que as expressões na *lista-de-variáveis* sejam acessadas, elas não podem ser avaliadas.
- Erros que são gerados durante a avaliação das expressões na *expressão-let* são propagados.

Uma expressão let pode ser vista como uma simplificação sintática em uma expressão de registro implícita. A seguinte expressão é equivalente ao exemplo acima:

```
[     x = 1 + 1,
      y = 2 + 2,
      z = y + 1,
      result = x + y + z
][result]
```

Condicionais

08/05/2020 • 2 minutes to read

A *expressão-if* seleciona entre duas expressões com base no valor de um valor de entrada lógico e avalia apenas a expressão selecionada.

expressão-if:
    `if` *condição-if* `then` *expressão-verdadeira* `else` *expressão-falsa*
condição-if:
    expressão
expressão-verdadeira:
    expressão
expressão-falsa:
    expressão

Estes são exemplos de *expressões-if*.

```
if 2 > 1 then 2 else 1          // 2
if 1 = 1 then "yes" else "no"   // "yes"
```

As seguintes condições se verificam ao avaliar uma *expressão-if*.

- Se o valor produzido pela avaliação da *condição-if* não for um valor lógico, um erro com o código de motivo `"Expression.Error"` será gerado.
- A *expressão-verdadeira* será avaliada apenas se a *condição-if* for avaliada como o valor `true`.
- A *expressão-falsa* será avaliada somente se a *condição-if* for avaliada como o valor `false`.
- O resultado da *expressão-if* será o valor da *expressão-verdadeira* se a *condição-if* for `true` e será o valor da *expressão-falsa* se a *condição-if* for `false`.
- Erros gerados durante a avaliação da *condição-if*, *expressão-verdadeira* ou *expressão-falsa* são propagados.

Funções

26/05/2020 • 12 minutes to read

Uma *função* é um valor que representa um mapeamento de um conjunto de valores de argumento para um valor. Uma função é invocada por um conjunto de valores de entrada (os valores de argumento) fornecido e produz um valor de saída (o valor retornado).

Escrita de funções

As funções são gravadas usando uma *expressão-de-função*:

expressão-de-função:
    `(` *lista-de-parâmetros*$_{opt}$ `)` *tipo-de-retorno-de-função*$_{opt}$ `=>` *corpo-da-função*
corpo-da-função:
    expressão
lista-de-parâmetros:
    lista-de-parâmetros-fixos
    lista-de-parâmetros-fixos `,` *lista-de-parâmetros-opcionais*
    lista-de-parâmetros-opcionais
lista-de-parâmetros-fixos:
    parâmetro
    parâmetro `,` *lista-de-parâmetros-fixos*
parâmetro:
    nome-do-parâmetro tipo-de-parâmetro$_{opt}$
nome-do-parâmetro:
    identificador
tipo-de-parâmetro:
    asserção
tipo-de-retorno-de-função:
    asserção
asserção:
    `as` *tipo-primitivo-que-permite-valor-nulo*
lista-de-parâmetros-opcionais:
    parâmetro-opcional
    parâmetro-opcional `,` *lista-de-parâmetros-opcionais*
parâmetro-opcional:
    `optional` *parâmetro*
tipo-primitivo-que-permite-valor-nulo
    `nullable` $_{opt}$ tipo-primitivo_

Veja a seguir um exemplo de uma função que requer exatamente dois valores `x` e `y` e produz o resultado da aplicação do operador `+` a esses valores. `x` e `y` são *parâmetros* que fazem parte da *lista-de-parâmetros-formais* da função, e `x + y` é o *corpo da função*:

```
(x, y) => x + y
```

O resultado da avaliação de uma *expressão-de-função* é produzir um valor de função (não avaliar o *corpo-da-função*). Como uma convenção neste documento, os valores de função (em lugar das expressões-de-função) são mostrados com a *lista-de-parâmetros-formais*, mas com reticências (`...`) em vez do *corpo-da-função*. Por

exemplo, depois que a expressão de função acima tiver sido avaliada, ela será mostrada como o seguinte valor de função:

```
(x, y) => ...
```

Os seguintes operadores são definidos para valores de função:

OPERADOR	RESULTADO
`x = y`	Igual
`x <> y`	Diferente

O tipo nativo de valores de função é um tipo de função personalizada (derivado do tipo intrínseco `function`) que lista os nomes de parâmetro e especifica todos os tipos de parâmetro e o tipo de retorno para `any`. (Confira Tipos de função para obter detalhes sobre os tipos de função.)

Invocação de funções

O *corpo-da-função* de uma função é executado *invocando-se* o valor da função pelo uso de uma *expressão-de-invocação*. Invocar um valor de função significa que o *corpo-da-função* do valor da função é avaliado e um valor é retornado ou um erro é gerado.

expressão-de-invocação:
    expressão-primária `(` *lista-de-argumentos*$_{opt}$ `)`
lista-de-argumentos:
    lista-de-expressões

Cada vez que um valor de função é invocado, um conjunto de valores é especificado como uma *lista-de-argumentos*, também chamados de *argumentos* para a função.

Uma *lista-de-argumentos* é usada para especificar um número fixo de argumentos diretamente como uma lista de expressões. O seguinte exemplo define um registro com um valor de função em um campo e, em seguida, invoca a função de outro campo do registro:

```
[
    MyFunction = (x, y, z) => x + y + z,
    Result1 = MyFunction(1, 2, 3)           // 6
]
```

Os seguintes preceitos são válidos ao invocar uma função:

- O ambiente usado para avaliar o *corpo-da-função* da função inclui uma variável que corresponde a cada parâmetro, com o mesmo nome que o parâmetro. O valor de cada parâmetro corresponde a um valor construído com base na *lista-de-argumentos* da *expressão-de-invocação*, conforme definido em Parâmetros.
- Todas as expressões correspondentes aos argumentos da função são avaliadas antes que o *corpo-da-função* seja avaliado.
- Os erros gerados ao avaliar as expressões na *lista-de-expressões* ou *expressão-de-função* são propagados.
- O número de argumentos construídos com base na *lista-de-argumentos* deverá ser compatível com os parâmetros formais da função. Caso contrário, um erro será gerado com o código de motivo

`"Expression.Error"`. O processo para determinar a compatibilidade é definido em Parâmetros.

Parâmetros

Há dois tipos de parâmetros formais que podem estar presentes em uma *lista-de-parâmetros-formais*:

- Um parâmetro *obrigatório* indica que um argumento correspondente ao parâmetro precisa sempre ser especificado quando uma função é invocada. Os parâmetros obrigatórios precisam ser especificados primeiro na *lista-de-parâmetros-formais*. A função neste exemplo define os parâmetros necessários `x` e `y`:

```
[
    MyFunction = (x, y) => x + y,

    Result1 = MyFunction(1, 1),     // 2
    Result2 = MyFunction(2, 2)      // 4
]
```

- Um parâmetro *opcional* indica que um argumento correspondente ao parâmetro pode ser especificado quando uma função é invocada, mas não obrigatoriamente. Se um argumento que corresponder a um parâmetro opcional não for especificado quando a função for invocada, o valor `null` será usado. Os parâmetros opcionais devem aparecer após qualquer parâmetro obrigatório em uma *lista-de-parâmetros-formais*. A função neste exemplo define um parâmetro fixo `x` e um parâmetro opcional `y`:

```
[
    MyFunction = fn(x, optional y) =>
                    if (y = null) x else x + y,
    Result1 = MyFunction(1),        // 1
    Result2 = MyFunction(1, null),  // 1
    Result3 = MyFunction(2, 2),     // 4
]
```

O número de argumentos que são especificados quando uma função é invocada deve ser compatível com a lista de parâmetros formais. A compatibilidade de um conjunto de argumentos `A` para uma função `F` é calculada da seguinte maneira:

- Deixe o valor *N* representar o número de argumentos `A` construídos com base na *lista-de-argumentos*. Por exemplo:

```
MyFunction()            // N = 0
MyFunction(1)           // N = 1
MyFunction(null)        // N = 1
MyFunction(null, 2)     // N = 2
MyFunction(1, 2, 3)     // N = 3
MyFunction(1, 2, null)  // N = 3
MyFunction(1, 2, {3, 4}) // N = 3
```

- Deixe o valor *Required* representar o número de parâmetros fixos de `F` e *Optional* representar o número de parâmetros opcionais de `F`. Por exemplo:

```
()              // Required = 0, Optional = 0
(x)             // Required = 1, Optional = 0
(optional x)    // Required = 0, Optional = 1
(x, optional y) // Required = 1, Optional = 1
```

- Os argumentos `A` serão compatíveis com a função `F` se as seguintes condições forem verdadeiras:

- (*N >= Fixo*) e (*N <= (Fixo + Opcional)*)
- Os tipos de argumento são compatíveis com os tipos de parâmetro correspondentes do `F`

- Se a função tiver um tipo de retorno declarado, o valor de resultado do corpo da função `F` será compatível com o tipo de retorno de `F` se o seguinte for verdadeiro:
  - O valor produzido pela avaliação do corpo da função com os argumentos fornecidos para os parâmetros de função tem um tipo compatível com o tipo de retorno.
- Se o corpo da função gerar um valor incompatível com o tipo de retorno da função, um erro com o código de motivo `"Expression.Error"` será gerado.

Funções recursivas

Para gravar um valor de função que é recursivo, é necessário usar o operador de escopo (`@`) para fazer referência à função dentro do escopo dela. Por exemplo, o seguinte registro contém um campo que define a função `Factorial` e outro campo que a invoca:

```
[
    Factorial = (x) =>
                if x = 0 then 1 else x * @Factorial(x - 1),
    Result = Factorial(3)  // 6
]
```

Da mesma forma, funções mutuamente recursivas podem ser gravadas, desde que cada função que precise ser acessada tenha um nome. No exemplo a seguir, parte da função `Factorial` foi refatorada em uma segunda função `Factorial2`.

```
[
    Factorial = (x) => if x = 0 then 1 else Factorial2(x),
    Factorial2 = (x) => x * Factorial(x - 1),
    Result = Factorial(3)     // 6
]
```

Fechamentos

Uma função pode retornar outra função como um valor. Essa função pode, por sua vez, depender de um ou mais parâmetros da função original. No seguinte exemplo, a função associada ao campo `MyFunction` retorna uma função que retorna o parâmetro especificado para ela:

```
[
    MyFunction = (x) => () => x,
    MyFunction1 = MyFunction(1),
    MyFunction2 = MyFunction(2),
    Result = MyFunction1() + MyFunction2()  // 3
]
```

Cada vez que a função é invocada, um novo valor de função será retornado mantendo o valor do parâmetro, de modo que, quando a função for invocada, o valor do parâmetro será retornado.

Funções e ambientes

Além dos parâmetros, o *corpo-da-função* de uma *expressão-de-função* pode referenciar variáveis que estão presentes no ambiente quando a função é inicializada. Por exemplo, a função definida pelo campo `MyFunction` acessa o campo `C` do registro delimitador `A`:

```
[
A =
    [
        MyFunction = () => C,
        C = 1
    ],
B = A[MyFunction]()           // 1
]
```

Quando `MyFunction` é invocado, ele acessa o valor da variável `C`, embora ele esteja sendo invocado de um ambiente (`B`) que não contém uma variável `C`.

Declarações simplificadas

A *expressão-each* é uma abreviação sintática para declarar funções não tipadas usando um parâmetro formal chamado `_` (sublinhado).

expressão-each:
    `each` *corpo-da-expressão-each*
corpo-da-expressão-each:
    corpo-da-função

Declarações simplificadas são normalmente usadas para melhorar a legibilidade da invocação de uma função de ordem superior.

Por exemplo, os seguintes pares de declarações são semanticamente equivalentes:

```
each _ + 1
(_) => _ + 1
each [A]
(_) => _[A]

Table.SelectRows( aTable, each [Weight] > 12 )
Table.SelectRows( aTable, (_) => _[Weight] > 12 )
```

Tratamento de erros

08/05/2020 • 8 minutes to read

O resultado da avaliação de uma expressão do M produz um dos seguintes resultados:

- Apenas um valor é produzido.
- Um *erro é gerado*, indicando que o processo de avaliação da expressão não pôde produzir um valor. Um erro contém um valor de registro que pode ser usado para fornecer informações adicionais sobre o que causou a avaliação incompleta.

Os erros podem ser gerados de dentro de uma expressão e podem ser manipulados de dentro de uma expressão.

Como gerar erros

A sintaxe para gerar um erro é a seguinte:

expressão-para-geração-de-erros:
`error` *expressão*

Valores de texto podem ser usados como abreviação para valores de erro. Por exemplo:

```
error "Hello, world" // error with message "Hello, world"
```

Os valores de erro completos são registros e podem ser construídos usando a função `Error.Record`:

```
error Error.Record("FileNotFound", "File my.txt not found",
     "my.txt")
```

A expressão acima é equivalente a:

```
error [
    Reason = "FileNotFound",
    Message = "File my.txt not found",
    Detail = "my.txt"
]
```

Gerar um erro fará com que a avaliação da expressão atual pare e a pilha de avaliação da expressão se desenrole até que uma das seguintes situações ocorra:

- Um campo de registro, um membro de seção ou uma variável let—coletivamente: uma *entrada*—é alcançada. A entrada está marcada como tendo um erro, o valor do erro é salvo com essa entrada e, em seguida, propagada. Qualquer acesso subsequente a essa entrada fará com que um erro idêntico seja gerado. Outras entradas do registro, da seção ou da expressão let não serão necessariamente afetadas (a menos que acessem uma entrada marcada anteriormente como tendo um erro).
- A expressão de nível superior é alcançada. Nesse caso, o resultado da avaliação da expressão de nível superior é um erro em vez de um valor.
- Uma expressão `try` é alcançada. Nesse caso, o erro é capturado e retornado como um valor.

Gerenciando erros

Uma *expressão-de-tratamento-de-erro* é usada para gerenciar um erro:

_expressão-de-tratamento-de-erro:
    `try` *expressão-protegida cláusula-otherwise*$_{opt}$
expressão-protegida:
    expressão
cláusula-otherwise:
    `otherwise` *expressão-padrão*
expressão-padrão:
    expressão

Os seguintes preceitos são válidos ao avaliar uma *expressão-de-tratamento-de-erro* sem uma *cláusula-otherwise*.

- Se a avaliação da expressão-protegida não resultar em um erro e produzir um valor x, o valor produzido pela expressão-de-tratamento-de-erro será um registro com o seguinte formato:

```
    [ HasErrors = false, Value = x ]
```

- Se a avaliação da expressão-protegida gerar um valor de erro e o resultado da expressão-de-tratamento-de-erro for um registro com o seguinte formato:

```
    [ HasErrors = true, Error = e ]
```

Os seguintes preceitos são válidos ao avaliar uma *expressão-de-tratamento-de-erro* com uma *cláusula-otherwise*.

- A expressão-protegida precisa ser avaliada antes da cláusula-otherwise.
- A cláusula-otherwise precisará ser avaliada se e somente se a avaliação da expressão-protegida gerar um erro.
- Se a avaliação da *expressão-protegida* gerar um erro, o valor produzido pela *expressão-de-tratamento-de-erro* será o resultado da avaliação da cláusula-otherwise.
- Erros gerados durante a avaliação da cláusula-otherwise são propagados.

O seguinte exemplo ilustra uma *expressão-de-tratamento-de-erro* em um caso em que nenhum erro é gerado:

```
let
    x = try "A"
in
    if x[HasError] then x[Error] else x[Value]
// "A"
```

Este exemplo mostra como gerar um erro e tratá-lo:

```
let
    x = try error "A"
in
    if x[HasError] then x[Error] else x[Value]
// [ Reason = "Expression.Error", Message = "A", Detail = null ]
```

Uma cláusula-otherwise pode ser usada para substituir erros gerenciados por uma expressão try com um valor alternativo:

```
try error "A" otherwise 1
// 1
```

Se a cláusula-otherwise também gerar um erro, a expressão try inteira fará o mesmo:

```
try error "A" otherwise error "B"
// error with message "B"
```

Erros no registro e inicializadores let

O exemplo a seguir mostra um inicializador de registro com um campo `A` que gera um erro e é acessado por dois outros campos, `B` e `C`. O campo `B` não gerencia o erro que é gerado pelo `A`, mas `C` o gerencia. O campo final `D` não acessa `A` e, portanto, não é afetado pelo erro em `A`.

```
[
    A = error "A",
    B = A + 1,
    C = let x =
            try A in
                if not x[HasError] then x[Value]
                else x[Error],
    D = 1 + 1
]
```

O resultado da avaliação da expressão acima é:

```
[
    A = // error with message "A"
    B = // error with message "A"
    C = "A",
    D = 2
]
```

O tratamento de erro em M deve ser executado próximo à causa de erros para lidar com os efeitos da inicialização de campo lenta e das avaliações de fechamento adiadas. O seguinte exemplo mostra uma tentativa malsucedida de tratar um erro usando uma expressão `try`:

```
let
    f = (x) => [ a = error "bad", b = x ],
    g = try f(42) otherwise 123
in
    g[a]  // error "bad"
```

Neste exemplo, a definição `g` foi destinada a tratar o erro gerado ao chamar `f`. No entanto, o erro é gerado por um inicializador de campo que só é executado quando necessário. Assim, isso só ocorrerá após o registro ter sido retornado de f e passado pela expressão `try`.

Erro "não implementado"

Enquanto uma expressão está sendo desenvolvida, um autor pode querer não incluir a implementação para algumas partes da expressão, mas ainda pode querer ser capaz de executar essa expressão. Uma forma de lidar com esse caso é gerar um erro para as partes não implementadas. Por exemplo:

```
(x, y) =>
     if x > y then
         x - y
     else
         error Error.Record("Expression.Error",
            "Not Implemented")
```

O símbolo de reticências (`...`) pode ser usado como um atalho para `error`.

expressão-não-implementada:

`...`

Por exemplo, o seguinte é equivalente ao exemplo anterior:

```
(x, y) => if x > y then x - y else ...
```

Seções

09/05/2020 • 7 minutes to read

Um *documento-de-seção* é um programa do M que consiste em várias expressões nomeadas.

documento-de-seção:
    seção
seção:
    atributos-literais$_{opt}$ `section` *nome-da-seção* `;` *membros-da-seção*$_{opt}$
nome-da-seção:
    identificador
membros-da-seção:
    membro-da-seção membro-da-seção$_{opt}$
membros-da-seção:
    atributos-literais$_{opt}$ `shared`$_{opt}$ *nome-do-membro-da-seção* `=` *expressão* `;`
nome-do-membro-da-sessão:
    identificador

No M, uma seção é um conceito organizacional que permite que expressões relacionadas sejam nomeadas e agrupadas em um documento. Cada seção tem um *nome-da-seção*, que identifica a seção e qualifica os nomes dos *membros-da-seção* declarados na seção em questão. Um *membro-da-sessão* consiste em um *nome-do-membro* e em uma *expressão*. As expressões de membro da seção podem se referir a outros membros da mesma seção, diretamente pelo nome do membro.

O seguinte exemplo mostra um documento-de-seção que contém uma seção:

```
section Section1;

A = 1;                          //1
B = 2;                          //2
C = A + B;                      //3
```

As expressões de membro da seção podem se referir a membros da seção localizados em outras seções por meio de uma *expressão-de-acesso-à-seção*, que qualifica um nome de membro da seção com o nome da seção que o contém.

expressão-de-acesso-à-seção:
    identificador `!` *identificador*

O seguinte exemplo mostra um documento que contém duas seções que são mutuamente referenciais:

```
section Section1;
A = "Hello";                    //"Hello"
B = 1 + Section2!A;             //3

section Section2;
A = 2;                          //2
B = Section1!A & " world!";     /"Hello, world"
```

Os membros da seção podem ser opcionalmente declarados como `shared`, o que omite a necessidade de usar uma *expressão-de-acesso-à-seção* ao se referir a membros compartilhados que existam fora da seção recipiente. Membros compartilhados em seções externas podem ser referenciados pelos respectivos nomes de membro não

qualificados, desde que nenhum membro do mesmo nome seja declarado na seção que os referencia e nenhuma outra seção tenha um membro compartilhado de mesmo nome.

O seguinte exemplo ilustra o comportamento de membros compartilhados quando usados entre seções dentro do mesmo documento:

```
section Section1;
shared A = 1;        // 1

section Section2;
B = A + 2;           // 3 (refers to shared A from Section1)

section Section3;
A = "Hello";         // "Hello"
B = A + " world";    // "Hello world" (refers to local A)
C = Section1!A + 2;  // 3
```

A definição de um membro compartilhado com o mesmo nome em seções diferentes produzirá um ambiente global válido, no entanto, o acesso ao membro compartilhado resultará em um erro.

```
section Section1;
shared A = 1;

section Section2;
shared A = "Hello";

section Section3;
B = A;    //Error: shared member A has multiple definitions
```

Os seguintes preceitos são válidos ao avaliar um documento-de-seção:

- Cada *nome-de-seção* precisa ser exclusivo no ambiente global.
- Em uma seção, cada *membro-da-seção* precisa ter um *nome-do-membro-da-sessão* exclusivo.
- Membros da seção compartilhados com mais de uma definição geram um erro quando o membro compartilhado é acessado.
- O componente de expressão de um *membro-da-seção* não pode ser avaliado antes que o membro da seção seja acessado.
- Os erros gerados enquanto o componente de expressão de um *membro-da-seção* é avaliado são associados a esse membro da seção antes da propagação para o exterior e, em seguida, gerados novamente cada vez que o membro da seção é acessado.

Vinculação de documentos

Um conjunto de documentos de seção do M pode ser vinculado a um valor de registro opaco que tem um campo por membro compartilhado dos documentos da seção. Se os membros compartilhados tiverem nomes ambíguos, um erro será gerado.

O valor de registro resultante fecha totalmente o ambiente global no qual o processo de vinculação foi executado. Esses registros são, portanto, componentes adequados para compor documentos do M de outros conjuntos (vinculados) de documentos do M. Não há oportunidades para conflitos de nomenclatura.

As funções `Embedded.Value` de biblioteca padrão podem ser usadas para recuperar valores de registro "inseridos" que correspondem a componentes do M reutilizados.

Introspecção de documento

O M fornece acesso programático ao ambiente global por meio das palavras-chave `#sections` e `#shared`.

#sections

A variável intrínseca `#sections` retorna todas as seções dentro do ambiente global como um registro. Esse registro é codificado pelo nome da seção e cada valor é uma representação de registro da seção correspondente indexada pelo nome do membro da seção.

O seguinte exemplo mostra um documento que consiste em duas seções e o registro produzido pela avaliação da variável intrínseca `#sections` dentro do contexto desse documento:

```
section Section1;
A = 1;
B = 2;

section Section2;
C = "Hello";
D = "world";

#sections
//[
//  Section1 = [ A = 1, B = 2],
//  Section2 = [ C = "Hello", D = "world" ]
//]
```

Os seguintes preceitos são válidos ao avaliar `#sections`:

- A variável intrínseca `#sections` preserva o estado de avaliação de todas as expressões de membro da seção dentro do documento.
- A variável intrínseca `#sections` não força a avaliação de nenhum membro não avaliado da seção.

#shared

A variável intrínseca `#shared` retorna um registro que contém os nomes e valores de todos os membros da seção compartilhados que estão atualmente no escopo.

O seguinte exemplo mostra um documento com dois membros compartilhados e o registro correspondente produzido pela avaliação da variável intrínseca #shared dentro do contexto desse documento:

```
section Section1;
shared A = 1;
B = 2;

Section Section2;
C = "Hello";
shared D = "world";

//[
//  A = 1,
//  D = "world"
//]
```

Os seguintes preceitos são válidos ao avaliar `#shared`:

- A variável intrínseca `#shared` preserva o estado de avaliação de todas as expressões de membro compartilhado dentro do documento.
- A variável intrínseca `#shared` não força a avaliação de nenhum membro não avaliado da seção.

Gramática consolidada

04/06/2020 • 9 minutes to read

Gramática lexical

unidade-lexical:
    elementos-lexicais$_{opt}$
elementos-lexicais:
    elemento-lexical elementos-lexicais$_{opt}$
elemento-lexical:
    espaço em branco
    comentário de token

Espaço em branco

espaço em branco:
    Qualquer caractere com classe Unicode ZS
    Caractere de tabulação horizontal (`U+0009`)
    Caractere de tabulação vertical (`U+000B`)
    Caractere de feed de formulário (`U+000C`)
    Caractere de retorno de carro (`U+000D`) seguido de caractere de alimentação de linha (`U+000A`) *caractere-de-nova-linha*
caractere-de-nova-linha:
    Caractere de retorno de carro (`U+000D`)
    Caractere de feed de linha (`U+000A`)
    Caractere de próxima linha (`U+0085`)
    Caractere separador de linha (`U+2028`)
    Caractere separador de parágrafo (`U+2029`)

Comentário

comentário:
    comentário-de-linha-única
    comentário-delimitado
comentário-de-linha-única:
    `//` *caracteres-de-comentário-de-linha-única*$_{opt}$
caracteres-de-comentário-de-linha-única:
    caracteres-de-comentário-de-linha-única caractere-de-comentário-de-linha-única$_{opt}$
caractere-de-comentário-de-linha-única:
    Qualquer caractere Unicode, exceto um comentário-delimitado por *caractere-de-nova-linha*
:
    `/*` *texto-de-comentário-delimitado*$_{opt}$ *asteriscos* `/`
texto-de-comentário-delimitado:
    seção-de-comentário-delimitado texto-de-comentário-delimitado$_{opt}$
seção-de-comentário-delimitado:
    `/`
    asteriscos$_{opt}$ *caracteres-diferentes-de-barra-e-asterisco*
asteriscos:
    `*` *asteriscos*$_{opt}$
caracteres-diferentes-de-barra-e-asterisco asteriscos:

Qualquer caractere Unicode, exceto `*` ou `/`

Tokens

token:
identificador
palavra-chave
literal
operador-ou-pontuador

Sequências de escape de caractere

sequência-de-escape-de-caracteres:
`#(` *lista-de-sequências-de-escape* `)`
lista-de-sequências-de-escape:
sequência-de-escape-única
lista-de-sequências-de-escape `,` *sequência-de-escape-única*
sequência-de-escape-única:
sequência-de-escape-unicode-longa
sequência-de-escape-unicode-curta
sequência-de-escape-de-caracteres-de-controle
escape-escape
sequência-de-escape-unicode-longa:
dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal
sequência-de-escape-unicode-curta:
dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal dígito-hexadecimal
sequência-de-escape-de-caracteres-de-controle:
caractere-de-controle
caractere-de-controle:
`cr`
`lf`
`tab`
escape-escape:
`#`

Literais

literal:
literal-lógico
literal-de-número
literal-de-texto
literal-nulo
literal-textual
literal-lógico:
`true`
`false`
literal-de-número:
literal-de-número-decimal
literal-de-número-hexadecimal
dígitos-decimais:
dígito-decimal dígitos-decimais$_{opt}$
dígito-decimal: um entre
`0 1 2 3 4 5 6 7 8 9`
literal-de-número-hexadecimal:

`0x` *dígitos-hexadecimais*

`0X` *dígitos-hexadecimais*

dígitos-hexadecimais:

dígito-hexadecimal dígitos-hexadecimais$_{opt}$

dígito-hexadecimal: um entre

`0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f`

literal-de-número-decimal:

dígitos-decimais `.` *dígitos-decimais parte-do-expoente*$_{opt}$

`.` *dígitos-decimais parte-do-expoente*$_{opt}$

dígitos-decimais parte-do-expoente$_{opt}$

parte-do-expoente:

`e` *sinal*$_{opt}$ *dígitos-decimais*

`E` *sinal*$_{opt}$ *dígitos-decimais*

sinal: um entre

`+ -`

literal-de-texto:

`"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

caracteres-de-literal-de-texto:

caractere-de-literal-de-texto caracteres-de-literal-de-texto$_{opt}$

caractere-de-literal-de-texto:

caractere-de-texto-único

sequência-de-escape-de-caracteres

sequência-de-escape-com-aspas-duplas

caractere-de-texto-único:

Qualquer caractere, exceto `"` (`U+0022`) ou `#` (`U+0023`) seguido por `(` (`U+0028`)

sequência-de-escape-com-aspas-duplas:

`""` (`U+0022`, `U+0022`)

literal-nulo:

`null`

literal-textual:

`#!"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

Identificadores

identificador:

identificador-comum

identificador-entre-aspas

identificador-comum:

identificador-disponível

identificador-disponível caractere-de-ponto identificador-comum

identificador-disponível:

Uma *palavra-chave-ou-identificador* que não é uma *palavra-chave*

palavra-chave-ou-identificador:

caractere-de-letra

caractere-de-sublinhado

caractere-identificador-de-início caracteres-identificadores-de-parte

caractere-identificador-de-início:

caractere-de-letra

caractere-de-sublinhado

caracteres-identificadores-de-parte:

caractere-identificador-de-parte caracteres-identificadores-de-parte$_{opt}$

caractere-identificador-de-parte:

caractere-de-letra

caractere-de-dígito-decimal
caractere-de-sublinhado
caractere-de-conexão
caractere-de-combinação
caractere-de-formatação

identificador-generalizado:
parte-do-identificador-generalizado
identificador-generalizado separado somente por espaços em branco (`U+0020`) *parte-do-identificador-generalizado*

parte-do-identificador-generalizado:
segmento-do-identificador-generalizado
caractere-do-dígito-decimal segmento-identificador-generalizado

segmento-identificador-generalizado:
palavra-chave-ou-identificador
palavra-chave-ou-identificador caractere-de-ponto palavra-chave-ou-identificador

caractere-de-ponto:
`.` (`U+002E`)

caractere-de-sublinhado:
`_` (`U+005F`)

caractere-de-letra:
Um caractere Unicode de uma das classes Lu, Ll, Lt, Lm, Lo ou Nl

caractere-de-combinação:
Um caractere Unicode de uma das classes Mn ou Mc

caractere-de-dígito-decimal:
Um caractere Unicode da classe Nd

caractere-de-conexão:
Um caractere Unicode da classe Pc

caractere-de-formatação:
Um caractere Unicode da classe Cf

identificador-entre-aspas:
`#"` *caracteres-de-literal-de-texto*$_{opt}$ `"`

Palavras-chave e identificadores predefinidos

Palavras-chave e identificadores predefinidos não podem ser redefinidos. Um identificador entre aspas pode ser usado para lidar com identificadores que, de outra forma, colidiriam com identificadores ou palavras-chave predefinidos.

palavra-chave: um entre
`and as each else error false if in is let meta not null or otherwise`
`section shared then true try type #binary #date #datetime`
`#datetimezone #duration #infinity #nan #sections #shared #table #time`

Operadores e pontuadores

operador-ou-pontuador: um entre
`, ; = < <= > >= <> + - * / & ( ) [ ] { } @ ? => .. ...`

Gramática sintática

Documentos

documento:
documento-de-seção
documento-de-expressão

Documentos da seção

documento-de-seção:

seção

seção:

atributos-literais$_{opt}$ `section` *nome-da-seção* `;` *membros-da-seção*$_{opt}$

nome-da-seção:

identificador

membros-da-seção:

membro-da-seção membro-da-seção$_{opt}$

membros-da-seção:

atributos-literais$_{opt}$ *compartilhado*$_{opt}$ *nome-do-membro-da-seção* `=` *expressão* `;`

nome-do-membro-da-sessão:

identificador

Documentos de expressão

Expressões

documento-de-expressão:

expressão

expressão:

expressão-or-lógica

expressão-each

expressão-de-função

expressão-let

expressão-if

expressão-de-geração-de-erros

expressão-de-tratamento-de-erros

Expressões lógicas

expressão-or-lógica:

expressão-and-lógica

expressão-and-lógica `or` *expressão-or-lógica*

expressão-and-lógica:

expressão-is

expressão-and-lógica `and` *expressão-is*

Expressão Is

expressão-is:

expressão-as

expressão-is `is` *tipo-primitivo-que-permite-valor-nulo*

tipo-primitivo-que-permite-valor-nulo:

`nullable`$_{opt}$ *tipo-primitivo*

Expressão As

expressão-as:

expressão-de-igualdade

expressão-as `as` *tipo-primitivo-que-permite-valor-nulo*

Expressão de igualdade

expressão-de-igualdade:

expressão-relacional

expressão-relacional `=` *expressão-de-igualdade*

expressão-relacional `<>` *expressão-de-igualdade*

Expressão relacional

expressão-relacional:

expressão-aditiva
expressão-aditiva `<` *expressão-relacional*
expressão-aditiva `>` *expressão-relacional*
expressão-aditiva `<=` *expressão-relacional*
expressão-aditiva `>=` *expressão-relacional*

Expressões aritméticas

expressão-aditiva:
expressão-multiplicativa
expressão-multiplicativa `+` *expressão-aditiva*
expressão-multiplicativa `-` *expressão-aditiva*
expressão-multiplicativa `&` _expressão-aditiva
expressão-multiplicativa:
expressão-de-metadados
expressão-de-metadados `*` *expressão-multiplicativa*
expressão-de-metadados `/` *expressão-multiplicativa*

Expressão de metadados

expressão-de-metadados:
expressão-unária
expressão-unária `meta` *expressão-unária*

Expressão unária

expressão-unária:
expressão-de-tipo
`+` *expressão-unária*
`-` *expressão-unária*
`not` *expressão-unária*

Expressão primária

expressão-primária:
expressão-literal
expressão-de-lista
expressão-de-registro
expressão-de-identificador
expressão-de-acesso-à-seção
expressão-entre-parênteses
expressão-de-acesso-ao-campo
expressão-de-acesso-ao-item
expressão-de-invocação
expressão-não-implementada

Expressão literal

expressão-literal:
literal

Expressão de identificador

expressão-de-identificador:
referência-de-identificador
referência-de-identificador:
referência-de-identificador-exclusiva
referência-de-identificador-inclusiva
referência-do-identificador-exclusivo:
identificador
referência-do-identificador-inclusivo:

`@` *identificador*

Expressão de acesso à seção

expressão-de-acesso-à-seção:
identificador `!` *identificador*

Expressão entre parênteses

expressão-entre-parênteses:
`(` *expressão* `)`

Expressão não implementada

expressão-não-implementada:
`...`

Expressão de invocação

expressão-de-invocação:
expressão-primária `(` *lista-de-argumentos*$_{opt}$ `)`
lista-de-argumentos:
expressão
expressão `,` *lista-de-argumentos*

Expressão de lista

expressão-de-lista:
`{` *lista-de-itens*$_{opt}$ `}`
lista-de-itens:
item
item `,` *lista-de-itens*
item:
expressão
expressão `..` *expressão*

Expressão de registro

expressão-de-registro:
`[` *lista-de-campos*$_{opt}$ `]`
lista-de-campos:
campo
campo `,` *lista-de-campos*
campo:
nome-do-campo `=` *expressão*
nome-do-campo:
identificador-generalizado
identificador-entre-aspas

Expressão de acesso ao item

expressão-de-acesso-a-item:
seleção-de-item
seleção-de-item-opcional
seleção-de-item:
expressão-primária `{` *seletor-de-item* `}`
seleção-de-item-opcional:
expressão-primária `{` *seletor-de-item* `}?`
seletor-de-item:
expressão

Expressões de acesso ao campo

expressão-de-acesso-ao-campo:

seleção-de-campo
seleção-de-campo-de-destino-implícita
projeção
projeção-de-destino-implícita

seleção-de-campo:
expressão-primária seletor-de-campo

seletor-de-campo:
seletor-de-campo-obrigatório
seletor-de-campo-opcional

seletor-de-campo-obrigatório:
`[` *nome-do-campo* `]`

seletor-de-campo-opcional:
`[` *nome-do-campo* `]?`

nome-do-campo:
identificador-generalizado
identificador-entre-aspas

seleção-de-campo-de-destino-implícita:
seletor-de-campo

projeção:
expressão-primária projeção-obrigatória
expressão-primária projeção-opcional

projeção-obrigatória:
`[` *lista-de-seletores-obrigatórios* `]`

projeção-opcional:
`[` *lista-de-seletores-obrigatórios* `]?`

lista-de-seletores-obrigatórios:
seletor-de-campo-obrigatório
seletor-de-campo-obrigatório `,` *lista-de-seletores-obrigatórios*

projeção-de-destino-implícita:
projeção-obrigatória
projeção-opcional

Expressão de função

expressão-de-função:
`(` *lista-de-parâmetros*$_{opt}$ `)` *tipo-de-retorno*$_{opt}$ `=>` *corpo-da-função*

corpo-da-função:
expressão

lista-de-parâmetros:
lista-de-parâmetros-fixos
lista-de-parâmetros-fixos `,` *lista-de-parâmetros-opcionais*
lista-de-parâmetros-opcionais

lista-de-parâmetros-fixos:
parâmetro
parâmetro `,` *lista-de-parâmetros-fixos*

parâmetro:
nome-do-parâmetro tipo-de-parâmetro$_{opt}$

nome-do-parâmetro:
identificador

tipo-de-parâmetro:
asserção

tipo-de-retorno:
asserção

asserção:
`as` *tipo-primitivo-que-permite-valor-nulo*
lista-de-parâmetros-opcionais:
parâmetro-opcional
parâmetro-opcional `,` *lista-de-parâmetros-opcionais*
parâmetro-opcional:
`optional` *parâmetro*

Expressão each

expressão-each:
`each` *corpo-da-expressão-each*
corpo-da-expressão-each:
corpo-da-função

Expressão let

expressão-let:
`let` *lista-de-variáveis* `in` *expressão*
lista-de-variáveis:
variável
variável `,` *lista-de-variáveis*
variável:
nome-da-variável `=` *expressão*
nome-da-variável:
identificador

Expressão if

expressão-if:
`if` *condição-if* `then` *expressão-verdadeira* `else` *expressão-falsa*
condição-if:
expressão
expressão-verdadeira:
expressão
expressão-falsa:
expressão

Expressão de tipo

expressão-de-tipo:
expressão-primária
`type` *tipo-primário*
tipo:
expressão-entre-parênteses
tipo-primário
tipo-primário:
tipo-primitivo
tipo-de-registro
tipo-de-lista
tipo-de-função
tipo-de-tabela
tipo-anulável
tipo-primitivo: um entre
`any anynonnull binary date datetime datetimezone duration function`
`list logical none null number record table text type`
tipo-de-registro:
`[` *marcador-de-registro-aberto* `]`

`[` *lista-de-especificação-de-campos*$_{opt}$ `]`

`[` *lista-de-especificações-de-campo* `,` *marcador-de-registro-aberto* `]`

lista-de-especificações-de-campo:

especificação-de-campo

especificação-de-campo `,` *lista-de-especificações-de-campo*

especificação-de-campo:

`optional`$_{opt}$ *nome-do-campo especificação-de-tipo-de-campo*$_{opt}$

especificação-de-tipo-de-campo:

`=` *tipo-de-campo*

tipo-de-campo:

tipo

marcador-de-registro-aberto:

`...`

tipo-de-lista:

`{` *tipo-de-item* `}`

tipo-de-item:

tipo

tipo-de-função:

`function (` *lista-de-especificações-de-parâmetroopt* `)` *tipo-retornado*

lista-de-especificações-de-parâmetro:

lista-de-especificações-de-parâmetro-obrigatórias

lista-de-especificações-de-parâmetro-obrigatórias `,` *lista-de-especificações-de-parâmetro-opcionais*

lista-de-especificações-de-parâmetro-opcionais

lista-de-especificações-de-parâmetro-obrigatórias:

especificação-de-parâmetro-obrigatória

especificação-de-parâmetro-obrigatória `,` *lista-de-especificações-de-parâmetro-obrigatórias*

especificação-de-parâmetro-obrigatória:

especificação-de-parâmetro

lista-de-especificações-de-parâmetro-opcionais:

especificação-de-parâmetro-opcional

especificação-de-parâmetro-opcional `,` *lista-de-especificações-de-parâmetro-opcionais*

especificação-de-parâmetro-opcional:

`optional` *especificação-de-parâmetro*

especificação-de-parâmetro:

nome-do-parâmetro tipo-de-parâmetro

tipo-de-tabela:

`table` *tipo-de-linha*

tipo-de-linha:

`[` *lista-de-especificações-de-campo* `]`

tipo-anulável:

`nullable` *tipo*

Expressão para geração de erros

expressão-para-geração-de-erros:

`error` expressão_

Expressão de tratamento de erro

expressão-de-tratamento-de-erro:

`try` *expressão-protegida cláusula-otherwise*$_{opt}$

expressão-protegida:

expressão

cláusula-otherwise:

`otherwise` *expressão-padrão*

expressão-padrão:

expressão

Atributos literais

atributos-literais:

literal-de-registro

literal-de-registro:

`[` *lista-de-campos-literais*$_{opt}$ `]`

lista-de-campos-literais:

campo-literal

campo-literal `,` *lista-de-campos-literais*

campo-literal:

nome-do-campo `=` *qualquer-literal*

literal-de-lista:

`{` *lista-de-itens-literais*$_{opt}$ `}`

lista-de-itens-de-literal:

qualquer-literal

qualquer-literal `,` *lista-de-itens-de-literal*

qualquer-literal:

literal-de-registro

literal-de-lista

literal-lógico

literal-de-número

literal-de-texto

literal-nulo

Tipos na linguagem de fórmula do Power Query M

08/05/2020 • 15 minutes to read

A linguagem de fórmula do Power Query M é uma linguagem de mashup de dados útil e expressiva. No entanto, ela tem algumas limitações. Por exemplo, não há nenhuma imposição forte do sistema de tipos. Em alguns casos, uma validação mais rigorosa é necessária. Felizmente, o M fornece uma biblioteca interna compatível com tipos para viabilizar uma validação mais forte.

Os desenvolvedores devem ter uma compreensão abrangente do sistema de tipos para que possam fazer isso com qualquer generalidade. E embora a especificação da linguagem do Power Query M explique bem o sistema de tipos, ela ainda deixa algumas surpresas. Por exemplo, a validação de instâncias de função requer uma forma de comparar tipos quanto à compatibilidade.

Ao explorar o sistema de tipos do M mais cuidadosamente, muitos desses problemas poderão ser esclarecidos e os desenvolvedores se capacitarão a criar as soluções de que precisam.

O conhecimento de cálculo de predicados e a teoria ingênua dos conjuntos devem ser adequados para a compreensão da notação usada.

ETAPAS PRELIMINARES

(1) $B := \{ true, false \}$
B é o conjunto típico de valores boolianos

(2) $N :=$ { identificadores M válidos }
N é o conjunto de todos os nomes válidos em M. Isso é definido em outro lugar.

(3) $P := \langle B, T \rangle$
P é o conjunto de parâmetros de função. Cada um é possivelmente opcional e tem um tipo. Os nomes de parâmetro são irrelevantes.

(4) $P^n := \bigcup_{0 \le i \le n} \langle i, P^i \rangle$
P^n é o conjunto de todas as sequências ordenadas de n parâmetros de função.

(5) $P^* := \bigcup_{0 \le i \le \infty} P^i$
P^* é o conjunto de todas as sequências possíveis de parâmetros de função, em ordem crescente, começando pelas de comprimento 0.

(6) $F := \langle B, N, T \rangle$
F é o conjunto de todos os campos de registro. Cada campo é possivelmente opcional, tem um nome e um tipo.

(7) $F^n := \prod_{0 \le i \le n} F$
F^n é o conjunto de todos os campos de registro.

(8) $F^* := \left(\bigcup_{0 \le i \le \infty} F^i \right) \setminus \{ F \mid \langle b_1, n_1, t_1 \rangle, \langle b_2, n_2, t_2 \rangle \in F \wedge n_1 = n_2 \}$
F^* é o conjunto de todos os conjuntos (de qualquer tamanho) de campos de registro, exceto pelos conjuntos em que mais de um campo tem o mesmo nome.

(9) $C := \langle N, T \rangle$
C é o conjunto de tipos de coluna para tabelas. Cada coluna tem um nome e um tipo.

(10) $C^n \subset \bigcup_{0 \le i \le n} \langle i, C \rangle$
C^n é o conjunto de todas as sequências de tipos de coluna n.

(11) $C^* := \left(\bigcup_{0 \le i \le \infty} C^i \right) \setminus \{ C^m \mid \square a, \square n_1, t_1 \square\square, \square b, \square n_2, t_2 \square\square \in C^m \wedge n_1 = n_2 \}$
C^* é o conjunto de todas as combinações (de qualquer tamanho) de tipos de coluna, exceto aquelas em que mais de uma coluna tem o mesmo nome.

TIPOS DE M

(12) $T_F := \square P, P^* \square$
Um tipo de função consiste em um tipo de retorno e uma lista ordenada de zero ou mais parâmetros de função.

(13) $T_L := [\![T]\!]$
Um tipo de lista é indicado por um determinado tipo (chamado de "tipo de item") encapsulado entre chaves. Já que as chaves são usadas na metalinguagem, os *colchetes* ⟦ ⟧ *são usados neste documento.*

(14) $T_R := \square B, F^* \square$
Um tipo de registro tem um sinalizador que indica se ele está "aberto" e tem zero ou mais campos de registro não ordenados.

(15) $T_R^o := \square true, F \square$

(16) $T_R^\bullet := \square false, F \square$
T_R^o e $T_R^\bullet$ são atalhos de notação para tipos de registro abertos e fechados, respectivamente.

(17) $T_T := C^*$
Um tipo de tabela é uma sequência ordenada de zero ou mais tipos de coluna, em que não há conflitos de nome.

(18) T_P := { any; none; null; logical; number; time; date; datetime; datetimezone; duration; text; binary; type; list; record; table; function; anynonnull }
Um tipo primitivo é um contido nesta lista de palavras-chave do M.

(19) $T_N := \{ t_n, u \in T \mid t_n = u + \text{null} \} = \text{nullable } t$
Qualquer tipo também pode ser marcado como permitindo valor nulo, usando a palavra-chave "nullable".

(20) $T := T_F \cup T_L \cup T_R \cup T_T \cup T_P \cup T_N$
O conjunto de todos os tipos do M é a união de todos estes seis conjuntos de tipos: Tipos de função, tipos de lista, tipos de registro, tipos de tabela, tipos primitivos e tipos que permitem valor nulo.

FUNÇÕES

É necessário definir uma função: $NonNullable : T \leftarrow T$
Essa função usa um tipo e retorna um tipo equivalente, exceto pelo fato de que esse segundo tipo não está em conformidade com o valor nulo.

IDENTIDADES

Algumas identidades são necessárias para definir alguns casos especiais e também podem ajudar a elucidar os casos acima.

(21) nullable any = any
(22) nullable anynonnull = any
(23) nullable null = null
(24) nullable none = null
(25) nullable nullable $t \in T$ = nullable t
(26) $NonNullable(\text{nullable } t \in T) = NonNullable(t)$
(27) $NonNullable(\text{any}) = \text{anynonnull}$

COMPATIBILIDADE DE TIPO

Conforme ressaltado em outra parte deste documento, um tipo M será compatível com outro tipo M se e somente se todos os valores que estiverem em conformidade com o primeiro tipo também estiverem em conformidade com o segundo tipo.

Temos definida aqui uma relação de compatibilidade que não depende de valores estarem em conformidade e é baseada nas propriedades dos tipos propriamente ditos. Espera-se que essa relação, conforme definido neste documento, seja completamente equivalente à definição semântica original.

A relação "é compatível com": $\leq : B \leftarrow T \times T$
Na seção abaixo, uma letra *t* minúscula sempre representará um tipo do M, um elemento de *T*.

Um Φ representará um subconjunto de F^* ou de C^*.

(28) $t \leq t$
Essa relação é reflexiva.

(29) $t_a \leq t_b \land t_b \leq t_c \rightarrow t_a \leq t_c$
Essa relação é transitiva.

(30) none $\leq t \leq$ any
Os tipos do M formam uma malha sobre essa relação; none *é a parte inferior e* any *é a parte superior.*

(31) $t_a , t_b \in T_N \land t_a \leq t_a \rightarrow NonNullable(t_a) \leq NonNullable(t_b)$
Se dois tipos forem compatíveis, os equivalentes NonNullable também serão compatíveis.

(32) null $\leq t \in T_N$
O tipo primitivo null *é compatível com todos os tipos que permitem valor nulo.*

(33) $t \notin T_N \leq$ anynonnull
Todos os tipos NonNullable são compatíveis com anynonnull.

(34) $NonNullable(t) \leq t$
Um tipo NonNullable é compatível com o equivalente anulável.

(35) $t \in T_F \rightarrow t \leq$ function
Todos os tipos de função são compatíveis com function.

(36) $t \in T_L \rightarrow t \leq$ list
Todos os tipos de lista são compatíveis com list.

(37) $t \in T_R \rightarrow t \leq$ record
Todos os tipos de registro são compatíveis com record.

(38) $t \in T_T \rightarrow t \leq$ table
Todos os tipos de tabela são compatíveis com table.

(39) $t_a \leq t_b \leftrightarrow [\![t_a]\!] \leq [\![t_b]\!]$
Um tipo de lista será compatível com outro tipo de lista se os tipos de item forem compatíveis e vice-versa.

(40) $t_a \in T_F = [\ p_a , p^* \ ], t_b \in T_F = [\ p_b , p^* \ ] \land p_a \leq p_b \rightarrow t_a \leq t_b$
Um tipo de função será compatível com outro tipo de função se os tipos de retorno forem compatíveis e as listas de parâmetros forem idênticas.

(41) $t_a \in T_R^o, t_b \in T_R^\bullet \rightarrow t_a \not\leq t_b$
Um tipo de registro aberto nunca é compatível com um tipo de registro fechado.

(42) $t_a \in T_R^\bullet = [false, \Phi], t_b \in T_R^o = [true, \Phi] \rightarrow t_a \leq t_b$
Um tipo de registro fechado é compatível com outro tipo de registro quase idêntico, exceto por ser aberto.

(43) $t_a \in T_R^o = [true, (\ \Phi , [true, n, \text{any}])], t_b \in T_R^o = [true, \Phi] \rightarrow t_a \leq t_b \land t_b \leq t_a$

Um campo opcional com o tipo any *pode ser ignorado ao comparar dois tipos de registros abertos.*

(44) $t_a \in T_R = [b, (\Phi, [\beta, n, u_a])], t_b \in T_R = [b, (\Phi, [\beta, n, u_b])] \wedge u_a \leq u_b \rightarrow t_a \leq t_b$
Dois tipos de registro que diferem somente por um campo serão compatíveis se o nome e a opcionalidade do campo forem idênticos e os tipos desse campo forem compatíveis.

(45) $t_a \in T_R = [b, (\Phi, [false, n, u])], t_b \in T_R = [b, (\Phi, [true, n, u])] \rightarrow t_a \leq t_b$
Um tipo de registro com um campo não opcional é compatível com um tipo de registro quase idêntico, exceto por ter esse campo como opcional.

(46) $t_a \in T_R^o = [true, (\Phi, [b, n, u])], t_b \in T_R^o = [true, \Phi] \rightarrow t_a \leq t_b$
Um tipo de registro aberto é compatível com outro tipo de registro aberto que tem um campo a menos.

(47) $t_a \in T_T = (\Phi, [i, [n, u_a]]), t_b \in T_T = (\Phi, [i, [n, u_b]]) \wedge u_a \leq u_b \rightarrow t_a \leq t_b$
Um tipo de tabela é compatível com um segundo tipo de tabela quase idêntico, exceto por ter uma coluna com um tipo diferente, quando os tipos dessa coluna são compatíveis.

REFERENCES

Microsoft Corporation (agosto de 2015)
Especificação de linguagem de fórmula do Microsoft Power Query para Excel [PDF]
Recuperado de https://msdn.microsoft.com/library/mt807488.aspx

Microsoft Corporation. (n. d.)
Referência de função do Power Query M [página da Web]
Recuperado de https://msdn.microsoft.com/library/mt779182.aspx

Expressões, valores e expressão Let

09/05/2020 • 10 minutes to read

Uma consulta de linguagem de fórmula Power Query M é composta pelas etapas da **expressão** da fórmula que criam uma consulta de mashup. Uma expressão de fórmula pode ser avaliada (computada), produzindo um valor. A expressão **let** encapsula um conjunto de valores a serem computados, nomes atribuídos, então usados em uma expressão subsequente que segue a instrução **in**. Por exemplo, uma expressão let pode conter uma variável **Fonte** igual ao valor de **Text.Proper()** e produz um valor de texto com o uso correto de maiúsculas e minúsculas.

Expressão Let

```
let
    Source = Text.Proper("hello world")
in
    Source
```

No exemplo acima, Text.Proper("olá, mundo") é avaliado como "Olá, Mundo".

As seções a seguir descrevem os tipos de valor no idioma.

Valor primitivo

Um valor **primitivo** é um valor de parte única, como um número, lógico, texto ou nulo. Um valor nulo pode ser usado para indicar a ausência de dados.

TIPO	VALOR DE EXEMPLO
Binário	00 00 00 02 // número de pontos (2)
Data	23/5/2015
DateTime	5/23/2015 12:00:00 AM
DateTimeZone	5/23/2015 12:00:00 AM -08:00
Duração	15:35:00
Logical	true e false
Nulo	nulo
Número	0, 1, -1, 1,5 e 2.3e-5
Texto	"abc"
Hora	12:34:12 PM

Valor da função

Uma **Função** é um valor que, quando invocado com argumentos, produz um novo valor. As funções são gravadas pela listagem dos **parâmetros** da função entre parênteses, seguidos pelo símbolo de ir para =>, seguido pela expressão que define a função. Por exemplo, para criar uma função chamada "MyFunction" que tem dois parâmetros e executa um cálculo no parâmetro1 e parâmetro2:

```
let
    MyFunction = (parameter1, parameter2) => (parameter1 + parameter2) / 2
in
    MyFunction

Calling the MyFunction() returns the result:

let
    Source = MyFunction(2, 4)
in
    Source
```

Esse código produz o valor de 3.

Valores de dados estruturados

A linguagem M dá suporte aos seguintes valores de dados estruturados:

- Lista
- Registro
- Tabela
- Exemplos de dados estruturados adicionais

> **NOTE**
> Os dados estruturados podem conter qualquer valor de M. Para ver alguns exemplos, confira Exemplos de dados estruturados adicionais.

Lista

Uma lista é uma sequência ordenada com base em zero de valores entre chaves { }. As chave { } também são usadas para recuperar um item de uma lista por posição de índice. Confira [List value](#_List_value).

> **NOTE**
> O Power Query M dá suporte a um tamanho de lista infinito, mas se uma lista for gravada como um literal, ela terá um comprimento fixo. Por exemplo, {1, 2, 3} tem um comprimento fixo de 3.

A seguir estão alguns exemplos de **Lista**.

VALOR	TIPO
{123, true, "A"}	Lista que contém um número, um lógico e um texto.
{1, 2, 3}	Lista de números

VALOR	TIPO

{
{1, 2, 3},
{4, 5, 6}
}
	Lista da lista de números

{
[CustomerID = 1, Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"]
}
	Lista de Registros

{123, true, "A"}{0}	Obtenha o valor do primeiro item em uma lista. Essa expressão retorna o valor 123.
{ {1, 2, 3}, {4, 5, 6} }{0}{1}	Obtenha o valor do segundo item do primeiro elemento List. Essa expressão retorna o valor 2.

Registro

Um **Registro** é um conjunto de campos. Um **campo** é um par de nome/valor em que o nome é um valor de texto exclusivo dentro do registro do campo. A sintaxe para valores de registro permite que os nomes sejam gravados sem aspas, uma forma também conhecida como **identificadores**. Um identificador pode ter as duas formas a seguir:

- identifier_name como OrderID.
- #"nome do identificador", como #"A data de hoje é: ".

A seguir está um registro que contém os campos chamados "OrderID", "CustomerID", "Item" e "Price" com os valores 1, 1, "Vara de pescar" e 100,00. Os caracteres de colchete [] denotam o início e o fim de uma expressão de registro e são usados para obter um valor de campo de um registro. Os exemplos a seguir mostram um registro e como obter o valor do campo item.

Aqui está um exemplo de registro:

```
let Source =
        [
              OrderID = 1,
              CustomerID = 1,
              Item = "Fishing rod",
              Price = 100.00
        ]
in Source
```

Para obter o valor de um Item, use colchetes como Source[item]:

```
let Source =
    [
          OrderID = 1,
          CustomerID = 1,
          Item = "Fishing rod",
          Price = 100.00
    ]
in Source[Item] //equals "Fishing rod"
```

Tabela

Uma **Tabela** é um conjunto de valores organizados em colunas e linhas nomeadas. O tipo de coluna pode ser implícito ou explícito. Você pode usar #table para criar uma lista de nomes de coluna e lista de linhas. Uma **Tabela** de valores é uma Lista em uma **Lista**. Os caracteres de chave {} também são usados para recuperar uma linha de uma **Tabela** por posição de índice (confira Exemplo 3 – Obter uma linha de uma tabela pela posição de índice).

Exemplo 1 – Criar uma tabela com tipos de coluna implícitos

```
let
  Source = #table(
    {"OrderID", "CustomerID", "Item", "Price"},
      {
          {1, 1, "Fishing rod", 100.00},
          {2, 1, "1 lb. worms", 5.00}
      })
in
    Source
```

Exemplo 2 – Criar uma tabela com tipos de coluna explícitos

```
let
    Source = #table(
    type table [OrderID = number, CustomerID = number, Item = text, Price = number],
        {
                {1, 1, "Fishing rod", 100.00},
             {2, 1, "1 lb. worms", 5.00}
        }
    )
in
    Source
```

Os dois exemplos acima criam uma tabela com a seguinte forma:

ORDERID	CUSTOMERID	ITEM	PREÇO
1	1	Vara de pescar	100,00
2	1	1 lb de minhocas	5,00

Exemplo 3 – Obter uma linha de uma tabela por posição de índice

```
let
    Source = #table(
    type table [OrderID = number, CustomerID = number, Item = text, Price = number],
        {
              {1, 1, "Fishing rod", 100.00},
              {2, 1, "1 lb. worms", 5.00}
         }
    )
in
    Source{1}
```

Essa expressão retorna o registro a seguir:

OrderID	2
CustomerID	1

Item	1 lb de minhocas
Preço	5

Exemplos de dados estruturados adicionais

Os dados estruturados podem conter qualquer valor de M. Aqui estão alguns exemplos:

Exemplo 1 – Lista com valores [Primitive](#_Primitive_value_1), [Function](#_Function_value) e [Record](#_Record_value)

```
let
    Source =
{
   1,
   "Bob",
   DateTime.ToText(DateTime.LocalNow(), "yyyy-MM-dd"),
   [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0]
}
in
    Source
```

Avaliar essa expressão pode ser visualizado como:

A List containing a Record	
1	
"Bob"	
2015-05-22	
OrderID	1
CustomerID	1
Item	"Fishing rod"
Price	100.0

Exemplo 2 – Registro que contém valores Primitivos e Registros aninhados

```
let
    Source = [CustomerID = 1, Name = "Bob", Phone = "123-4567", Orders =
        {
              [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
            [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0]
        }]
in
    Source
```

Avaliar essa expressão pode ser visualizado como:

A record containing a List of Records	
CustomerID	1
Name	"Bob"
Phone	"123-4567"
Orders	OrderID: 1 CustomerID: 1 Item: "Fishing rod" Price: 100.0
	OrderID: 2 CustomerID: 1 Item: "1 lb. worms" Price: 5.0

> **NOTE**
>
> Embora muitos valores possam ser escritos literalmente como uma expressão, um valor não é uma expressão. Por exemplo, a expressão 1 é avaliada como o valor 1; a expressão 1+1 é avaliada como o valor 2. Essa distinção é sutil, mas importante. As expressões são receitas para avaliação; os valores são os resultados da avaliação.

Expressão If

A expressão **if** seleciona entre duas expressões com base em uma condição lógica. Por exemplo:

```
if 2 > 1 then
    2 + 2
else
    1 + 1
```

A primeira expressão (2 + 2) será selecionada se a expressão lógica (2 > 1) for verdadeira e a segunda expressão (1 + 1) será selecionada se for false. A expressão selecionada (nesse caso, 2 + 2) é avaliada e torna-se o resultado da expressão **if** (4).

Comentários

09/05/2020 • 2 minutes to read

Você pode adicionar comentários ao seu código com comentários de linha única `//` ou de várias linhas que começam com `/*` e terminam com `*/`.

Exemplo – comentário de linha única

```
let
   //Convert to proper case.
    Source = Text.Proper("hello world")
in
    Source
```

Exemplo – comentário de várias linhas

```
/* Capitalize each word in the Item column in the Orders table. Text.Proper
is evaluated for each Item in each table row. */
let
    Orders = Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "fishing rod", Price = 100.0],
          [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
          [OrderID = 3, CustomerID = 2, Item = "fishing net", Price = 25.0]}),
        #"Capitalized Each Word" = Table.TransformColumns(Orders, {"Item", Text.Proper})
in
        #"Capitalized Each Word"
```

Modelo de avaliação

09/05/2020 • 4 minutes to read

O modelo de avaliação da linguagem de fórmulas Power Query M é modelado após o modelo de avaliação normalmente encontrado em planilhas, em que a ordem dos cálculos pode ser determinada com base nas dependências entre as fórmulas nas células.
Se você tiver escrito fórmulas em uma planilha como o Excel, poderá reconhecer que as fórmulas à esquerda resultarão nos valores à direita quando calculadas:

	A
1	=A2 * 2
2	=A3 + 1
3	1

	A
1	4
2	2
3	1

Em M, uma expressão pode referenciar expressões anteriores por nome e o processo de avaliação determinará automaticamente a ordem na qual as expressões referenciadas serão calculadas.

Vamos usar um registro para produzir uma expressão que seja equivalente ao exemplo da planilha acima. Ao inicializar o valor de um campo, você faz referência a outros campos dentro do registro pelo nome do campo, da seguinte maneira:

```
[
    A1 = A2 * 2,
    A2 = A3 + 1,
    A3 = 1
]
```

A expressão acima é avaliada como o seguinte registro:

```
[
    A1 = 4,
    A2 = 2,
    A3 = 1
]
```

Os registros podem estar contidos ou **aninhados** em outros registros. É possível usar o **operador de pesquisa** ([ ]) para acessar os campos de um registro pelo nome. Por exemplo, o registro a seguir tem um campo chamado "Sales" com um registro e um campo chamado "Total" que acessa os campos "FirstHalf" e "SecondHalf" do registro "Sales":

```
[
    Sales = [ FirstHalf = 1000, SecondHalf = 1100 ],
    Total = Sales[FirstHalf] + Sales[SecondHalf]
]
```

A expressão acima é avaliada como o seguinte registro:

```
[
    Sales = [ FirstHalf = 1000, SecondHalf = 1100 ],
    Total = 2100
]
```

Use o **operador de índice posicional** ({ }) para acessar um item em uma lista por seu índice numérico. Os valores dentro de uma lista são referenciados usando um índice de base zero desde o início da lista. Por exemplo, os índices 0 e 1 são usados para referenciar o primeiro e o segundo itens na lista abaixo:

```
[
    Sales =
        {
            [
                Year = 2007,
                FirstHalf = 1000,
                SecondHalf = 1100,
                Total = FirstHalf + SecondHalf // equals 2100
            ],
            [
                Year = 2008,
                FirstHalf = 1200,
                SecondHalf = 1300,
                Total = FirstHalf + SecondHalf // equals 2500
            ]
        },
    #"Total Sales" = Sales{0}[Total] + Sales{1}[Total] // equals 4600
]
```

Avaliação lenta e rápida

As expressões de membro **List**, **Record** e **Table**, bem como as expressões **let** (confira Expressões, valores e expressão Let), são avaliadas por meio da **avaliação lenta**, ou seja, quando necessário. Todas as outras expressões são avaliadas usando a **avaliação rápida**, ou seja, imediatamente, quando encontradas durante o processo de avaliação. Uma boa maneira de pensar sobre nisso é lembrar de que a avaliação de uma expressão de lista ou registro retornará um valor de lista ou registro que sabe como seus itens de lista ou campos de registro precisam ser computados, quando solicitado (por operadores de pesquisa ou índice).

Operadores

12/05/2020 • 5 minutes to read

A linguagem de fórmula Power Query M inclui um conjunto de operadores que podem ser usados em uma expressão. Os **operadores** são aplicados a **operandos** para formar expressões simbólicas. Por exemplo, na expressão 1 + 2, os números 1 e 2 são operandos e o operador é o operador de adição (+).

O significado de um operador pode variar dependendo do tipo de valores do operando. A linguagem tem os seguintes operadores:

Operador de adição (+)

EXPRESSÃO	IGUAL A
1 + 2	Adição numérica: 3
#time(12,23,0) + #duration(0,0,2,0)	Aritmética de tempo: #time(12,25,0)

Operador de combinação (&)

FUNÇÃO	IGUAL A
"A" & "BC"	Concatenação de texto: "ABC"
{1} & {2, 3}	Concatenação de lista: {1, 2, 3}
[a = 1] & [b = 2]	Mesclagem de registro: [a = 1, b = 2]

Lista de operadores M

Operadores comuns que se aplicam a nulo, lógico, número, hora, data, datetime, datetimezone, duração, texto, binário)

OPERADOR	DESCRIÇÃO
>	Maior que
>=	Maior ou igual
<	Menor que
<=	Inferior ou igual
=	Igual
<>	Diferente

Operadores lógicos (além de **Operadores comuns**)

OPERADOR	DESCRIÇÃO
ou	OR lógico condicional
e	AND lógico condicional
não	NOT lógico

Operadores numéricos (além de Operadores comuns)

OPERADOR	DESCRIÇÃO
+	Soma
-	Diferença
*	Produto
/	Quociente
+x	Adição de unário
-x	Negação

Operadores de texto (além de Operadores comuns)

OPERADOR	DESCRIÇÃO
&	Concatenação

Operadores de lista, registro, tabela

OPERADOR	DESCRIÇÃO
=	Igual
<>	Diferente
&	Concatenação

Operador de pesquisa de registro

OPERADOR	DESCRIÇÃO
[]	Acesse os campos de um registro por nome.

Operador do indexador de lista

OPERADOR	DESCRIÇÃO
{}	Acesse um item em uma lista pelo índice numérico baseado em zero.

Compatibilidade de tipos e operadores de asserção

OPERADOR	DESCRIÇÃO
é	A expressão x é y retornará true se o tipo de x for compatível com y e retornará false se o tipo de x não for compatível com y.
como	A expressão x como y declara que o valor x é compatível com y, conforme o operador is.

Operadores de data

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
x + y	hora	duração	Deslocamento de Date por duração
x + y	duração	hora	Deslocamento de Date por duração
x - y	hora	duração	Deslocamento de data por duração negada
x - y	time	hora	Duração entre datas
x & y	data	hora	Data/hora mescladas

Operadores datetime

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
x + y	datetime	duração	Deslocamento de data/hora por duração
x + y	duração	datetime	Deslocamento de data/hora por duração
x - y	datetime	duração	Deslocamento de Datetime por duração negada
x - y	datetime	datetime	Duração entre datetimes

Operadores datetimezone

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
x + y	datetimezone	duração	Deslocamento DateTimeZone por duração
x + y	duração	datetimezone	Deslocamento DateTimeZone por duração

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
x - y	datetimezone	duração	Deslocamento de datetimezone por duração negada
x - y	datetimezone	datetimezone	Duração entre datetimezones

Operadores de duração

OPERADOR	OPERANDO ESQUERDO	OPERANDO DIREITO	SIGNIFICADO
x + y	datetime	duração	Deslocamento de data/hora por duração
x + y	duração	datetime	Deslocamento de data/hora por duração
x + y	duração	duração	Soma das durações
x - y	datetime	duração	Deslocamento de Datetime por duração negada
x - y	datetime	datetime	Duração entre datetimes
x - y	duração	duração	Diferença de durações
x * y	duração	número	N vezes uma duração
x * y	número	duração	N vezes uma duração
x / y	duração	número	Fração de uma duração

> **NOTE**
> Um operador pode não ter suporte para todas as combinações de valores. As expressões que, quando avaliadas, encontram condições de operador indefinidas são avaliadas como erros. Para obter mais informações sobre erros em M, confira Erros

Exemplo de erro:

FUNÇÃO	IGUAL A
1 + "2"	Erro: não há suporte para adicionar número e texto

Conversão de tipo

09/05/2020 • 3 minutes to read

A linguagem de fórmula do Power Query M tem fórmulas para converter entre tipos. A seguir está um resumo de fórmulas de conversão em M.

Número

CONVERSÃO DE TIPO	DESCRIÇÃO
Number.FromText(texto como texto) como número	Retorna um valor numérico de um valor de texto.
Número.ToText(número como número) como texto	Retorna um valor de texto de um valor numérico.
Number.From(value as any) as number	Retorna um valor numérico de um valor.
Int32.From(valor como qualquer) como número	Retorna um valor numérico de inteiro de 32 bits do valor especificado.
Int64.From(valor como qualquer) como número	Retorna um valor numérico de inteiro de 64 bits do valor especificado.
Single.From(valor como qualquer) como número	Retorna um valor numérico único do valor especificado.
Double.From(valor como qualquer) como número	Retorna um valor de número duplo do valor especificado.
Decimal.From(valor como qualquer) como número	Retorna um valor numérico Decimal do valor especificado.
Currency.From(valor como qualquer) como número	Retorna um valor de número Currency do valor especificado.

Texto

CONVERSÃO DE TIPO	DESCRIÇÃO
Text.From(valor como qualquer) como texto	Retorna a representação de texto de um valor de número, data, time, datetime, datetimezone, lógico, duração ou binário.

Logical

CONVERSÃO DE TIPO	DESCRIÇÃO
Logical.FromText(texto como texto) como lógico	Retorna um valor lógico de true ou false de um valor de texto.
Logical.ToText(lógico como lógico) como texto	Retorna um valor de texto de um valor lógico.
Logical.From(valor como qualquer) como lógico	Retorna um valor lógico de um valor.

Date, Time, DateTime e DateTimeZone

CONVERSÃO DE TIPO	DESCRIÇÃO
.FromText(texto como texto) como date, time, datetime ou datetimezone	Retorna um valor de date, time, datetime ou datetimezone de um conjunto de formatos de data e valor de cultura.
.ToText(date, time, dateTime ou dateTimeZone como date, time, datetime ou datetimezone) como texto	Retorna um valor de texto de um valor date, time, datetime ou datetimezone.
.From(value as any)	Retorna um valor date, time, datetime ou datetimezone de um valor.
.ToRecord(date, time, dateTime ou dateTimeZone como date, time, datetime ou datetimezone)	Retorna um registro que contém partes de um valordate, time, datetime ou datetimezone.

Metadados

09/05/2020 • 2 minutes to read

Os **metadados** são informações sobre um valor que está associado a um valor. Os **metadados** são representados como um valor de registro, chamado de registro de metadados. Os campos de um **registro de metadados** podem ser usados para armazenar os metadados de um valor. Cada valor tem um registro de metadados. Se o valor do registro de metadados não tiver sido especificado, o registro de metadados estará vazio (não terá nenhum campo). A associação de um registro de metadados com um valor não altera o comportamento do valor em avaliações, exceto aqueles que inspecionam explicitamente os registros de metadados.

Um valor de registro de metadados é associado a um valor x usando o valor de sintaxe meta [record]. Por exemplo, o seguinte associa um registro de metadados aos campos de classificação e marcações com o valor de texto "Mozart":

```
"Mozart" meta [ Rating = 5,
Tags = {"Classical"} ]
```

Um registro de metadados pode ser acessado para um valor usando a função `Value.Metadata`. No exemplo a seguir, a expressão no campo ComposerRating acessa o registro de metadados do valor no campo do compositor e, em seguida, acessa o campo de classificação do registro de metadados.

```
[
    Composer = "Mozart" meta [ Rating = 5, Tags = {"Classical"} ],
    ComposerRating = Value.Metadata(Composer)[Rating]   // 5
]
```

Os registros de metadados não são preservados quando um valor é usado com um operador ou função que cria um novo valor. Por exemplo, se dois valores de texto forem concatenados usando o operador &, os metadados do valor de texto resultante serão um registro vazio [].

As funções de biblioteca padrão `Value.RemoveMetadata` e `Value.ReplaceMetadata` podem ser usadas para remover todos os metadados de um valor e para substituir os metadados de um valor.

Errors

09/05/2020 • 2 minutes to read

Um **erro** na linguagem de fórmulas do Power Query M é uma indicação de que o processo de avaliação de uma expressão não pôde produzir um valor. Os erros são gerados por operadores e funções que encontraram condições de **erros** ou usaram a expressão de **erro**. Os erros são tratados usando a expressão **try**. Quando um erro é gerado, é especificado um valor que pode ser usado para indicar por que o erro ocorreu.

Expressão try

Uma expressão try converte valores e erros em um valor de registro que indica se a expressão try tratou um erro ou não, bem como o valor apropriado ou o registro de erro que ela extraiu ao tratar o erro. Por exemplo, considere a seguinte expressão que gera um erro e o trata imediatamente:

```
try error "negative unit count"
```

Essa expressão avalia o seguinte valor de registro aninhado, explicando as pesquisas de campo `[HasError], [Error]` e `[Message]` no exemplo de preço unitário antes.

Registro de erro

```
[
    HasError = true,
    Error =
        [
            Reason = "Expression.Error",
            Message = "negative unit count",
            Detail = null
        ]
]
```

Um caso comum é substituir erros por valores padrão. A expressão try também pode ser usada com uma cláusula opcional para obter apenas isso em um formato compacto:

```
try error "negative unit count" otherwise 42
// equals 42
```

Exemplo de erro

```
let Sales =
        [
        ProductName = "Fishing rod",
            Revenue = 2000,
            Units = 1000,
            UnitPrice = if Units = 0 then error "No Units"
                    else Revenue / Units
        ],

    //Get UnitPrice from Sales record
        textUnitPrice = try Number.ToText(Sales[UnitPrice]),
    Label = "Unit Price: " &
        (if textUnitPrice[HasError] then textUnitPrice[Error][Message]
        //Continue expression flow
            else textUnitPrice[Value])
in
    Label
```

O exemplo acima acessa o campo `Sales[UnitPrice]` e formata o valor que produz o resultado:

```
"Unit Price: 2"
```

Se o campo "Unidades" tivesse sido zero, o campo `UnitPrice` teria gerado um erro que teria sido tratado pela expressão try. O valor resultante teria sido:

```
"No Units"
```

Referência de função do Power Query M

09/05/2020 • 2 minutes to read

A referência de função do Power Query M inclui artigos para cada uma das mais de 700 funções. Os artigos de referência que você vê aqui em docs.microsoft.com são gerados automaticamente na ajuda no produto. Para saber mais sobre as funções e como elas funcionam em uma expressão, confira Entender as funções do Power Query M.

Funções por categoria

- Como acessar funções de dados
- Funções binárias
- Funções de combinação
- Funções de comparação
- Funções de data
- Funções de DateTime
- Funções de DateTimeZone
- Funções de duração
- Tratamento de erro
- Funções de expressão
- Valores de função
- Funções de lista
- Funções de linhas
- Funções lógicas
- Funções numéricas
- Funções de registro
- Funções de substituto
- Funções de divisor
- Funções de tabela
- Funções de texto
- Funções de hora
- Funções de tipo
- Funções de URI
- Funções de valor

Como entender as funções do Power Query M

12/05/2020 • 3 minutes to read

Na linguagem de fórmula Power Query M, uma **função** é um mapeamento de um conjunto de valores de entrada para um único valor de saída. Uma função é gravada primeiro nomeando os parâmetros de função e, em seguida, fornecendo uma expressão para calcular o resultado da função. O corpo da função segue o símbolo de ir para (=>). Opcionalmente, as informações de tipo podem ser incluídas em parâmetros e o valor retornado da função. Uma função é definida e invocada no corpo de uma instrução **let**. Os parâmetros e/ou o valor retornado podem ser implícitos ou explícitos. Os parâmetros e/ou o valor retornado implícitos são do tipo **qualquer**. O tipo **qualquer** é semelhante a um tipo de objeto em outras linguagens. Todos os tipos em M derivam do tipo **qualquer**.

Uma **função** é um valor como um número ou um valor de texto e pode ser incluída em linha, assim como qualquer outra expressão. O exemplo a seguir mostra uma função que é o valor de uma variável Add, que é invocada ou executada de várias outras variáveis. Quando uma função é chamada, um conjunto de valores é especificado, que são substituídos logicamente pelo conjunto necessário de valores de entrada dentro da expressão do corpo da função.

Exemplo – parâmetros explícitos e valor retornado

```
let
    AddOne = (x as number) as number => x + 1,
    //additional expression steps
    CalcAddOne = AddOne(5)
in
    CalcAddOne
```

Exemplo – parâmetros implícitos e valor retornado

```
let
    Add = (x, y) => x + y,
    AddResults =
        [
            OnePlusOne = Add(1, 1),     // equals 2
            OnePlusTwo = Add(1, 2)      // equals 3
    ]
in
    AddResults
```

Localizar o primeiro elemento de uma lista maior que 5 ou nulo, caso contrário

```
let
    FirstGreaterThan5 = (list) =>
        let
            GreaterThan5 = List.Select(list, (n) => n> 5),
            First = List.First(GreaterThan5)
        in
            First,
    Results =
    [
            Found    = FirstGreaterThan5({3,7,9}),  // equals 7
            NotFound = FirstGreaterThan5({1,3,4})   // equals null
    ]
in
    Results
```

As funções podem ser usadas recursivamente. Para referenciar recursivamente a função, prefixe o identificador com @.

```
let
    fact = (num) => if num = 0 then 1 else num * @fact (num-1)
in
    fact(5) // equals 120
```

Palavra-chave each

A palavra-chave **each** é usada para criar facilmente funções simples. "each..." é uma simplificação sintática para uma assinatura de função que usa o _ parâmetro "(_) => ..."

Each é útil quando combinada ao operador lookup, que é aplicado por padrão a _
Por exemplo, each [CustomerID] é o mesmo que each _[CustomerID], que é o mesmo que (_) => _[CustomerID]

Exemplo – usando each um no filtro de linha de tabela

```
Table.SelectRows(
      Table.FromRecords({
            [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
            [CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
            [CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
            [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
      }),
      each [CustomerID] = 2
)[Name]

// equals "Jim"
```

Como acessar funções de dados

30/09/2020 • 17 minutes to read

Como acessar dados

Estas funções acessam dados e retornam valores de tabela. A maioria dessas funções retorna um valor de tabela chamado **tabela de navegação**. As tabelas de navegação são usadas principalmente pela interface do usuário do Power Query para proporcionar uma experiência de navegação sobre os conjuntos de dados hierárquicos potencialmente grandes retornados.

FUNÇÃO	DESCRIÇÃO
AccessControlEntry.ConditionToIdentities	Retorna uma lista de identidades que a condição aceitará.
AccessControlKind.Allow	O acesso é permitido.
AccessControlKind.Deny	O acesso é negado.
Access.Database	Retorna uma representação estrutural de um banco de dados do Microsoft Access.
ActiveDirectory.Domains	Retorna uma lista de domínios do Active Directory na mesma floresta do domínio especificado ou do domínio do computador atual, caso nenhuma tenha sido especificada.
AdobeAnalytics.Cubes	Retorna os conjuntos de relatórios ao Adobe Analytics.
AdoDotNet.DataSource	Retorna a coleção de esquema para uma fonte de dados do ADO.NET.
AdoDotNet.Query	Retorna a coleção de esquema para uma fonte de dados do ADO.NET.
AnalysisServices.Database	Retorna uma tabela de cubos multidimensionais ou modelos de tabela do banco de dados do Analysis Services.
AnalysisServices.Databases	Retorna os bancos de dados do Analysis Services em um host especificado.
AzureStorage.BlobContents	Retorna o conteúdo do blob especificado de um cofre de armazenamento do Azure.
AzureStorage.Blobs	Retorna uma tabela de navegação contendo todos os contêineres encontrados na conta de Armazenamento do Azure. Cada linha tem o nome do contêiner e um link para os blobs de contêiner.
AzureStorage.DataLake	Retorna uma tabela navegacional contendo os documentos localizados no contêiner especificado e nas suas subpastas do Azure Data Lake Storage.

FUNÇÃO	DESCRIÇÃO
AzureStorage.DataLakeContents	Retorna o conteúdo do arquivo especificado de um sistema de arquivos do Azure Data Lake Storage.
AzureStorage.Tables	Retorna uma tabela de navegação que contém uma linha para cada tabela encontrado na URL da conta de um cofre de armazenamento do Azure. Cada linha contém um link para a tabela do Azure.
Cdm.Contents	Esta função não está disponível porque requer o .NET 4.5.
Csv.Document	Retorna o conteúdo de um documento CSV como uma tabela usando a codificação especificada.
CsvStyle.QuoteAfterDelimiter	As aspas em um campo são significativas apenas imediatamente depois do delimitador.
CsvStyle.QuoteAlways	As aspas em um campo são sempre significativas, independentemente do local em que aparecerem.
Cube.AddAndExpandDimensionColumn	Mescla a tabela de dimensão especificada, dimensionSelector, no contexto de filtro do cubo e altera a granularidade dimensional expandindo o conjunto especificado, attributeNames, de atributos de dimensão.
Cube.AddMeasureColumn	Adiciona uma coluna com o nome de coluna ao cubo que contém os resultados da medida measureSelector aplicada ao contexto de cada linha.
Cube.ApplyParameter	Retorna um cubo depois de aplicar o parâmetro com argumentos ao cubo.
Cube.AttributeMemberId	Retorna o identificador de membro exclusivo de um valor da propriedade do membro.
Cube.AttributeMemberProperty	Retorna a propriedade `propertyName` do atributo de dimensão `attribute`.
Cube.CollapseAndRemoveColumns	Altera a granularidade dimensional do contexto de filtro do cubo, recolhendo os atributos mapeados para as colunas especificadas, columnNames.
Cube.Dimensions	Retorna a tabela que contém o conjunto de dimensões disponíveis dentro do cubo.
Cube.DisplayFolders	Retorna uma árvore de tabelas aninhada que representa a hierarquia da pasta de exibição dos objetos (por exemplo, dimensões e medidas) disponíveis para uso no cubo.
Cube.MeasureProperties	Retorna uma tabela contendo o conjunto de propriedades disponíveis para medidas que são expandidas no cubo.
Cube.MeasureProperty	Retorna a propriedade de uma medida.

FUNÇÃO	DESCRIÇÃO
Cube.Measures	Retorna a tabela que contém o conjunto de medidas disponíveis no cubo.
Cube.Parameters	Retorna uma tabela contendo o conjunto de parâmetros que pode ser aplicado ao cubo.
Cube.Properties	Retorna uma tabela contendo o conjunto de propriedades disponíveis para dimensões expandidas no cubo.
Cube.PropertyKey	Retorna a chave da propriedade `property`.
Cube.ReplaceDimensions	
Cube.Transform	Aplica as funções de cubo da lista, transformações, no cubo.
DB2.Database	Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados Db2.
Essbase.Cubes	Retorna os cubos em uma instância do Essbase agrupados por servidor Essbase.
Excel.CurrentWorkbook	Retorna as tabelas na pasta de trabalho do Excel atual.
Excel.Workbook	Retorna uma tabela representando planilhas na pasta de trabalho específica do Excel.
Exchange.Contents	Retorna um sumário de uma conta do Microsoft Exchange.
File.Contents	Retorna o conteúdo binário do arquivo localizado em um caminho.
Folder.Contents	Retorna uma tabela que contém as propriedades e o conteúdo dos arquivos e das pastas encontrados no caminho.
Folder.Files	Retorna uma tabela que contém uma linha para cada arquivo encontrado em um caminho de pasta e subpastas. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.
GoogleAnalytics.Accounts	Retorna as contas do Google Analytics para a credencial atual.
Hdfs.Contents	Retorna uma tabela que contém uma linha para cada pasta e arquivo encontrado na URL da pasta, {0}, de um sistema de arquivos do Hadoop. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.
Hdfs.Files	Retorna uma tabela que contém uma linha para cada arquivo encontrado na URL da pasta, {0}, e subpastas de um sistema de arquivos do Hadoop. Cada linha contém propriedades do arquivo e um link para o conteúdo.
HdInsight.Containers	Retorna uma tabela de navegação contendo todos os contêineres encontrados na conta do HDInsight. Cada linha tem o nome do contêiner e a tabela contendo seus arquivos.

FUNÇÃO	DESCRIÇÃO
HdInsight.Contents	Retorna uma tabela de navegação contendo todos os contêineres encontrados na conta do HDInsight. Cada linha tem o nome do contêiner e a tabela contendo seus arquivos.
HdInsight.Files	Retorna uma tabela que contém uma linha para cada pasta e arquivo encontrado na URL do contêiner e as subpastas de uma conta do HDInsight. Cada linha contém propriedades do arquivo/pasta e um link para o conteúdo.
Html.Table	Retorna uma tabela com os resultados da execução dos seletores de CSS especificados em relação ao html fornecido.
Identity.From	Cria uma identidade.
Identity.IsMemberOf	Determina se uma identidade é um membro de uma coleção de identidades.
IdentityProvider.Default	O provedor de identidade padrão do host atual.
Informix.Database	Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados do Informix no servidor `server` na instância de banco de dados chamada `database`.
Json.Document	Retorna o conteúdo de um documento JSON. O conteúdo pode ser passado diretamente para a função como texto ou pode ser o valor binário retornado por uma função como File.Contents.
Json.FromValue	Produz uma representação JSON de um valor especificado com uma codificação de texto especificada por codificação.
MySQL.Database	Retorna uma tabela contendo dados relacionados às tabelas no banco de dados MySQL especificado.
OData.Feed	Retorna uma tabela de feeds OData oferecidos por um OData serviceUri.
ODataOmitValues.Nulls	Permite que o serviço OData omita valores nulos.
Odbc.DataSource	Retorna uma tabela de exibições e tabelas SQL da fonte de dados ODBC especificada pela cadeia de conexão `connectionString`.
Odbc.InferOptions	Retorna o resultado de uma tentativa de inferir as funcionalidades do SQL de um driver ODBC.
Odbc.Query	Conecta-se a um provedor genérico com a cadeia de conexão fornecida e retorna o resultado da avaliação da consulta.

FUNÇÃO	DESCRIÇÃO
OleDb.DataSource	Retorna uma tabela de exibições e tabelas SQL da fonte de dados OLE DB especificada pela cadeia de conexão.
OleDb.Query	Retorna o resultado da execução de uma consulta nativa em uma fonte de dados do OLE DB.
Oracle.Database	Retorna uma tabela contendo dados relacionados às tabelas no Oracle Database especificado.
Parquet.Document	Retorna o conteúdo do documento Parquet como uma tabela.
Pdf.Tables	Retorna todas as tabelas encontradas em pdf.
PostgreSQL.Database	Retorna uma tabela contendo dados relacionados às tabelas no banco de dados PostgreSQL especificado.
RData.FromBinary	Retorna um registro de quadros de dados do arquivo RData.
Salesforce.Data	Conecta-se à API de Objetos do Salesforce e retorna o conjunto de objetos disponíveis (ou seja, Contas).
Salesforce.Reports	Conecta-se à API de Relatórios do Salesforce e retorna o conjunto de relatórios disponíveis.
SapBusinessWarehouse.Cubes	Retorna InfoCubes e consultas em um sistema SAP Business Warehouse agrupado por InfoArea.
SapBusinessWarehouseExecutionMode.DataStream	Opção 'Modo de mesclagem de DataStream' para execução de MDX no SAP Business Warehouse.
SapBusinessWarehouseExecutionMode.BasXml	Opção 'Modo de mesclagem de bXML' para execução de MDX no SAP Business Warehouse.
SapBusinessWarehouseExecutionMode.BasXmlGzip	Opção 'Modo de mesclagem de bXML compactado em Gzip' para execução de MDX no SAP Business Warehouse. Recomendado para consultas de alto volume ou baixa latência.
SapHana.Database	Retorna os pacotes em um banco de dados SAP HANA.
SapHanaDistribution.All	Retorna os pacotes em um banco de dados SAP HANA.
SapHanaDistribution.Connection	Opção de distribuição 'Connection' para SAP HANA.
SapHanaDistribution.Off	Opção de distribuição 'Off' para SAP HANA.
SapHanaDistribution.Statement	Opção de distribuição 'Statement' para SAP HANA.
SapHanaRangeOperator.Equals	Operador de intervalo 'equals' para parâmetros de entrada do SAP HANA.
SapHanaRangeOperator.GreaterThan	Operador de intervalo 'greater than' para parâmetros de entrada do SAP HANA.

FUNÇÃO	DESCRIÇÃO
SapHanaRangeOperator.GreaterThanOrEquals	Operador de intervalo 'greater than or equals' para parâmetros de entrada do SAP HANA.
SapHanaRangeOperator.LessThan	Operador de intervalo 'less than' para parâmetros de entrada do SAP HANA.
SapHanaRangeOperator.LessThanOrEquals	Operador de intervalo 'less than or equals' para parâmetros de entrada do SAP HANA.
SapHanaRangeOperator.NotEquals	Operador de intervalo 'not equals' para parâmetros de entrada do SAP HANA.
SharePoint.Contents	Retorna uma tabela que contém uma linha para cada pasta e documento encontrados na URL site do SharePoint especificada. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.
SharePoint.Files	Retorna uma tabela que contém uma linha para cada documento encontrado na URL do SharePoint e subpastas. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.
SharePoint.Tables	Retorna uma tabela que contém o resultado de uma lista do SharePoint como um feed OData.
Soda.Feed	Retorna a tabela resultante de um arquivo CSV que pode ser acessado usando a API SODA 2.0. A URL deve apontar para uma fonte em conformidade com SODA válida que termina em uma extensão .csv.
Sql.Database	Retorna uma tabela que contém tabelas SQL localizadas em um banco de dados de instância do SQL Server.
Sql.Databases	Retorna uma tabela com referências a bancos de dados localizados em uma instância do SQL Server. Retorna uma tabela de navegação.
Sybase.Database	Retorna uma tabela contendo dados relacionados às tabelas no Sybase Database especificado.
Teradata.Database	Retorna uma tabela contendo dados relacionados às tabelas no Teradata Database especificado.
WebAction.Request	Cria uma ação que, quando executada, retornará os resultados de realizar uma solicitação de método com relação à URL usando HTTP como um valor binário.
Web.BrowserContents	Retorna o HTML da URL especificada, conforme exibido por um navegador da Web.
Web.Contents	Retorna o conteúdo baixado de uma URL da Web como um valor binário.
Web.Page	Retorna o conteúdo de uma página da Web HTML como uma tabela.

FUNÇÃO	DESCRIÇÃO
WebMethod.Delete	Especifica o método DELETE para HTTP.
WebMethod.Get	Especifica o método GET para HTTP.
WebMethod.Head	Especifica o método HEAD para HTTP.
WebMethod.Patch	Especifica o método PATCH para HTTP.
WebMethod.Post	Especifica o método POST para HTTP.
WebMethod.Put	Especifica o método PUT para HTTP.
Xml.Document	Retorna o conteúdo de um documento XML como uma tabela hierárquica (lista de registros).
Xml.Tables	Retorna o conteúdo de um documento XML como uma coleção aninhada de tabelas niveladas.

AccessControlEntry.ConditionToIdentities

09/05/2020 • 2 minutes to read

Sintaxe

```
AccessControlEntry.ConditionToIdentities(identityProvider as function, condition as function) as
list
```

Sobre

Usando o `identityProvider` especificado, converte o `condition` na lista de identidades para a qual `condition` retornaria `true` em todos os contextos de autorização com `identityProvider` como o provedor de identidade. Um erro será gerado se não for possível converter `condition` em uma lista de identidades; por exemplo, se `condition` consultar atributos diferentes de identidades de usuário ou de grupo para tomar uma decisão.

Observe que a lista de identidades representa as identidades conforme elas aparecem na `condition` e nenhuma normalização (como a expansão do grupo) é executada nelas.

AccessControlKind.Allow

09/05/2020 • 2 minutes to read

Sobre

O acesso é permitido.

AccessControlKind.Deny

09/05/2020 • 2 minutes to read

Sobre

O acesso é negado.

Access.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
Access.Database(database as binary, optional options as nullable record) as table
```

Sobre

Retorna uma representação estrutural de um banco de dados do Access, `database`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: Uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é false).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...], por exemplo.

ActiveDirectory.Domains

09/05/2020 • 2 minutes to read

Sintaxe

```
ActiveDirectory.Domains(optional forestRootDomainName as nullable text) as table
```

Sobre

Retorna uma lista de domínios do Active Directory na mesma floresta do domínio especificado ou do domínio do computador atual, caso nenhuma tenha sido especificada.

AdobeAnalytics.Cubes

09/05/2020 • 2 minutes to read

Sintaxe

```
AdobeAnalytics.Cubes(optional options as nullable record) as table
```

Sobre

Retorna uma tabela de pacotes multidimensionais do Adobe Analytics. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).
- `MaxRetryCount`: O número de tentativas a serem executadas ao sondar o resultado da consulta. O valor padrão é 120.
- `RetryInterval`: A duração do tempo entre as tentativas de repetição. O valor padrão é de 1 segundo.

AdoDotNet.DataSource

09/05/2020 • 2 minutes to read

Sintaxe

```
AdoDotNet.DataSource(providerName as text, connectionString as any, optional options as nullable record) as table
```

Sobre

Retorna a coleção de esquemas para a fonte de dados ADO.NET com o nome do provedor `providerName` e a cadeia de conexão `connectionString`. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.
- `TypeMap`

AdoDotNet.Query

09/05/2020 • 2 minutes to read

Sintaxe

```
AdoDotNet.Query(providerName as text, connectionString as any, query as text, optional options as nullable record) as table
```

Sobre

Retorna o resultado da execução de `query` com a cadeia de conexão `connectionString` usando o provedor ADO.NET `providerName`. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.

AnalysisServices.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
AnalysisServices.Database(server as text, database as text, optional options as nullable record)
as table
```

Sobre

Retorna uma tabela de cubos multidimensionais ou modelos de tabela do banco de dados do Analysis Services `database` no servidor `server`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `Query`: Uma consulta MDX nativa usada para recuperar dados.
- `TypedMeasureColumns`: Um valor lógico indicando se os tipos especificados no modelo multidimensional ou de tabela serão usados para os tipos das colunas de medida adicionadas. Quando definido como false, o tipo "número" será usado para todas as colunas de medida. O valor padrão dessa opção é false.
- `Culture`: Um nome de cultura especificando a cultura dos dados. Isso corresponde à propriedade da cadeia de conexão 'Identificador de Localidade'.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dependente do driver.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `SubQueries`: Um número (0, 1 ou 2) que define o valor da propriedade "SubQueries" na cadeia de conexão. Isso controla o comportamento de membros calculados em subseleções ou subcubos. (O valor padrão é 2.)
- `Implementation`

AnalysisServices.Databases

09/05/2020 • 2 minutes to read

Sintaxe

```
AnalysisServices.Databases(server as text, optional options as nullable record) as table
```

Sobre

Retornará bancos de dados em uma instância do Analysis Services, `server`. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `TypedMeasureColumns`: Um valor lógico indicando se os tipos especificados no modelo multidimensional ou de tabela serão usados para os tipos das colunas de medida adicionadas. Quando definido como false, o tipo "número" será usado para todas as colunas de medida. O valor padrão dessa opção é false.
- `Culture`: Um nome de cultura especificando a cultura dos dados. Isso corresponde à propriedade da cadeia de conexão 'Identificador de Localidade'.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dependente do driver.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `SubQueries`: Um número (0, 1 ou 2) que define o valor da propriedade "SubQueries" na cadeia de conexão. Isso controla o comportamento de membros calculados em subseleções ou subcubos. (O valor padrão é 2.)
- `Implementation`

AzureStorage.BlobContents

09/05/2020 • 2 minutes to read

Sintaxe

```
AzureStorage.BlobContents(url as text, optional options as nullable record) as binary
```

Sobre

Retorna o conteúdo do blob na URL, `url`, de um cofre de armazenamento do Azure. `options` pode ser especificado para controlar as seguintes opções:

- `BlockSize`: O número de bytes a serem lidos antes de aguardar o consumidor de dados. O valor padrão é de 4 MB.
- `RequestSize`: O número de bytes cuja leitura será tentada em uma única solicitação HTTP para o servidor. O valor padrão é de 4 MB.
- `ConcurrentRequests`: A opção ConcurrentRequests dá suporte ao download mais rápido de dados, especificando o número de solicitações a serem feitas em paralelo, ao custo da utilização da memória. A memória necessária é (ConcurrentRequest * RequestSize). O valor padrão é 16.

AzureStorage.Blobs

09/05/2020 • 2 minutes to read

Sintaxe

```
AzureStorage.Blobs(account as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de navegação que contém uma linha para cada contêiner encontrado na URL da conta, `account`, de um cofre de armazenamento do Azure. Cada linha contém um link para os blobs de contêiner. `options` pode ser especificado para controlar as seguintes opções:

- `BlockSize`: O número de bytes a serem lidos antes de aguardar o consumidor de dados. O valor padrão é de 4 MB.
- `RequestSize`: O número de bytes cuja leitura será tentada em uma única solicitação HTTP para o servidor. O valor padrão é de 4 MB.
- `ConcurrentRequests`: A opção ConcurrentRequests dá suporte ao download mais rápido de dados, especificando o número de solicitações a serem feitas em paralelo, ao custo da utilização da memória. A memória necessária é (ConcurrentRequest * RequestSize). O valor padrão é 16.

AzureStorage.DataLake

09/05/2020 • 2 minutes to read

Sintaxe

```
AzureStorage.DataLake(endpoint as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela navegacional contendo os documentos localizados no contêiner especificado e nas suas subpastas na URL da conta, `endpoint`, de um sistema de arquivos do Azure Data Lake Storage. `options` pode ser especificado para controlar as seguintes opções:

- `BlockSize`: O número de bytes a serem lidos antes de aguardar o consumidor de dados. O valor padrão é de 4 MB.
- `RequestSize`: O número de bytes cuja leitura será tentada em uma única solicitação HTTP para o servidor. O valor padrão é de 4 MB.
- `ConcurrentRequests`: A opção ConcurrentRequests dá suporte ao download mais rápido de dados, especificando o número de solicitações a serem feitas em paralelo, ao custo da utilização da memória. A memória necessária é (ConcurrentRequest * RequestSize). O valor padrão é 16.
- `HierarchicalNavigation`: Uma lógica (true/false) que controla se os arquivos são retornados em um modo de exibição de diretório semelhante ao de árvore ou em uma lista plana. O valor padrão é false.

AzureStorage.DataLakeContents

09/05/2020 • 2 minutes to read

Sintaxe

```
AzureStorage.DataLakeContents(url as text, optional options as nullable record) as binary
```

Sobre

Retorna o conteúdo do arquivo na URL, `url`, de um sistema de arquivos do Azure Data Lake Storage. `options` pode ser especificado para controlar as seguintes opções:

- `BlockSize`: O número de bytes a serem lidos antes de aguardar o consumidor de dados. O valor padrão é de 4 MB.
- `RequestSize`: O número de bytes cuja leitura será tentada em uma única solicitação HTTP para o servidor. O valor padrão é de 4 MB.
- `ConcurrentRequests`: A opção ConcurrentRequests dá suporte ao download mais rápido de dados, especificando o número de solicitações a serem feitas em paralelo, ao custo da utilização da memória. A memória necessária é (ConcurrentRequest * RequestSize). O valor padrão é 16.

AzureStorage.Tables

09/05/2020 • 2 minutes to read

Sintaxe

```
AzureStorage.Tables(account as text) as table
```

Sobre

Retorna uma tabela de navegação que contém uma linha para cada tabela encontrado na URL da conta, `account`, de um cofre de armazenamento do Azure. Cada linha contém um link para a tabela do Azure.

Cdm.Contents

08/05/2020 • 2 minutes to read

Sintaxe

```
Cdm.Contents(table as table) as table
```

Sobre

Esta função não está disponível porque requer o .NET 4.5.

Csv.Document

08/05/2020 • 3 minutes to read

Sintaxe

```
Csv.Document(source as any, optional columns as any, optional delimiter as any, optional
extraValues as nullable number, optional encoding as nullable number) as table
```

Sobre

Retorna o conteúdo do documento CSV como uma tabela.

- `columns` pode ser nulo, o número de colunas, uma lista de nomes de colunas, um tipo de tabela ou um registro de opções. (Confira abaixo mais detalhes sobre o registro de opções.)
- `delimiter` pode ser um único caractere ou uma lista de caracteres. Padrão: `","`.
- Veja `ExtraValues.Type` para obter os valores de `extraValues` com suporte.
- `encoding` especifica o tipo de codificação de texto.

Se um registro for especificado para `columns` (e `delimiter`, `extraValues` e `encoding` forem nulos), os seguintes campos de registro poderão ser fornecidos:

- `Delimiter`: o delimitador de coluna. Padrão: `","`.
- `Columns`: pode ser nulo, o número de colunas, uma lista de nomes de colunas ou um tipo de tabela. Se o número de colunas for menor do que o número encontrado na entrada, as colunas adicionais serão ignoradas. Se o número de colunas for maior do que o número encontrado na entrada, as colunas adicionais serão nulas. Quando não for especificado, o número de colunas será determinado pelo que é encontrado na entrada.
- `Encoding`: a codificação de texto do arquivo. Padrão: 65001 (UTF-8).
- `CsvStyle`: especifica como as aspas são manipuladas. `CsvStyle.QuoteAfterDelimiter` (padrão): As aspas em um campo são significativas apenas imediatamente depois do delimitador. `CsvStyle.QuoteAlways`: as aspas em um campo são sempre significativas, independentemente do local em que aparecerem.
- `QuoteStyle`: especifica como as quebras de linha entre aspas são manipuladas. `QuoteStyle.None` (padrão): Todas as quebras de linha são tratadas como o final da linha atual, mesmo quando elas ocorrem dentro de um valor entre aspas. `QuoteStyle.Csv`: as quebras de linha entre aspas são tratadas como parte dos dados, e não como o fim da linha atual.

Exemplo 1

Processa texto CSV com cabeçalhos de coluna.

```
let
    csv = Text.Combine({"OrderID,Item", "1,Fishing rod", "2,1 lb. worms"}, "#(cr)#(lf)")
in
    Table.PromoteHeaders(Csv.Document(csv))
```

ORDERID	ITEM
1	Vara de pescar

ORDERID	ITEM
2	1 lb de minhocas

CsvStyle.QuoteAfterDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
CsvStyle.QuoteAfterDelimiter
```

Sobre

As aspas em um campo são significativas apenas imediatamente depois do delimitador.

CsvStyle.QuoteAlways

09/05/2020 • 2 minutes to read

Sintaxe

```
CsvStyle.QuoteAlways
```

Sobre

As aspas em um campo são sempre significativas, independentemente do local em que aparecerem.

Cube.AddAndExpandDimensionColumn

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.AddAndExpandDimensionColumn(**cube** as table, **dimensionSelector** as any,
**attributeNames** as list, optional **newColumnNames** as any) as table
```

Sobre

Mescla a tabela de dimensão especificada, `dimensionSelector`, no contexto de filtro do cubo, `cube`, e altera a granularidade dimensional expandindo o conjunto especificado, `attributeNames`, de atributos de dimensão. Os atributos de dimensão são adicionados à exibição tabular com colunas nomeadas `newColumnNames`, ou `attributeNames`, se não for especificado.

Cube.AddMeasureColumn

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.AddMeasureColumn(**cube** as table, **column** as text, **measureSelector** as any) as table
```

Sobre

Adiciona uma coluna com o nome `column` ao `cube` que contém os resultados da medida `measureSelector` aplicada ao contexto de cada linha. O aplicativo de medida é afetado por alterações na granularidade e na divisão da dimensão. Os valores da medida serão ajustados depois que determinadas operações de cubo forem executadas.

Cube.ApplyParameter

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.ApplyParameter(cube as table, parameter as any, optional arguments as nullable list) as table
```

Sobre

Retorna um cubo após a aplicação de `parameter` com `arguments` a `cube`.

Cube.AttributeMemberId

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.AttributeMemberId(attribute as any) as any
```

Sobre

Retorna o identificador de membro exclusivo de um valor da propriedade do membro. `attribute`. Retorna nulo para qualquer outro valor.

Cube.AttributeMemberProperty

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.AttributeMemberProperty(attribute as any, propertyName as text) as any
```

Sobre

Retorna a propriedade `propertyName` do atributo de dimensão `attribute`.

Cube.CollapseAndRemoveColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.CollapseAndRemoveColumns(**cube** as table, **columnNames** as list) as table
```

Sobre

Altera a granularidade dimensional do contexto de filtro do `cube`, recolhendo os atributos mapeados para as colunas especificadas `columnNames`. As colunas também são removidas da exibição de tabela do cubo.

Cube.Dimensions

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.Dimensions(**cube** as table) as table
```

Sobre

Retorna a tabela que contém o conjunto de dimensões disponíveis dentro do `cube`. Cada dimensão é uma tabela que contém um conjunto de atributos de dimensão e cada atributo de dimensão é representado como uma coluna na tabela de dimensões. As dimensões podem ser expandidas no cubo usando Cube.AddAndExpandDimensionColumn.

Cube.DisplayFolders

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.DisplayFolders(**cube** as table) as table
```

Sobre

Retorna uma árvore de tabelas aninhada que representa a hierarquia da pasta de exibição dos objetos (por exemplo, dimensões e medidas) disponíveis para uso no `cube`.

Cube.MeasureProperties

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.MeasureProperties(cube as table) as table
```

Sobre

Retorna uma tabela contendo o conjunto de propriedades disponíveis para medidas que são expandidas no cubo.

Cube.MeasureProperty

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.MeasureProperty(measure as any, propertyName as text) as any
```

Sobre

Retorna a propriedade `propertyName` da medida `measure`.

Cube.Measures

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.Measures(**cube** as any) as table
```

Sobre

Retorna a tabela que contém o conjunto de medidas disponíveis no `cube`. Cada medida é representada como uma função. As medidas podem ser aplicadas ao cubo usando Cube.AddMeasureColumn.

Cube.Parameters

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.Parameters(cube as table) as table
```

Sobre

Retorna uma tabela contendo o conjunto de parâmetros que pode ser aplicado a `cube`. Cada parâmetro é uma função que pode ser invocada para obter `cube` com o parâmetro e seus argumentos aplicados.

Cube.Properties

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.Properties(cube as table) as table
```

Sobre

Retorna uma tabela contendo o conjunto de propriedades disponíveis para dimensões expandidas no cubo.

Cube.PropertyKey

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.PropertyKey(property as any) as any
```

Sobre

Retorna a chave da propriedade `property`.

Cube.ReplaceDimensions

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.ReplaceDimensions(cube as table, dimensions as table) as table
```

Sobre

Cube.ReplaceDimensions

Cube.Transform

09/05/2020 • 2 minutes to read

Sintaxe

```
Cube.Transform(cube as table, transforms as list) as table
```

Sobre

Aplica as funções de cubo da lista, `transforms`, ao `cube`.

DB2.Database

09/05/2020 • 3 minutes to read

Sintaxe

```
DB2.Database(server as text, database as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados do Db2 no servidor `server` na instância de banco de dados chamada `database`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).
- `Implementation`: Especifica a implementação do provedor de banco de dados interno a ser usado. Os valores válidos são: "IBM" e "Microsoft".
- `BinaryCodePage`: Um número para o CCSID (Identificador do conjunto de caracteres codificados) para decodificar dados binários Db2 FOR BIT em cadeias de caracteres. Aplica-se à Implementação = "Microsoft". Defina 0 para desabilitar a conversão (padrão). Defina 1 para converter com base na codificação do banco de dados. Defina outro número de CCSID para converter em codificação de aplicativo.
- `PackageCollection`: Especifica um valor de cadeia de caracteres para a coleção de pacotes (o padrão é "NULLID") para habilitar o uso de pacotes compartilhados necessários para processar instruções SQL. Aplica-se à Implementação = "Microsoft".
- `UseDb2ConnectGateway`: Especifica se a conexão está sendo estabelecida por meio de um gateway do Db2 Connect. Aplica-se à Implementação = "Microsoft".

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

Essbase.Cubes

09/05/2020 • 2 minutes to read

Sintaxe

```
Essbase.Cubes(url as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de cubos agrupados pelo servidor Essbase de uma instância do Essbase no servidor APS `url`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.

Excel.CurrentWorkbook

30/09/2020 • 2 minutes to read

Sintaxe

```
Excel.CurrentWorkbook() as table
```

Sobre

Retorna o conteúdo da pasta de trabalho atual do Excel.

Excel.Workbook

30/09/2020 • 2 minutes to read

Sintaxe

```
Excel.Workbook(workbook as binary, optional useHeaders as nullable logical, optional delayTypes as nullable logical) as table
```

Sobre

Retorna o conteúdo da pasta de trabalho do Excel.

Exchange.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
Exchange.Contents (optional mailboxAddress as nullable text) as table
```

Sobre

Retorna um sumário de uma conta do Microsoft Exchange `mailboxAddress`. Se `mailboxAddress` não for especificado, a conta padrão da credencial será usada.

File.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
File.Contents(path as text, optional options as nullable record) as binary
```

Sobre

Retorna o conteúdo do arquivo, `path`, como binário.

Folder.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
Folder.Contents(path as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada pasta e arquivo encontrado no caminho de pasta, `path`. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.

Folder.Files

09/05/2020 • 2 minutes to read

Sintaxe

```
Folder.Files(path as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada arquivo encontrado no caminho de pasta, `path`, e subpastas. Cada linha contém propriedades do arquivo e um link para o conteúdo.

GoogleAnalytics.Accounts

09/05/2020 • 2 minutes to read

Sintaxe

```
GoogleAnalytics.Accounts() as table
```

Sobre

Retorna as contas do Google Analytics que são acessíveis pela credencial atual.

Hdfs.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
Hdfs.Contents(url as text) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada pasta e arquivo encontrado na URL da pasta, `url`, de um sistema de arquivos do Hadoop. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo.

Hdfs.Files

09/05/2020 • 2 minutes to read

Sintaxe

```
Hdfs.Files(url as text) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada arquivo encontrado na URL da pasta, `url`, e subpastas de um sistema de arquivos do Hadoop. Cada linha contém propriedades do arquivo e um link para o conteúdo.

HdInsight.Containers

09/05/2020 • 2 minutes to read

Sintaxe

```
HdInsight.Containers(account as text) as table
```

Sobre

Retorna uma tabela de navegação que contém uma linha para cada contêiner encontrado na URL da conta, `account`, de um cofre de armazenamento do Azure. Cada linha contém um link para os blobs de contêiner.

HdInsight.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
HdInsight.Contents(account as text) as table
```

Sobre

Retorna uma tabela de navegação que contém uma linha para cada contêiner encontrado na URL da conta, `account`, de um cofre de armazenamento do Azure. Cada linha contém um link para os blobs de contêiner.

HdInsight.Files

09/05/2020 • 2 minutes to read

Sintaxe

```
HdInsight.Files(account as text, containerName as text) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada arquivo de blob encontrado na URL do contêiner, `account`, de um cofre de armazenamento do Azure. Cada linha contém propriedades do arquivo e um link para o conteúdo.

Html.Table

21/09/2020 • 2 minutes to read

Sintaxe

```
Html.Table(html as any, columnNameSelectorPairs as list, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém os resultados da execução dos seletores de CSS especificados em relação ao `html` fornecido. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `RowSelector`

Exemplo 1

Retorna uma tabela de um valor de texto de html de exemplo.

```
Html.Table("<div class=""name"">Jo</div><span>Manager</span>", {{"Name", ".name"}, {"Title", "span"}}, [RowSelector=".name"])\
```

```
#table({"Name", "Title"}, {{"Jo", "Manager"}})
```

Exemplo 2

Extrai todos os hrefs de um valor de texto de html de exemplo.

```
Html.Table("<a href=""/test.html"">Test</a>", {{"Link", "a", each [Attributes][href]}})
```

```
#table({"Link"}, {{"/test.html"}})
```

Identity.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Identity.From(identityProvider as function, value as any) as record
```

Sobre

Cria uma identidade.

Identity.IsMemberOf

09/05/2020 • 2 minutes to read

Sintaxe

```
Identity.IsMemberOf(identity as record, collection as record) as logical
```

Sobre

Determina se uma identidade é um membro de uma coleção de identidades.

IdentityProvider.Default

09/05/2020 • 2 minutes to read

Sintaxe

```
IdentityProvider.Default() as any
```

Sobre

O provedor de identidade padrão do host atual.

Informix.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
Informix.Database(server as text, database as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados do Informix no servidor `server` na instância de banco de dados chamada `database`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

Json.Document

09/05/2020 • 2 minutes to read

Sintaxe

```
Json.Document(jsonText as any, optional encoding as nullable number) as any
```

Sobre

Retorna o conteúdo do documento JSON.

Json.FromValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Json.FromValue(value as any, optional encoding as nullable number) as binary
```

Sobre

Produz uma representação JSON de um valor especificado `value` com uma codificação de texto especificada por `encoding`. Se `encoding` for omitido, UTF8 será usado. Os valores são representados da seguinte maneira:

- Valores nulos, lógicos e de texto são representados como os tipos JSON correspondentes
- Os números são representados como números em JSON, exceto que `#infinity`, `-#infinity` e `#nan` são convertidos em NULL
- As listas são representadas como matrizes JSON
- Os registros são representados como objetos JSON
- As tabelas são representadas como uma matriz de objetos
- Datas, horas, data e hora, fusos horários e durações são representados como texto ISO-8601
- Os valores binários são representados como texto codificado em base64
- Tipos e funções produzem um erro

Exemplo 1

Converter um valor complexo em JSON.

```
Text.FromBinary(Json.FromValue([A = {1, true, "3"}, B = #date(2012, 3, 25)]))
```

```
"{""A"":[1,true,""3""],""B"":""2012-03-25""}"
```

MySQL.Database

09/05/2020 • 3 minutes to read

Sintaxe

```
MySQL.Database(server as text, database as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas SQL, exibições e funções escalares armazenadas disponíveis em um banco de dados MySQL no servidor `server` na instância do banco de dados chamada `database`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `Encoding`: Um valor de TextEncoding que especifica o conjunto de caracteres usado para codificar todas as consultas enviadas ao servidor (o padrão é nulo).
- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `TreatTinyAsBoolean`: uma lógica (true/false) que determina se será necessário forçar as colunas tinyint no servidor como valores lógicos. O valor padrão é true.
- `OldGuids`: uma lógica (true/false) que define se as colunas char(36) (em caso de false) ou binary(16) (em caso de true) serão tratadas como GUIDs. O valor padrão é false.
- `ReturnSingleDatabase`: uma lógica (true/false) que define se todas as tabelas de todos os banco de dados serão retornadas (em caso de false) ou se as tabelas e visualizações do banco de dados especificado serão retornadas (em caso de true). O valor padrão é false.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

OData.Feed

09/05/2020 • 5 minutes to read

Sintaxe

```
OData.Feed(serviceUri as text, optional headers as nullable record, optional options as any) as
any
```

Sobre

Retorna uma tabela de feeds OData oferecidos por um serviço OData de um URI `serviceUri`, cabeçalhos `headers`. Um valor booliano que especifica se é necessário usar conexões simultâneas ou um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `Query`: adicione programaticamente parâmetros de consulta à URL sem precisar se preocupar com a saída.
- `Headers`: especificar esse valor como um registro fornecerá cabeçalhos adicionais a uma solicitação HTTP.
- `ExcludedFromCacheKey`: especificar esse valor como uma lista excluirá essas chaves de cabeçalho HTTP de fazerem parte do cálculo para armazenar dados em cache.
- `ApiKeyName`: se o site de destino tiver uma noção de uma chave de API, esse parâmetro poderá ser usado para especificar o nome (não o valor) do parâmetro de chave que deve ser usado na URL. O valor real da chave é fornecido na credencial.
- `Timeout`: especificar esse valor como uma duração alterará o tempo limite de uma solicitação HTTP. O valor padrão é de 600 segundos.
- `EnableBatch`: uma lógica (true/false) que define se a geração de uma solicitação de $batch OData deverá ser permitida se o MaxUriLength for excedido (o padrão é false).
- `MaxUriLength`: um número que indica o comprimento máximo de um URI permitido enviado a um serviço OData. Se excedido e EnableBatch for true, a solicitação será feita a um ponto de extremidade de $batch OData, caso contrário, falhará (o padrão é 2048).
- `Concurrent`: em uma lógica (true/false), quando definida como true, as solicitações ao serviço serão feitas simultaneamente. Quando definida como false, as solicitações serão feitas em sequência. Quando não especificado, o valor será determinado pela anotação AsynchronousRequestsSupported do serviço. Se o serviço não especificar se há suporte para AsynchronousRequestsSupported, as solicitações serão feitas em sequência.
- `ODataVersion`: Um número (3 ou 4) que especifica a versão do protocolo OData a ser usada para esse serviço de OData. Quando não especificadas, todas as versões com suporte serão solicitadas. A versão do serviço será determinada pelo cabeçalho OData-Version retornado pelo serviço.
- `FunctionOverloads`: Uma lógica (true/false) quando definida como true: as sobrecargas de importação de função estarão listadas no navegador como entradas separadas; quando definida como false: as sobrecargas de importação de função estarão listadas como uma função de união no navegador. Valor padrão para V3: false. Valor padrão para V4: true.
- `MoreColumns`: uma lógica (true/false) quando definida como true, adiciona uma coluna "Mais colunas" a cada feed de entidade que contém tipos abertos e polimórficos. Conterá os campos não declarados no tipo base. Quando for false, este campo não estará presente. O padrão é false.
- `IncludeAnnotations`: uma lista separada por vírgulas de padrões ou nomes de termos qualificados de namespace a ser incluída com "*" como um caractere curinga. Por padrão, nenhuma das anotações estão incluídas.
- `IncludeMetadataAnnotations`: uma lista separada por vírgulas de padrões ou nomes de termos qualificados de

namespace a ser incluída em solicitações de documento de metadados, com "*" como um caractere curinga. Por padrão, inclui as mesmas anotações que IncludeAnnotations.

- `OmitValues`: permite que o serviço OData evite gravar determinados valores em respostas. Se for confirmado, vamos inferir esses valores dos campos omitidos. As opções incluem:
- `ODataOmitValues.Nulls`: permite que o serviço OData omita valores nulos.
- `Implementation`: Especifica a implementação do conector OData a ser usada. Os valores válidos são "2.0" ou nulo.

ODataOmitValues.Nulls

09/05/2020 • 2 minutes to read

Sobre

Permite que o serviço OData omita valores nulos.

Odbc.DataSource

09/05/2020 • 2 minutes to read

Sintaxe

```
Odbc.DataSource(connectionString as any, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de exibições e tabelas SQL da fonte de dados ODBC especificada pela cadeia de conexão `connectionString`. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é de 15 segundos.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.

Odbc.InferOptions

09/05/2020 • 2 minutes to read

Sintaxe

```
Odbc.InferOptions(connectionString as any) as record
```

Sobre

Retorna o resultado da tentativa de inferir as funcionalidades SQL com a cadeia de conexão `connectionString` usando o ODBC. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número.

Odbc.Query

09/05/2020 • 2 minutes to read

Sintaxe

```
Odbc.Query(connectionString as any, query as text, optional options as nullable record) as table
```

Sobre

Retorna o resultado da execução de `query` com a cadeia de conexão `connectionString` usando ODBC. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é de 15 segundos.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.

OleDb.DataSource

09/05/2020 • 2 minutes to read

Sintaxe

```
OleDb.DataSource(connectionString as any, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de exibições e tabelas SQL da fonte de dados OLE DB especificada pela cadeia de conexão `connectionString`. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas pelos respectivos nomes de esquema serão exibidas (o padrão é true).
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

OleDb.Query

09/05/2020 • 2 minutes to read

Sintaxe

```
OleDb.Query(connectionString as any, query as text, optional options as nullable record) as table
```

Sobre

Retorna o resultado da execução de `query` com a cadeia de conexão `connectionString` usando OLE DB. `connectionString` pode ser um texto ou um registro de pares de valor da propriedade. Os valores de propriedade podem ser texto ou número. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `SqlCompatibleWindowsAuth`: um valor lógico (true/false) que determina se as opções de cadeia de conexão compatíveis com o SQL Server devem ser produzidas para a autenticação do Windows. O valor padrão é true.

Oracle.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
Oracle.Database(server as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL do Oracle Database no servidor `server`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

Parquet.Document

21/09/2020 • 2 minutes to read

Sintaxe

```
Parquet.Document(binary as binary, optional options as nullable record) as any
```

Sobre

Retorna o conteúdo do documento Parquet como uma tabela.

Pdf.Tables

30/09/2020 • 2 minutes to read

Sintaxe

```
Pdf.Tables(pdf as binary, optional options as nullable record) as table
```

Sobre

Retorna todas as tabelas encontradas em `pdf`. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `Implementation`: a versão do algoritmo a ser usada ao identificar tabelas. Os valores válidos são "1,1" ou nulo.
- `StartPage`: Especifica a primeira página no intervalo de páginas a ser analisado. Padrão: 1.
- `EndPage`: especifica a última página no intervalo de páginas a ser analisado. Padrão: a última página do documento.
- `MultiPageTables`: Controla se tabelas semelhantes em páginas consecutivas serão automaticamente combinadas em uma única tabela. Padrão: true.
- `EnforceBorderLines`: Controla se linhas de borda são sempre impostas como limites da célula (quando for true) ou simplesmente usadas como uma dica dentre muitas para determinar os limites da célula (quando for false). Padrão: falso.

Exemplo 1

Retorna as tabelas contidas em sample.pdf.

```
Pdf.Tables(File.Contents("c:\sample.pdf"))
```

```
#table({"Name", "Kind", "Data"}, ...)
```

PostgreSQL.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
PostgreSQL.Database(server as text, database as text, optional options as nullable record) as
table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados do PostgreSQL no servidor `server` na instância de banco de dados chamada `database`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

RData.FromBinary

09/05/2020 • 2 minutes to read

Sintaxe

```
RData.FromBinary(stream as binary) as any
```

Sobre

Retorna um registro de quadros de dados do arquivo RData.

Salesforce.Data

09/05/2020 • 2 minutes to read

Sintaxe

```
Salesforce.Data(optional loginUrl as any, optional options as nullable record) as table
```

Sobre

Retorna os objetos na conta do Salesforce fornecida nas credenciais. A conta será conectada por meio da `loginUrl` do ambiente fornecido. Se nenhum ambiente for fornecido, a conta será conectada à produção (https://login.salesforce.com). Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `CreateNavigationProperties`: Uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é false).
- `ApiVersion`: A versão da API do Salesforce a ser usada para esta consulta. Quando não especificado, a versão 29.0 da API é usada.
- `Timeout`: Uma duração que controla por quanto tempo esperar antes de abandonar a solicitação para o servidor. O valor padrão é específico da fonte.

Salesforce.Reports

08/05/2020 • 2 minutes to read

Sintaxe

```
Salesforce.Reports(optional loginUrl as nullable text, optional options as nullable record) as
table
```

Sobre

Retorna os relatórios na conta do Salesforce fornecida nas credenciais. A conta será conectada por meio da `loginUrl` do ambiente fornecido. Se nenhum ambiente for fornecido, a conta será conectada à produção (https://login.salesforce.com). Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `ApiVersion`: A versão da API do Salesforce a ser usada para esta consulta. Quando não especificado, a versão 29.0 da API é usada.
- `Timeout`: Uma duração que controla por quanto tempo esperar antes de abandonar a solicitação para o servidor. O valor padrão é específico da fonte.

SapBusinessWarehouse.Cubes

09/05/2020 • 2 minutes to read

Sintaxe

```
SapBusinessWarehouse.Cubes(server as text, systemNumberOrSystemId as text, clientId as text,
optional optionsOrLogonGroup as any, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de InfoCubes e consultas agrupadas por InfoArea de uma instância do SAP Business Warehouse no servidor `server` com o número do sistema `systemNumberOrSystemId` e a ID do cliente `clientId`. Um parâmetro de registro opcional, `optionsOrLogonGroup`, pode ser especificado para controlar as opções.

sapbusinesswarehouseexecutionmode.datastream

09/05/2020 • 2 minutes to read

Sobre

Opção 'Modo de mesclagem de DataStream' para execução de MDX no SAP Business Warehouse.

SapBusinessWarehouseExecutionMode.BasXml

09/05/2020 • 2 minutes to read

Sobre

Opção 'Modo de mesclagem de bXML' para execução de MDX no SAP Business Warehouse.

SapBusinessWarehouseExecutionMode.BasXmlGzip

09/05/2020 • 2 minutes to read

Sobre

Opção 'Modo de mesclagem de bXML compactado em Gzip' para execução de MDX no SAP Business Warehouse. Recomendado para consultas de alto volume ou baixa latência.

SapHana.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
SapHana.Database(**server** as text, optional **options** as nullable record) as table
```

Sobre

Retorna uma tabela de pacotes multidimensionais do banco de dados SAP HANA `server`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `Distribution`: Uma SapHanaDistribution que define o valor da propriedade "Distribuição" na cadeia de conexão. Os roteiros de instruções são o método para avaliar o nó de servidor correto de um sistema distribuído antes da execução da instrução. O valor padrão é SapHanaDistribution.All.

SapHanaDistribution.All

09/05/2020 • 2 minutes to read

Sobre

Opção de distribuição 'All' para SAP HANA.

SapHanaDistribution.Connection

09/05/2020 • 2 minutes to read

Sobre

Opção de distribuição 'Connection' para SAP HANA.

SapHanaDistribution.Off

09/05/2020 • 2 minutes to read

Sobre

Opção de distribuição 'Off' para SAP HANA.

SapHanaDistribution.Statement

09/05/2020 • 2 minutes to read

Sobre

Opção de distribuição 'Statement' para SAP HANA.

SapHanaRangeOperator.Equals

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'equals' para parâmetros de entrada do SAP HANA.

SapHanaRangeOperator.GreaterThan

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'greater than' para parâmetros de entrada do SAP HANA.

SapHanaRangeOperator.GreaterThanOrEquals

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'greater than or equals' para parâmetros de entrada do SAP HANA.

SapHanaRangeOperator.LessThan

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'less than' para parâmetros de entrada do SAP HANA.

SapHanaRangeOperator.LessThanOrEquals

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'less than or equals' para parâmetros de entrada do SAP HANA.

SapHanaRangeOperator.NotEquals

09/05/2020 • 2 minutes to read

Sobre

Operador de intervalo 'not equals' para parâmetros de entrada do SAP HANA.

SharePoint.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
SharePoint.Contents(url as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada pasta e documento encontrados no site do SharePoint especificado, `url`. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo. `options` pode ser especificado para controlar as seguintes opções:

- `ApiVersion`: Um número (14 ou 15) ou o texto "Automático" que especifica a versão da API do SharePoint a ser usada para este site. Quando não especificado, a versão 14 da API é usada. Quando Auto é especificado, a versão do servidor será descoberta automaticamente, se possível; caso contrário, a versão usará 14 como padrão. Sites do SharePoint que não estejam em inglês exigem pelo menos a versão 15.

SharePoint.Files

09/05/2020 • 2 minutes to read

Sintaxe

```
SharePoint.Files(url as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada documento encontrado no site do SharePoint especificado, `url`, e subpastas. Cada linha contém as propriedades da pasta ou do arquivo e um link para seu conteúdo. `options` pode ser especificado para controlar as seguintes opções:

- `ApiVersion`: Um número (14 ou 15) ou o texto "Automático" que especifica a versão da API do SharePoint a ser usada para este site. Quando não especificado, a versão 14 da API é usada. Quando Auto é especificado, a versão do servidor será descoberta automaticamente, se possível; caso contrário, a versão usará 14 como padrão. Sites do SharePoint que não estejam em inglês exigem pelo menos a versão 15.

SharePoint.Tables

19/10/2020 • 2 minutes to read

Sintaxe

```
SharePoint.Tables(url as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela que contém uma linha para cada item de Lista encontrado na lista do SharePoint especificada, `url`. Cada linha contém propriedades da Lista. `options` pode ser especificado para controlar as seguintes opções:

- `ApiVersion`: Um número (14 ou 15) ou o texto "Automático" que especifica a versão da API do SharePoint a ser usada para este site. Quando não especificado, a versão 14 da API é usada. Quando Auto é especificado, a versão do servidor será descoberta automaticamente, se possível; caso contrário, a versão usará 14 como padrão. Sites do SharePoint que não estejam em inglês exigem pelo menos a versão 15.
- `Implementation`
- `ViewMode`

Soda.Feed

09/05/2020 • 2 minutes to read

Sintaxe

```
Soda.Feed(url as text) as table
```

Sobre

Retorna uma tabela do conteúdo na URL `url` especificada formatada de acordo com a API SODA 2.0. A URL deve apontar para uma fonte em conformidade com SODA válida que termina em uma extensão .csv.

Sql.Database

22/09/2020 • 2 minutes to read

Sintaxe

```
Sql.Database(server as text, database as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas SQL, exibições e funções armazenadas do banco de dados do SQL Server `database` no servidor `server`. A porta pode ser especificada com o servidor, separada por dois-pontos ou vírgula. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `MaxDegreeOfParallelism`: um número que define o valor da cláusula de consulta "maxdop" na consulta SQL gerada.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).
- `MultiSubnetFailover`: uma lógica (true/false) que define o valor da propriedade "MultiSubnetFailover" na cadeia de conexão (o padrão é false).
- `UnsafeTypeConversions`
- `ContextInfo`: Um valor binário que é usado para definir o CONTEXT_INFO antes de executar cada comando.
- `OmitSRID`

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

Sql.Databases

22/09/2020 • 2 minutes to read

Sintaxe

```
Sql.Databases(server as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de bancos de dados no SQL Server especificado, `server`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `MaxDegreeOfParallelism`: um número que define o valor da cláusula de consulta "maxdop" na consulta SQL gerada.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).
- `MultiSubnetFailover`: uma lógica (true/false) que define o valor da propriedade "MultiSubnetFailover" na cadeia de conexão (o padrão é false).
- `UnsafeTypeConversions`
- `ContextInfo`: Um valor binário que é usado para definir o CONTEXT_INFO antes de executar cada comando.
- `OmitSRID`

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...], por exemplo.
Não oferece suporte à definição de uma consulta SQL a ser executada no servidor. `Sql.Database` deve ser usado em vez de executar uma consulta SQL.

Sybase.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
Sybase.Database(server as text, database as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL disponíveis em um banco de dados Sybase no servidor `server` na instância de banco de dados chamada `database`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

Teradata.Database

09/05/2020 • 2 minutes to read

Sintaxe

```
Teradata.Database(server as text, optional options as nullable record) as table
```

Sobre

Retorna uma tabela de tabelas e exibições SQL do banco de dados Teradata no servidor `server`. A porta pode ser especificada com o servidor, separada por dois-pontos. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `CreateNavigationProperties`: uma lógica (true/false) que define se as propriedades de navegação nos valores retornados serão geradas (o padrão é true).
- `NavigationPropertyNameGenerator`: uma função usada para a criação de nomes de propriedades de navegação.
- `Query`: uma consulta SQL nativa usada para recuperar dados. Se a consulta produzir vários conjuntos de resultados, somente o primeiro será retornado.
- `CommandTimeout`: uma duração que controla quanto tempo a consulta do servidor tem permissão para ser executada até ser cancelada. O valor padrão é dez minutos.
- `ConnectionTimeout`: uma duração que controla quanto tempo esperar antes de abandonar uma tentativa de fazer conexão com o servidor. O valor padrão é dependente do driver.
- `HierarchicalNavigation`: uma lógica (true/false) que define se as tabelas agrupadas por seus nomes de esquema serão exibidas (o padrão é false).

O parâmetro de registro é especificado como [option1 = value1, option2 = value2...] ou [Query = "select ..."], por exemplo.

WebAction.Request

09/05/2020 • 2 minutes to read

Sintaxe

```
WebAction.Request(method as text, url as text, optional options as nullable record) as action
```

Sobre

Cria uma ação que, quando executada, retornará os resultados de realizar uma solicitação `method` com relação a `url` usando HTTP como um valor binário. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `Query`: adicione programaticamente parâmetros de consulta à URL sem precisar se preocupar com a saída.
- `ApiKeyName`: se o site de destino tiver uma noção de uma chave de API, esse parâmetro poderá ser usado para especificar o nome (não o valor) do parâmetro de chave que deve ser usado na URL. O valor real da chave é fornecido na credencial.
- `Content`: a especificação desse valor altera a solicitação da Web de GET para POST, usando o valor do campo `Content` como o conteúdo de POST.
- `Headers`: especificar esse valor como um registro fornecerá cabeçalhos adicionais a uma solicitação HTTP.
- `Timeout`: especificar esse valor como uma duração alterará o tempo limite de uma solicitação HTTP. O valor padrão é de 100 segundos.
- `IsRetry`: a especificação desse valor lógico como verdadeiro ignorará qualquer resposta existente no cache ao buscar dados.
- `ManualStatusHandling`: especificar esse valor como uma lista impedirá qualquer manipulação interna para solicitações HTTP cuja resposta tenha um desses códigos de status.
- `RelativePath`: especificar esse valor como texto o acrescentará à URL base antes de fazer a solicitação.

Web.BrowserContents

21/09/2020 • 2 minutes to read

Sintaxe

```
Web.BrowserContents(url as text, optional options as nullable record) as text
```

Sobre

Retorna o HTML da `url` especificada, conforme exibido por um navegador da Web. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `WaitFor`: especifica uma condição a aguardar antes de baixar o HTML, além de aguardar o carregamento da página (que sempre é feito). Pode ser um registro que contém campos de Tempo limite e/ou Seletor. Se apenas um Tempo limite for especificado, a função aguardará a quantidade de tempo especificada antes de baixar o HTML. Se um Seletor e um Tempo limite forem especificados e o Tempo limite expirar antes que o Seletor exista na página, um erro será gerado. Se um Seletor for especificado sem Tempo limite, um Tempo limite padrão de 30 segundos será aplicado.

Exemplo 1

Retorna o HTML para https://microsoft.com.

```
Web.BrowserContents("https://microsoft.com")
```

```
"<!DOCTYPE html><html xmlns=..."
```

Exemplo 2

Retorna o HTML para https://microsoft.com depois de aguardar pela existência de um seletor de CSS.

```
Web.BrowserContents("https://microsoft.com", [WaitFor = [Selector = "div.ready"]])
```

```
"<!DOCTYPE html><html xmlns=..."
```

Exemplo 3

Retorna o HTML para https://microsoft.com após esperar dez segundos.

```
Web.BrowserContents("https://microsoft.com", [WaitFor = [Timeout = #duration(0,0,0,10)]])
```

```
"<!DOCTYPE html><html xmlns=..."
```

Exemplo 4

Retorna o HTML para https://microsoft.com depois de aguardar até dez segundos pela existência de um seletor de CSS.

```
Web.BrowserContents("https://microsoft.com", [WaitFor = [Selector = "div.ready", Timeout =
#duration(0,0,0,10)]])
```

```
"<!DOCTYPE html><html xmlns=..."
```

Web.Contents

09/05/2020 • 2 minutes to read

Sintaxe

```
Web.Contents(url as text, optional options as nullable record) as binary
```

Sobre

Retorna o conteúdo baixado de `url` como binário. Um parâmetro de registro opcional, `options`, pode ser fornecido para especificar propriedades adicionais. O registro pode conter os seguintes campos:

- `Query`: adicione programaticamente parâmetros de consulta à URL sem precisar se preocupar com a saída.
- `ApiKeyName`: se o site de destino tiver uma noção de uma chave de API, esse parâmetro poderá ser usado para especificar o nome (não o valor) do parâmetro de chave que deve ser usado na URL. O valor real da chave é fornecido na credencial.
- `Content`: a especificação desse valor altera a solicitação da Web de GET para POST, usando o valor do campo `Content` como o conteúdo de POST.
- `Headers`: especificar esse valor como um registro fornecerá cabeçalhos adicionais a uma solicitação HTTP.
- `Timeout`: especificar esse valor como uma duração alterará o tempo limite de uma solicitação HTTP. O valor padrão é de 100 segundos.
- `ExcludedFromCacheKey`: especificar esse valor como uma lista excluirá essas chaves de cabeçalho HTTP de fazerem parte do cálculo para armazenar dados em cache.
- `IsRetry`: a especificação desse valor lógico como verdadeiro ignorará qualquer resposta existente no cache ao buscar dados.
- `ManualStatusHandling`: especificar esse valor como uma lista impedirá qualquer manipulação interna para solicitações HTTP cuja resposta tenha um desses códigos de status.
- `RelativePath`: especificar esse valor como texto o acrescentará à URL base antes de fazer a solicitação.

Web.Page

09/05/2020 • 2 minutes to read

Sintaxe

```
Web.Page(html as any) as table
```

Sobre

Retorna o conteúdo do documento HTML decomposto em suas estruturas constitutivas, bem como uma representação do documento completo e respectivo texto após a remoção das marcas.

WebMethod.Delete

09/05/2020 • 2 minutes to read

Sobre

Especifica o método DELETE para HTTP.

WebMethod.Get

09/05/2020 • 2 minutes to read

Sobre

Especifica o método GET para HTTP.

WebMethod.Head

09/05/2020 • 2 minutes to read

Sobre

Especifica o método HEAD para HTTP.

WebMethod.Patch

09/05/2020 • 2 minutes to read

Sobre

Especifica o método PATCH para HTTP.

WebMethod.Post

09/05/2020 • 2 minutes to read

Sobre

Especifica o método POST para HTTP.

WebMethod.Put

09/05/2020 • 2 minutes to read

Sobre

Especifica o método PUT para HTTP.

Xml.Document

09/05/2020 • 2 minutes to read

Sintaxe

```
Xml.Document(contents as any, optional encoding as nullable number) as table
```

Sobre

Retorna o conteúdo do documento XML como uma tabela hierárquica.

Xml.Tables

09/05/2020 • 2 minutes to read

Sintaxe

```
Xml.Tables(contents as any, optional options as nullable record, optional encoding as nullable
number) as table
```

Sobre

Retorna o conteúdo do documento XML como uma coleção aninhada de tabelas niveladas.

Funções binárias

30/09/2020 • 7 minutes to read

Essas funções criam e manipulam dados binários.

Formatos binários

Leitura de números

FUNÇÃO	DESCRIÇÃO
BinaryFormat.7BitEncodedSignedInteger	Um formato binário que lê um inteiro com sinal de 64 bits que foi codificado usando a codificação de tamanho variável de 7 bits.
BinaryFormat.7BitEncodedUnsignedInteger	Um formato binário que lê um inteiro sem sinal de 64 bits que foi codificado usando a codificação de tamanho variável de 7 bits.
BinaryFormat.Binary	Retorna um formato binário que lê um valor binário.
BinaryFormat.Byte	Um formato binário que lê um inteiro sem sinal de 8 bits.
BinaryFormat.Choice	Retorna um formato binário que escolhe o próximo formato binário com base em um valor que já foi lido.
BinaryFormat.Decimal	Um formato binário que lê um valor decimal de 16 bytes do .NET.
BinaryFormat.Double	Um formato binário que lê um valor de ponto flutuante de precisão dupla IEEE de 8 bytes.
BinaryFormat.Group	Retorna um formato binário que lê um grupo de itens. Cada valor de item é precedido por um valor de chave exclusivo. O resultado é uma lista de valores de itens.
BinaryFormat.Length	Retorna um formato binário que limita a quantidade de dados que pode ser lida. Tanto BinaryFormat.List quanto BinaryFormat.Binary podem ser usados para fazer a leitura até o final dos dados. BinaryFormat.Length pode ser usado para limitar o número de bytes lidos.
BinaryFormat.List	Retorna um formato binário que lê uma sequência de itens e retorna uma lista.
BinaryFormat.Null	Um formato binário que lê zero bytes e retorna nulo.
BinaryFormat.Record	Retorna um formato binário que lê um registro. Cada campo do registro pode ter um formato binário diferente.
BinaryFormat.SignedInteger16	Um formato binário que lê um inteiro com sinal de 16 bits.

FUNÇÃO	DESCRIÇÃO
BinaryFormat.SignedInteger32	Um formato binário que lê um inteiro com sinal de 32 bits.
BinaryFormat.SignedInteger64	Um formato binário que lê um inteiro com sinal de 64 bits.
BinaryFormat.Single	Um formato binário que lê um valor de ponto flutuante de precisão simples IEEE de 4 bytes.
BinaryFormat.Text	Retorna um formato binário que lê um valor de texto. O valor de codificação opcional especifica a codificação do texto.
BinaryFormat.Transform	Retorna um formato binário que transformará os valores lidos por outro formato binário.
BinaryFormat.UnsignedInteger16	Um formato binário que lê um inteiro sem sinal de 16 bits.
BinaryFormat.UnsignedInteger32	Um formato binário que lê um inteiro sem sinal de 32 bits.
BinaryFormat.UnsignedInteger64	Um formato binário que lê um inteiro sem sinal de 64 bits.

CONTROLE DA ORDEM DE BYTE	DESCRIÇÃO
BinaryFormat.ByteOrder	Retorna um formato binário com a ordem de byte especificada por uma função.
Table.PartitionValues	Retorna informações sobre como uma tabela é particionada.

Binário

FUNÇÃO	DESCRIÇÃO
Binary.Buffer	Armazena em buffer o valor binário na memória. O resultado dessa chamada é um valor binário estável, o que significa que ele terá uma ordem de byte e um tamanho determinísticos.
Binary.Combine	Combina uma lista de binários em um só binário.
Binary.Compress	Compacta um valor binário usando o tipo de compactação especificado.
Binary.Decompress	Descompacta um valor binário usando o tipo de compactação especificado.
Binary.From	Retorna um valor binário do valor especificado.
Binary.FromList	Converte uma lista de números em um valor binário
Binary.FromText	Decodifica dados de um formato de texto em binário.
Binary.InferContentType	Retorna um registro com o campo Content.Type que contém o tipo MIME inferido.

FUNÇÃO	DESCRIÇÃO
Binary.Length	Retorna o tamanho dos valores binários.
Binary.ToList	Converte um valor binário em lista de números
Binary.ToText	Codifica dados binários em um formato de texto.
BinaryEncoding.Base64	Constante a ser usada como o tipo de codificação quando a codificação de Base 64 é necessária.
BinaryEncoding.Hex	Constante a ser usada como o tipo de codificação quando a codificação hexadecimal é necessária.
BinaryOccurrence.Optional	O item não deverá aparecer nenhuma vez ou deverá aparecer uma vez na entrada.
BinaryOccurrence.Repeating	O item não deverá aparecer nenhuma vez ou deverá aparecer mais vezes na entrada.
BinaryOccurrence.Required	O item deverá aparecer uma vez na entrada.
ByteOrder.BigEndian	Um valor possível para o parâmetro `byteOrder` em `BinaryFormat.ByteOrder`. O byte mais significativo aparece primeiro na ordem de byte big endian.
ByteOrder.LittleEndian	Um valor possível para o parâmetro `byteOrder` em `BinaryFormat.ByteOrder`. O byte menos significativo aparece primeiro na ordem de byte little endian.
Compression.Brotli	Os dados compactados estão no formato 'Brotli'.
Compression.Deflate	Os dados compactados estão no formato 'Deflate'.
Compression.GZip	Os dados compactados estão no formato 'GZip'.
Compression.LZ4	Os dados compactados estão no formato 'LZ4'.
Compression.None	Os dados estão descompactados.
Compression.Snappy	Os dados compactados estão no formato 'Snappy'.
Compression.Zstandard	Os dados compactados estão no formato 'Zstandard'.
Occurrence.Optional	O item não deverá aparecer nenhuma vez ou deverá aparecer uma vez na entrada.
Occurrence.Repeating	O item não deverá aparecer nenhuma vez ou deverá aparecer mais vezes na entrada.
Occurrence.Required	O item deverá aparecer uma vez na entrada.
#binary	Cria um valor binário usando números ou texto.

Binary.Buffer

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.Buffer(binary as nullable binary) as nullable binary
```

Sobre

Armazena em buffer o valor binário na memória. O resultado dessa chamada é um valor binário estável, o que significa que ele terá uma ordem de byte e um tamanho determinísticos.

Exemplo 1

Crie uma versão estável do valor binário.

```
Binary.Buffer(Binary.FromList({0..10}))
```

```
#binary({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10})
```

Binary.Combine

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.Combine(binaries as list) as binary
```

Sobre

Combina uma lista de binários em um só binário.

Binary.Compress

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.Compress(binary as nullable binary, compressionType as number) as nullable binary
```

Sobre

Compacta um valor binário usando o tipo de compactação especificado. O resultado dessa chamada é uma cópia compactada da entrada. Os tipos de compactação incluem:

- `Compression.GZip`
- `Compression.Deflate`

Exemplo 1

Comprime o valor binário.

```
Binary.Compress(Binary.FromList(List.Repeat({10}, 1000)), Compression.Deflate)
```

```
#binary({227, 226, 26, 5, 163, 96, 20, 12, 119, 0, 0})
```

Binary.Decompress

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.Decompress(binary as nullable binary, compressionType as number) as nullable binary
```

Sobre

Descompacta um valor binário usando o tipo de compactação especificado. O resultado dessa chamada é uma cópia descompactada da entrada. Os tipos de compactação incluem:

- `Compression.GZip`
- `Compression.Deflate`

Exemplo 1

Descompacte o valor binário.

```
Binary.Decompress(#binary({115, 103, 200, 7, 194, 20, 134, 36, 134, 74, 134, 84, 6, 0}), Compression.Deflate)
```

```
#binary({71, 0, 111, 0, 111, 0, 100, 0, 98, 0, 121, 0, 101, 0})
```

Binary.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.From(value as any, optional encoding as nullable number) as nullable binary
```

Sobre

Retorna um valor `binary` do `value` especificado. Se o `value` fornecido for `null`, `Binary.From` retornará `null`. Se o `value` fornecido for `binary`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `binary`:

- `text`: Um valor `binary` da representação de texto. Confira `Binary.FromText` para obter detalhes.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Obtenha o valor `binary` de `"1011"`.

```
Binary.From("1011")
```

`Binary.FromText("1011", BinaryEncoding.Base64)`

Binary.FromList

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.FromList(list as list) as binary
```

Sobre

Converte uma lista de números em um valor binário.

Binary.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.FromText(text as nullable text, optional encoding as nullable number) as nullable binary
```

Sobre

Retorna o resultado da conversão do valor de texto `text` em um binário (lista de `number`). `encoding` pode ser especificado para indicar a codificação usada no valor de texto. Os valores de `BinaryEncoding` a seguir podem ser usados para `encoding`.

- `BinaryEncoding.Base64`: Codificação base 64
- `BinaryEncoding.Hex`: Codificação hexadecimal

Exemplo 1

Decodifique `"1011"` em binário.

```
Binary.FromText("1011")
```

```
Binary.FromText("1011", BinaryEncoding.Base64)
```

Exemplo 2

Decodifique `"1011"` em binário com codificação Hexadecimal.

```
Binary.FromText("1011", BinaryEncoding.Hex)
```

```
Binary.FromText("EBE=", BinaryEncoding.Base64)
```

Binary.InferContentType

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.InferContentType(source as binary) as record
```

Sobre

Retorna um registro com o campo Content.Type que contém o tipo MIME inferido. Se o tipo de conteúdo inferido for text/* e uma página de código de codificação for detectada, retornará também o campo Content.Encoding que contém a codificação do fluxo. Se o tipo de conteúdo inferido for texto/csv e o formato for delimitado, retornará também o campo Csv.PotentialDelimiter que contém uma tabela para análise de possíveis delimitadores. Se o tipo de conteúdo inferido for texto/csv e o formato for de largura fixa, retornará também o campo Csv.PotentialPositions contendo uma lista para a análise de posições de coluna de largura fixa potencial.

Binary.Length

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.Length(binary as nullable binary) as nullable number
```

Sobre

Retorna o número de caracteres.

Binary.ToList

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.ToList(binary as binary) as list
```

Sobre

Converte um valor binário em lista de números.

Binary.ToText

09/05/2020 • 2 minutes to read

Sintaxe

```
Binary.ToText(binary as nullable binary, optional encoding as nullable number) as nullable text
```

Sobre

Retorna o resultado da conversão de uma lista binária de números `binary` em um valor de texto. Opcionalmente, `encoding` pode ser especificado para indicar que a codificação a ser usada no valor de texto produzido. Os seguintes valores `BinaryEncoding` podem ser usados para `encoding`.

- `BinaryEncoding.Base64`: Codificação base 64
- `BinaryEncoding.Hex`: Codificação hexadecimal

BinaryEncoding.Base64

09/05/2020 • 2 minutes to read

Sobre

Constante a ser usada como o tipo de codificação quando a codificação de Base 64 é necessária.

BinaryEncoding.Hex

09/05/2020 • 2 minutes to read

Sobre

Constante a ser usada como o tipo de codificação quando a codificação hexadecimal é necessária.

BinaryFormat.7BitEncodedSignedInteger

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.7BitEncodedSignedInteger(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro com sinal de 64 bits que foi codificado usando a codificação de tamanho variável de 7 bits.

BinaryFormat.7BitEncodedUnsignedInteger

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.7BitEncodedUnsignedInteger(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro sem sinal de 64 bits que foi codificado usando a codificação de tamanho variável de 7 bits.

BinaryFormat.Binary

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Binary(optional length as any) as function
```

Sobre

Retorna um formato binário que lê um valor binário. Se `length` for especificado, o valor binário conterá esse número de bytes. Se `length` não for especificado, o valor binário conterá os bytes restantes. O `length` pode ser especificado como um número ou como um formato binário do tamanho que precede os dados binários.

BinaryFormat.Byte

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Byte(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro sem sinal de 8 bits.

BinaryFormat.ByteOrder

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.ByteOrder(binaryFormat as function, byteOrder as number) as function
```

Sobre

Retorna um formato binário com a ordem de byte especificada por `binaryFormat`. A ordem de byte padrão é `ByteOrder.BigEndian`.

BinaryFormat.Choice

09/05/2020 • 3 minutes to read

Sintaxe

```
BinaryFormat.Choice(binaryFormat as function, chooseFunction as function, optional type as
nullable type, optional combineFunction as nullable function) as function
```

Sobre

Retorna um formato binário que escolhe o próximo formato binário com base em um valor que já foi lido. O valor de formato binário produzido por essa função funciona em estágios:

- O formato binário especificado pelo parâmetro `binaryFormat` é usado para ler um valor.
- O valor é passado para a função choice especificada pelo parâmetro `chooseFunction`.
- A função choice inspeciona o valor e retorna um segundo formato binário.
- O segundo formato binário é usado para ler um segundo valor.
- Se a função combine for especificada, o primeiro e o segundo valores serão passados para a função combine e o valor resultante será retornado.
- Se a função combine não for especificada, o segundo valor será retornado.
- O segundo valor é retornado.

O parâmetro opcional `type` indica o tipo de formato binário que será retornado pela função choice. `type any`, `type list` ou `type binary` pode ser especificado. Se o parâmetro `type` não for especificado, `type any` será usado. Se `type list` ou `type binary` for usado, o sistema poderá retornar um valor `binary` ou `list` de streaming, em vez de um armazenado em buffer, o que pode reduzir a quantidade de memória necessária para ler o formato.

Exemplo 1

Leia uma lista de bytes em que o número de elementos é determinado pelo primeiro byte.

```
let
    binaryData = #binary({2, 3, 4, 5}),
    listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) => BinaryFormat.List(BinaryFormat.Byte, length)
    )
in
    listFormat(binaryData)
```

```
3
```

```
4
```

Exemplo 2

Leia uma lista de bytes em que o número de elementos seja determinado pelo primeiro byte e preserve o primeiro byte lido.

```
let
    binaryData = #binary({2, 3, 4, 5}),
    listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) => BinaryFormat.Record([
            length = length,
            list = BinaryFormat.List(BinaryFormat.Byte, length)
        ])
    )
in
    listFormat(binaryData)
```

COMPRIMENTO	2
LISTA	[Lista]

Exemplo 3

Leia uma lista de bytes em que o número de elementos seja determinado pelo primeiro byte usando uma lista de fluxo.

```
let
    binaryData = #binary({2, 3, 4, 5}),
    listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) => BinaryFormat.List(BinaryFormat.Byte, length),
        type list
    )
in
    listFormat(binaryData)
```

3
4

BinaryFormat.Decimal

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Decimal(binary as binary) as any
```

Sobre

Um formato binário que lê um valor decimal de 16 bytes do .NET.

BinaryFormat.Double

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Double(binary as binary) as any
```

Sobre

Um formato binário que lê um valor de ponto flutuante de precisão dupla IEEE de 8 bytes.

BinaryFormat.Group

09/05/2020 • 5 minutes to read

Sintaxe

```
BinaryFormat.Group(binaryFormat as function, group as list, optional extra as nullable function,
optional lastKey as any) as function
```

Sobre

Os parâmetros são os seguintes:

- O parâmetro `binaryFormat` especifica o formato binário do valor de chave.
- O parâmetro `group` fornece informações sobre o grupo de itens conhecidos.
- O parâmetro opcional `extra` pode ser usado para especificar uma função que retornará um valor de formato binário para o valor após qualquer chave inesperada. Se o parâmetro `extra` não for especificado, um erro será gerado se houver valores de chave inesperados.

O parâmetro `group` especifica uma lista de definições de item. Cada definição de item é uma lista contendo de três a cinco valores, como segue:

- Valor da chave. O valor da chave que corresponde ao item. Deve ser exclusivo dentro do conjunto de itens.
- Formato do item. O formato binário correspondente ao valor do item. Isso permite que cada item tenha um formato diferente.
- Ocorrência do item. O valor `BinaryOccurrence.Type` para quantas vezes o item deve aparecer no grupo. Os itens necessários que não estão presentes causam um erro. Itens duplicados necessários ou opcionais são tratados como valores de chave inesperados.
- Valor do item padrão (opcional). Se o valor do item padrão for exibido na lista definição de item e não for nulo, ele será usado em vez do padrão. O padrão para itens repetidos ou opcionais é NULL e o padrão para valores repetitivos é uma lista vazia { }.
- Transformação de valor de item (opcional). Se a função de transformação valor do item estiver presente na lista definição de item e não for nula, ela será chamada para transformar o valor do item antes que ele seja retornado. A função de transformação só será chamada se o item aparecer na entrada (nunca será chamada com o valor padrão).

Exemplo 1

O seguinte pressupõe um valor de chave que seja um único byte, com quatro itens esperados no grupo, todos com um byte de dados após a chave. Os itens aparecem na entrada da seguinte maneira:

- A chave 1 é necessária e é exibida com o valor 11.
- A chave 2 é repetida e aparece duas vezes com o valor 22 e resulta em um valor de { 22, 22 }.
- A chave 3 é opcional e não aparece e resulta em um valor de null.
- A chave 4 repete-se, mas não aparece, e resulta em um valor de { }.
- A chave 5 não faz parte do grupo, mas é exibida uma vez com o valor 55. A função extra é chamada com o valor de chave 5 e retorna o formato correspondente a esse valor (BinaryFormat.Byte). O valor 55 é lido e descartado.

```
let
    b = #binary({
        1, 11,
        2, 22,
        2, 22,
        5, 55,
        1, 11
    }),
    f = BinaryFormat.Group(
        BinaryFormat.Byte,
        {
            {1, BinaryFormat.Byte, BinaryOccurrence.Required},
            {2, BinaryFormat.Byte, BinaryOccurrence.Repeating},
            {3, BinaryFormat.Byte, BinaryOccurrence.Optional},
            {4, BinaryFormat.Byte, BinaryOccurrence.Repeating}
        },
        (extra) => BinaryFormat.Byte
    )
in
    f(b)
```

11
[Lista]
[Lista]

Exemplo 2

O exemplo a seguir ilustra a transformação de valor de item e o valor de item padrão. O item repetido com a chave 1 soma a lista de valores lidos usando List.Sum. O item opcional com a chave 2 tem um valor padrão de 123, em vez de null.

```
let
    b = #binary({
        1, 101,
        1, 102
    }),
    f = BinaryFormat.Group(
        BinaryFormat.Byte,
        {
            {1, BinaryFormat.Byte, BinaryOccurrence.Repeating,
              0, (list) => List.Sum(list)},
            {2, BinaryFormat.Byte, BinaryOccurrence.Optional, 123}
        }
    )
in
    f(b)
```

203
123

BinaryFormat.Length

08/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Length(binaryFormat as function, length as any) as function
```

Sobre

Retorna um formato binário que limita a quantidade de dados que pode ser lida. Tanto `BinaryFormat.List` quanto `BinaryFormat.Binary` podem ser usados para ler até o final dos dados. `BinaryFormat.Length` pode ser usado para limitar o número de bytes lidos. O parâmetro `binaryFormat` especifica o formato binário a ser limitado. O parâmetro `length` especifica o número de bytes a serem lidos. O parâmetro `length` pode ser um valor numérico ou um valor de formato binário que especifica o formato do valor de comprimento que aparece e que precede o valor que está sendo lido.

Exemplo 1

Limite o número de bytes lido a 2 ao ler uma lista de bytes.

```
let
    binaryData = #binary({1, 2, 3}),
    listFormat = BinaryFormat.Length(
        BinaryFormat.List(BinaryFormat.Byte),
        2
    )
in
    listFormat(binaryData)
```

```
1
```

```
2
```

Exemplo 2

Limite o número de bytes lidos durante a leitura de uma lista de bytes para o valor de byte anterior na lista.

```
let
    binaryData = #binary({1, 2, 3}),
    listFormat = BinaryFormat.Length(
        BinaryFormat.List(BinaryFormat.Byte),
        BinaryFormat.Byte
    )
in
    listFormat(binaryData)
```

```
2
```

BinaryFormat.List

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.List(binaryFormat as function, optional countOrCondition as any) as function
```

Sobre

Retorna um formato binário que lê uma sequência de itens e retorna uma `list`. O parâmetro `binaryFormat` especifica o formato binário de cada item. Há três maneiras de determinar o número de itens lidos:

- Se o `countOrCondition` não for especificado, o formato binário será lido até que não haja mais itens.
- Se o `countOrCondition` for um número, o formato binário lerá aquele número de itens.
- Se o `countOrCondition` for uma função, essa função será invocada para cada item lido. A função retorna true para continuar e false para interromper a leitura de itens. O item final está incluído na lista.
- Se a countOrCondition for um formato binário, a contagem de itens deverá anteceder à lista e o formato especificado será usado para ler a contagem.

Exemplo 1

Bytes lidos até o fim dos dados.

```
let
    binaryData = #binary({1, 2, 3}),
    listFormat = BinaryFormat.List(BinaryFormat.Byte)
in
    listFormat(binaryData)
```

1
2
3

Exemplo 2

Dois bytes lidos.

```
let
    binaryData = #binary({1, 2, 3}),
    listFormat = BinaryFormat.List(BinaryFormat.Byte, 2)
in
    listFormat(binaryData)
```

1

```
2
```

Exemplo 3

Bytes lidos até que o valor de byte seja maior ou igual a dois.

```
let
    binaryData = #binary({1, 2, 3}),
    listFormat = BinaryFormat.List(BinaryFormat.Byte, (x) => x < 2)
in
    listFormat(binaryData)
```

```
1
```

```
2
```

BinaryFormat.Null

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Null(binary as binary) as any
```

Sobre

Um formato binário que lê zero bytes e retorna nulo.

BinaryFormat.Record

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Record(record as record) as function
```

Sobre

Retorna um formato binário que lê um registro. O parâmetro `record` especifica o formato do registro. Cada campo do registro pode ter um formato binário diferente. Se um campo contiver um valor que não seja um valor de formato binário, nenhum dado será lido para esse campo e o valor do campo será ecoado para o resultado.

Exemplo 1

Leia um registro contendo um número inteiro de 16 bits e um número inteiro de 32 bits.

```
let
    binaryData = #binary({
        0x00, 0x01,
        0x00, 0x00, 0x00, 0x02
    }),
    recordFormat = BinaryFormat.Record([
        A = BinaryFormat.UnsignedInteger16,
        B = BinaryFormat.UnsignedInteger32
    ])
in
    recordFormat(binaryData)
```

A	1
B	2

BinaryFormat.SignedInteger16

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.SignedInteger16(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro com sinal de 16 bits.

BinaryFormat.SignedInteger32

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.SignedInteger32(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro com sinal de 32 bits.

BinaryFormat.SignedInteger64

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.SignedInteger64(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro com sinal de 64 bits.

BinaryFormat.Single

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Single(binary as binary) as any
```

Sobre

Um formato binário que lê um valor de ponto flutuante de precisão simples IEEE de 4 bytes.

BinaryFormat.Text

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Text(length as any, optional encoding as nullable number) as function
```

Sobre

Retorna um formato binário que lê um valor de texto. O `length` especifica o número de bytes a serem decodificados ou o formato binário do comprimento que precede o texto. O valor de `encoding` opcional especifica a codificação do texto. Se o `encoding` não for especificado, a codificação será determinada das marcas de ordem de byte Unicode. Se nenhuma marca de ordem de byte estiver presente, `TextEncoding.Utf8` será usado.

Exemplo 1

Decodifique dois bytes como texto ASCII.

```
let
    binaryData = #binary({65, 66, 67}),
    textFormat = BinaryFormat.Text(2, TextEncoding.Ascii)
in
    textFormat(binaryData)
```

`"AB"`

Exemplo 2

Decodifique o texto ASCII em que o comprimento do texto em bytes aparece antes do texto como um byte.

```
let
    binaryData = #binary({2, 65, 66}),
    textFormat = BinaryFormat.Text(
        BinaryFormat.Byte,
        TextEncoding.Ascii
    )
in
    textFormat(binaryData)
```

`"AB"`

BinaryFormat.Transform

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.Transform(binaryFormat as function, function as function) as function
```

Sobre

Retorna um formato binário que transformará os valores lidos por outro formato binário. O parâmetro `binaryFormat` especifica o formato binário que será usado para ler o valor. O `function` é invocado com o valor lido e retorna o valor transformado.

Exemplo 1

Leia um byte e adicione-o a ele.

```
let
    binaryData = #binary({1}),
    transformFormat = BinaryFormat.Transform(
        BinaryFormat.Byte,
        (x) => x + 1
    )
in
    transformFormat(binaryData)
```

`2`

BinaryFormat.UnsignedInteger16

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.UnsignedInteger16(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro sem sinal de 16 bits.

BinaryFormat.UnsignedInteger32

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.UnsignedInteger32(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro sem sinal de 32 bits.

BinaryFormat.UnsignedInteger64

09/05/2020 • 2 minutes to read

Sintaxe

```
BinaryFormat.UnsignedInteger64(binary as binary) as any
```

Sobre

Um formato binário que lê um inteiro sem sinal de 64 bits.

BinaryOccurrence.Optional

09/05/2020 • 2 minutes to read

Sobre

O item não deverá aparecer nenhuma vez ou deverá aparecer uma vez na entrada.

BinaryOccurrence.Repeating

09/05/2020 • 2 minutes to read

Sobre

O item não deverá aparecer nenhuma vez ou deverá aparecer mais vezes na entrada.

BinaryOccurrence.Required

09/05/2020 • 2 minutes to read

Sobre

O item deverá aparecer uma vez na entrada.

ByteOrder.BigEndian

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro `byteOrder` em `BinaryFormat.ByteOrder`. O byte mais significativo aparece primeiro na ordem de byte big endian.

ByteOrder.LittleEndian

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro `byteOrder` em `BinaryFormat.ByteOrder`. O byte menos significativo aparece primeiro na ordem de byte little endian.

Compression.Brotli

30/09/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'Brotli'.

Compression.Deflate

09/05/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'Deflate'.

Compression.GZip

09/05/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'GZip'.

Compression.LZ4

30/09/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'LZ4'.

Compression.None

30/09/2020 • 2 minutes to read

Sobre

Os dados estão descompactados.

Compression.Snappy

30/09/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'Snappy'.

Compression.Zstandard

30/09/2020 • 2 minutes to read

Sobre

Os dados compactados estão no formato 'Zstandard'.

Occurrence.Optional

09/05/2020 • 2 minutes to read

Sobre

O item não deverá aparecer nenhuma vez ou deverá aparecer uma vez na entrada.

Occurrence.Repeating

09/05/2020 • 2 minutes to read

Sobre

O item não deverá aparecer nenhuma vez ou deverá aparecer mais vezes na entrada.

Occurrence.Required

09/05/2020 • 2 minutes to read

Sobre

O item deverá aparecer uma vez na entrada.

#binary

09/05/2020 • 2 minutes to read

Sintaxe

```
#binary(value as any) as any
```

Sobre

Cria um valor binário usando uma lista de números ou um valor de texto codificado em base 64.

Exemplo 1

Criar um valor binário usando uma lista de números.

```
#binary({0x30, 0x31, 0x32})
```

`Text.ToBinary("012")`

Exemplo 2

Criar um valor binário usando um valor de texto codificado em base 64.

```
#binary("1011")
```

`Binary.FromText("1011", BinaryEncoding.Base64)`

Funções de combinação

08/05/2020 • 2 minutes to read

Essas funções são usadas por outras funções de biblioteca que mesclam valores. Por exemplo, `Table.ToList` e `Table.CombineColumns` aplicam uma função de combinação a cada linha em uma tabela para produzir um valor único para cada linha.

Combinador

FUNÇÃO	DESCRIÇÃO
Combiner.CombineTextByDelimiter	Retorna uma função que combina uma lista de texto em texto único usando os delimitadores especificados.
Combiner.CombineTextByEachDelimiter	Retorna uma função que combina uma lista de texto em texto único usando cada delimitador especificado na sequência.
Combiner.CombineTextByLengths	Retorna uma função que combina uma lista de texto em texto único usando os tamanhos especificados.
Combiner.CombineTextByPositions	Retorna uma função que combina uma lista de texto em texto único usando as posições especificadas.
Combiner.CombineTextByRanges	Retorna uma função que combina uma lista de texto em texto único usando as posições e os tamanhos especificados.

Combiner.CombineTextByDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Combiner.CombineTextByDelimiter(delimiter as text, optional quoteStyle as nullable number) as
function
```

Sobre

Retorna uma função que combina uma lista de texto em texto único usando os delimitadores especificados.

Combiner.CombineTextByEachDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Combiner.CombineTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number) as function
```

Sobre

Retorna uma função que combina uma lista de texto em texto único usando cada delimitador especificado na sequência.

Combiner.CombineTextByLengths

09/05/2020 • 2 minutes to read

Sintaxe

```
Combiner.CombineTextByLengths(lengths as list, optional template as nullable text) as function
```

Sobre

Retorna uma função que combina uma lista de texto em texto único usando os tamanhos especificados.

Combiner.CombineTextByPositions

09/05/2020 • 2 minutes to read

Sintaxe

```
Combiner.CombineTextByPositions(positions as list, optional template as nullable text) as function
```

Sobre

Retorna uma função que combina uma lista de texto em texto único usando as posições especificadas.

Combiner.CombineTextByRanges

09/05/2020 • 2 minutes to read

Sintaxe

```
Combiner.CombineTextByRanges(ranges as list, optional template as nullable text) as function
```

Sobre

Retorna uma função que combina uma lista de texto em texto único usando as posições e os tamanhos especificados.

Funções de comparação

08/05/2020 • 2 minutes to read

Essas funções testam a igualdade e determinam a ordem.

Comparador

FUNÇÃO	DESCRIÇÃO
Comparer.Equals	Retorna um valor lógico baseado na verificação de igualdade sobre os dois valores fornecidos.
Comparer.FromCulture	Retorna uma função de comparador, dada a cultura e um valor lógico para diferenciação de maiúsculas e minúsculas para a comparação. O valor padrão para ignoreCase é false. O valor de cultura são representações de texto conhecidas de localidades usadas no .NET Framework.
Comparer.Ordinal	Retorna uma função de comparador que usa regras do Ordinal para comparar valores.
Comparer.OrdinalIgnoreCase	Retorna uma função de comparador sem diferenciação de maiúsculas e minúsculas que usa regras Ordinais para comprar os valores fornecidos x e y.
Culture.Current	Retorna a cultura atual do sistema.

Comparer.Equals

09/05/2020 • 2 minutes to read

Sintaxe

```
Comparer.Equals(comparer as function, x as any, y as any) as logical
```

Sobre

Retorna um valor `logical` com base na verificação de igualdade sobre os dois valores especificados, `x` e `y`, usando o `comparer` especificado.

`comparer` é um `Comparer` usado para controlar a comparação. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Comparar "1" e "A" usando a localidade "pt-BR" para determinar se os valores são iguais.

```
Comparer.Equals(Comparer.FromCulture("en-us"), "1", "A")
```

`false`

Comparer.FromCulture

09/05/2020 • 2 minutes to read

Sintaxe

```
Comparer.FromCulture(culture as text, optional ignoreCase as nullable logical) as function
```

Sobre

Retorna uma função de comparador, dada a `culture` e um valor lógico `ignoreCase` para diferenciação de maiúsculas e minúsculas para a comparação. O valor padrão para `ignoreCase` é false. O valor de cultura são representações de texto conhecidas de localidades usadas no .NET Framework.

Exemplo 1

Comparar "a" e "A" usando a localidade "pt-BR" para determinar se os valores são iguais.

```
Comparer.FromCulture("en-us")("a", "A")
```

`-1`

Exemplo 2

Comparar "a" e "A" usando a localidade "pt-BR", ignorando maiúsculas e minúsculas, para determinar se os valores são iguais.

```
Comparer.FromCulture("en-us", true)("a", "A")
```

`0`

Comparer.Ordinal

09/05/2020 • 2 minutes to read

Sintaxe

```
Comparer.Ordinal(x as any, y as any) as number
```

Sobre

Retorna uma função de comparador que usa regras Ordinais para comparar os valores `x` e `y` fornecidos.

Exemplo 1

Usando regras Ordinais, compare se "encyclopædia" e "encyclopaedia" são equivalentes. Observe que elas são equivalentes usando `Comparer.FromCulture("en-us")`.

```
Comparer.Equals(Comparer.Ordinal, "encyclopædia", "encyclopaedia")
```

`false`

Comparer.OrdinalIgnoreCase

09/05/2020 • 2 minutes to read

Sintaxe

```
Comparer.OrdinalIgnoreCase(x as any, y as any) as number
```

Sobre

Retorna uma função de comparador sem diferenciação de maiúsculas e minúsculas que usa regras Ordinais para comprar os valores fornecidos `x` e `y`.

Exemplo

Usando regras ordinais que não diferenciam maiúsculas de minúsculas, compare "Abc" com "abc". Observação: "Abc" é menor que "abc" usando `Comparer.Ordinal`.

```
Comparer.OrdinalIgnoreCase("Abc", "abc")
```

```
0
```

Culture.Current

09/05/2020 • 2 minutes to read

Sobre

Retorna o nome da cultura atual do aplicativo.

Funções de data

08/05/2020 • 11 minutes to read

Essas funções criam e manipulam o componente de data dos valores date, datetime e datetimezone.

Date

FUNÇÃO	DESCRIÇÃO
Date.AddDays	Retorna um valor de Date/DateTime/DateTimeZone com a parte de dia incrementada pelo número de dias fornecido. Também cuida do incremento das partes de mês e ano do valor, conforme apropriado.
Date.AddMonths	Retorna um valor de DateTime com a parte de mês incrementada por n meses.
Date.AddQuarters	Retorna um valor de Date/DateTime/DateTimeZone incrementado pelo número de trimestres fornecido. Cada trimestre é definido como uma duração de três meses. Também cuida do incremento da parte de ano do valor, conforme apropriado.
Date.AddWeeks	Retorna um valor de Date/DateTime/DateTimeZone incrementado pelo número de semanas fornecido. Cada semana é definida como uma duração de sete dias. Também cuida do incremento das partes de mês e ano do valor, conforme apropriado.
Date.AddYears	Retorna um valor de DateTime com a parte de ano incrementada por n anos.
Date.Day	Retorna o dia de um valor de DateTime.
Date.DayOfWeek	Retorna um número (de 0 a 6) que indica o dia da semana do valor fornecido.
Date.DayOfWeekName	Retorna o nome do dia da semana.
Date.DayOfYear	Retorna um número que representa o dia do ano de um valor de DateTime.
Date.DaysInMonth	Retorna o número de dias no mês de um valor de DateTime.
Date.EndOfDay	Retorna um valor de DateTime para o final do dia.
Date.EndOfMonth	Retorna um valor de DateTime para o final do mês.
Date.EndOfQuarter	Retorna um valor de Date/DateTime/DateTimeZone que representa o final do trimestre. As partes de data e hora são redefinidas para os valores de término para o trimestre. As informações de fuso horário são persistidas.

FUNÇÃO	DESCRIÇÃO
Date.EndOfWeek	Retorna um valor de DateTime para o fim da semana.
Date.EndOfYear	Retorna um valor de DateTime para o final do ano.
Date.From	Retorna um valor de data com base em um valor.
Date.FromText	Retorna um valor de Date de um conjunto de formatos de data e valor de cultura.
Date.IsInCurrentDay	Indica se o valor de datetime `dateTime` especificado ocorre durante o dia atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInCurrentMonth	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o mês atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInCurrentQuarter	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o trimestre atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInCurrentWeek	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante a semana atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInCurrentYear	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o ano atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextDay	Indica se o valor de datetime especificado dateTime ocorre durante o dia seguinte, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextMonth	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o mês seguinte, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextNDays	Indica se o valor de datetime especificado dateTime ocorre durante o próximo número de dias, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextNMonths	Indica se o valor de datetime especificado dateTime ocorre durante o próximo número de meses, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextNQuarters	Indica se o valor de datetime especificado dateTime ocorre durante o próximo número de trimestres, conforme determinado pela data e pela hora atuais do sistema.

FUNÇÃO	DESCRIÇÃO
Date.IsInNextNWeeks	Indica se o valor de datetime especificado dateTime ocorre durante o próximo número de semanas, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextNYears	Indica se o valor de datetime especificado dateTime ocorre durante o próximo número de anos, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextQuarter	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o próximo trimestre, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextWeek	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante a próxima semana, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInNextYear	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o ano seguinte, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousDay	Indica se o valor de datetime especificado dateTime ocorre durante o dia anterior, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousMonth	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o mês anterior, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousNDays	Indica se o valor de datetime especificado dateTime ocorre durante o número anterior de dias, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousNMonths	Indica se o valor de datetime especificado dateTime ocorre durante o número anterior de meses, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousNQuarters	Indica se o valor de datetime especificado dateTime ocorre durante o número anterior de trimestres, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousNWeeks	Indica se o valor de datetime especificado dateTime ocorre durante o número anterior de semanas, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousNYears	Indica se o valor de datetime especificado dateTime ocorre durante o número anterior de anos, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousQuarter	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o trimestre anterior, conforme determinado pela data e pela hora atuais do sistema.

FUNÇÃO	DESCRIÇÃO
Date.IsInPreviousWeek	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante a semana anterior, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInPreviousYear	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu durante o ano anterior, conforme determinado pela data e pela hora atuais do sistema.
Date.IsInYearToDate	Retorna um valor lógico que indica se o Date/DateTime/DateTimeZone especificado ocorreu no período que começa em 1º de janeiro do ano atual e termina no dia atual, conforme determinado pela data e pela hora atuais do sistema.
Date.IsLeapYear	Retorna um valor lógico que indica se a parte de ano de um valor de DateTime é um ano bissexto.
Date.Month	Retorna o mês de um valor de DateTime.
Date.MonthName	Retorna o componente do nome do mês.
Date.QuarterOfYear	Retorna um número entre 1 e 4 para o trimestre do ano de um valor de DateTime.
Date.StartOfDay	Retorna um valor de DateTime para o início do dia.
Date.StartOfMonth	Retorna um valor de DateTime que representa o início do mês.
Date.StartOfQuarter	Retorna um valor de DateTime que representa o início do trimestre.
Date.StartOfWeek	Retorna um valor de DateTime que representa o início da semana.
Date.StartOfYear	Retorna um valor de DateTime que representa o início do ano.
Date.ToRecord	Retorna um registro que contém partes de valor de Date.
Date.ToText	Retorna um valor de texto de um valor de Date.
Date.WeekOfMonth	Retorna um número para a contagem da semana no mês atual.
Date.WeekOfYear	Retorna um número para a contagem da semana no ano atual.
Date.Year	Retorna o ano de um valor de DateTime.
#date	Cria um valor de data do ano, do mês e do dia.

VALORES DE PARÂMETROS	DESCRIÇÃO
Day.Sunday	Representa o domingo.
Day.Monday	Representa a segunda-feira.
Day.Tuesday	Representa a terça-feira.
Day.Wednesday	Representa a quarta-feira.
Day.Thursday	Representa a quinta-feira.
Day.Friday	Representa a sexta-feira.
Day.Saturday	Representa o sábado.

Date.AddDays

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.AddDays(dateTime as any, numberOfDays as number) as any
```

Sobre

Retorna o resultado `date`, `datetime` ou `datetimezone` de adicionar `numberOfDays` dias ao valor `datetime`., `dateTime`.

- `dateTime`: O valor `date`, `datetime` ou `datetimezone` ao qual os dias estão sendo adicionado.
- `numberOfDays`: O número de dias a serem adicionados.

Exemplo 1

Adicione cinco dias ao valor `date`, `datetime` ou `datetimezone` que representa a data 14/05/2011.

```
Date.AddDays(#date(2011, 5, 14), 5)
```

`#date(2011, 5, 19)`

Date.AddMonths

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.AddMonths(dateTime as any, numberOfMonths as number) as any
```

Sobre

Retorna o resultado `date`, `datetime` ou `datetimezone` de adicionar `numberOfMonths` meses ao valor `datetime`, `dateTime`.

- `dateTime`: O valor `date`, `datetime` ou `datetimezone` ao qual os meses estão sendo adicionado.
- `numberOfMonths`: O número de meses a serem adicionados.

Exemplo 1

Adicione cinco meses ao valor `date`, `datetime` ou `datetimezone` que representa a data 14/05/2011.

```
Date.AddMonths(#date(2011, 5, 14), 5)
```

`#date(2011, 10, 14)`

Exemplo 2

Adicione 18 meses ao valor `date`, `datetime` ou `datetimezone` que representa a data e a hora de 14/05/2011 08:15:22 AM.

```
Date.AddMonths(#datetime(2011, 5, 14, 8, 15, 22), 18)
```

`#datetime(2012, 11, 14, 8, 15, 22)`

Date.AddQuarters

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.AddQuarters(dateTime as any, numberOfQuarters as number) as any
```

Sobre

Retorna o resultado `date`, `datetime` ou `datetimezone` de adicionar `numberOfQuarters` trimestres ao valor `datetime`., `dateTime`.

- `dateTime`: O valor `date`, `datetime` ou `datetimezone` ao qual os trimestres estão sendo adicionado.
- `numberOfQuarters`: O número de trimestres a serem adicionados.

Exemplo 1

Adicione um trimestre ao valor `date`, `datetime` ou `datetimezone` que representa a data 14/05/2011.

```
Date.AddQuarters(#date(2011, 5, 14), 1)
```

`#date(2011, 8, 14)`

Date.AddWeeks

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.AddWeeks(dateTime as any, numberOfWeeks as number) as any
```

Sobre

Retorna o resultado `date`, `datetime` ou `datetimezone` de adicionar `numberOfWeeks` semanas ao valor `datetime`., `dateTime`.

- `dateTime`: O valor `date`, `datetime` ou `datetimezone` ao qual as semanas estão sendo adicionadas.
- `numberOfWeeks`: O número de semanas a serem adicionadas.

Exemplo 1

Adicione duas semanas ao valor `date`, `datetime` ou `datetimezone` que representa a data 14/05/2011.

```
Date.AddWeeks(#date(2011, 5, 14), 2)
```

`#date(2011, 5, 28)`

Date.AddYears

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.AddYears(dateTime as any, numberOfYears as number) as any
```

Sobre

Retorna o resultado `date`, `datetime` ou `datetimezone` da adição de `numberOfYears` a um valor `datetime`, `dateTime`.

- `dateTime`: O valor `date`, `datetime` ou `datetimezone` ao qual os anos são adicionados.
- `numberOfYears`: O número de anos a serem adicionados.

Exemplo 1

Adicione quatro anos ao valor `date`, `datetime` ou `datetimezone` que representa a data 14/05/2011.

```
Date.AddYears(#date(2011, 5, 14), 4)
```

`#date(2015, 5, 14)`

Exemplo 2

Adicione 10 anos ao valor `date`, `datetime` ou `datetimezone` que representa a data e a hora de 14/05/2011 08:15:22 AM.

```
Date.AddYears(#datetime(2011, 5, 14, 8, 15, 22), 10)
```

`#datetime(2021, 5, 14, 8, 15, 22)`

Date.Day

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.Day(dateTime as any) as nullable number
```

Sobre

Retorna o componente de dia de um valor `date`, `datetime` ou `datetimezone`.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` do qual o componente de dia é extraído.

Exemplo 1

Obtém o componente de dia de um valor `date`, `datetime` ou `datetimezone` que representa a data e a hora de 14/-5/2011 05:00:00 PM.

```
Date.Day(#datetime(2011, 5, 14, 17, 0, 0))
```

`14`

Date.DayOfWeek

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.DayOfWeek(dateTime as any, optional firstDayOfWeek as nullable number) as nullable number
```

Sobre

Retorna um número (de 0 a 6) que indica o dia da semana do `dateTime` fornecido.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone`.
- `firstDayOfWeek`: Um valor `Day` indicando qual dia deve ser considerado o primeiro dia da semana. Os valores permitidos são Day.Sunday, Day.Monday, Day.Tuesday, Day.Wednesday, Day.Thursday, Day.Friday ou Day.Saturday. Se não for especificado, um padrão dependente de cultura será usado.

Exemplo 1

Obter o dia da semana representado por segunda-feira, 21 de fevereiro de 2011, tratando o domingo como o primeiro dia da semana.

```
Date.DayOfWeek(#date(2011, 02, 21), Day.Sunday)`
```

`1`

Exemplo 2

Obter o dia da semana representado por segunda-feira, 21 de fevereiro de 2011, tratando a segunda-feira como o primeiro dia da semana.

```
Date.DayOfWeek(#date(2011, 02, 21), Day.Monday)
```

`0`

Date.DayOfWeekName

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.DayOfWeekName(date as any, optional culture as nullable text)
```

Sobre

Retorna o nome do dia da semana para a `date` fornecida. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha o nome do dia da semana.

```
Date.DayOfWeekName(#date(2011, 12, 31), "en-US")
```

`"Saturday"`

Date.DayOfYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.DayOfYear(dateTime as any) as nullable number
```

Sobre

Retorna um número que representa o dia do ano no valor `date`, `datetime` ou `datetimezone` fornecido, `dateTime`.

Exemplo 1

O número do dia 1º de março de 2011 (`#date(2011, 03, 01)`).

```
Date.DayOfYear(#date(2011, 03, 01))
```

`60`

Date.DaysInMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.DaysInMonth(dateTime as any) as nullable number
```

Sobre

Retorna o número de dias do mês no valor de `dateTime` de `date`, `datetime` ou `datetimezone`.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` para o qual o número de dias do mês é retornado.

Exemplo 1

Número de dias do mês de dezembro, conforme representado por `#date(2011, 12, 01)`.

```
Date.DaysInMonth(#date(2011, 12, 01))
```

`31`

Date.EndOfDay

09/05/2020 • 2 minutes to read

```
Date.EndOfDay(dateTime as any) as any
```

Sobre

Retorna um valor `date`, `datetime` ou `datetimezone` que representa o fim do dia em `dateTime`. As informações de fuso horário são preservadas.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` do qual o fim do dia é calculado.

Exemplo 1

Obtenha o fim do dia de 14/5/2011 05:00: 00 PM.

```
Date.EndOfDay(#datetime(2011, 5, 14, 17, 0, 0))
```

```
#datetime(2011, 5, 14, 23, 59, 59.9999999)
```

Exemplo 2

Obtenha o fim do dia de 17/5/2011 05:00: 00 PM -7:00.

```
Date.EndOfDay(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
```

```
#datetimezone(2011, 5, 17, 23, 59, 59.9999999, -7, 0)
```

Date.EndOfMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.EndOfMonth(dateTime as any) as any
```

Sobre

Retorna o último dia do mês em `dateTime`.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` do qual o fim do mês é calculado

Exemplo 1

Obter o fim do mês de 14/5/2011.

```
Date.EndOfMonth(#date(2011, 5, 14))
```

`#date(2011, 5, 31)`

Exemplo 2

Obter o fim do mês de 17/5/2011 17h – 7:00.

```
Date.EndOfMonth(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
```

`#datetimezone(2011, 5, 31, 23, 59, 59.9999999, -7, 0)`

Date.EndOfQuarter

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.EndOfQuarter(dateTime as any) as any
```

Sobre

Retorna um valor `date`, `datetime` ou `datetimezone` que representa o fim do trimestre em `dateTime`. As informações de fuso horário são preservadas.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` do qual o fim do trimestre é calculado.

Exemplo 1

Localizar o fim do trimestre de 10 de outubro de 2011, 8:00 AM (`#datetime(2011, 10, 10, 8, 0, 0)`).

```
Date.EndOfQuarter(#datetime(2011, 10, 10, 8, 0, 0))
```

`#datetime(2011, 12, 31, 23, 59, 59.9999999)`

Date.EndOfWeek

12/05/2020 • 2 minutes to read

Sintaxe

```
Date.EndOfWeek(dateTime as any, optional firstDayOfWeek as nullable number) as any
```

Sobre

Retorna o último dia da semana em `date`, `datetime` ou `datetimezone``dateTime` fornecido. Essa função usa um `Day` opcional, `firstDayOfWeek`, para definir o primeiro dia da semana para esse cálculo relativo. O valor padrão é `Day.Sunday`.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` com base no qual o último dia da semana é calculado
- `firstDayOfWeek`: *[Opcional]* Um valor `Day.Type` que representa o primeiro dia da semana. Os valores possíveis são `Day.Sunday`, `Day.Monday`, `Day.Tuesday`, `Day.Wednesday`, `Day.Thursday`, `Day.Friday` e `Day.Saturday.`. O valor padrão é `Day.Sunday`.

Exemplo 1

Obter o fim da semana de 14/5/2011.

```
Date.EndOfWeek(#date(2011, 5, 14))
```

`#date(2011, 5, 14)`

Exemplo 2

Obter o fim da semana de 17/5/2011 17h – 7h, considerando o domingo como primeiro dia da semana.

```
Date.EndOfWeek(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0), Day.Sunday)
```

`#datetimezone(2011, 5, 21, 23, 59, 59.9999999, -7, 0)`

Date.EndOfYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.EndOfYear(dateTime as any) as any
```

Sobre

Retorna um valor que representa o fim do ano em `dateTime`, incluindo segundos fracionários. As informações de fuso horário são preservadas.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` do qual o fim do ano é calculado.

Exemplo 1

Obtenha o fim do ano de 14/5/2011 05:00: 00 PM.

```
Date.EndOfYear(#datetime(2011, 5, 14, 17, 0, 0))
```

```
#datetime(2011, 12, 31, 23, 59, 59.9999999)
```

Exemplo 2

Obtenha o fim da hora de 17/5/2011 05:00: 00 PM -7:00.

```
Date.EndOfYear(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
```

```
#datetimezone(2011, 12, 31, 23, 59, 59.9999999, -7, 0)
```

Date.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.From(value as any, optional culture as nullable text) as nullable date
```

Sobre

Retorna um valor `date` do `value` especificado. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR"). Se o `value` fornecido for `null`, `Date.From` retornará `null`. Se o `value` fornecido for `date`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `date`:

- `text`: Um valor `date` da representação textual. Confira `Date.FromText` para obter detalhes.
- `datetime`: O componente de data do `value`.
- `datetimezone`: O componente de data do equivalente de datetime local de `value`.
- `number`: O componente de data do equivalente de datetime da Data de Automação OLE expresso por `value`.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `43910` em um valor `date`.

```
Date.From(43910)
```

`#date(2020, 3, 20)`

Exemplo 2

Converter `#datetime(1899, 12, 30, 06, 45, 12)` em um valor `date`.

```
Date.From(#datetime(1899, 12, 30, 06, 45, 12))
```

`#date(1899, 12, 30)`

Date.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.FromText(text as nullable text, optional culture as nullable text) as nullable date
```

Sobre

Cria um valor de `date` com base em uma representação textual, `text`, seguindo o padrão de formato ISO 8601. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

- `Date.FromText("2010-02-19")` // Data, aaaa-MM-dd

Exemplo 1

Converter `"December 31, 2010"` em um valor de data.

```
Date.FromText("2010-12-31")
```

`#date(2010, 12, 31)`

Exemplo 2

Converter `"December 31, 2010"` em um valor de data, com um formato diferente

```
Date.FromText("2010, 12, 31")
```

`#date(2010, 12, 31)`

Exemplo 3

Converter `"December, 2010"` em um valor de data.

```
Date.FromText("2010, 12")
```

`#date(2010, 12, 1)`

Exemplo 4

Converter `"2010"` em um valor de data.

```
Date.FromText("2010")
```

`#date(2010, 1, 1)`

Date.IsInCurrentDay

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInCurrentDay(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o dia atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo

Determine se a hora atual do sistema ocorre no dia atual.

```
Date.IsInCurrentDay(DateTime.FixedLocalNow())
```

`true`

Date.IsInCurrentMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInCurrentMonth(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o mês atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora atual do sistema ocorre no mês atual.

```
Date.IsInCurrentMonth(DateTime.FixedLocalNow())
```

`true`

Date.IsInCurrentQuarter

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInCurrentQuarter(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o trimestre atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se a hora atual do sistema ocorre no trimestre atual.

```
Date.IsInCurrentQuarter(DateTime.FixedLocalNow())
```

`true`

Date.IsInCurrentWeek

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInCurrentWeek(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a semana atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora atual do sistema ocorre na semana atual.

```
Date.IsInCurrentWeek(DateTime.FixedLocalNow())
```

`true`

Date.IsInCurrentYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInCurrentYear(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o ano atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se a hora atual do sistema ocorre no ano atual.

```
Date.IsInCurrentYear(DateTime.FixedLocalNow())
```

`true`

Date.IsInNextDay

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextDay(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o dia seguinte, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no dia atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o dia após a hora atual do sistema ocorre no dia seguinte.

```
Date.IsInNextDay(Date.AddDays(DateTime.FixedLocalNow(), 1))
```

`true`

Date.IsInNextMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextMonth(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo mês, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no mês atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se o mês após a hora atual do sistema é o mês seguinte.

```
Date.IsInNextMonth(Date.AddMonths(DateTime.FixedLocalNow(), 1))
```

`true`

Date.IsInNextNDays

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextNDays(dateTime as any, days as number) as nullable logical
```

Sobre

Indica se o valor `dateTime` de datetime especificado ocorre durante o próximo número de dias, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no dia atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `days`: O número de dias.

Exemplo 1

Determina se o dia após a hora atual do sistema é nos próximos dois dias.

```
Date.IsInNextNDays(Date.AddDays(DateTime.FixedLocalNow(), 1), 2)
```

`true`

Date.IsInNextNMonths

09/05/2020 • 2 minutes to read

```
Date.IsInNextNMonths(dateTime as any, months as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo número de meses, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no mês atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `months`: O número de meses.

Exemplo 1

Determina se o mês após a hora atual do sistema é nos próximos dois meses.

```
Date.IsInNextNMonths(Date.AddMonths(DateTime.FixedLocalNow(), 1), 2)
```

`true`

Date.IsInNextNQuarters

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextNQuarters(dateTime as any, quarters as number) as nullable logical
```

Sobre

Indica se o valor `dateTime` de datetime especificado ocorre durante o próximo número de trimestres, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro do trimestre atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `quarters`: O número de trimestres.

Exemplo 1

Determine se o trimestre após a hora atual do sistema é nos próximos dois trimestres.

```
Date.IsInNextNQuarters(Date.AddQuarters(DateTime.FixedLocalNow(), 1), 2)
```

`true`

Date.IsInNextNWeeks

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextNWeeks(dateTime as any, weeks as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo número de semanas, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre na semana atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `weeks`: O número de semanas.

Exemplo 1

Determina se o semana após a hora atual do sistema é nas próximas duas semanas.

```
Date.IsInNextNWeeks(Date.AddDays(DateTime.FixedLocalNow(), 7), 2)
```

`true`

Date.IsInNextNYears

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextNYears(dateTime as any, years as number) as nullable logical
```

Sobre

Indica se o valor `dateTime` de datetime especificado ocorre durante o próximo número de anos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará falso quando receber um valor que ocorre no ano atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `years`: O número de anos.

Exemplo 1

Determina se o ano após a hora atual do sistema é nos próximos dois anos.

```
Date.IsInNextNYears(Date.AddYears(DateTime.FixedLocalNow(), 1), 2)
```

`true`

Date.IsInNextQuarter

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextQuarter(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo trimestre, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro do trimestre atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

####Exemplo 1 Determine se o trimestre após a hora atual do sistema é o próximo trimestre.

```
Date.IsInNextQuarter(Date.AddQuarters(DateTime.FixedLocalNow(), 1))
```

`true`

Date.IsInNextWeek

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextWeek(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a semana a seguir, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre na semana atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se a semana após a hora atual do sistema é a semana seguinte.

```
Date.IsInNextWeek(Date.AddDays(DateTime.FixedLocalNow(), 7))
```

`true`

Date.IsInNextYear

08/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInNextYear(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo ano, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará falso quando receber um valor que ocorre no ano atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se o ano após a hora atual do sistema é o ano seguinte.

```
Date.IsInNextYear(Date.AddYears(DateTime.FixedLocalNow(), 1))
```

`true`

Date.IsInPreviousDay

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousDay(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime especificado `dateTime` ocorre durante o dia anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no dia atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se o dia antes da hora atual do sistema é o dia anterior.

```
Date.IsInPreviousDay(Date.AddDays(DateTime.FixedLocalNow(), -1))
```

`true`

Date.IsInPreviousMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousMonth(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o mês anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no mês atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se o mês antes da hora atual do sistema é o mês anterior.

```
Date.IsInPreviousMonth(Date.AddMonths(DateTime.FixedLocalNow(), -1))
```

`true`

Date.IsInPreviousNDays

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousNDays(dateTime as any, days as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de dias, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no dia atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `days`: O número de dias.

Exemplo 1

Determina se o dia anterior à hora atual do sistema é nos dois dias anteriores.

```
Date.IsInPreviousNDays(Date.AddDays(DateTime.FixedLocalNow(), -1), 2)
```

`true`

Date.IsInPreviousNMonths

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousNMonths(dateTime as any, months as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de meses, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no mês atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `months`: O número de meses.

Exemplo 1

Determina se o mês anterior à hora atual do sistema ocorre nos dois meses anteriores.

```
Date.IsInPreviousNMonths(Date.AddMonths(DateTime.FixedLocalNow(), -1), 2)
```

`true`

Date.IsInPreviousNQuarters

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousNQuarters(dateTime as any, quarters as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de trimestres, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro do trimestre atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `quarters`: O número de trimestres.

Exemplo 1

Determina se o trimestre anterior à hora atual do sistema é nos dois trimestres anteriores.

```
Date.IsInPreviousNQuarters(Date.AddQuarters(DateTime.FixedLocalNow(), -1), 2)
```

`true`

Date.IsInPreviousNWeeks

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousNWeeks(dateTime as any, weeks as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de semanas, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre na semana atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `weeks`: O número de semanas.

Exemplo 1

Determinar se a semana anterior à hora atual do sistema ocorre nas duas semanas anteriores.

```
Date.IsInPreviousNWeeks(Date.AddDays(DateTime.FixedLocalNow(), -7), 2)
```

`true`

Date.IsInPreviousNYears

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousNYears(dateTime as any, years as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de anos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará falso quando receber um valor que ocorre no ano atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.
- `years`: O número de anos.

Exemplo 1

Determina se o ano anterior à hora atual do sistema é nos dois anos anteriores.

```
Date.IsInPreviousNYears(Date.AddYears(DateTime.FixedLocalNow(), -1), 2)
```

`true`

Date.IsInPreviousQuarter

08/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousQuarter(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o trimestre anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro do trimestre atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se o trimestre anterior à hora atual do sistema ocorre no trimestre anterior.

```
Date.IsInPreviousQuarter(Date.AddQuarters(DateTime.FixedLocalNow(), -1))
```

`true`

Date.IsInPreviousWeek

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousWeek(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a semana anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre na semana atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se a semana antes da hora atual do sistema é a semana anterior.

```
Date.IsInPreviousWeek(Date.AddDays(DateTime.FixedLocalNow(), -7))
```

`true`

Date.IsInPreviousYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInPreviousYear(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o ano anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará falso quando receber um valor que ocorre no ano atual.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o ano anterior à hora atual do sistema ocorre no ano anterior.

```
Date.IsInPreviousYear(Date.AddYears(DateTime.FixedLocalNow(), -1))
```

`true`

Date.IsInYearToDate

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsInYearToDate(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o ano atual e se ocorre no dia atual ou antes dele, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determine se a hora atual do sistema ocorre desde o início do ano.

```
Date.IsInYearToDate(DateTime.FixedLocalNow())
```

`true`

Date.IsLeapYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.IsLeapYear(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado cai em um ano bissexto.

- `dateTime`: Um valor `date`, `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determina se o ano de 2012, conforme representado por `#date(2012, 01, 01)`, é um ano bissexto.

```
Date.IsLeapYear(#date(2012, 01, 01))
```

`true`

Date.Month

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.Month(dateTime as any) as nullable number
```

Sobre

Retorna o componente de mês do valor `datetime` fornecido, `dateTime`.

Exemplo 1

Localize o mês em #datetime (2011, 12, 31, 9, 15, 36).

```
Date.Month(#datetime(2011, 12, 31, 9, 15, 36))
```

`12`

Date.MonthName

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.MonthName(date as any, optional culture as nullable text) as nullable text
```

Sobre

Retorna o nome do componente de mês para a `date` fornecida. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo

Obter o nome do mês.

```
Date.MonthName(#datetime(2011, 12, 31, 5, 0, 0), "en-US")
```

`"December"`

Date.QuarterOfYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.QuarterOfYear(dateTime as any) as nullable number
```

Sobre

Retorna um número de 1 a 4 que indica em qual trimestre do ano a data `dateTime` cairá. `dateTime` pode ser um valor `date`, `datetime` ou `datetimezone`.

Exemplo 1

Localize em qual trimestre do ano a data #date(2011, 12, 31) cairá.

```
Date.QuarterOfYear(#date(2011, 12, 31))
```

`4`

Date.StartOfDay

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.StartOfDay(dateTime as any) as any
```

Sobre

Retorna o primeiro valor do dia `dateTime`. `dateTime` precisa ser um valor `date`, `datetime` ou `datetimezone`.

Exemplo 1

Localizar o início do dia de 10 de outubro de 2011, 8h (`#datetime(2011, 10, 10, 8, 0, 0)`).

```
Date.StartOfDay(#datetime(2011, 10, 10, 8, 0, 0))
```

`#datetime(2011, 10, 10, 0, 0, 0)`

Date.StartOfMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.StartOfMonth(dateTime as any) as any
```

Sobre

Retorna o primeiro valor do mês, considerando um tipo `date` ou `datetime`.

Exemplo 1

Localizar o início do mês de 10 de outubro de 2011, 8h10min32 (`#datetime(2011, 10, 10, 8, 10, 32)`).

```
Date.StartOfMonth(#datetime(2011, 10, 10, 8, 10, 32))
```

`#datetime(2011, 10, 1, 0, 0, 0)`

Date.StartOfQuarter

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.StartOfQuarter(dateTime as any) as any
```

Sobre

Retorna o primeiro valor do trimestre < `dateTime` . `dateTime` precisa ser um valor `date` , `datetime` ou `datetimezone` .

Exemplo 1

Localizar o início do trimestre de 10 de outubro de 2011, 8h (`#datetime(2011, 10, 10, 8, 0, 0)`).

```
Date.StartOfQuarter(#datetime(2011, 10, 10, 8, 0, 0))
```

`#datetime(2011, 10, 1, 0, 0, 0)`

Date.StartOfWeek

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.StartOfWeek(dateTime as any, optional firstDayOfWeek as nullable number) as any
```

Sobre

Retorna o primeiro valor da semana, considerando um valor `date`, `datetime` ou `datetimezone`.

Exemplo 1

Localizar o início da semana de 10 de outubro de 2011, 8h10min32 (`#datetime(2011, 10, 10, 8, 10, 32)`).

```
Date.StartOfWeek(#datetime(2011, 10, 10, 8, 10, 32))
```

`#datetime(2011, 10, 9, 0, 0, 0)`

Date.StartOfYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.StartOfYear(dateTime as any) as any
```

Sobre

Retorna o primeiro valor do ano, considerando um valor `date`, `datetime` ou `datetimezone`.

Exemplo 1

Localize o início do ano para 10 de outubro de 2011, 8h10min32 (`#datetime(2011, 10, 10, 8, 10, 32)`).

```
Date.StartOfYear(#datetime(2011, 10, 10, 8, 10, 32))
```

`#datetime(2011, 1, 1, 0, 0, 0)`

Date.ToRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.ToRecord(date as date) as record
```

Sobre

Retorna um registro que contém as partes do valor de data especificado, `date`.

- `date`: Um valor `date` com base no qual o registro das partes deve ser calculado.

Exemplo 1

Converter o valor de `#date(2011, 12, 31)` em um registro contendo partes do valor de data.

```
Date.ToRecord(#date(2011, 12, 31))
```

ANO	2011
MÊS	12
DIA	31

Date.ToText

26/05/2020 • 2 minutes to read

Sintaxe

```
Date.ToText(date as nullable date, optional format as nullable text, optional culture as nullable text) as nullable text
```

Sobre

Retorna uma representação textual de `date`. Um `format` opcional pode ser fornecido para personalizar a formatação do texto. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha uma representação textual #date(2010, 12, 31).

```
Date.ToText(#date(2010, 12, 31))
```

`"12/31/2010"`

Exemplo 2

Obtenha uma representação textual de #date(2010, 12, 31) com opção de formato.

```
Date.ToText(#date(2010, 12, 31), "yyyy/MM/dd")
```

`"2010/12/31"`

Date.WeekOfMonth

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.WeekOfMonth(dateTime as any, optional firstDayOfWeek as nullable number) as nullable number
```

Sobre

Retorna um número de 1 a 5 que indica em qual semana do ano e do mês a data `dateTime` cai.

- `dateTime`: Um valor `datetime` para o qual a semana do mês é determinada.

Exemplo 1

Determinar em qual semana o dia 15 de março cai em 2011 (`#date(2011, 03, 15)`).

```
Date.WeekOfMonth(#date(2011, 03, 15))
```

`3`

Date.WeekOfYear

09/05/2020 • 2 minutes to read

Sintaxe

```
Date.WeekOfYear(dateTime as any, optional firstDayOfWeek as nullable number) as nullable number
```

Sobre

Retorna um número de 1 a 54 que indica em qual semana do ano a data, `dateTime`, cairá.

- `dateTime`: Um valor `datetime` em que a semana do ano é determinada.
- `firstDayOfWeek`: Um valor de `Day.Type` opcional que indica qual dia é considerado o início de uma nova semana (por exemplo, `Day.Sunday`. Se não for especificado, um padrão dependente de cultura será usado.

Exemplo 1

Determina em qual semana do ano 27 de março cairá em 2011 (`#date(2011, 03, 27)`).

```
Date.WeekOfYear(#date(2011, 03, 27))
```

`14`

Exemplo 2

Determina em qual semana do ano 27 de março cairá em 2011 (`#date(2011, 03, 27)`), usando a segunda-feira como o início de uma nova semana.

```
Date.WeekOfYear(#date(2011, 03, 27), Day.Monday)
```

`13`

Date.Year

09/05/2020 • 2 minutes to read

```
Date.Year(dateTime as any) as nullable number
```

Sobre

Retorna o componente de ano do valor `datetime` fornecido, `dateTime`.

Exemplo 1

Localizar o ano em #datetime(2011, 12, 31, 9, 15, 36).

```
Date.Year(#datetime(2011, 12, 31, 9, 15, 36))
```

`2011`

Day.Friday

09/05/2020 • 2 minutes to read

Sobre

Retorna 6, o número que representa a sexta-feira.

Day.Monday

09/05/2020 • 2 minutes to read

Sobre

Retorna 2, o número que representa a segunda-feira.

Day.Saturday

09/05/2020 • 2 minutes to read

Sobre

Retorna 7, o número que representa o sábado.

Day.Sunday

09/05/2020 • 2 minutes to read

Sobre

Retorna 1, o número que representa o domingo.

Day.Thursday

09/05/2020 • 2 minutes to read

Sobre

Retorna 5, o número que representa a quinta-feira.

Day.Tuesday

09/05/2020 • 2 minutes to read

Sobre

Retorna 3, o número que representa a terça-feira.

Day.Wednesday

09/05/2020 • 2 minutes to read

Sobre

Retorna 4, o número que representa a quarta-feira.

#date

09/05/2020 • 2 minutes to read

Sintaxe

```
#date(year as number, month as number, day as number) as date
```

Sobre

Cria um valor de data do ano `year`, do mês `month` e do dia `day`. Gera um erro se isto não é verdadeiro:

- 1 ≤ ano ≤ 9999
- 1 ≤ mês ≤ 12
- 1 ≤ dia ≤ 31

Funções de DateTime

08/05/2020 • 5 minutes to read

Essas funções criam e manipulam valores datetime e datetimezone.

DateTime

FUNÇÃO	DESCRIÇÃO
DateTime.AddZone	Adiciona o timezonehours como uma diferença para o valor de datetime inserido e retorna um novo valor datetimezone.
DateTime.Date	Retorna uma parte de data de um valor de DateTime
DateTime.FixedLocalNow	Retorna um valor de DateTime definido como a data e a hora atuais do sistema.
DateTime.From	Retorna o valor de datetime de um valor.
DateTime.FromFileTime	Retorna o valor de DateTime de um valor fornecido.
DateTime.FromText	Retorna um valor de DateTime de um conjunto de formatos de data e valor de cultura.
DateTime.IsInCurrentHour	Indica se o valor de datetime especificado ocorre durante a hora atual, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInCurrentMinute	Indica se o valor de datetime especificado ocorre durante o minuto atual, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInCurrentSecond	Indica se o valor de datetime especificado ocorre durante o segundo atual, conforme determinado pela hora e pela data atuais do sistema.
DateTime.IsInNextHour	Indica se o valor de datetime especificado ocorre durante a hora a seguir, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInNextMinute	Indica se o valor de datetime especificado ocorre durante o minuto a seguir, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInNextNHours	Indica se o valor de datetime especificado ocorre durante o próximo número de horas, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInNextNMinutes	Indica se o valor de datetime especificado ocorre durante o próximo número de minutos, conforme determinado pela data e pela hora atuais do sistema.

FUNÇÃO	DESCRIÇÃO
DateTime.IsInNextNSeconds	Indica se o valor de datetime especificado ocorre durante o próximo número de segundos, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInNextSecond	Indica se o valor de datetime especificado ocorre durante o segundo a seguir, conforme determinado pela hora e pela data atuais do sistema.
DateTime.IsInPreviousHour	Indica se o valor de datetime especificado ocorre durante a hora anterior, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInPreviousMinute	Indica se o valor de datetime especificado ocorre durante o minuto anterior, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInPreviousNHours	Indica se o valor de datetime especificado ocorre durante o número anterior de horas, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInPreviousNMinutes	Indica se o valor de datetime especificado ocorre durante o número anterior de minutos, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInPreviousNSeconds	Indica se o valor de datetime especificado ocorre durante o número anterior de segundos, conforme determinado pela data e pela hora atuais do sistema.
DateTime.IsInPreviousSecond	Indica se o valor de datetime especificado ocorre durante o próximo segundo, conforme determinado pela hora e pela data atuais do sistema.
DateTime.LocalNow	Retorna um valor de datetime definido como a data e a hora atuais do sistema.
DateTime.Time	Retorna uma parte de hora de um valor de DateTime.
DateTime.ToRecord	Retorna um registro contendo partes do valor de DateTime.
DateTime.ToText	Retorna o valor de texto de um valor de DateTime.
#datetime	Cria um valor de datetime usando ano, mês, dia, hora, minuto e segundo.

DateTime.AddZone

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.AddZone(dateTime as nullable datetime, timezoneHours as number, optional timezoneMinutes
as nullable number) as nullable datetimezone
```

Sobre

Define as informações de fuso horário em um valor de datetime `dateTime`. As informações de fuso horário incluirão `timezoneHours` e, opcionalmente, `timezoneMinutes`.

Exemplo 1

Define a informação de fuso horário de #datetime(2010, 12, 31, 11, 56, 02) para 7 horas, 30 minutos.

```
DateTime.AddZone(#datetime(2010, 12, 31, 11, 56, 02), 7, 30)
```

`#datetimezone(2010, 12, 31, 11, 56, 2, 7, 30)`

DateTime.Date

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.Date(dateTime as any) as nullable date
```

Sobre

Retorna o componente de data de `dateTime`, o valor de `date`, `datetime` ou `datetimezone` especificado.

Exemplo 1

Encontrar o valor de data de #datetime(2010, 12, 31, 11, 56, 02).

```
DateTime.Date(#datetime(2010, 12, 31, 11, 56, 02))
```

```
#date(2010, 12, 31)
```

DateTime.FixedLocalNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.FixedLocalNow() as datetime
```

Sobre

Retorna um valor `datetime` definido como a data e a hora atuais do sistema. O valor é fixo e não será alterado com as chamadas sucessivas, diferentemente de DateTime.LocalNow, que poderá retornar valores diferentes no decorrer da execução de uma expressão.

DateTime.From

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.From(value as any, optional culture as nullable text) as nullable datetime
```

Sobre

Retorna um valor `datetime` do `value` especificado. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR"). Se o `value` fornecido for `null`, `DateTime.From` retornará `null`. Se o `value` fornecido for `datetime`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `datetime`:

- `text`: Um valor `datetime` da representação textual. Confira `DateTime.FromText` para obter detalhes.
- `date`: Um `datetime` com `value` como o componente de data e `12:00:00 AM` como o componente de hora.
- `datetimezone`: O `datetime` local equivalente de `value`.
- `time`: Um `datetime` com o equivalente de data da Data de Automação OLE de `0` como o componente de data e `value` como o componente de hora.
- `number`: Um `datetime` equivalente da Data de Automação OLE expresso por `value`.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `#time(06, 45, 12)` em um valor `datetime`.

```
DateTime.From(#time(06, 45, 12))
```

`#datetime(1899, 12, 30, 06, 45, 12)`

Exemplo 2

Converter `#date(1975, 4, 4)` em um valor `datetime`.

```
DateTime.From(#date(1975, 4, 4))
```

`#datetime(1975, 4, 4, 0, 0, 0)`

DateTime.FromFileTime

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.FromFileTime(fileTime as nullable number) as nullable datetime
```

Sobre

Cria um valor `datetime` com base no valor de `fileTime` e o converte no fuso horário local. O filetime é um valor temporal de arquivo do Windows que representa o número de intervalos de 100 a nanossegundos decorridos desde a meia-noite de 1º de janeiro de 1601 D.C. (C.E.) Tempo Universal Coordenado (UTC).

Exemplo 1

Converter `129876402529842245` em um valor de datetime.

```
DateTime.FromFileTime(129876402529842245)
```

```
#datetime(2012, 7, 24, 14, 50, 52.9842245)
```

DateTime.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.FromText(text as nullable text, optional culture as nullable text) as nullable datetime
```

Sobre

Cria um valor de `datetime` com base em uma representação textual, `text`, seguindo o padrão de formato ISO 8601. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

- `DateTime.FromText("2010-12-31T01:30:00")` // aaaa-MM-ddThh:mm:ss

Exemplo 1

Converter `"2010-12-31T01:30:25"` em um valor de datetime.

```
DateTime.FromText("2010-12-31T01:30:25")
```

`#datetime(2010, 12, 31, 1, 30, 25)`

Exemplo 2

Converter `"2010-12-31T01:30"` em um valor de datetime.

```
DateTime.FromText("2010-12-31T01:30")
```

`#datetime(2010, 12, 31, 1, 30, 0)`

Exemplo 3

Converter `"20101231T013025"` em um valor de datetime.

```
DateTime.FromText("20101231T013025")
```

`#datetime(2010, 12, 31, 1, 30, 25)`

Exemplo 4

Converter `"20101231T01:30:25"` em um valor de datetime.

```
DateTime.FromText("20101231T01:30:25")
```

`#datetime(2010, 12, 31, 1, 30, 25)`

Exemplo 5

Converter `"20101231T01:30:25.121212"` em um valor de datetime.

```
DateTime.FromText("20101231T01:30:25.121212")
```

`#datetime(2010, 12, 31, 1, 30, 25.121212)`

DateTime.IsInCurrentHour

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInCurrentHour(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a hora atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora atual do sistema ocorre na hora atual.

```
DateTime.IsInCurrentHour(DateTime.FixedLocalNow())
```

`true`

DateTime.IsInCurrentMinute

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInCurrentMinute(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o minuto atual, conforme determinado pela data e pela hora atuais do sistema.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o horário atual do sistema está no minuto atual.

```
DateTime.IsInCurrentMinute(DateTime.FixedLocalNow())
```

`true`

DateTime.IsInCurrentSecond

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInCurrentSecond(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o segundo atual, conforme determinado pela hora e pela data atuais do sistema.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora atual do sistema ocorre no segundo atual.

```
DateTime.IsInCurrentSecond(DateTime.FixedLocalNow())
```

`true`

DateTime.IsInNextHour

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextHour(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a hora a seguir, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro da hora atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora após o horário atual do sistema está na próxima hora.

```
DateTime.IsInNextHour(DateTime.FixedLocalNow() + #duration(0, 1, 0, 0))
```

`true`

DateTime.IsInNextMinute

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextMinute(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o minuto seguinte, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no minuto atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o minuto após a hora atual do sistema ocorre no minuto seguinte.

```
DateTime.IsInNextMinute(DateTime.FixedLocalNow() + #duration(0, 0, 1, 0))
```

`true`

DateTime.IsInNextNHours

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextNHours(dateTime as any, hours as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo número de horas, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro da hora atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `hours`: O número de horas.

Exemplo 1

Determinar se a hora após o horário atual do sistema está nas próximas duas horas.

```
DateTime.IsInNextNHours(DateTime.FixedLocalNow() + #duration(0, 2, 0, 0), 2)
```

`true`

DateTime.IsInNextNMinutes

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextNMinutes(dateTime as any, minutes as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo número de minutos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no minuto atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `minutes`: O número de minutos.

Exemplo 1

Determinar se o minuto após a hora atual do sistema ocorre nos próximos dois minutos.

```
DateTime.IsInNextNMinutes(DateTime.FixedLocalNow() + #duration(0, 0, 2, 0), 2)
```

`true`

DateTime.IsInNextNSeconds

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextNSeconds(dateTime as any, seconds as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo número de segundos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no segundo atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `seconds`: O número de segundos.

Exemplo 1

Determinar se o segundo após o horário atual do sistema está nos próximos dois segundos.

```
DateTime.IsInNextNSeconds(DateTime.FixedLocalNow() + #duration(0, 0, 0, 2), 2)
```

`true`

DateTime.IsInNextSecond

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInNextSecond(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o segundo a seguir, conforme determinado pela hora e pela data atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no segundo atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o segundo após o horário atual do sistema está no próximo segundo.

```
DateTime.IsInNextSecond(DateTime.FixedLocalNow() + #duration(0, 0, 0, 1))
```

`true`

DateTime.IsInPreviousHour

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousHour(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante a hora anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro da hora atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se a hora antes do horário atual do sistema está na hora anterior.

```
DateTime.IsInPreviousHour(DateTime.FixedLocalNow() - #duration(0, 1, 0, 0))
```

`true`

DateTime.IsInPreviousMinute

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousMinute(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o minuto anterior, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no minuto atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o minuto antes do horário atual do sistema está no minuto anterior.

```
DateTime.IsInPreviousMinute(DateTime.FixedLocalNow() - #duration(0, 0, 1, 0))
```

`true`

DateTime.IsInPreviousNHours

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousNHours(dateTime as any, hours as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de horas, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor ocorrido dentro da hora atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `hours`: O número de horas.

Exemplo 1

Determinar se a hora anterior à hora atual do sistema ocorre nas duas horas anteriores.

```
DateTime.IsInPreviousNHours(DateTime.FixedLocalNow() - #duration(0, 2, 0, 0), 2)
```

`true`

DateTime.IsInPreviousNMinutes

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousNMinutes(dateTime as any, minutes as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de minutos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no minuto atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `minutes`: O número de minutos.

Exemplo 1

Determinar se o minuto antes do horário atual do sistema está nos dois minutos anteriores.

```
DateTime.IsInPreviousNMinutes(DateTime.FixedLocalNow() - #duration(0, 0, 2, 0), 2)
```

`true`

DateTime.IsInPreviousNSeconds

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousNSeconds(dateTime as any, seconds as number) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o número anterior de segundos, conforme determinado pela data e pela hora atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no segundo atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.
- `seconds`: O número de segundos.

Exemplo 1

Determinar se o segundo antes da hora atual do sistema ocorre nos dois segundos anteriores.

```
DateTime.IsInPreviousNSeconds(DateTime.FixedLocalNow() - #duration(0, 0, 0, 2), 2)
```

`true`

DateTime.IsInPreviousSecond

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.IsInPreviousSecond(dateTime as any) as nullable logical
```

Sobre

Indica se o valor de datetime `dateTime` especificado ocorre durante o próximo segundo, conforme determinado pela hora e pela data atuais do sistema. Observe que essa função retornará false quando receber um valor que ocorre no segundo atual.

- `dateTime`: Um valor `datetime` ou `datetimezone` a ser avaliado.

Exemplo 1

Determinar se o segundo antes do horário atual do sistema está no segundo anterior.

```
DateTime.IsInPreviousSecond(DateTime.FixedLocalNow() - #duration(0, 0, 0, 1))
```

`true`

DateTime.LocalNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.LocalNow() as datetime
```

Sobre

Retorna um valor `datetime` definido como a data e a hora atuais do sistema.

DateTime.Time

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.Time(dateTime as any) as nullable time
```

Sobre

Retorna a parte de hora do valor datetime especificado, `dateTime`.

Exemplo 1

Localize o valor temporal de #datetime(2010, 12, 31, 11, 56, 02).

```
DateTime.Time(#datetime(2010, 12, 31, 11, 56, 02))
```

`#time(11, 56, 2)`

DateTime.ToRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.ToRecord(dateTime as datetime) as record
```

Sobre

Retorna um registro que contém as partes do valor de datetime especificado, `dateTime`.

- `dateTime`: Um valor `datetime` com base no qual o registro das partes deve ser calculado.

Exemplo 1

Converter o valor `#datetime(2011, 12, 31, 11, 56, 2)` em um registro contendo valores de Date e Time.

```
DateTime.ToRecord(#datetime(2011, 12, 31, 11, 56, 2))
```

ANO	2011
MÊS	12
DIA	31
HORA	11
MINUTE	56
SECOND	2

DateTime.ToText

26/05/2020 • 2 minutes to read

Sintaxe

```
DateTime.ToText(dateTime as nullable datetime, optional format as nullable text, optional culture
as nullable text) as nullable text
```

Sobre

Retorna uma representação textual de `dateTime`. Um `format` opcional pode ser fornecido para personalizar a formatação do texto. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha uma representação textual #datetime(2011, 12, 31, 11, 56, 2).

```
DateTime.ToText(#datetime(2010, 12, 31, 11, 56, 2))
```

```
"12/31/2010 11:56:02 AM"
```

Exemplo 2

Obtenha uma representação textual de #datetime(2011, 12, 31, 11, 56, 2) com opção de formato.

```
DateTime.ToText(#datetime(2010, 12, 31, 11, 56, 2), "yyyy/MM/ddThh:mm:ss")
```

```
"2010/12/31T11:56:02"
```

#datetime

09/05/2020 • 2 minutes to read

Sintaxe

```
#datetime(year as number, month as number, day as number, hour as number, minute as number, second as number) as any
```

Sobre

Cria um valor DateTime de números inteiros `year`, mês `month`, dia `day`, hora `hour`, minuto `minute` e segundo `second` (fracionário). Gera um erro se isto não é verdadeiro:

- 1 ≤ ano ≤ 9999
- 1 ≤ mês ≤ 12
- 1 ≤ dia ≤ 31
- 0 ≤ hora ≤ 23
- 0 ≤ minuto ≤ 59
- 0 ≤ segundo ≤ 59

Funções de DateTimeZone

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam valores datetimezone.

DateTimeZone

FUNÇÃO	DESCRIÇÃO
DateTimeZone.FixedLocalNow	Retorna um valor de DateTimeZone definido como a data, a hora e o fuso horário de diferença no sistema.
DateTimeZone.FixedUtcNow	Retorna a data e a hora atuais em UTC (fuso horário GMT).
DateTimeZone.From	Retorna o valor de datetimezone de um valor.
DateTimeZone.FromFileTime	Retorna o DateTimeZone de um valor numérico.
DateTimeZone.FromText	Retorna um valor de DateTimeZone de um conjunto de formatos de data e valor de cultura.
DateTimeZone.LocalNow	Retorna um valor de DateTime definido como a data e a hora atuais do sistema.
DateTimeZone.RemoveZone	Retorna um valor de datetime com as informações de zona removidas do valor de datetimezone inserido.
DateTimeZone.SwitchZone	Altera as informações de fuso horário para o DateTimeZone inserido.
DateTimeZone.ToLocal	Retorna um valor de DateTime do fuso horário local.
DateTimeZone.ToRecord	Retorna um registro contendo partes do valor de DateTime.
DateTimeZone.ToText	Retorna o valor de texto de um valor de DateTime.
DateTimeZone.ToUtc	Retorna um valor de DateTime para o fuso horário UTC.
DateTimeZone.UtcNow	Retorna um valor de DateTime definido como a data e a hora atuais do sistema no fuso horário UTC.
DateTimeZone.ZoneHours	Retorna um valor de hora de fuso horário de um valor de DateTime.
DateTimeZone.ZoneMinutes	Retorna um valor de minuto de fuso horário de um valor de DateTime.
#datetimezone	Cria um valor de datetimezone usando ano, mês, dia, hora, minuto, segundo, horas de diferença e minutos de diferença.

DateTimeZone.FixedLocalNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.FixedLocalNow() as datetimezone
```

Sobre

Retorna um valor `datetime` definido como a data e a hora atuais do sistema. O valor retornado contém informações de fuso horário que representam o fuso horário local. O valor é fixo e não será alterado com as chamadas sucessivas, ao contrário de DateTimeZone.LocalNow, que poderá retornar valores diferentes no decorrer da execução de uma expressão.

DateTimeZone.FixedUtcNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.FixedUtcNow() as datetimezone
```

Sobre

Retorna a data e a hora atuais em UTC (fuso horário GMT). Esse valor é fixo e não será alterado com chamadas sucessivas.

DateTimeZone.From

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.From(value as any, optional culture as nullable text) as nullable datetimezone
```

Sobre

Retorna um valor `datetimezone` do `value` especificado. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR"). Se o `value` fornecido for `null`, `DateTimeZone.From` retornará `null`. Se o `value` fornecido for `datetimezone`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `datetimezone`:

- `text`: Um valor `datetimezone` da representação textual. Confira `DateTimeZone.FromText` para obter detalhes.
- `date`: Um `datetimezone` com `value` como o componente de data, `12:00:00 AM` como o componente de hora e o deslocamento correspondente ao fuso horário local.
- `datetime`: Um `datetimezone` com `value` como o datetime e o deslocamento correspondente ao fuso horário local.
- `time`: Um `datetimezone` com o equivalente de data da Data de Automação OLE de `0` como o componente de data, `value` como o componente de hora e o deslocamento correspondente ao fuso horário local.
- `number`: Um `datetimezone` com o equivalente de datetime da Data de Automação OLE expresso por `value` e o deslocamento correspondente ao fuso horário local.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `"2020-10-30T01:30:00-08:00"` em um valor `datetimezone`.

```
DateTimeZone.From("2020-10-30T01:30:00-08:00")
```

`#datetimezone(2020, 10, 30, 01, 30, 00, -8, 00)`

DateTimeZone.FromFileTime

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.FromFileTime(fileTime as nullable number) as nullable datetimezone
```

Sobre

Cria um valor `datetimezone` com base no valor de `fileTime` e o converte no fuso horário local. O filetime é um valor temporal de arquivo do Windows que representa o número de intervalos de 100 a nanossegundos decorridos desde a meia-noite de 1º de janeiro de 1601 D.C. (C.E.) Tempo Universal Coordenado (UTC).

Exemplo 1

Converter `129876402529842245` em um valor de datetimezone.

```
DateTimeZone.FromFileTime(129876402529842245)
```

```
#datetimezone(2012, 7, 24, 14, 50, 52.9842245, -7, 0)
```

DateTimeZone.FromText

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.FromText(text as nullable text, optional culture as nullable text) as nullable
datetimezone
```

Sobre

Cria um valor de `datetimezone` com base em uma representação textual, `text`, seguindo o padrão de formato ISO 8601. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

- `DateTimeZone.FromText("2010-12-31T01:30:00-08:00")` // aaaa-MM-ddThh:mm:ssZ

Exemplo 1

Converter `"2010-12-31T01:30:00-08:00"` em um valor de datetimezone.

```
DateTimeZone.FromText("2010-12-31T01:30:00-08:00")
```

`#datetimezone(2010, 12, 31, 1, 30, 0, -8, 0)`

Exemplo 2

Converter `"2010-12-31T01:30:00.121212-08:00"` em um valor de datetimezone.

```
DateTimeZone.FromText("2010-12-31T01:30:00.121212-08:00")
```

`#datetimezone(2010, 12, 31, 1, 30, 0.121212, -8, 0)`

Exemplo 3

Converter `"2010-12-31T01:30:00Z"` em um valor de datetimezone.

```
DateTimeZone.FromText("2010-12-31T01:30:00Z")
```

`#datetimezone(2010, 12, 31, 1, 30, 0, 0, 0)`

Exemplo 4

Converter `"20101231T013000+0800"` em um valor de datetimezone.

```
DateTimeZone.FromText("20101231T013000+0800")
```

`#datetimezone(2010, 12, 31, 1, 30, 0, 8, 0)`

DateTimeZone.LocalNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.LocalNow() as datetimezone
```

Sobre

Retorna um valor `datetimezone` definido como a data e a hora atuais do sistema. O valor retornado contém informações de fuso horário que representam o fuso horário local.

DateTimeZone.RemoveZone

08/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.RemoveZone(dateTimeZone as nullable datetimezone) as nullable datetime
```

Sobre

Retorna um valor de #datetime de `dateTimeZone` com as informações de fuso horário removidas.

Exemplo 1

Remover as informações de fuso horário do valor de #datetimezone(2011, 12, 31, 9, 15, 36, -7, 0).

```
DateTimeZone.RemoveZone(#datetimezone(2011, 12, 31, 9, 15, 36, -7, 0))
```

```
#datetime(2011, 12, 31, 9, 15, 36)
```

DateTimeZone.SwitchZone

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.SwitchZone(dateTimeZone as nullable datetimezone, timezoneHours as number, optional
timezoneMinutes as nullable number) as nullable datetimezone
```

Sobre

Altera as informações de fuso horário no valor de datetimezone `dateTimeZone` para as novas informações de fuso horário fornecidas por `timezoneHours` e, opcionalmente, `timezoneMinutes`. Se `dateTimeZone` não tem um componente de fuso horário, uma exceção é gerada.

Exemplo 1

Alterar as informações de fuso horário de #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) para 8 horas.

```
DateTimeZone.SwitchZone(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30), 8)
```

```
#datetimezone(2010, 12, 31, 12, 26, 2, 8, 0)
```

Exemplo 2

Alterar as informações de fuso horário de #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) para -30 minutos.

```
DateTimeZone.SwitchZone(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30), 0, -30)
```

```
#datetimezone(2010, 12, 31, 3, 56, 2, 0, -30)
```

DateTimeZone.ToLocal

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ToLocal(dateTimeZone as nullable datetimezone) as nullable datetimezone
```

Sobre

Altera as informações de fuso horário do valor de datetimezone `dateTimeZone` para as informações de fuso horário local. Se `dateTimeZone` não tiver um componente de fuso horário, as informações de fuso horário local serão adicionadas.

Exemplo 1

Alterar as informações de fuso horário de #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) para o fuso horário local (pressupondo PST).

```
DateTimeZone.ToLocal(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30))
```

```
#datetimezone(2010, 12, 31, 12, 26, 2, -8, 0)
```

DateTimeZone.ToRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ToRecord(dateTimeZone as datetimezone) as record
```

Sobre

Retorna um registro contendo as partes do valor de datetimezone especificado, `dateTimeZone`.

- `dateTimeZone`: Um valor `datetimezone` com base no qual o registro das partes deve ser calculado.

Exemplo 1

Converter o valor `#datetimezone(2011, 12, 31, 11, 56, 2, 8, 0)` em um registro contendo valores de Date, Time e Zone.

```
DateTimeZone.ToRecord(#datetimezone(2011, 12, 31, 11, 56, 2, 8, 0))
```

ANO	2011
MÊS	12
DIA	31
HORA	11
MINUTE	56
SECOND	2
ZONEHOURS	8
ZONEMINUTES	0

DateTimeZone.ToText

26/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ToText(dateTimeZone as nullable datetimezone, optional format as nullable text,
optional culture as nullable text) as nullable text
```

Sobre

Retorna uma representação textual de `dateTimeZone`. Um `format` opcional pode ser fornecido para personalizar a formatação do texto. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha uma representação textual #datetimezone(2011, 12, 31, 11, 56, 2, 8, 0).

```
DateTimeZone.ToText(#datetimezone(2010, 12, 31, 11, 56, 2, 8, 0))
```

`"12/31/2010 11:56:02 AM +08:00"`

Exemplo 2

Obtenha uma representação textual de #datetimezone(2010, 12, 31, 11, 56, 2, 10, 12) com opção de formato.

```
DateTimeZone.ToText(#datetimezone(2010, 12, 31, 11, 56, 2, 10, 12), "yyyy/MM/ddThh:mm:sszzz")
```

`"2010/12/31T11:56:02+10:12"`

DateTimeZone.ToUtc

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ToUtc(dateTimeZone as nullable datetimezone) as nullable datetimezone
```

Sobre

Altera as informações de fuso horário do valor de datetime `dateTimeZone` para as informações de fuso horário UTC ou Tempo Universal. Se `dateTimeZone` não tiver um componente de fuso horário, as informações de fuso horário UTC serão adicionadas.

Exemplo 1

Alterar as informações de fuso horário de #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) para o fuso horário UTC.

```
DateTimeZone.ToUtc(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30))
```

```
#datetimezone(2010, 12, 31, 4, 26, 2, 0, 0)
```

DateTimeZone.UtcNow

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.UtcNow() as datetimezone
```

Sobre

Retorna a data e a hora atuais em UTC (fuso horário GMT).

Exemplo 1

Obtenha a data e a hora atuais em UTC.

```
DateTimeZone.UtcNow()
```

```
#datetimezone(2011, 8, 16, 23, 34, 37.745, 0, 0)
```

DateTimeZone.ZoneHours

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ZoneHours(dateTimeZone as nullable datetimezone) as nullable number
```

Sobre

Altera o fuso horário do valor.

DateTimeZone.ZoneMinutes

09/05/2020 • 2 minutes to read

Sintaxe

```
DateTimeZone.ZoneMinutes(dateTimeZone as nullable datetimezone) as nullable number
```

Sobre

Altera o fuso horário do valor.

#datetimezone

09/05/2020 • 2 minutes to read

Sintaxe

```
#datetimezone(year as number, month as number, day as number, hour as number, minute as number,
second as number, offsetHours as number, offsetMinutes as number) as any
```

Sobre

Cria um valor datetimezone dos números inteiros `year`, mês `month`, dia `day`, hora `hour`, minuto `minute`, segundo `second` (fracionário), horas de deslocamento `offsetHours` (fracionário) e minutos de deslocamento `offsetMinutes`. Gera um erro se isto não é verdadeiro:

- 1 ≤ ano ≤ 9999
- 1 ≤ mês ≤ 12
- 1 ≤ dia ≤ 31
- 0 ≤ hora ≤ 23
- 0 ≤ minuto ≤ 59
- 0 ≤ segundo ≤ 59
- -14 ≤ horas de deslocamento + minutos de deslocamento / 60 ≤ 14

Funções de duração

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam valores de duração.

Duração

FUNÇÃO	DESCRIÇÃO
Duration.Days	Retorna o componente de dia do valor de duração.
Duration.From	Retorna o valor de duração de um valor.
Duration.FromText	Retorna o valor de duração de um valor de texto.
Duration.Hours	Retorna o componente de hora do valor de duração.
Duration.Minutes	Retorna o componente de minuto do valor de duração.
Duration.Seconds	Retorna o componente de segundo do valor de duração.
Duration.ToRecord	Retorna um registro com partes de um valor de duração.
Duration.TotalDays	Retorna a magnitude total de dias com base em um valor de duração.
Duration.TotalHours	Retorna a magnitude total de horas de um valor de duração.
Duration.TotalMinutes	Retorna a magnitude total de minutos de um valor de duração.
Duration.TotalSeconds	Retorna a magnitude total de segundos de um valor de duração.
Duration.ToText	Retorna um valor de texto de um valor de duração.
#duration	Cria um valor de duração usando dias, hora, minuto e segundo.

Duration.Days

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.Days(duration as nullable duration) as nullable number
```

Sobre

Retorna o componente de dia do valor `duration` fornecido, `duration`.

Exemplo 1

Localizar o dia em #duration(5, 4, 3, 2).

```
Duration.Days(#duration(5, 4, 3, 2))
```

`5`

Duration.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.From(value as any) as nullable duration
```

Sobre

Retorna um valor `duration` do `value` especificado. Se o `value` fornecido for `null`, `Duration.From` retornará `null`. Se o `value` fornecido for `duration`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `duration`:

- `text`: Um valor `duration` do formato textual de tempo decorrido (d.h:m:s). Confira `Duration.FromText` para obter detalhes.
- `number`: Um `duration` equivalente ao número de dias inteiros e fracionários expressos por `value`.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `2.525` em um valor `duration`.

```
Duration.From(2.525)
```

```
#duration(2, 12, 36, 0)
```

Duration.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.FromText(text as nullable text) as nullable duration
```

Sobre

Retorna um valor de duração do texto especificado, `text`. Os seguintes formatos podem ser analisados por esta função:

- (-)hh:mm(:ss(.ff))
- (-)ddd(.hh:mm(:ss(.ff)))

(Todos os intervalos são inclusivos)
ddd: Número de dias.
hh: Número de horas, entre 0 e 23.
mm: Número de minutos, entre 0 e 59.
ss: Número de segundos, entre 0 e 59.
ff: Fração de segundos, entre 0 e 9999999.

Exemplo 1

Converter `"2.05:55:20"` em um valor `duration`.

```
Duration.FromText("2.05:55:20")
```

`#duration(2, 5, 55, 20)`

Duration.Hours

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.Hours(duration as nullable duration) as nullable number
```

Sobre

Retorna o componente de hora do valor `duration` fornecido, `duration`.

Exemplo 1

Localizar as horas em #duration(5, 4, 3, 2).

```
Duration.Hours(#duration(5, 4, 3, 2))
```

`4`

Duration.Minutes

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.Minutes(duration as nullable duration) as nullable number
```

Sobre

Retorna o componente de minutos do valor `duration` fornecido, `duration`.

Exemplo 1

Localize os minutos em #duration(5, 4, 3, 2).

```
Duration.Minutes(#duration(5, 4, 3, 2))
```

`3`

Duration.Seconds

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.Seconds(duration as nullable duration) as nullable number
```

Sobre

Retorna o componente de segundos do valor `duration` fornecido, `duration`.

Exemplo 1

Localizar os segundos em #duration(5, 4, 3, 2).

```
Duration.Seconds(#duration(5, 4, 3, 2))
```

`2`

Duration.ToRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.ToRecord(duration as duration) as record
```

Sobre

Retorna um registro contendo as partes do valor de duração, `duration`.

- `duration`: Uma `duration` em que o registro é criado.

Exemplo 1

Converta `#duration(2, 5, 55, 20)` em um registro das partes, incluindo dias, horas, minutos e segundos, se aplicável.

```
Duration.ToRecord(#duration(2, 5, 55, 20))
```

DIAS	2
HORAS	5
MINUTOS	55
SEGUNDOS	20

Duration.TotalDays

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.TotalDays(duration as nullable duration) as nullable number
```

Sobre

Retorna o total de dias abrangidos pelo valor de `duration` fornecido, `duration`.

Exemplo 1

Localizar o total de dias abrangidos em #duration(5, 4, 3, 2).

```
Duration.TotalDays(#duration(5, 4, 3, 2))
```

`5.1687731481481478`

Duration.TotalHours

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.TotalHours(duration as nullable duration) as nullable number
```

Sobre

Retorna o total de horas abarcado pelo valor `duration` fornecido, `duration`.

Exemplo 1

Localize o total de horas abarcados em #duration(5, 4, 3, 2).

```
Duration.TotalHours(#duration(5, 4, 3, 2))
```

`124.05055555555555`

Duration.TotalMinutes

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.TotalMinutes(duration as nullable duration) as nullable number
```

Sobre

Retorna o total de minutos abrangidos pelo valor de `duration` fornecido, `duration`.

Exemplo 1

Localizar o total de minutos abrangidos em #duration(5, 4, 3, 2).

```
Duration.TotalMinutes(#duration(5, 4, 3, 2))
```

`7443.0333333333338`

Duration.TotalSeconds

09/05/2020 • 2 minutes to read

Sintaxe

```
Duration.TotalSeconds(duration as nullable duration) as nullable number
```

Sobre

Retorna o total de segundos abarcado pelo valor `duration` fornecido, `duration`.

Exemplo 1

Localize o total de segundos abarcados em #duration(5, 4, 3, 2).

```
Duration.TotalSeconds(#duration(5, 4, 3, 2))
```

`446582`

Duration.ToText

30/09/2020 • 2 minutes to read

Sintaxe

```
Duration.ToText(duration as nullable duration, optional format as nullable text) as nullable text
```

Sobre

Retorna uma representação textual no formato "day.hour:mins:sec" do valor de duração especificado, `duration`.

- `duration`: Uma `duration` com base na qual a representação textual é calculada.
- `format`: *[opcional]* preterida; gera um erro se o valor não é nulo.

Exemplo 1

Converter `#duration(2, 5, 55, 20)` em um valor de texto.

```
Duration.ToText(#duration(2, 5, 55, 20))
```

`"2.05:55:20"`

#duration

09/05/2020 • 2 minutes to read

Sintaxe

```
#duration(days as number, hours as number, minutes as number, seconds as number) as duration
```

Sobre

Cria um valor de duração usando os números de dias `days`, horas `hours`, minutos `minutes` e segundos `seconds`.

Tratamento de erro

08/05/2020 • 2 minutes to read

Essas funções retornam rastreamentos de diagnóstico em diferentes níveis de detalhamento, além de gerar registros de erros.

Erro

FUNÇÃO	DESCRIÇÃO
Diagnostics.ActivityId	Retorna um identificador opaco para a avaliação atualmente em execução.
Diagnostics.Trace	Grava uma entrada de rastreamento se o rastreamento está habilitado e retorna o valor.
Error.Record	Retorna um registro que contém os campos "Motivo", "Mensagem" e "Detalhes" definidos com os valores fornecidos. O registro pode ser usado para gerar um erro.
TraceLevel.Critical	Retorna 1, o valor para o nível de rastreamento Crítico.
TraceLevel.Error	Retorna 2, o valor para o nível de rastreamento de Erro.
TraceLevel.Information	Retorna 4, o valor do nível de rastreamento de Informações.
TraceLevel.Verbose	Retorna 5, o valor para o nível de rastreamento Detalhado.
TraceLevel.Warning	Retorna 3, o valor do nível de rastreamento de Aviso.

Diagnostics.ActivityId

09/05/2020 • 2 minutes to read

`

Sintaxe

```
Diagnostics.ActivityId() as nullable text
```

Sobre

Retorna um identificador opaco para a avaliação atualmente em execução.

Diagnostics.Trace

09/05/2020 • 2 minutes to read

Sintaxe

```
Diagnostics.Trace(traceLevel as number, message as anynonnull, value as any, optional delayed as nullable logical) as any
```

Sobre

Grava uma `message` de rastreamento, se o rastreamento está habilitado e retorna `value`. Um parâmetro opcional `delayed` especifica se a avaliação de `value` deve ser atrasada até a mensagem ser rastreada. `traceLevel` pode usar um dos seguintes valores:

- `TraceLevel.Critical` - `TraceLevel.Error`

- `TraceLevel.Warning`
- `TraceLevel.Information`
- `TraceLevel.Verbose`

Exemplo 1

Rastrear a mensagem antes de invocar a função Text.From e retornar o resultado.

```
Diagnostics.Trace(TraceLevel.Information, "TextValueFromNumber", () => Text.From(123), true)
```

`"123"`

Error.Record

09/05/2020 • 2 minutes to read

Sintaxe

```
Error.Record(reason as text, optional message as nullable text, optional detail as any) as record
```

Sobre

Retorna um registro de erro dos valores de texto fornecidos para motivo, mensagem e detalhes.

TraceLevel.Critical

09/05/2020 • 2 minutes to read

Sobre

Retorna 1, o valor para o nível de rastreamento Crítico.

TraceLevel.Error

09/05/2020 • 2 minutes to read

Sobre

Retorna 2, o valor para o nível de rastreamento de Erro.

TraceLevel.Information

09/05/2020 • 2 minutes to read

Sobre

Retorna 4, o valor do nível de rastreamento de Informações.

TraceLevel.Verbose

09/05/2020 • 2 minutes to read

Sobre

Retorna 5, o valor para o nível de rastreamento Detalhado.

TraceLevel.Warning

09/05/2020 • 2 minutes to read

Sobre

Retorna 3, o valor do nível de rastreamento de Aviso.

Funções de expressão

08/05/2020 • 2 minutes to read

Essas funções permitem a construção e a avaliação do código M.

Expressão

FUNÇÃO	DESCRIÇÃO
Expression.Constant	Retorna a representação do código-fonte M de um valor constante.
Expression.Evaluate	Retorna o resultado da avaliação de uma expressão M.
Expression.Identifier	Retorna a representação do código-fonte M de um identificador.

Expression.Constant

09/05/2020 • 2 minutes to read

Sintaxe

```
Expression.Constant(value as any) as text
```

Sobre

Retorna a representação do código-fonte M de um valor constante.

Exemplo 1

Obter a representação do código-fonte M de um valor numérico.

```
Expression.Constant(123)
```

```
"123"
```

Exemplo 2

Obter a representação do código-fonte M de um valor de data.

```
Expression.Constant(#date(2035, 01, 02))
```

```
"#date(2035, 1, 2)"
```

Exemplo 3

Obter a representação do código-fonte M de um valor de texto.

```
Expression.Constant("abc")
```

```
"""abc"""
```

Expression.Evaluate

09/05/2020 • 2 minutes to read

Sintaxe

```
Expression.Evaluate(document as text, optional environment as nullable record) as any
```

Sobre

Retorna o resultado da avaliação de uma expressão M `document`, com os identificadores disponíveis que podem ser referenciados por `environment`.

Exemplo 1

Avaliar uma soma simples.

```
Expression.Evaluate("1 + 1")
```

`2`

Exemplo 2

Avaliar uma soma mais complexa.

```
Expression.Evaluate("List.Sum({1, 2, 3})", [List.Sum = List.Sum])
```

`6`

Exemplo 3

Avaliar a concatenação de um valor de texto com um identificador.

```
Expression.Evaluate(Expression.Constant("""abc") & " & " & Expression.Identifier("x"), [x = "def"""])
```

`"""abcdef"""`

Expression.Identifier

09/05/2020 • 2 minutes to read

Sintaxe

```
Expression.Identifier(name as text) as text
```

Sobre

Retorna a representação do código-fonte M de um identificador `name`.

Exemplo 1

Obter a representação do código-fonte M de um identificador.

```
Expression.Identifier("MyIdentifier")
```

```
"MyIdentifier"
```

Exemplo 2

Obter a representação do código-fonte M de um identificador que contenha um espaço.

```
Expression.Identifier("My Identifier")
```

```
"#""My Identifier"""
```

Valores de função

08/05/2020 • 2 minutes to read

Essas funções criam e invocam outras funções M.

Função

FUNÇÃO	DESCRIÇÃO
Function.From	Usa uma função unária `function` e cria uma nova função com o tipo `functionType`, que cria uma lista dos seus argumentos e a transfere para `function`.
Function.Invoke	Chama a função fornecida usando o especificado e retorna o resultado.
Function.InvokeAfter	Retorna o resultado da função de invocação após o atraso de duração ter passado.
Function.IsDataSource	Retorna se a função é considerada ou não uma fonte de dados.
Function.ScalarVector	Retorna uma função escalar do tipo scalarFunctionType que invoca vectorFunction com uma única linha de argumentos e retorna a única saída.

Function.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Function.From(functionType as type, function as function) as function
```

Sobre

Usa uma função unária `function` e cria uma nova função com o tipo `functionType`, que cria uma lista dos seus argumentos e a transfere para `function`.

Exemplo 1

Converte List.Sum em uma função de dois argumentos que são adicionados de modo conjunto.

```
Function.From(type function (a as number, b as number) as number, List.Sum)(2, 1)
```

`3`

Exemplo 2

Converte uma função que leva uma lista para uma função de dois argumentos.

```
Function.From(type function (a as text, b as text) as text, (list) => list{0} & list{1})("2", "1")
```

`"21"`

Function.Invoke

08/05/2020 • 2 minutes to read

Sintaxe

```
Function.Invoke(function as function, args as list) as any
```

Sobre

Chama a função fornecida usando a lista de argumentos especificada e retorna o resultado.

Exemplo 1

Chama Record.FieldNames com um argumento [A=1,B=2]

```
Function.Invoke(Record.FieldNames, {[A = 1, B = 2]}
```

A
B

Function.InvokeAfter

09/05/2020 • 2 minutes to read

Sintaxe

```
Function.InvokeAfter(function as function, delay as duration) as any
```

Sobre

Retorna o resultado da invocação de `function` após o decorrer da duração `delay`.

Function.IsDataSource

09/05/2020 • 2 minutes to read

Sintaxe

```
Function.IsDataSource(function as function) as logical
```

Sobre

Retorna se `function` é considerado ou não uma fonte de dados.

Function.ScalarVector

09/05/2020 • 2 minutes to read

Sintaxe

```
Function.ScalarVector(scalarFunctionType as type, vectorFunction as function) as function
```

Sobre

Retorna uma função escalar do tipo `scalarFunctionType` que invoca `vectorFunction` com uma única linha de argumentos e retorna a única saída. Além disso, quando a função escalar é aplicada repetidamente a cada linha de uma tabela de entradas, como em Table.AddColumn, em vez disso, `vectorFunction` será aplicada uma vez a todas as entradas.

`vectorFunction` receberá uma tabela cujas colunas têm nomes correspondentes e posicionam os parâmetros de `scalarFunctionType`. Cada linha dessa tabela contém os argumentos para uma chamada à função escalar, com as colunas correspondentes aos parâmetros de `scalarFunctionType`.

`vectorFunction` precisa retornar uma lista com o mesmo tamanho da tabela de entrada, cujo item em cada posição precisa ter o mesmo resultado da avaliação da função escalar na linha de entrada da mesma posição.

A tabela de entrada deve ser transmitida e, portanto, `vectorFunction` deve transmitir a saída conforme a entrada é recebida, trabalhando apenas com uma parte da entrada de cada vez. Em particular, `vectorFunction` não deve enumerar a tabela de entrada mais de uma vez.

Funções de linha

08/05/2020 • 2 minutes to read

Essas funções convertem listas de texto de e para valores de texto único e binário.

Linhas

FUNÇÃO	DESCRIÇÃO
Lines.FromBinary	Converte um valor binário em uma lista de valores de texto dividida em quebras de linha.
Lines.FromText	Converte um valor de texto em uma lista de valores de texto dividida em quebras de linha.
Lines.ToBinary	Converte uma lista de texto em valor binário usando a codificação especificada e o lineSeparator. O lineSeparator especificado é acrescentado a cada linha. Se não for especificado, o retorno de carro e os caracteres de alimentação de linha serão usados.
Lines.ToText	Converte uma lista de texto em um só texto. O lineSeparator especificado é acrescentado a cada linha. Se não for especificado, o retorno de carro e os caracteres de alimentação de linha serão usados.

Lines.FromBinary

09/05/2020 • 2 minutes to read

Sintaxe

```
Lines.FromBinary(binary as binary, optional quoteStyle as nullable number, optional
includeLineSeparators as nullable logical, optional encoding as nullable number) as list
```

Sobre

Converte um valor binário em uma lista de valores de texto dividida em quebras de linha. Se um estilo de aspas for especificado, as quebras de linha poderão aparecer entre aspas. Se includeLineSeparators for true, os caracteres de quebra de linha serão incluídos no texto.

Lines.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Lines.FromText(text as text, optional quoteStyle as nullable number, optional
includeLineSeparators as nullable logical) as list
```

Sobre

Converte um valor de texto em uma lista de valores de texto dividida em quebras de linha. Se includeLineSeparators for true, os caracteres de quebra de linha serão incluídos no texto.

- `QuoteStyle.None:` (padrão) Nenhum comportamento de aspas é necessário.
- `QuoteStyle.Csv:` As aspas são de acordo com o CSV. Um caractere de aspas duplas é usado para demarcar essas regiões e um par de caracteres de aspas duplas é usado para indicar um único caractere de aspas duplas nessa região.

Lines.ToBinary

09/05/2020 • 2 minutes to read

Sintaxe

```
Lines.ToBinary(lines as list, optional lineSeparator as nullable text, optional encoding as
nullable number, optional includeByteOrderMark as nullable logical) as binary
```

Sobre

Converte uma lista de texto em valor binário usando a codificação especificada e o lineSeparator. O lineSeparator especificado é acrescentado a cada linha. Se não for especificado, o retorno de carro e os caracteres de alimentação de linha serão usados.

Lines.ToText

09/05/2020 • 2 minutes to read

Sintaxe

```
Lines.ToText(lines as list, optional lineSeparator as nullable text) as text
```

Sobre

Converte uma lista de texto em um só texto. O lineSeparator especificado é acrescentado a cada linha. Se não for especificado, o retorno de carro e os caracteres de alimentação de linha serão usados.

Funções de lista

30/09/2020 • 16 minutes to read

Essas funções criam e manipulam valores de lista.

Informações

FUNÇÃO	DESCRIÇÃO
List.Count	Retorna o número de itens em uma lista.
List.NonNullCount	Retorna o número de itens em uma lista excluindo valores nulos
List.IsEmpty	Retorna uma mensagem informando se uma lista está vazia.

Seleção

FUNÇÃO	DESCRIÇÃO
List.Alternate	Retorna uma lista com os itens alternados da lista original baseado em uma contagem, repeatInterval opcional e um deslocamento opcional.
List.Buffer	Armazena em buffer a lista na memória. O resultado dessa chamada é uma lista estável, o que significa que ela terá uma ordem dos itens e uma contagem determinísticas.
List.Distinct	Filtra uma lista removendo duplicatas. Um valor de critérios de equação opcional pode ser especificado para controlar a comparação de igualdade. O primeiro valor de cada grupo de igualdade é escolhido.
List.FindText	Pesquisa uma lista de valores, incluindo campos de registro, para um valor de texto.
List.First	Retorna o primeiro valor da lista ou o padrão especificado se ela está vazia. Retorna o primeiro item da lista ou o valor padrão opcional se a lista está vazia. Se a lista estiver vazia e um valor padrão não for especificado, a função retornará.
List.FirstN	Retorna o primeiro conjunto de itens da lista especificando quantos itens devem ser retornados ou uma condição de qualificação fornecida por **countOrCondition**.
List.InsertRange	Insere itens de valores no índice fornecido na lista de entrada.
List.IsDistinct	Retorna uma mensagem informando se uma lista é distinta.
List.Last	Retorna o último conjunto de itens da lista especificando quantos itens devem ser retornados ou uma condição de qualificação fornecida por **countOrCondition**.

FUNÇÃO	DESCRIÇÃO
List.LastN	Retorna o último conjunto de itens de uma lista especificando quantos itens devem ser retornados ou uma condição de qualificação.
List.MatchesAll	Retorna verdadeiro se todos os itens de uma lista atendem a uma condição.
List.MatchesAny	Retorna verdadeiro se qualquer item de uma lista atende a uma condição.
List.Positions	Retorna uma lista de posições de uma lista de entrada.
List.Range	Retorna uma contagem de itens começando em um deslocamento.
List.Select	Seleciona os itens que correspondem a uma condição.
List.Single	Retorna o único item da lista ou gera um Expression.Error se a lista tem mais de um item.
List.SingleOrDefault	Retorna um único item de uma lista.
List.Skip	Ignora o primeiro item da lista. Considerando uma lista vazia, ele retorna uma lista vazia. Essa função usa um parâmetro opcional countOrCondition para dar suporte ao recurso de ignorar vários valores.

Funções de transformação

FUNÇÃO	DESCRIÇÃO
List.Accumulate	Acumula um resultado da lista. Começando na semente de valor inicial, essa função aplica a função de acumulador e retorna o resultado final.
List.Combine	Mescla uma lista de listas em uma só lista.
List.ConformToPageReader	Esta função destina-se somente a uso interno.
List.RemoveRange	Retorna uma lista que remove itens de contagem começando no deslocamento. A contagem padrão é 1.
List.RemoveFirstN	Retorna uma lista com o número especificado de elementos removidos da lista, começando no primeiro elemento. O número de elementos removidos depende do parâmetro opcional countOrCondition.
List.RemoveItems	Remove itens de list1 que estão presentes em list2 e retorna uma nova lista.
List.RemoveLastN	Retorna uma lista com o número especificado de elementos removidos da lista, começando no último elemento. O número de elementos removidos depende do parâmetro opcional countOrCondition.

FUNÇÃO	DESCRIÇÃO
List.Repeat	Retorna uma lista que repete o conteúdo do número de contagens de uma lista de entrada.
List.ReplaceRange	Retorna uma lista que substitui valores de contagem em uma lista por uma lista replaceWith começando em um índice.
List.RemoveMatchingItems	Remove todas as ocorrências dos valores especificados na lista.
List.RemoveNulls	Remove os valores nulos de uma lista.
List.ReplaceMatchingItems	Substitui as ocorrências de valores existentes na lista por novos valores usando o equationCriteria fornecido. Os valores novos e antigos são fornecidos pelos parâmetros de substituição. Um valor de critérios de equação opcional pode ser especificado para controlar comparações de igualdade. Para obter detalhes sobre operações de substituição e critérios de equação, confira Valores de parâmetros.
List.ReplaceValue	Pesquisa o valor em uma lista de valores e substitui cada ocorrência pelo valor de substituição.
List.Reverse	Retorna uma lista que reverte os itens em uma lista.
List.Split	Divide a lista especificada em uma lista de listas usando o tamanho da página especificado.
List.Transform	Executa a função em cada item da lista e retorna a nova lista.
List.TransformMany	Retorna uma lista cujos elementos são projetados com base na lista de entrada.

Funções de associação

Como todos os valores podem ser testados quanto à igualdade, essas funções podem operar em listas heterogêneas.

FUNÇÃO	DESCRIÇÃO
List.AllTrue	Retorna verdadeiro se todas as expressões de uma lista são verdadeiras
List.AnyTrue	Retorna verdadeiro se qualquer expressão de uma lista é verdadeira
List.Contains	Retorna verdadeiro se um valor é encontrado em uma lista.
List.ContainsAll	Retorna verdadeiro se todos os itens nos valores são encontrados em uma lista.
List.ContainsAny	Retorna verdadeiro se qualquer item nos valores é encontrado em uma lista.
List.PositionOf	Encontra a primeira ocorrência de um valor em uma lista e retorna a posição dela.

FUNÇÃO	DESCRIÇÃO
List.PositionOfAny	Encontra a primeira ocorrência de qualquer valor nos valores e retorna a posição dela.

Operações de conjuntos

FUNÇÃO	DESCRIÇÃO
List.Difference	Retorna os itens da lista 1 que não aparecem na lista 2. Há suporte para valores duplicados.
List.Intersect	Retorna uma lista de uma lista de listas e interseciona itens comuns em listas individuais. Há suporte para valores duplicados.
List.Union	Retorna uma lista de uma lista de listas e une os itens nas listas individuais. A lista retornada contém todos os itens em listas de entrada. É feita a correspondência dos valores duplicados como parte da união.
List.Zip	Retorna uma lista de listas combinando itens na mesma posição.

Ordenando

As funções de ordenação fazem comparações. Todos os valores comparados precisam ser comparáveis entre si. Isso significa que todos eles precisam ser provenientes do mesmo tipo de dados (ou incluir nulo, que sempre compara o menor). Caso contrário, um Expression.Error é gerado.

Tipos de dados comparáveis

- Número
- Duração
- Datetime
- Texto
- Lógico
- Nulo

FUNÇÃO	DESCRIÇÃO
List.Max	Retorna o item máximo em uma lista ou o valor padrão opcional se a lista está vazia.
List.MaxN	Retorna os valores máximos na lista. Depois que as linhas são classificadas, parâmetros opcionais podem ser especificados para filtrar ainda mais o resultado
List.Median	Retorna o item mediano de uma lista.
List.Min	Retorna o item mínimo em uma lista ou o valor padrão opcional se a lista está vazia.

FUNÇÃO	DESCRIÇÃO
List.MinN	Retorna os valores mínimos em uma lista.
List.Sort	Retorna uma lista classificada usando o critério de comparação.

Averages

Essas funções operam em listas homogêneas de Numbers, DateTimes e Durations.

FUNÇÃO	DESCRIÇÃO
List.Average	Retorna um valor médio de uma lista no tipo de dados dos valores na lista.
List.Mode	Retorna um item que aparece mais comumente em uma lista.
List.Modes	Retorna todos os itens que aparecem com a mesma frequência máxima.
List.StandardDeviation	Retorna o desvio padrão de uma lista de valores. List.StandardDeviation faz uma estimativa baseada em amostra. O resultado é um número de números e uma duração de DateTimes e Durations.

Adição

Essas funções operam em listas homogêneas de Numbers ou Durations.

FUNÇÃO	DESCRIÇÃO
List.Sum	Retorna a soma de uma lista.

Numéricos

Essas funções só operam em números.

FUNÇÃO	DESCRIÇÃO
List.Covariance	Retorna a covariância de duas listas como um número.
List.Product	Retorna o produto de uma lista de números.

Geradores

Essas funções geram uma lista de valores.

FUNÇÃO	DESCRIÇÃO
List.Dates	Retorna uma lista de valores de data da contagem de tamanho, começando no início, e adiciona um incremento a cada valor.
List.DateTimes	Retorna uma lista de valores de datetime da contagem de tamanho, começando no início, e adiciona um incremento a cada valor.

FUNÇÃO	DESCRIÇÃO
List.DateTimeZones	Retorna uma lista de valores de datetimezone da contagem de tamanho, começando no início, e adiciona um incremento a cada valor.
List.Durations	Retorna uma lista de valores de durações da contagem de tamanho, começando no início, e adiciona um incremento a cada valor.
List.Generate	Gera uma lista com base em uma função de valor, uma função de condição, uma função next e uma função de transformação opcional nos valores.
List.Numbers	Retorna uma lista de números da contagem de tamanho, começando no início, e adiciona um incremento. O incremento usa 1 como padrão.
List.Random	Retorna uma lista de números aleatórios de contagem, com um parâmetro de semente opcional.
List.Times	Retorna uma lista de valores de tempo da contagem de tamanho, começando no início.

Valores de parâmetro

Especificação de ocorrência

- Occurrence.First = 0;
- Occurrence.Last = 1;
- Occurrence.All = 2;

Ordem de classificação

- Order.Ascending = 0;
- Order.Descending = 1;

Critérios de equação

Os critérios de equação para valores de lista podem ser especificados como

- Um valor de função que é
  - Um seletor de chave que determina o valor na lista para aplicar os critérios de igualdade ou
  - Uma função de comparador que é usada para especificar o tipo de comparação a ser aplicado. As funções de comparador internas podem ser especificadas; confira a seção Funções de comparação.
- Um valor de lista que tenha
  - Exatamente dois itens
  - O primeiro elemento é o seletor de chave, conforme especificado acima
  - O segundo elemento é um comparador, conforme especificado acima.

Para obter mais informações e exemplos, confira List.Distinct.

Critérios de comparação

O critério de comparação pode ser fornecido como um dos seguintes valores:

- Um valor numérico para especificar uma ordem de classificação. Para obter mais informações, confira a ordem de classificação em Valores de parâmetros.
- Para calcular uma chave a ser usada para classificação, uma função de um argumento pode ser usada.
- Para selecionar uma ordem de chave e de controle, o critério de comparação pode ser uma lista que contém a chave e a ordem.
- Para controlar por completo a comparação, é possível usar uma função de dois argumentos que retorna -1, 0 ou 1, considerando a relação entre as entradas esquerda e direita. Value.Compare é um método que pode ser usado para delegar essa lógica.

Para obter mais informações e exemplos, confira List.Sort.

Operações de substituição

As operações de substituição são especificadas por um valor de lista. Cada item dessa lista precisa ser

- Um valor de lista com exatamente dois itens
- O primeiro item é o valor antigo na lista, a ser substituído
- O segundo item é o novo, que deve substituir todas as ocorrências do valor antigo na lista

List.Accumulate

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Accumulate(list as list, seed as any, accumulator as function) as any
```

Sobre

Acumula um valor de resumo com base nos itens da lista `list`, usando `accumulator`. Um parâmetro semente opcional, `seed`, pode ser definido.

Exemplo 1

Acumula o valor de resumo dos itens na lista {1, 2, 3, 4, 5} usando ((state, current) => state + current).

```
List.Accumulate({1, 2, 3, 4, 5}, 0, (state, current) => state + current)
```

```
15
```

List.AllTrue

09/05/2020 • 2 minutes to read

Sintaxe

```
List.AllTrue(list as list) as logical
```

Sobre

Retorna verdadeiro se todas as expressões da lista `list` são verdadeiras.

Exemplo 1

Determinar se todas as expressões da lista {true, true, 2 > 0} são verdadeiras.

```
List.AllTrue({true, true, 2 > 0})
```

`true`

Exemplo 2

Determinar se todas as expressões da lista {true, true, 2 < 0} são verdadeiras.

```
List.AllTrue({true, false, 2 < 0})
```

`false`

List.Alternate

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Alternate(list as list, count as number, optional repeatInterval as nullable number, optional offset as nullable number) as list
```

Sobre

Retorna uma lista composta por todos os elementos de deslocamento de número ímpar em uma lista. Alterna entre usar e ignorar valores da lista `list`, dependendo dos parâmetros.

- `count`: Especifica o número de valores que são ignorados a cada vez.
- `repeatInterval`: Um intervalo de repetição opcional para indicar quantos valores são adicionados entre os valores ignorados.
- `offset`: Um parâmetro de deslocamento opcional para começar a ignorar os valores no deslocamento inicial.

Exemplo 1

Crie uma lista com base em {1..10} que ignore o primeiro número.

```
List.Alternate({1..10}, 1)
```

2
3
4
5
6
7
8
9
10

Exemplo 2

Crie uma lista com base em {1..10} que ignore todos os outros números.

```
List.Alternate({1..10}, 1, 1)
```

2
4
6
8
10

Exemplo 3

Crie uma lista com base em {1..10} que comece em 1 e ignore todos os outros números.

```
List.Alternate({1..10}, 1, 1, 1)
```

1
3
5
7
9

Exemplo 4

Crie uma lista com base em {1..10} que comece em 1, ignore um valor, mantenha dois valores e assim por diante.

```
List.Alternate({1..10}, 1, 2, 1)
```

1
3
4
6
7
9
10

List.AnyTrue

09/05/2020 • 2 minutes to read

Sintaxe

```
List.AnyTrue(list as list) as logical
```

Sobre

Retorna verdadeiro se qualquer expressão da lista `list` é verdadeira.

Exemplo 1

Determinar se uma das expressões da lista {true, false, 2 > 0} é verdadeira.

```
List.AnyTrue({true, false, 2>0})
```

`true`

Exemplo 2

Determinar se uma das expressões da lista {2 = 0, false, 2 < 0} é verdadeira.

```
List.AnyTrue({2 = 0, false, 2 < 0})
```

`false`

List.Average

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Average(list as list, optional precision as nullable number) as any
```

Sobre

Retorna o valor médio dos itens na lista `list`. O resultado é fornecido no mesmo tipo de dados que os valores na lista. Funciona somente com valores number, date, datetime, datetimezone e duration. Se a lista estiver vazia, nulo será retornado.

Exemplo 1

Localize a média da lista de números, `{3, 4, 6}`.

```
List.Average({3, 4, 6})
```

`4.333333333333333`

Exemplo 2

Localize a média dos valores date 1º de janeiro de 2011, 2 de janeiro de 2011 e 3 de janeiro de 2011.

```
List.Average({#date(2011, 1, 1), #date(2011, 1, 2), #date(2011, 1, 3)})
```

`#date(2011, 1, 2)`

List.Buffer

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Buffer(list as list) as list
```

Sobre

Armazena em buffer a lista `list` na memória. O resultado dessa chamada é uma lista estável.

Exemplo 1

Criar uma cópia estável da lista {1..10}.

```
List.Buffer({1..10})
```

1
2
3
4
5
6
7
8
9
10

List.Combine

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Combine(lists as list) as list
```

Sobre

Faça uma lista das listas, `lists`, e mescle-as em uma única lista nova.

Exemplo 1

Combine as duas listas simples {1, 2} e {3, 4}.

```
List.Combine({{1, 2}, {3, 4}})
```

1
2
3
4

Exemplo 2

Combine as duas listas, {1, 2} e {3, {4, 5}}, uma delas contendo uma lista aninhada.

```
List.Combine({{1, 2}, {3, {4, 5}}})
```

1
2
3
[Lista]

List.ConformToPageReader

30/09/2020 • 2 minutes to read

Sintaxe

```
List.ConformToPageReader(list as list, optional options as nullable record) as table
```

Sobre

Esta função destina-se somente a uso interno.

List.Contains

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Contains(list as list, value as any, optional equationCriteria as any) as logical
```

Sobre

Indica se a lista `list` contém o valor `value`. Retorna true se o valor é encontrado na lista; caso contrário, false. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Descobrir se a lista {1, 2, 3, 4, 5} contém 3.

```
List.Contains({1, 2, 3, 4, 5}, 3)
```

`true`

Exemplo 2

Descobrir se a lista {1, 2, 3, 4, 5} contém 6.

```
List.Contains({1, 2, 3, 4, 5}, 6)
```

`false`

List.ContainsAll

09/05/2020 • 2 minutes to read

Sintaxe

```
List.ContainsAll(list as list, values as list, optional equationCriteria as any) as logical
```

Sobre

Indica se a lista `list` inclui todos os valores em outra lista, `values`. Retorna true se o valor é encontrado na lista; caso contrário, false. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Descubra se a lista {1, 2, 3, 4, 5} contém 3 e 4.

```
List.ContainsAll({1, 2, 3, 4, 5}, {3, 4})
```

`true`

Exemplo 2

Descubra se a lista {1, 2, 3, 4, 5} contém 5 e 6.

```
List.ContainsAll({1, 2, 3, 4, 5}, {5, 6})
```

`false`

List.ContainsAny

09/05/2020 • 2 minutes to read

Sintaxe

```
List.ContainsAny(list as list, values as list, optional equationCriteria as any) as logical
```

Sobre

Indica se a lista `list` inclui um dos valores em outra lista, `values`. Retorna true se o valor é encontrado na lista; caso contrário, false. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Descobrir se a lista {1, 2, 3, 4, 5} contém 3 ou 9.

```
List.ContainsAny({1, 2, 3, 4, 5}, {3, 9})
```

`true`

Exemplo 2

Descobrir se a lista {1, 2, 3, 4, 5} contém 6 ou 7.

```
List.ContainsAny({1, 2, 3, 4, 5}, {6, 7})
```

`false`

List.Count

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Count(list as list) as number
```

Sobre

Retorna o número de itens na lista `list`.

Exemplo 1

Localize o número de valores na lista {1, 2, 3}.

```
List.Count({1, 2, 3})
```

`3`

List.Covariance

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Covariance(numberList1 as list, numberList2 as list) as nullable number
```

Sobre

Retorna a covariância entre duas listas, `numberList1` e `numberList2`. `numberList1` e `numberList2` precisam conter o mesmo número de valores `number`.

Exemplo 1

Calcular a covariância entre duas listas.

```
List.Covariance({1, 2, 3}, {1, 2, 3})
```

```
0.66666666666666607
```

List.Dates

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Dates(start as date, count as number, step as duration) as list
```

Sobre

Retorna uma lista de valores `date` do tamanho `count`, começando em `start`. O incremento especificado, `step`, é um valor `duration` que é adicionado a cada valor.

Exemplo 1

Crie uma lista de 5 valores que comece na véspera do ano novo (#date(2011, 12, 31)) e cujos valores sejam incrementados em um dia (#duration(1, 0, 0, 0)).

```
List.Dates(#date(2011, 12, 31), 5, #duration(1, 0, 0, 0))
```

31/12/2011 12:00:00 AM
1/1/2012 12:00:00 AM
2/1/2012 12:00:00 AM
3/1/2012 12:00:00 AM
4/1/2012 12:00:00 AM

List.DateTimes

09/05/2020 • 2 minutes to read

Sintaxe

```
List.DateTimes(start as datetime, count as number, step as duration) as list
```

Sobre

Retorna uma lista de valores `datetime` do tamanho `count`, começando em `start`. O incremento especificado, `step`, é um valor `duration` que é adicionado a cada valor.

Exemplo

Criar uma lista de dez valores que começa em cinco minutos antes do primeiro dia do ano [#datetime(2011, 12, 31, 23, 55, 0)] e com um incremento de um minuto [#duration(0, 0, 1, 0)].

```
List.DateTimes(#datetime(2011, 12, 31, 23, 55, 0), 10, #duration(0, 0, 1, 0))
```

31/12/2011 11:55:00 PM
31/12/2011 11:56:00 PM
31/12/2011 11:57:00 PM
31/12/2011 11:58:00 PM
31/12/2011 11:59:00 PM
1/1/2012 12:00:00 AM
1/1/2012 12:01:00 AM
1/1/2012 12:02:00 AM
1/1/2012 12:03:00 AM
1/1/2012 12:04:00 AM

List.DateTimeZones

09/05/2020 • 2 minutes to read

Sintaxe

```
List.DateTimeZones(start as datetimezone, count as number, step as duration) as list
```

Sobre

Retorna uma lista de valores `datetimezone` do tamanho `count`, começando em `start`. O incremento especificado, `step`, é um valor `duration` que é adicionado a cada valor.

Exemplo 1

Crie uma lista de dez valores que comece em 5 minutos antes do dia do ano novo (#datetimezone(2011, 12, 31, 23, 55, 0, -8, 0)) e cujos valores sejam incrementados em um minuto (#duration(0, 0, 1, 0)).

```
List.DateTimeZones(#datetimezone(2011, 12, 31, 23, 55, 0, -8, 0), 10, #duration(0, 0, 1, 0))
```

12/31/2011 11:55:00 PM -08:00
12/31/2011 11:56:00 PM -08:00
12/31/2011 11:57:00 PM -08:00
12/31/2011 11:58:00 PM -08:00
12/31/2011 11:59:00 PM -08:00
1/1/2012 12:00:00 AM -08:00
1/1/2012 12:01:00 AM -08:00
1/1/2012 12:02:00 AM -08:00
1/1/2012 12:03:00 AM -08:00
1/1/2012 12:04:00 AM -08:00

List.Difference

08/05/2020 • 2 minutes to read

```
List.Difference(list1 as list, list2 as list, optional equationCriteria as any) as list
```

Sobre

Retorna os itens da lista `list1` que não aparecem na lista `list2`. Há suporte para valores duplicados. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Encontrar os itens da lista {1, 2, 3, 4, 5} que não aparecem em {4, 5, 3}.

```
List.Difference({1, 2, 3, 4, 5}, {4, 5, 3})
```

1
2

Exemplo 2

Encontrar os itens da lista {1, 2} que não aparecem em {1, 2, 3}.

```
List.Difference({1, 2}, {1, 2, 3})
```

List.Distinct

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Distinct(list as list, optional equationCriteria as any) as list
```

Sobre

Retorna uma lista que contém todos os valores na lista `list` com duplicatas removidas. Se a lista estiver vazia, o resultado será uma lista vazia.

Exemplo 1

Remova as duplicatas da lista {1, 1, 2, 3, 3, 3}.

```
List.Distinct({1, 1, 2, 3, 3, 3})
```

1
2
3

List.Durations

12/05/2020 • 2 minutes to read

Sintaxe

```
List.Durations(start as duration, count as number, step as duration) as list
```

Sobre

Retorna uma lista de valores `count` `duration`, começando em `start` e incrementada pelo `duration` `step` fornecido.

Exemplo

Criar uma lista de cinco valores que começa em uma hora e com um incremento de uma hora.

```
List.Durations(#duration(0, 1, 0, 0), 5, #duration(0, 1, 0, 0))
```

01:00:00
02:00:00
03:00:00
04:00:00
05:00:00

List.FindText

09/05/2020 • 2 minutes to read

Sintaxe

```
List.FindText(list as list, text as text) as list
```

Sobre

Retorna uma lista de valores da lista `list` que continha o valor `text`.

Exemplo 1

Localize os valores de texto da lista {"a", "b", "ab"} que correspondam a "a".

```
List.FindText({"a", "b", "ab"}, "a")
```

um
ab

List.First

09/05/2020 • 2 minutes to read

Sintaxe

```
List.First(list as list, optional defaultValue as any) as any
```

Sobre

Retorna o primeiro item da lista `list` ou o valor padrão opcional, `defaultValue`, se a lista está vazia. Se a lista estiver vazia e um valor padrão não for especificado, a função retornará `null`.

Exemplo 1

Encontrar o primeiro valor da lista {1, 2, 3}.

```
List.First({1, 2, 3})
```

`1`

Exemplo 2

Encontrar o primeiro valor da lista {}. Se a lista estiver vazia, retornará -1.

```
List.First({}, -1)
```

`-1`

List.FirstN

08/05/2020 • 2 minutes to read

Sintaxe

```
List.FirstN(list as list, countOrCondition as any) as any
```

Sobre

- Se um número for especificado, até que esse número de itens seja retornado.
- Se uma condição for especificada, serão retornados todos os itens que atendam inicialmente à condição. Quando um item falha na condição, nenhum outro item é considerado.

Exemplo 1

Localize os valores iniciais na lista {3, 4, 5, -1, 7, 8, 2} que sejam maiores que 0.

```
List.FirstN({3, 4, 5, -1, 7, 8, 2}, each _ > 0)
```

3
4
5

List.Generate

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Generate(initial as function, condition as function, next as function, optional selector as nullable function) as list
```

Sobre

Gera uma lista de valores com base em quatro funções que geram o valor inicial `initial`, realizam o teste com base em uma condição `condition` e, se o teste é bem-sucedido, selecionam o resultado e geram o próximo valor `next`. Um parâmetro opcional, `selector`, também pode ser especificado.

Exemplo 1

Crie uma lista que comece em 10, permaneça maior que 0 e decremente em 1.

```
List.Generate(() => 10, each _ > 0, each _ - 1)
```

10
9
8
7
6
5
4
3
2
1

Exemplo 2

Gere uma lista de registros contendo x e y, em que x é um valor e y é uma lista. x deve permanecer menor que 10 e representar o número de itens na lista y. Depois que a lista for gerada, retorne apenas os valores x.

```
List.Generate(
    () => [x = 1, y = {}],
    each [x] < 10,
    each [x = List.Count([y]), y = [y] & {x}],
    each [x]
)
```

1
0
1
2
3
4
5
6
7
8
9

List.InsertRange

09/05/2020 • 2 minutes to read

Sintaxe

```
List.InsertRange(list as list, index as number, values as list) as list
```

Sobre

Retorna uma nova lista produzida inserindo os valores de `values` em `list` no `index`. A primeira posição na lista é no índice 0.

- `list`: A lista de destino em que os valores devem ser inseridos.
- `index`: O índice da lista de destino (`list`) em que os valores devem ser inseridos. A primeira posição na lista é no índice 0.
- `values`: A lista de valores que devem ser inseridos em `list`.

Exemplo 1

Inserir a lista ({3, 4}) na lista de destino ({1, 2, 5}) no índice 2.

```
List.InsertRange({1, 2, 5}, 2, {3, 4})
```

1
2
3
4
5

Exemplo 2

Inserir uma lista com uma lista aninhada ({1, {1.1, 1.2}}) em uma lista de destino ({2, 3, 4}) no índice 0.

```
List.InsertRange({2, 3, 4}, 0, {1, {1.1, 1.2}})
```

1
[Lista]
2
3

4

List.Intersect

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Intersect(lists as list, optional equationCriteria as any) as list
```

Sobre

Retorna a interseção dos valores de lista encontrados na lista de entrada `lists`. Um parâmetro opcional, `equationCriteria`, pode ser especificado.

Exemplo 1

Localize a interseção das listas {1..5}, {2..6}, {3..7}.

```
List.Intersect({{1..5}, {2..6}, {3..7}})
```

3
4
5

List.IsDistinct

09/05/2020 • 2 minutes to read

Sintaxe

```
List.IsDistinct(list as list, optional equationCriteria as any) as logical
```

Sobre

Retorna um valor lógico se há duplicatas na lista `list`; `true` se a lista é distinta; `false` se há valores duplicados.

Exemplo 1

Descobrir se a lista {1, 2, 3} é distinta (ou seja, se não há duplicatas).

```
List.IsDistinct({1, 2, 3})
```

`true`

Exemplo 2

Descobrir se a lista {1, 2, 3, 3} é distinta (ou seja, se não há duplicatas).

```
List.IsDistinct({1, 2, 3, 3})
```

`false`

List.IsEmpty

09/05/2020 • 2 minutes to read

Sintaxe

```
List.IsEmpty(list as list) as logical
```

Sobre

Retorna `true` se a lista, `list`, não contém valores (comprimento 0). Se a lista contém valores (comprimento > 0), retorna `false`.

Exemplo 1

Descubra se a lista {} está vazia.

```
List.IsEmpty({})
```

`true`

Exemplo 2

Descubra se a lista {1, 2} está vazia.

```
List.IsEmpty({1, 2})
```

`false`

List.Last

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Last(list as list, optional defaultValue as any) as any
```

Sobre

Retorna o último item da lista `list` ou o valor padrão opcional, `defaultValue`, se a lista está vazia. Se a lista estiver vazia e um valor padrão não for especificado, a função retornará `null`.

Exemplo 1

Localizar o último valor da lista {1, 2, 3}.

```
List.Last({1, 2, 3})
```

`3`

Exemplo 2

Localizar o último valor da lista {} ou -1 se ela estiver vazia.

```
List.Last({}, -1)
```

`-1`

List.LastN

08/05/2020 • 2 minutes to read

Sintaxe

```
List.LastN(list as list, optional countOrCondition as any) as any
```

Sobre

Retorna o último item da lista `list`. Se a lista estiver vazia, uma exceção será gerada. Essa função usa um parâmetro opcional, `countOrCondition`, para dar suporte à coleta de vários itens ou à filtragem de itens. `countOrCondition` pode ser especificado de três maneiras:

- Se um número for especificado, até que esse número de itens seja retornado.
- Se uma condição for especificada, serão retornados todos os itens que atendam inicialmente à condição, começando no fim da lista. Quando um item falha na condição, nenhum outro item é considerado.
- Se esse parâmetro for nulo, o último item da lista será retornado.

Exemplo 1

Localize o último valor da lista {3, 4, 5, -1, 7, 8, 2}.

```
List.LastN({3, 4, 5, -1, 7, 8, 2}, 1)
```

```
2
```

Exemplo 2

Localize os últimos valores da lista {3, 4, 5, -1, 7, 8, 2} que sejam maiores que 0.

```
List.LastN({3, 4, 5, -1, 7, 8, 2}, each _ > 0)
```

```
7
```

```
8
```

```
2
```

List.MatchesAll

08/05/2020 • 2 minutes to read

Sintaxe

```
List.MatchesAll(list as list, condition as function) as logical
```

Sobre

Retornará `true` se a função condition, `condition`, for atendida por todos os valores da lista `list`, caso contrário, retornará `false`.

Exemplo 1

Determine se todos os valores da lista {11, 12, 13} serão maiores que 10.

```
List.MatchesAll({11, 12, 13}, each _  > 10)
```

`true`

Exemplo 2

Determine se todos os valores da lista {1, 2, 3} serão maiores que 10.

```
List.MatchesAll({1, 2, 3}, each _  > 10)
```

`false`

List.MatchesAny

08/05/2020 • 2 minutes to read

Sintaxe

```
List.MatchesAny(list as list, condition as function) as logical
```

Sobre

Retornará `true` se a função condition, `condition`, for atendida por algum dos valores da lista `list`; caso contrário, retornará `false`.

Exemplo 1

Descubra se algum valor da lista {9, 10, 11} é maior que 10.

```
List.MatchesAny({9, 10, 11}, each _  > 10)
```

`true`

Exemplo 2

Descubra se algum valor da lista {1, 2, 3} é maior que 10.

```
List.MatchesAny({1, 2, 3}, each _  > 10)
```

`false`

List.Max

08/05/2020 • 2 minutes to read

Sintaxe

```
List.Max(list as list, optional default as any, optional comparisonCriteria as any, optional
includeNulls as nullable logical) as any
```

Sobre

Retorna o item máximo na lista `list` ou o valor padrão opcional `default`, se a lista está vazia. Um valor opcional de comparisonCriteria, `comparisonCriteria`, pode ser especificado para determinar como comparar os itens na lista. Se esse parâmetro for nulo, o comparador padrão será usado.

Exemplo 1

Localize o item máximo da lista {1, 4, 7, 3, -2, 5}.

```
List.Max({1, 4, 7, 3, -2, 5}, 1)
```

`7`

Exemplo 2

Localize o item máximo da lista {} ou retorne -1, se ela estiver vazia.

```
List.Max({}, -1)
```

`-1`

List.MaxN

09/05/2020 • 2 minutes to read

Sintaxe

```
List.MaxN(list as list, countOrCondition as any, optional comparisonCriteria as any, optional
includeNulls as nullable logical) as list
```

Sobre

Retorna os valores máximos na lista, `list`. Depois que as linhas são classificadas, parâmetros opcionais podem ser especificados para filtrar ainda mais o resultado. O parâmetro opcional, `countOrCondition`, especifica o número de valores a serem retornados ou uma condição de filtragem. O parâmetro opcional, `comparisonCriteria`, especifica como comparar valores na lista.

- `list`: Lista de valores.
- `countOrCondition`: Se um número for especificado, uma lista de até `countOrCondition` itens em ordem crescente será retornada. Se uma condição for especificada, uma lista de itens que atendam inicialmente à condição será retornada. Quando um item falha na condição, nenhum outro item é considerado.
- `comparisonCriteria`: *[Opcional]* Um valor de `comparisonCriteria` opcional pode ser especificado para determinar como comparar os itens na lista. Se esse parâmetro for nulo, o comparador padrão será usado.

List.Median

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Median(list as list, optional comparisonCriteria as any) as any
```

Sobre

Retorna o item mediano da lista `list`. Essa função retornará `null` se a lista não contiver valores não `null`. Se houver um número par de itens, a função escolherá o menor dos dois itens medianos, a menos que a lista seja composta inteiramente por datas/horas, durações, números ou horas, caso em que retornará a média dos dois itens.

Exemplo 1

Localize o item mediano da lista `{5, 3, 1, 7, 9}`.

```
powerquery-mList.Median({5, 3, 1, 7, 9})
```

`5`

List.Min

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Min(list as list, optional default as any, optional comparisonCriteria as any, optional
includeNulls as nullable logical) as any
```

Sobre

Retorna o item mínimo na lista `list` ou o valor padrão opcional `default`, se a lista está vazia. Um valor opcional de comparisonCriteria, `comparisonCriteria`, pode ser especificado para determinar como comparar os itens na lista. Se esse parâmetro for nulo, o comparador padrão será usado.

Exemplo 1

Localize o item mínimo da lista {1, 4, 7, 3, -2, 5}.

```
List.Min({1, 4, 7, 3, -2, 5})
```

`-2`

Exemplo 2

Localize o item mínimo da lista {} ou retorne -1, se ela estiver vazia.

```
List.Min({}, -1)
```

`-1`

List.MinN

09/05/2020 • 2 minutes to read

Sintaxe

```
List.MinN(list as list, countOrCondition as any, optional comparisonCriteria as any, optional
includeNulls as nullable logical) as list
```

Sobre

Retorna os valores mínimos na lista `list`. O parâmetro, `countOrCondition`, especifica o número de valores a serem retornados ou uma condição de filtragem. O parâmetro opcional, `comparisonCriteria`, especifica como comparar valores na lista.

- `list`: Lista de valores.
- `countOrCondition`: Se um número for especificado, uma lista de até `countOrCondition` itens em ordem crescente será retornada. Se uma condição for especificada, uma lista de itens que atendam inicialmente à condição será retornada. Quando um item falha na condição, nenhum outro item é considerado. Se esse parâmetro for nulo, o menor valor mínimo da lista será retornado.
- `comparisonCriteria`: *[Opcional]* Um valor de `comparisonCriteria` opcional pode ser especificado para determinar como comparar os itens na lista. Se esse parâmetro for nulo, o comparador padrão será usado.

Exemplo 1

Localize os cinco menores valores da lista `{3, 4, 5, -1, 7, 8, 2}`.

```
List.MinN({3, 4, 5, -1, 7, 8, 2}, 5)
```

-1
2
3
4
5

List.Mode

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Mode(list as list, optional equationCriteria as any) as any
```

Sobre

Retorna o item que aparece com mais frequência na lista, `list`. Se a lista estiver vazia, uma exceção será gerada. Se vários itens aparecerem com a mesma frequência máxima, o último será escolhido. Um valor de `comparisonCriteria` opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Localiza o item que aparece com mais frequência na lista, `{"A", 1, 2, 3, 3, 4, 5}`.

```
List.Mode({"A", 1, 2, 3, 3, 4, 5})
```

`3`

Exemplo 2

Localiza o item que aparece com mais frequência na lista, `{"A", 1, 2, 3, 3, 4, 5, 5}`.

```
List.Mode({"A", 1, 2, 3, 3, 4, 5, 5})
```

`5`

List.Modes

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Modes(list as list, optional equationCriteria as any) as list
```

Sobre

Retorna o item que aparece com mais frequência na lista, `list`. Se a lista estiver vazia, uma exceção será gerada. Se vários itens aparecerem com a mesma frequência máxima, o último será escolhido. Um valor de `comparisonCriteria` opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Localiza os itens que aparecem com mais frequência na lista, `{"A", 1, 2, 3, 3, 4, 5, 5}`.

```
List.Modes({"A", 1, 2, 3, 3, 4, 5, 5})
```

3
5

List.NonNullCount

09/05/2020 • 2 minutes to read

Sintaxe

```
List.NonNullCount(list as list) as number
```

Sobre

Retorna o número de itens não nulos da lista `list`.

List.Numbers

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Numbers(start as number, count as number, optional increment as nullable number) as list
```

Sobre

Retorna uma lista de números de um valor inicial, uma contagem e um valor de incremento opcional. O valor de incremento padrão é 1.

- `start`: O valor inicial na lista.
- `count`: O número de valores a serem criados.
- `increment`: *[opcional]* o valor a ser incrementado. Se os valores omitidos forem incrementados em 1.

Exemplo 1

Gere uma lista de 10 números consecutivos começando por 1.

```
List.Numbers(1, 10)
```

1
2
3
4
5
6
7
8
9
10

Exemplo 2

Gere uma lista de 10 números começando por 1, com um incremento de 2 para cada número subsequente.

```
List.Numbers(1, 10, 2)
```

1
3
5
7
9
11
13
15
17
19

List.PositionOf

09/05/2020 • 2 minutes to read

Sintaxe

```
List.PositionOf(list as list, value as any, optional occurrence as nullable number, optional
equationCriteria as any) as any
```

Sobre

Retorna o deslocamento no qual o valor `value` aparece na lista `list`. Retornará -1 se o valor não for exibido. Um parâmetro de ocorrência opcional `occurrence` pode ser especificado.

- `occurrence`: O número máximo de ocorrências a serem relatadas.

Exemplo 1

Localize a posição na lista {1, 2, 3} em que o valor 3 aparece.

```
List.PositionOf({1, 2, 3}, 3)
```

`2`

List.PositionOfAny

09/05/2020 • 2 minutes to read

Sintaxe

```
List.PositionOfAny(list as list, values as list, optional occurrence as nullable number, optional equationCriteria as any) as any
```

Sobre

Retorna o deslocamento na lista `list` da primeira ocorrência de um valor em uma lista `values`. Retornará -1 se nenhuma ocorrência for encontrada. Um parâmetro de ocorrência opcional `occurrence` pode ser especificado.

- `occurrence`: O número máximo de ocorrências que podem ser retornadas.

Exemplo 1

Localize a primeira posição na lista {1, 2, 3} em que o valor 2 ou 3 aparece.

```
List.PositionOfAny({1, 2, 3}, {2, 3})
```

`1`

List.Positions

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Positions(list as list) as list
```

Sobre

Retorna uma lista de deslocamentos para a lista de entrada `list`. Ao usar List.Transform para alterar uma lista, a lista de posições pode ser usada para conceder acesso de transformação à posição.

Exemplo 1

Localize os deslocamentos de valores na lista {1, 2, 3, 4, null, 5}.

```
List.Positions({1, 2, 3, 4, null, 5})
```

0
1
2
3
4
5

List.Product

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Product(numbersList as list, optional precision as nullable number) as nullable number
```

Sobre

Retorna o produto dos números não nulos na lista, `numbersList`. Retornará nulo se não houver valores não nulos na lista.

Exemplo 1

Localize o produto dos números na lista `{1, 2, 3, 3, 4, 5, 5}`.

```
List.Product({1, 2, 3, 3, 4, 5, 5})
```

```
1800
```

List.Random

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Random(count as number, optional seed as nullable number) as list
```

Sobre

Retorna uma lista de números aleatórios entre 0 e 1, considerando o número de valores a serem gerados e um valor de semente opcional.

- `count`: O número de valores aleatórios a serem gerados.
- `seed`: *[Opcional]* um valor numérico usado para propagar o gerador de números aleatórios. Se omitido, uma lista exclusiva de números aleatórios será gerada cada vez que você chamar a função. Se você especificar o valor de semente com um número, cada chamada à função gerará a mesma lista de números aleatórios.

Exemplo 1

Crie uma lista de três números aleatórios.

```
List.Random(3)
```

0,992332
0,132334
0,023592

Exemplo 2

Crie uma lista de três números aleatórios, especificando o valor de semente.

```
List.Random(3, 2)
```

0,883002
0,245344
0,723212

List.Range

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Range(list as list, offset as number, optional count as nullable number) as list
```

Sobre

Retorna um subconjunto da lista começando em um deslocamento `list`. Um parâmetro opcional, `offset`, define o número máximo de itens no subconjunto.

Exemplo 1

Localize o subconjunto começando no deslocamento 6 da lista de números de 1 a 10.

```
List.Range({1..10}, 6)
```

7
8
9
10

Exemplo 2

Localize o subconjunto de comprimento 2 no deslocamento 6, na lista de números de 1 a 10.

```
List.Range({1..10}, 6, 2)
```

7
8

List.RemoveFirstN

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveFirstN(list as list, optional countOrCondition as any) as list
```

Sobre

Retorna uma lista que remove o primeiro elemento da lista `list`. Se `list` for uma lista vazia, uma lista vazia será retornada. Essa função usa um parâmetro opcional, `countOrCondition`, para dar suporte à remoção de vários valores, conforme listado abaixo.

- Se um número for especificado, até que esse número de itens seja removido.
- Se uma condição for especificada, a lista retornada começará com o primeiro elemento em `list` que cumpre os critérios. Quando um item falha na condição, nenhum outro item é considerado.
- Se esse parâmetro for nulo, o comportamento padrão será observado.

Exemplo 1

Crie uma lista com base em {1, 2, 3, 4, 5} sem os três primeiros números.

```
List.RemoveFirstN({1, 2, 3, 4, 5}, 3)
```

4
5

Exemplo 2

Crie uma lista com base em {5, 4, 2, 6, 1} que comece com um número menor que 3.

```
List.RemoveFirstN({5, 4, 2, 6, 1}, each _ > 3)
```

2
6
1

List.RemoveItems

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveItems(list1 as list, list2 as list) as list
```

Sobre

Remove todas as ocorrências dos valores especificados em `list2` de `list1`. Se os valores em `list2` não existirem em `list1`, a lista original será retornada.

Exemplo 1

Remova os itens na lista {2, 4, 6} da lista {1, 2, 3, 4, 2, 5, 5}.

```
List.RemoveItems({1, 2, 3, 4, 2, 5, 5}, {2, 4, 6})
```

1
3
5
5

List.RemoveLastN

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveLastN(list as list, optional countOrCondition as any) as list
```

Sobre

Retorna uma lista que remove os últimos elementos de `countOrCondition` do final da lista `list`. Se `list` tiver menos de `countOrCondition` elementos, uma lista vazia será retornada.

- Se um número for especificado, até que esse número de itens seja removido.
- Se uma condição for especificada, a lista retornada terminará com o primeiro elemento da parte inferior em `list` que cumpre os critérios. Quando um item falha na condição, nenhum outro item é considerado.
- Se esse parâmetro for nulo, apenas um item será removido.

Exemplo 1

Crie uma lista com base em {1, 2, 3, 4, 5} sem os três últimos números.

```
List.RemoveLastN({1, 2, 3, 4, 5}, 3)
```

1
2

Exemplo 2

Crie uma lista com base em {5, 4, 2, 6, 4} que termine com um número menor que 3.

```
List.RemoveLastN({5, 4, 2, 6, 4}, each _ > 3)
```

5
4
2

List.RemoveMatchingItems

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveMatchingItems(list1 as list, list2 as list, optional equationCriteria as any) as list
```

Sobre

Remove todas as ocorrências dos valores especificados em `list2` da lista `list1`. Se os valores em `list2` não existirem em `list1`, a lista original será retornada. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Crie uma lista em {1, 2, 3, 4, 5, 5} sem {1, 5}.

```
List.RemoveMatchingItems({1, 2, 3, 4, 5, 5}, {1, 5})
```

2
3
4

List.RemoveNulls

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveNulls(list as list) as list
```

Sobre

Remove todas as ocorrências de valores "null" na `list`. Se não houver nenhum valor 'nulo' na lista, a lista original será retornada.

Exemplo 1

Remova os valores "null" na lista {1, 2, 3, null, 4, 5, null, 6}.

```
List.RemoveNulls({1, 2, 3, null, 4, 5, null, 6})
```

1
2
3
4
5
6

List.RemoveRange

09/05/2020 • 2 minutes to read

Sintaxe

```
List.RemoveRange(list as list, index as number, optional count as nullable number) as list
```

Sobre

Remove os valores `count` na `list` começando na posição especificada, `index`.

Exemplo 1

Remova três valores na lista {1, 2, 3, 4, -6, -2, -1, 5} começando no índice. 4.

```
List.RemoveRange({1, 2, 3, 4, -6, -2, -1, 5}, 4, 3)
```

1
2
3
4
5

List.Repeat

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Repeat(list as list, count as number) as list
```

Sobre

Retorna uma lista que representa as repetições de `count` da lista original, `list`.

Exemplo 1

Criar uma lista que tenha 3 repetições de {1, 2}.

```
List.Repeat({1, 2}, 3)
```

1
2
1
2
1
2

List.ReplaceMatchingItems

09/05/2020 • 2 minutes to read

Sintaxe

```
List.ReplaceMatchingItems(list as list, replacements as list, optional equationCriteria as any) as
list
```

Sobre

Executa as substituições fornecidas na lista `list`. Uma operação de substituição `replacements` consiste em uma lista de dois valores, o antigo e o novo, fornecidos em uma lista. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Crie uma lista em {1, 2, 3, 4, 5} substituindo o valor 5 por -5 e o valor 1 por -1.

```
List.ReplaceMatchingItems({1, 2, 3, 4, 5}, {{5, -5}, {1, -1}})
```

-1
2
3
4
-5

List.ReplaceRange

09/05/2020 • 2 minutes to read

Sintaxe

```
List.ReplaceRange(list as list, index as number, count as number, replaceWith as list) as list
```

Sobre

Substitui os valores `count` na `list` pela lista `replaceWith`, começando na posição especificada, `index`.

Exemplo 1

Substitua {7, 8, 9} na lista {1, 2, 7, 8, 9, 5} por {3, 4}.

```
List.ReplaceRange({1, 2, 7, 8, 9, 5}, 2, 3, {3, 4})
```

1
2
3
4
5

List.ReplaceValue

09/05/2020 • 2 minutes to read

Sintaxe

```
List.ReplaceValue(list as list, oldValue as any, newValue as any, replacer as function) as list
```

Sobre

Pesquisa em uma lista de valores, `list`, o valor `oldValue` e substitui cada ocorrência pelo valor de substituição `newValue`.

Exemplo 1

Substitua todos os valores "a" na lista {"a", "B", "a", "a"} por "A".

v List.ReplaceValue({"a", "B", "a", "a"}, "a", "A", Replacer.ReplaceText)

```
<table> <tr><td>A</td></tr> <tr><td>B</td></tr> <tr><td>A</td></tr> <tr><td>A</td></tr> </table>
```

List.Reverse

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Reverse(list as list) as list
```

Sobre

Retorna uma lista com os valores da lista `list` na ordem inversa.

Exemplo 1

Crie uma lista com base em {1..10} na ordem inversa.

```
List.Reverse({1..10})
```

10
9
8
7
6
5
4
3
2
1

List.Select

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Select(list as list, selection as function) as list
```

Sobre

Retorna uma lista de valores da lista `list`, que corresponde à condição de seleção `selection`.

Exemplo 1

Localize os valores da lista {1, -3, 4, 9, -2} que sejam maiores que 0.

```
List.Select({1, -3, 4, 9, -2}, each _ > 0)
```

1
4
9

List.Single

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Single(list as list) as any
```

Sobre

Se houver apenas um item na lista `list`, retornará esse item. Se houver mais de um item ou a lista estiver vazia, a função gerará uma exceção.

Exemplo 1

Localize o valor único na lista {1}.

```
List.Single({1})
```

`1`

Exemplo 2

Localize o valor único na lista {1, 2, 3}.

```
List.Single({1, 2, 3})
```

```
[Expression.Error] There were too many elements in the enumeration to complete the operation.
```

List.SingleOrDefault

09/05/2020 • 2 minutes to read

Sintaxe

```
List.SingleOrDefault(list as list, optional default as any) as any
```

Sobre

Se houver apenas um item na lista `list`, retornará esse item. Se a lista estiver vazia, a função retornará null, a menos que um `default` opcional seja especificado. Se houver mais de um item na lista, a função retornará um erro.

Exemplo 1

Localize o valor único na lista {1}.

```
List.SingleOrDefault({1})
```

`1`

Exemplo 2

Localize o valor único na lista {}.

```
List.SingleOrDefault({})
```

`null`

Exemplo 3

Localize o valor único na lista {}. Se estiver vazio, retornará -1.

```
List.SingleOrDefault({}, -1)
```

`-1`

List.Skip

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Skip(list as list, optional countOrCondition as any) as list
```

Sobre

Retorna uma lista que ignora o primeiro elemento da lista `list`. Se `list` for uma lista vazia, uma lista vazia será retornada. Essa função usa um parâmetro opcional, `countOrCondition`, para dar suporte a ignorar vários valores, conforme listado abaixo.

- Se um número for especificado, até que esse número de itens seja ignorado.
- Se uma condição for especificada, a lista retornada começará com o primeiro elemento em `list` que cumpre os critérios. Quando um item falha na condição, nenhum outro item é considerado.
- Se esse parâmetro for nulo, o comportamento padrão será observado.

Exemplo 1

Crie uma lista com base em {1, 2, 3, 4, 5} sem os três primeiros números.

```
List.Skip({1, 2, 3, 4, 5}, 3)
```

4
5

Exemplo 2

Crie uma lista com base em {5, 4, 2, 6, 1} que comece com um número menor que 3.

```
List.Skip({5, 4, 2, 6, 1}, each _ > 3)
```

2
6
1

List.Sort

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Sort(list as list, optional comparisonCriteria as any) as list
```

Sobre

Classifica uma lista de dados, `list`, de acordo com os critérios opcionais especificados. Um parâmetro opcional, `comparisonCriteria`, pode ser especificado como o critério de comparação. Ele pode assumir os seguintes valores:

- Para controlar a ordem, o critério de comparação pode ser um valor de enumeração de Ordem. (`Order.Descending`, `Order.Ascending`).
- Para calcular uma chave a ser usada para classificação, uma função de um argumento pode ser usada.
- Para selecionar uma ordem de chave e de controle, o critério de comparação pode ser uma lista que contém a chave e a ordem (`{each 1 / _, Order.Descending}`).
- Para controlar por completo a comparação, é possível usar uma função de dois argumentos que retorna -1, 0 ou 1, considerando a relação entre as entradas esquerda e direita. Value.Compare é um método que pode ser usado para delegar essa lógica.

Exemplo 1

Classifique a lista {2, 3, 1}.

```
List.Sort({2, 3, 1})
```

1
2
3

Exemplo 2

Classifique a lista {2, 3, 1} em ordem decrescente.

```
List.Sort({2, 3, 1}, Order.Descending)
```

3
2
1

Exemplo 3

Classifique a lista {2, 3, 1} em ordem decrescente usando o método Value.Compare.

```
List.Sort({2, 3, 1}, (x, y) => Value.Compare(1/x, 1/y))
```

3
2
1

List.Split

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Split(list as list, pageSize as number) as list
```

Sobre

Divide o `list` em uma lista de listas em que o primeiro elemento da lista de saída é uma lista que contém os primeiros `pageSize` elementos da lista de origem, o próximo elemento da lista de saída é uma lista que contém os próximos `pageSize` elementos da lista de origem etc.

List.StandardDeviation

09/05/2020 • 2 minutes to read

Sintaxe

```
List.StandardDeviation(numbersList as list) as nullable number
```

Sobre

Retorna uma estimativa baseada em amostra do desvio padrão dos valores na lista, `numbersList`. Se `numbersList` for uma lista de números, um número será retornado. Uma exceção será gerada em uma lista vazia ou em uma lista de itens que não são do tipo `number`.

Exemplo 1

Localize o desvio padrão dos números de 1 a 5.

```
List.StandardDeviation({1..5})
```

```
1.5811388300841898
```

List.Sum

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Sum(list as list, optional precision as nullable number) as any
```

Sobre

Retorna a soma dos valores não nulos na lista `list`. Retornará nulo se não houver valores não nulos na lista.

Exemplo 1

Localize a soma dos números na lista `{1, 2, 3}`.

```
List.Sum({1, 2, 3})
```

`6`

List.Times

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Times(start as time, count as number, step as duration) as list
```

Sobre

Retorna uma lista de valores `time` do tamanho `count`, começando em `start`. O incremento especificado, `step`, é um valor `duration` que é adicionado a cada valor.

Exemplo 1

Crie uma lista de quatro valores começando ao meio-dia (#time(12, 0, 0)) e cujos valores sejam incrementados em uma hora (#duration(0, 1, 0, 0)).

```
List.Times(#time(12, 0, 0), 4, #duration(0, 1, 0, 0))
```

12:00:00
13:00:00
14:00:00
15:00:00

List.Transform

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Transform(list as list, transform as function) as list
```

Sobre

Retorna uma nova lista de valores aplicando a função transform `transform` à lista `list`.

Exemplo 1

Adicione 1 a cada valor na lista {1, 2}.

```
List.Transform({1, 2}, each _ + 1)
```

2
3

List.TransformMany

09/05/2020 • 2 minutes to read

Sintaxe

```
List.TransformMany(list as list, collectionTransform as function, resultTransform as function) as
list
```

Sobre

Retorna uma lista cujos elementos são projetados com base na lista de entrada. A função `collectionTransform` é aplicada a cada elemento e a função `resultTransform` é invocada para criar a lista resultante. O `collectionSelector` tem a assinatura (x como Qualquer) = >... em que x é um elemento na lista. O `resultTransform` projeta a forma do resultado e tem a assinatura (x como Qualquer, y como Qualquer) = >... em que x é o elemento na lista e y é o elemento obtido aplicando o `collectionTransform` a esse elemento.

List.Union

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Union(lists as list, optional equationCriteria as any) as list
```

Sobre

Usa uma lista de listas `lists`, une os itens nas listas individuais e os retorna na lista de saída. Como resultado, a lista retornada contém todos os itens em listas de entrada. Essa operação mantém a semântica da bolsa tradicional, portanto, valores duplicados são combinados como parte da União. Um valor de critérios de equação opcional, `equationCriteria`, pode ser especificado para controlar o teste de igualdade.

Exemplo 1

Crie uma união da lista {1..5}, {2..6}, {3..7}.

```
List.Union({{1..5}, {2..6}, {3..7}})
```

1
2
3
4
5
6
7

List.Zip

09/05/2020 • 2 minutes to read

Sintaxe

```
List.Zip(lists as list) as list
```

Sobre

Utiliza uma lista de listas, `lists`, e retorna uma lista de listas, combinando itens na mesma posição.

Exemplo 1

Compacta as duas listas simples {1, 2} e {3, 4}.

```
List.Zip({{1, 2}, {3, 4}})
```

[Lista]
[Lista]

Exemplo 2

Compacta as duas listas simples de comprimentos diferentes {1, 2} e {3}.

```
List.Zip({{1, 2}, {3}})
```

[Lista]
[Lista]

Funções lógicas

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam valores lógicos (ou seja, true/false).

Lógico

FUNÇÃO	DESCRIÇÃO
Logical.From	Retorna um valor lógico de um valor.
Logical.FromText	Retorna um valor lógico de true ou false de um valor de texto.
Logical.ToText	Retorna um valor de texto de um valor lógico.

Logical.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Logical.From(value as any) as nullable logical
```

Sobre

Retorna um valor `logical` do `value` especificado. Se o `value` fornecido for `null`, `Logical.From` retornará `null`. Se o `value` fornecido for `logical`, `value` será retornado.

Os valores dos seguintes tipos podem ser convertidos em um valor `logical`:

- `text`: Um valor de `logical` do valor de texto, seja `"true"` ou `"false"`. Confira `Logical.FromText` para obter detalhes.
- `number`: `false` se `value` é igual a `0`, `true` caso contrário.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `2` em um valor `logical`.

```
Logical.From(2)
```

`true`

Logical.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Logical.FromText(text as nullable text) as nullable logical
```

Sobre

Cria um valor lógico com base no valor de texto `text`, "true" ou "false". Se `text` contiver uma cadeia de caracteres diferente, uma exceção será gerada. O valor de texto `text` não diferencia maiúsculas de minúsculas.

Exemplo 1

Crie um valor lógico da cadeia de caracteres de texto "true".

```
Logical.FromText("true")
```

`true`

Exemplo 2

Crie um valor lógico da cadeia de caracteres de texto "a".

```
Logical.FromText("a")
```

`[Expression.Error] Could not convert to a logical.`

Logical.ToText

09/05/2020 • 2 minutes to read

Sintaxe

```
Logical.ToText(logicalValue as nullable logical) as nullable text
```

Sobre

Cria um valor de texto com base no valor lógico `logicalValue`, `true` ou `false`. Se `logicalValue` não for um valor lógico, uma exceção será gerada.

Exemplo 1

Crie um valor de texto com base no `true` lógico.

```
Logical.ToText(true)
```

`"true"`

Funções numéricas

08/05/2020 • 6 minutes to read

Essas funções criam e manipulam valores numéricos.

Número

Constantes

FUNÇÃO	DESCRIÇÃO
Number.E	Retorna 2,7182818284590451, o valor de e até 16 dígitos decimais.
Number.Epsilon	Retorna o menor número possível.
Number.NaN	Representa 0/0.
Number.NegativeInfinity	Representa -1/0.
Number.PI	Retorna 3,1415926535897931, o valor de Pi de até 16 dígitos decimais.
Number.PositiveInfinity	Representa 1/0.

Informações

FUNÇÃO	DESCRIÇÃO
Number.IsEven	Retorna true se um valor é um número par.
Number.IsNaN	Retorna true se um valor é Number.NaN.
Number.IsOdd	Retorna true se um valor é um número ímpar.

Conversão e formatação

FUNÇÃO	DESCRIÇÃO
Byte.From	Retorna um valor numérico de inteiro de 8 bits do valor especificado.
Currency.From	Retorna um valor de moeda de um determinado valor.
Decimal.From	Retorna um valor numérico decimal do valor especificado.
Double.From	Retorna um valor de número duplo do valor especificado.
Int8.From	Retorna um valor numérico de inteiro de 8 bits com sinal do valor especificado.

FUNÇÃO	DESCRIÇÃO
Int16.From	Retorna um valor numérico de inteiro de 16 bits do valor especificado.
Int32.From	Retorna um valor numérico de inteiro de 32 bits do valor especificado.
Int64.From	Retorna um valor numérico de inteiro de 64 bits do valor especificado.
Number.From	Retorna um valor numérico de um valor.
Number.FromText	Retorna um valor numérico de um valor de texto.
Number.ToText	Retorna um valor de texto de um valor numérico.
Percentage.From	Retorna um valor de percentual de um determinado valor.
Single.From	Retorna um valor numérico único do valor especificado.

Arredondamento

FUNÇÃO	DESCRIÇÃO
Number.Round	Retorna um número anulável (n) se o valor é um inteiro.
Number.RoundAwayFromZero	Retorna Number.RoundUp(value) quando valor >= 0 e Number.RoundDown(value) quando valor < 0.
Number.RoundDown	Retorna o maior número inteiro inferior ou igual a um valor numérico.
Number.RoundTowardZero	Retorna Number.RoundDown(x) quando x >= 0 e Number.RoundUp(x) quando x < 0.
Number.RoundUp	Retorna o número inteiro maior ou igual a um valor numérico.

Operações

FUNÇÃO	DESCRIÇÃO
Number.Abs	Retorna o valor absoluto de um número.
Number.Combinations	Retorna o número de combinações de um determinado número de itens para o tamanho de combinação opcional.
Number.Exp	Retorna um número que representa *e* elevado a uma potência.
Number.Factorial	Retorna o fatorial de um número.
Number.IntegerDivide	Divide dois números e retorna a parte inteira do número resultante.

FUNÇÃO	DESCRIÇÃO
Number.Ln	Retorna o logaritmo natural de um número.
Number.Log	Retorna o logaritmo de um número para a base.
Number.Log10	Retorna o logaritmo de base 10 de um número.
Number.Mod	Divide dois números e retorna o restante do número resultante.
Number.Permutations	Retorna o número total de permutações de um determinado número de itens para o tamanho de permuta opcional.
Number.Power	Retorna um número elevado por uma potência.
Number.Sign	Retorna 1 para números positivos,-1 para números negativos ou 0 para zero.
Number.Sqrt	Retorna a raiz quadrada de um número.

Random

FUNÇÃO	DESCRIÇÃO
Number.Random	Retorna um número fracionário aleatório entre 0 e 1.
Number.RandomBetween	Retorna um número aleatório entre os dois valores numéricos fornecidos.

Trigonometria

FUNÇÃO	DESCRIÇÃO
Number.Acos	Retorna o arco cosseno de um número.
Number.Asin	Retorna o arco seno de um número.
Number.Atan	Retorna o arco tangente de um número.
Number.Atan2	Retorna o arco tangente da divisão de dois números.
Number.Cos	Retorna o cosseno de um número.
Number.Cosh	Retorna o cosseno hiperbólico de um número.
Number.Sin	Retorna o seno de um número.
Number.Sinh	Retorna o seno hiperbólico de um número.
Number.Tan	Retorna a tangente de um número.
Number.Tanh	Retorna a tangente hiperbólica de um número.

Bytes

FUNÇÃO	DESCRIÇÃO
Number.BitwiseAnd	Retorna o resultado de uma operação AND bit a bit nos operandos fornecidos.
Number.BitwiseNot	Retorna o resultado de uma operação NOT bit a bit nos operandos fornecidos.
Number.BitwiseOr	Retorna o resultado de uma operação OR bit a bit nos operandos fornecidos.
Number.BitwiseShiftLeft	Retorna o resultado de uma operação de deslocamento à esquerda bit a bit nos operandos.
Number.BitwiseShiftRight	Retorna o resultado de uma operação de deslocamento à direita bit a bit nos operandos.
Number.BitwiseXor	Retorna o resultado de uma operação XOR bit a bit nos operandos fornecidos.

VALORES DE PARÂMETROS	DESCRIÇÃO
RoundingMode.AwayFromZero	RoundingMode.AwayFromZero
RoundingMode.Down	RoundingMode.Down
RoundingMode.ToEven	RoundingMode.ToEven
RoundingMode.TowardZero	RoundingMode.TowardZero
RoundingMode.Up	RoundingMode.Up

Byte.From

08/05/2020 • 2 minutes to read

Sintaxe

```
Byte.From(value as any, optional culture as nullable text, optional roundingMode as nullable
number) as nullable number
```

Sobre

Retorna um valor `number` > inteiro de 8 bits do `value` especificado. Se o `value` > fornecido for `null`, `Byte.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo do inteiro de 8 bits sem uma parte fracionária, `value` será retornado. Se ele tiver uma parte fracionária, o número será arredondado com o modo de arredondamento especificado. O modo de arredondamento padrão é `RoundingMode.ToEven`. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` inteiro de 8 bits é aplicável. Confira `Number.Round` para ver os modos de arredondamento disponíveis. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` inteiro de 8 bits de `"4"`.

```
Byte.From("4")
```

`4`

Exemplo 2

Obter o valor `number` inteiro de 8 bits de `"4.5"` usando `RoundingMode.AwayFromZero`.

```
Byte.From("4.5", null, RoundingMode.AwayFromZero)
```

`5`

=

Currency.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Currency.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number
```

Sobre

Retorna um valor `currency` do `value` especificado. Se o `value` fornecido for `null`, `Currency.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo de moeda, a parte fracionária do `value` será arredondada para quatro dígitos decimais e retornada. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `currency` é aplicável. O intervalo válido para a moeda é `-922,337,203,685,477.5808` para `922,337,203,685,477.5807`. Confira `Number.Round` para obter os modos de arredondamento disponíveis. O padrão é `RoundingMode.ToEven`. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha o valor `currency` de `"1.23455"`.

```
Currency.From("1.23455")
```

`1.2346`

Exemplo 2

Obtenha o valor `currency` de `"1.23455"` usando `RoundingMode.Down`.

```
Currency.From("1.23455", "en-Us", RoundingMode.Down)
```

`1.2345`

Decimal.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Decimal.From(value as any, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `number` Decimal do `value` especificado. Se o `value` fornecido for `null`, `Decimal.From` retornará `null`. Se o `value` fornecido for `number` dentro do intervalo de Decimal, `value` será retornado; caso contrário, um erro será retornado. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` Decimal é aplicável. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor de `number` Decimal de `"4.5"`.

```
Decimal.From("4.5")
```

`4.5`

Double.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Double.From(value as any, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `number` Double do `value` especificado. Se o `value` fornecido for `null`, `Double.From` retornará `null`. Se o `value` fornecido for `number` dentro do intervalo de Double, `value` será retornado; caso contrário, um erro será retornado. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` Double é aplicável. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` Double de `"4"`.

```
Double.From("4.5")
```

`4.5`

Int8.From

08/05/2020 • 2 minutes to read

Sintaxe

```
Int8.From(value as any, optional culture as nullable text, optional roundingMode as nullable
number) as nullable number
```

Sobre

Retorna um valor `number` inteiro com sinal de 8 bits do `value` especificado. Se o `value` fornecido for `null`, `Int8.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo do inteiro com sinal de 8 bits sem uma parte fracionária, `value` será retornado. Se ele tiver uma parte fracionária, o número será arredondado com o modo de arredondamento especificado. O modo de arredondamento padrão é `RoundingMode.ToEven`. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` inteiro com sinal de 8 bits é aplicável. Confira `Number.Round` para obter os modos de arredondamento disponíveis. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` inteiro com sinal de 8 bits de `"4"`.

```
Int8.From("4")
```

`4`

Exemplo 2

Obter o valor `number` inteiro com sinal de 8 bits de `"4.5"` usando `RoundingMode.AwayFromZero`.

```
Int8.From("4.5", null, RoundingMode.AwayFromZero)
```

`5`

Int16.From

08/05/2020 • 2 minutes to read

Sintaxe

```
Int16.From(value as any, optional culture as nullable text, optional roundingMode as nullable
number) as nullable number
```

Sobre

Retorna um valor `number` inteiro de 16 bits do `value` especificado. Se o `value` fornecido for `null`, `Int16.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo do inteiro de 16 bits sem uma parte fracionária, `value` será retornado. Se ele tiver uma parte fracionária, o número será arredondado com o modo de arredondamento especificado. O modo de arredondamento padrão é `RoundingMode.ToEven`. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` inteiro de 16 bits é aplicável. Confira `Number.Round` para obter os modos de arredondamento disponíveis. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` inteiro de 16 bits de `"4"`.

```
Int64.From("4")
```

`4`

Exemplo 2

Obter o valor `number` inteiro de 16 bits de `"4.5"` usando `RoundingMode.AwayFromZero`.

```
Int16.From("4.5", null, RoundingMode.AwayFromZero)
```

`5`

Int32.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Int32.From(value as any, optional culture as nullable text, optional roundingMode as nullable
number) as nullable number
```

Sobre

Retorna um valor `number` inteiro de 32 bits do `value` especificado. Se o `value` fornecido for `null`, `Int32.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo do inteiro de 32 bits sem uma parte fracionária, `value` será retornado. Se ele tiver uma parte fracionária, o número será arredondado com o modo de arredondamento especificado. O modo de arredondamento padrão é `RoundingMode.ToEven`. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` inteiro de 32 bits é aplicável. Confira `Number.Round` para obter os modos de arredondamento disponíveis. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` inteiro de 32 bits de `"4"`.

```
Int32.From("4")
```

`4`

Exemplo 2

Obter o valor `number` inteiro de 32 bits de `"4.5"` usando `RoundingMode.AwayFromZero`.

```
Int32.From("4.5", null, RoundingMode.AwayFromZero)
```

`5`

Int64.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Int64.From(value as any, optional culture as nullable text, optional roundingMode as nullable
number) as nullable number
```

Sobre

Retorna um valor `number` inteiro de 64 bits do `value` especificado. Se o `value` fornecido for `null`, `Int64.From` retornará `null`. Se o `value` especificado for `number` dentro do intervalo do inteiro de 64 bits sem uma parte fracionária, `value` será retornado. Se ele tiver uma parte fracionária, o número será arredondado com o modo de arredondamento especificado. O modo de arredondamento padrão é `RoundingMode.ToEven`. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` inteiro de 64 bits é aplicável. Confira `Number.Round` para obter os modos de arredondamento disponíveis. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` inteiro de 64 bits de `"4"`.

```
Int64.From("4")
```

`4`

Exemplo 2

Obter o valor `number` inteiro de 64 bits de `"4.5"` usando `RoundingMode.AwayFromZero`.

```
Int64.From("4.5", null, RoundingMode.AwayFromZero)
```

`5`

Number.Abs

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Abs(number as nullable number) as nullable number
```

Sobre

Retorna o valor absoluto de `number`. Se `number` for nulo, `Number.Abs` retornará nulo.

- `number`: Um `number` para o qual o valor absoluto deve ser calculado.

Exemplo 1

Valor absoluto de -3.

```
Number.Abs(-3)
```

`3`

Number.Acos

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Acos(number as nullable number) as nullable number
```

Sobre

Retorna o arco cosseno de `number`.

Number.Asin

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Asin(number as nullable number) as nullable number
```

Sobre

Retorna o arco seno de `number`.

Number.Atan

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Atan(number as nullable number) as nullable number
```

Sobre

Retorna o arco tangente de `number`.

Number.Atan2

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Atan2(y as nullable number, x as nullable number) as nullable number
```

Sobre

Retorna o arco tangente da divisão dos dois números. `y` e `x`. A divisão será interpretada como `y` / `x`.

Number.BitwiseAnd

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseAnd(number1 as nullable number, number2 as nullable number) as nullable number
```

Sobre

Retorna o resultado da execução de uma operação bit a bit "And" entre `number1` e `number2`.

Number.BitwiseNot

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseNot(number as any) as any
```

Sobre

Retorna o resultado da execução de uma operação bit a bit "Not" em `number`.

Number.BitwiseOr

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseOr(number1 as nullable number, number2 as nullable number) as nullable number
```

Sobre

Retorna o resultado da execução de uma operação bit a bit "Or" entre `number1` e `number2`.

Number.BitwiseShiftLeft

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseShiftLeft(number1 as nullable number, number2 as nullable number) as nullable number
```

Sobre

Retorna o resultado da execução de um deslocamento bit a bit para a esquerda em `number1`, com base no número especificado de bits `number2`.

Number.BitwiseShiftRight

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseShiftRight(number1 as nullable number, number2 as nullable number) as nullable number
```

Sobre

Retorna o resultado da execução de um deslocamento bit a bit para a direita em `number1`, com base no número especificado de bits `number2`.

Number.BitwiseXor

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.BitwiseXor(number1 as nullable number, number2 as nullable number) as nullable number
```

Sobre

Retorna o resultado da execução de uma operação bit a bit "XOR" (Exclusive-OR) entre `number1` e `number2`.

Number.Combinations

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Combinations(setSize as nullable number, combinationSize as nullable number) as nullable number
```

Sobre

Retorna o número de combinações exclusivas de uma lista de itens, `setSize`, com o tamanho de combinação especificado, `combinationSize`.

- `setSize`: O número de itens na lista.
- `combinationSize`: O número de itens em cada combinação.

Exemplo 1

Localize o número de combinações de um total de cinco itens quando cada combinação for um grupo de 3.

```
Number.Combinations(5, 3)
```

```
10
```

Number.Cos

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Cos(number as nullable number) as nullable number
```

Sobre

Retorna o cosseno de `number`.

Exemplo 1

Localize o cosseno do ângulo 0.

```
Number.Cos(0)
```

`1`

Number.Cosh

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Cosh(number as nullable number) as nullable number
```

Sobre

Retorna o cosseno hiperbólico de `number`.

Number.E

09/05/2020 • 2 minutes to read

Sobre

Uma constante que representa 2,7182818284590451, o valor de e até 16 dígitos decimais.

Number.Epsilon

09/05/2020 • 2 minutes to read

Sobre

Um valor de constante que representa ao menor número positivo que um número de ponto flutuante pode conter.

Number.Exp

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Exp(number as nullable number) as nullable number
```

Sobre

Retorna o resultado da elevação de e à potência de `number` (função exponencial).

- `number`: Um `number` para o qual a função exponencial deve ser calculada. Se `number` for nulo, `Number.Exp` retornará nulo.

Exemplo 1

Eleva e à potência 3.

```
Number.Exp(3)
```

```
20.085536923187668
```

Number.Factorial

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Factorial(number as nullable number) as nullable number
```

Sobre

Retorna o fatorial do número `number`.

Exemplo 1

Localize o fatorial 10.

```
Number.Factorial(10)
```

```
3628800
```

Number.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.From(value as any, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `number` do `value` especificado. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR"). Se o `value` fornecido for `null`, `Number.From` retornará `null`. Se o `value` fornecido for `number`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `number`:

- `text`: Um valor `number` da representação textual. Os formatos de texto comuns são tratados ("15", "3.423,10", "5.0E-10"). Confira `Number.FromText` para obter detalhes.
- `logical`: 1 para `true`, 0 para `false`.
- `datetime`: Um número de ponto flutuante de precisão dupla que contém uma data de Automação OLE equivalente.
- `datetimezone`: Um número de ponto flutuante de precisão dupla que contém uma data de Automação OLE equivalente da data e hora local de `value`.
- `date`: Um número de ponto flutuante de precisão dupla que contém uma data de Automação OLE equivalente.
- `time`: Expresso em dias fracionários.
- `duration`: Expresso em dias inteiros e fracionários.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Obtenha o valor `number` de `"4"`.

```
powerquery-mNumber.From("4")
```

`4`

Exemplo 2

Obtenha o valor `number` de `#datetime(2020, 3, 20, 6, 0, 0)`.

```
Number.From(#datetime(2020, 3, 20, 6, 0, 0))
```

`43910.25`

Exemplo 3

Obtenha o valor `number` de `"12.3%"`.

```
Number.From("12.3%")
```

`0.123`

Number.FromText

26/05/2020 • 2 minutes to read

Sintaxe

```
Number.FromText(text as nullable text, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `number` do valor de texto especificado, `text`.

- `text`: A representação textual de um valor numérico. A representação deve estar em um formato de número comum, como "15", "3423,10" ou "5.0E-10".
- `culture`: Uma cultura opcional que controla como `text` é interpretada (por exemplo, "pt-BR").

Exemplo 1

Obter o valor numérico de `"4"`.

```
Number.FromText("4")
```

`4`

Exemplo 2

Obter o valor numérico de `"5.0e-10"`.

```
Number.FromText("5.0e-10")
```

`5E-10`

Number.IntegerDivide

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.IntegerDivide(number1 as nullable number, number2 as nullable number, optional precision as nullable number) as nullable number
```

Sobre

Retorna a parte inteira do resultado da divisão de um número, `number1`, por outro número, `number2`. Se `number1` ou `number2` for nulo, `Number.IntegerDivide` retornará nulo.

- `number1`: O dividendo.
- `number2`: O divisor.

Exemplo 1

Divide 6 por 4.

```
Number.IntegerDivide(6, 4)
```

`1`

Exemplo 2

Divide 8,3 por 3.

```
Number.IntegerDivide(8.3, 3)
```

`2`

Number.IsEven

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.IsEven(number as number) as logical
```

Sobre

Indica se o valor, `number`, é par retornando `true` se for par, caso contrário, `false`.

Exemplo 1

Verifique se 625 é um número par.

```
Number.IsEven(625)
```

`false`

Exemplo 2

Verifique se 82 é um número par.

```
Number.IsEven(82)
```

`true`

Number.IsNaN

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.IsNaN(number as number) as logical
```

Sobre

Indica se o valor é NaN (e não um número). Retorna `true` se `number` for equivalente a `Number.IsNaN`, `false` caso contrário.

Exemplo 1

Verifique se 0 dividido por 0 é NaN.

```
Number.IsNaN(0/0)
```

`true`

Exemplo 2

Verifique se 1 dividido por 0 é NaN.

```
Number.IsNaN(1/0)
```

`false`

Number.IsOdd

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.IsOdd(number as number) as logical
```

Sobre

Indica se o valor é ímpar. Retorna `true` se `number` for um número ímpar, `false` caso contrário.

Exemplo 1

Verifique se 625 é um número ímpar.

```
Number.IsOdd(625)
```

`true`

Exemplo 2

Verifique se 82 é um número ímpar.

```
Number.IsOdd(82)
```

`false`

Number.Ln

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Ln(number as nullable number) as nullable number
```

Sobre

Retorna o logaritmo natural de um número, `number`. Se `number` for nulo, `Number.Ln` retornará nulo.

####Exemplo 1 Obtenha o logaritmo natural 15.

```
Number.Ln(15)
```

```
2.70805020110221
```

Number.Log

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Log(number as nullable number, optional base as nullable number) as nullable number
```

Sobre

Retorna o logaritmo de um número, `number`, para a base `base` especificada. Se `base` não for especificado, o valor padrão será Number.E. Se `number` for nulo, `Number.Log` retornará nulo.

Exemplo 1

Obtém o logaritmo de base 10 de 2.

```
Number.Log(2, 10)
```

```
0.3010299956639812
```

Exemplo 2

Obtém o logaritmo de base e de 2.

```
Number.Log(2)
```

```
0.69314718055994529
```

Number.Log10

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Log10(number as nullable number) as nullable number
```

Sobre

Retorna o logaritmo de base 10 de um número, `number`. Se `number` for nulo, `Number.Log10` retornará nulo.

Exemplo 1

Obtém o logaritmo de base 10 de 2.

```
Number.Log10(2)
```

```
0.3010299956639812
```

Number.Mod

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Mod(number as nullable number, divisor as nullable number, optional precision as nullable number) as nullable number
```

Sobre

Retorna o resto resultante da divisão de número inteiro de `number` por `divisor`. Se `number` ou `divisor` for nulo, `Number.Mod` retornará nulo.

- `number`: O dividendo.
- `divisor`: O divisor.

Exemplo 1

Localize o resto ao dividir 5 por 3.

```
Number.Mod(5, 3)
```

`2`

Number.NaN

09/05/2020 • 2 minutes to read

Sobre

Um valor de constante que representa 0 dividido por 0.

Number.NegativeInfinity

09/05/2020 • 2 minutes to read

Sobre

Um valor de constante que representa -1 dividido por 0.

Number.Permutations

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Permutations(setSize as nullable number, permutationSize as nullable number) as nullable number
```

Sobre

Retorna o número de permutações que podem ser geradas com base em um número de itens, `setSize`, com um tamanho de permutação especificado, `permutationSize`.

Exemplo 1

Localize o número de permutações em um total de cinco itens em grupos de três.

```
Number.Permutations(5, 3)
```

`60`

Number.PI

09/05/2020 • 2 minutes to read

Sobre

Uma constante que representa 3,1415926535897932, o valor de pi até 16 dígitos decimais.

Number.PositiveInfinity

09/05/2020 • 2 minutes to read

Sobre

Um valor de constante que representa 1 dividido por 0.

Number.Power

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Power(number as nullable number, power as nullable number) as nullable number
```

Sobre

Retorna o resultado da geração de `number` à potência de `power`. Se `number` ou `power` for nulo, `Number.Power` retornará nulo.

- `number`: A base.
- `power`: O expoente.

Exemplo 1

Localize o valor 5 elevado à potência de 3 (5 ao cubo).

```
Number.Power(5, 3)
```

```
125
```

Number.Random

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Random() as number
```

Sobre

Retorna um número aleatório entre 0 e 1.

Exemplo 1

Obtenha um número aleatório.

```
Number.Random()
```

```
0.919303
```

Number.RandomBetween

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.RandomBetween(bottom as number, top as number) as number
```

Sobre

Retorna um número aleatório entre `bottom` e `top`.

Exemplo 1

Obtenha um número aleatório entre 1 e 5.

```
Number.RandomBetween(1, 5)
```

`2.546797`

Number.Round

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Round(number as nullable number, optional digits as nullable number, optional roundingMode
as nullable number) as nullable number
```

Sobre

Retorna o resultado do arredondamento de `number` para o número mais próximo. Se `number` for nulo, `Number.Round` retornará nulo. `number` é arredondado para o número inteiro mais próximo, a menos que o parâmetro opcional `digits` seja especificado. Se `digits` for especificado, `number` será arredondado para o `digits` número de dígitos decimais. Um parâmetro opcional `roundingMode` especifica a direção do arredondamento quando há um vínculo entre os números possíveis de arredondar (confira `RoundingMode.Type` para obter os valores possíveis).

Exemplo 1

Arredonde 1,234 para o próximo número inteiro.

```
Number.Round(1.234)
```

`1`

Exemplo 2

Arredonde 1,56 para o próximo número inteiro.

```
Number.Round(1.56)
```

`2`

Exemplo 3

Arredonde 1,2345 para duas casas decimais.

```
Number.Round(1.2345, 2)
```

`1.23`

Exemplo 4

Arredonde 1,2345 para três casas decimais (arredondamento para cima).

```
Number.Round(1.2345, 3, RoundingMode.Up)
```

```
1.235
```

Exemplo 5

Arredonde 1,2345 para três casas decimais (arredondamento para baixo).

```
Number.Round(1.2345, 3, RoundingMode.Down)
```

```
1.234
```

Number.RoundAwayFromZero

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.RoundAwayFromZero(number as nullable number, optional digits as nullable number) as
nullable number
```

Sobre

Retornará o resultado de arredondar `number` com base no sinal do número. Essa função arredondará números positivos para cima e números negativos para baixo. Se `digits` for especificado, `number` será arredondado para o `digits` número de dígitos decimais.

Exemplo 1

Arredonde o número -1,2 em direção oposta a zero.

```
Number.RoundAwayFromZero(-1.2)
```

`-2`

Exemplo 2

Arredonde o número 1,2 em direção oposta a zero.

```
Number.RoundAwayFromZero(1.2)
```

`2`

Exemplo 3

Arredonda o número -1,234 em duas casas decimais após o zero.

```
Number.RoundAwayFromZero(-1.234, 2)
```

`-1.24`

Number.RoundDown

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.RoundDown(number as nullable number, optional digits as nullable number) as nullable number
```

Sobre

Retorna o resultado do arredondamento `number` até o número inteiro mais alto anterior. Se `number` for nulo, `Number.RoundDown` retornará nulo. Se `digits` for especificado, `number` será arredondado para o `digits` número de dígitos decimais.

Exemplo 1

Arredondar 1,234 para baixo, de modo que se torne um número inteiro.

```
Number.RoundDown(1.234)
```

`1`

Exemplo 2

Arredondar 1,999 para baixo, de modo que se torne um número inteiro.

```
Number.RoundDown(1.999)
```

`1`

Exemplo 3

Arredondar 1,999 para duas casas decimais.

```
Number.RoundDown(1.999, 2)
```

`1.99`

Number.RoundTowardZero

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.RoundTowardZero(number as nullable number, optional digits as nullable number) as nullable number
```

Sobre

Retornará o resultado de arredondar `number` com base no sinal do número. Essa função arredondará números positivos para baixo e números negativos para cima. Se `digits` for especificado, `number` será arredondado para o `digits` número de dígitos decimais.

Number.RoundUp

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.RoundUp(number as nullable number, optional digits as nullable number) as nullable number
```

Sobre

Retorna o resultado do arredondamento `number` até o número inteiro mais alto anterior. Se `number` for nulo, `Number.RoundDown` retornará nulo. Se `digits` for especificado, `number` será arredondado para o `digits` número de dígitos decimais.

Exemplo 1

Arredondar 1,234 para cima de modo que se torne um número inteiro.

```
Number.RoundUp(1.234)
```

`2`

Exemplo 2

Arredondar 1,999 para cima de modo que se torne um número inteiro.

```
Number.RoundUp(1.999)
```

`2`

Exemplo 3

Arredondar 1,234 para duas casas decimais.

```
Number.RoundUp(1.234, 2)
```

`1.24`

Number.Sign

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Sign(number as nullable number) as nullable number
```

Sobre

Retornará 1 se `number` for um número positivo,-1 se for um número negativo e 0 se for zero. Se `number` for nulo, `Number.Sign` retornará nulo.

Exemplo 1

Determina o sinal de 182.

```
Number.Sign(182)
```

`1`

Exemplo 2

Determina o sinal de -182.

```
Number.Sign(-182)
```

`-1`

Exemplo 3

Determina o sinal de 0.

```
Number.Sign(0)
```

`0`

Number.Sin

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Sin(number as nullable number) as nullable number
```

Sobre

Retorna o seno de `number`.

Exemplo 1

Localize o seno do ângulo 0.

```
Number.Sin(0)
```

`0`

Number.Sinh

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Sinh(number as nullable number) as nullable number
```

Sobre

Retorna o seno hiperbólico de `number`.

Number.Sqrt

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Sqrt(number as nullable number) as nullable number
```

Sobre

Retorna a raiz quadrada de `number`. Se `number` for nulo, `Number.Sqrt` retornará nulo. Se for um valor negativo, `Number.NaN` será retornado (não é um número).

Exemplo 1

Localize a raiz quadrada de 625.

```
Number.Sqrt(625)
```

`25`

Exemplo 2

Localize a raiz quadrada de 85.

```
Number.Sqrt(85)
```

`9.2195444572928871`

Number.Tan

08/05/2020 • 2 minutes to read

Sintaxe

```
Number.Tan(number as nullable number) as nullable number
```

Sobre

Retorna a tangente de `number`.

Exemplo 1

Localize a tangente do ângulo 1.

```
Number.Tan(1)
```

```
1.5574077246549023
```

Number.Tanh

09/05/2020 • 2 minutes to read

Sintaxe

```
Number.Tanh(number as nullable number) as nullable number
```

Sobre

Retorna a tangente hiperbólica de `number`.

Number.ToText

26/05/2020 • 2 minutes to read

Sintaxe

```
Number.ToText(number as nullable number, optional format as nullable text, optional culture as
nullable text) as nullable text
```

Sobre

Formata o valor numérico `number` em um valor de texto de acordo com o formato especificado por `format`. O formato é um código de caractere único, opcionalmente seguido por um especificador de precisão de número. Os códigos de caracteres a seguir podem ser usados para `format`.

- "D" ou "d": (Decimal) Formata o resultado como dígitos inteiros. O especificador de precisão controla o número de dígitos na saída.
- "E" ou "e": (Exponencial/científico) Notação exponencial. O especificador de precisão controla o número máximo de dígitos decimais (o padrão é 6).
- "F" ou "f": (Ponto fixo) Dígitos inteiros e decimais.
- "G" ou "g": (Geral) A forma mais compacta de um ponto fixo ou científico.
- "N" ou "n": (Número) Dígitos inteiros e decimais com separadores de grupo e um separador decimal.
- "P" ou "p": (Percentual) Número multiplicado por 100 e exibido com um símbolo de porcentagem.
- "R" ou "r": (Viagem de ida e volta) Um valor de texto que pode fazer uma viagem de ida e volta de um número idêntico. O especificador de precisão é ignorado.
- "X" ou "x": (Hexadecimal) Um valor de texto hexadecimal.

Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Formatar um número como texto sem formato especificado.

```
Number.ToText(4)
```

`"4"`

Exemplo 2

Formatar um número como texto em formato Exponencial.

```
Number.ToText(4, "e")
```

`"4.000000e+000"`

Exemplo 3

Formatar um número como texto em formato Decimal com precisão limitada.

```
Number.ToText(-0.1234, "P1")
```

```
"-12.3 %"
```

Percentage.From

08/05/2020 • 2 minutes to read

Sintaxe

```
Percentage.From(value as any, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `percentage` do `value` especificado. Se o `value` fornecido for `null`, `Percentage.From` retornará `null`. Se o `value` fornecido for `text` com um símbolo de porcentagem à direita, o número decimal convertido será retornado. Caso contrário, confira `Number.From` para convertê-lo no valor `number`. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha o valor `percentage` de `"12.3%"`.

```
Percentage.From("12.3%")
```

`0.123`

RoundingMode.AwayFromZero

09/05/2020 • 2 minutes to read

Sobre

RoundingMode.AwayFromZero

RoundingMode.Down

09/05/2020 • 2 minutes to read

Sobre

RoundingMode.Down

RoundingMode.ToEven

09/05/2020 • 2 minutes to read

Sobre

RoundingMode.ToEven

RoundingMode.TowardZero

09/05/2020 • 2 minutes to read

Sobre

RoundingMode.TowardZero

RoundingMode.Up

09/05/2020 • 2 minutes to read

Sobre

RoundingMode.Up

Single.From

08/05/2020 • 2 minutes to read

Sintaxe

```
Single.From(value as any, optional culture as nullable text) as nullable number
```

Sobre

Retorna um valor `number` Single do `value` especificado. Se o `value` fornecido for `null`, `Single.From` retornará `null`. Se o `value` fornecido for `number` dentro do intervalo de Single, `value` será retornado; caso contrário, um erro será retornado. Se o `value` fornecido for de qualquer outro tipo, confira `Number.FromText` para convertê-lo no valor `number`; portanto, a instrução anterior sobre a conversão do valor `number` no valor `number` Single é aplicável. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obter o valor `number` Single de `"1.5"`.

```
Single.From("1.5")
```

`1.5`

Funções de registro

30/09/2020 • 5 minutes to read

Essas funções criam e manipulam valores de registro.

Registro

Informações

FUNÇÃO	DESCRIÇÃO
Record.FieldCount	Retorna o número de campos em um registro.
Record.HasFields	Retornará true se houver nomes do campo em um registro.

Transformações

FUNÇÃO	DESCRIÇÃO
Geography.FromWellKnownText	Converte texto representando um valor geográfico no formato WKT (Texto Bem Conhecido) em um registro estruturado.
Geography.ToWellKnownText	Converte um valor de ponto geográfico estruturado em sua representação WKT (Texto Bem Conhecido).
GeographyPoint.From	Cria um registro que representa um ponto geográfico de partes.
Geometry.FromWellKnownText	Converte texto representando um valor geométrico no formato WKT (Texto Bem Conhecido) em um registro estruturado.
Geometry.ToWellKnownText	Converte um valor de ponto geométrico estruturado em sua representação WKT (Texto Bem Conhecido).
GeometryPoint.From	Cria um registro que representa um ponto geométrico de partes.
Record.AddField	Adiciona um campo de um valor e nome de campo.
Record.Combine	Combina os registros em uma lista.
Record.RemoveFields	Retorna um novo registro que reordena os campos especificados em relação uns aos outros. Todos os campos não especificados permanecem em seus locais originais.

FUNÇÃO	DESCRIÇÃO
Record.RenameFields	Retorna um novo registro que renomeia os campos especificados. Os campos resultantes reterão sua ordem original. Essa função dá suporte à troca e ao encadeamento de nomes de campo. No entanto, todos os nomes de destino mais os nomes de campo restantes devem constituir um conjunto exclusivo ou ocorrerá um erro.
Record.ReorderFields	Retorna um novo registro que reordena os campos em relação uns aos outros. Todos os campos não especificados permanecem em seus locais originais. Requer dois campos ou mais.
Record.TransformFields	Transforma campos aplicando transformOperations. Para obter mais informações sobre valores com suporte do transformOperations, confira Valores de parâmetro.

Seleção

FUNÇÃO	DESCRIÇÃO
Record.Field	Retorna o valor do campo fornecido. Essa função pode ser usada para criar dinamicamente a sintaxe de pesquisa de campo para um determinado registro. Dessa forma, é um versão dinâmica da sintaxe record[field].
Record.FieldNames	Retorna uma lista de nomes de campo na ordem dos campos do registro.
Record.FieldOrDefault	Retorna o valor de um campo de um registro ou o valor padrão se o campo não existe.
Record.FieldValues	Retorna uma lista de valores de campo na ordem dos campos do registro.
Record.SelectFields	Retorna um novo registro que contém os campos selecionados do registro de entrada. A ordem original dos campos é mantida.

Serialização

FUNÇÃO	DESCRIÇÃO
Record.FromList	Retorna um registro de acordo com uma lista de valores de campos e um conjunto de campos.
Record.FromTable	Retorna um registro de uma tabela de registros contendo nomes de campo e de valores.
Record.ToList	Retorna uma lista de valores que contêm valores de campo do registro de entrada.
Record.ToTable	Retorna uma tabela de registros contendo nomes de campo e de valores de um registro de entrada.

Valores dos parâmetros

As definições de tipo a seguir são usadas para descrever os valores de parâmetro referenciados em funções de Registro acima.

Opção MissingField	MissingField.Error = 0; MissingField.Ignore = 1; MissingField.UseNull = 2;
Operações de transformação	As operações de transformação podem ser especificadas por qualquer um dos seguintes valores: Um valor de lista de dois itens, sendo o primeiro item o nome do campo e o segundo item a função de transformação aplicada a esse campo para produzir um novo valor. Uma lista de transformações pode ser fornecida informando um valor de lista e sendo cada item o valor de lista de dois itens, conforme descrito acima. Para obter exemplos, confira a descrição de Record.TransformFields
Operações de renomear	As operações de renomear para um registro podem ser especificadas como uma das opções: Uma única operação de renomear, representada por uma lista de dois nomes de campo, antigo e novo. Para obter exemplos, confira a descrição de Record.RenameFields.

Geography.FromWellKnownText

30/09/2020 • 2 minutes to read

Sintaxe

```
Geography.FromWellKnownText(input as nullable text) as nullable record
```

Sobre

Converte texto representando um valor geográfico no formato WKT (Texto Bem Conhecido) em um registro estruturado. WKT é um formato padrão definido pelo OGC (Open Geospatial Consortium) e é o formato de serialização típico usado por bancos de dados, incluindo o SQL Server.

Geography.ToWellKnownText

30/09/2020 • 2 minutes to read

Sintaxe

```
Geography.ToWellKnownText(input as nullable record, optional omitSRID as nullable logical) as
nullable text
```

Sobre

Converte um valor de ponto geográfico estruturado em sua representação WKT (Texto Bem Conhecido), formato de serialização usado por muitos bancos de dados, incluindo SQL Server, conforme definido pelo OGC (Open Geospatial Consortium).

GeographyPoint.From

30/09/2020 • 2 minutes to read

Sintaxe

```
GeographyPoint.From(longitude as number, latitude as number, optional z as nullable number,
optional m as nullable number, optional srid as nullable number) as record
```

Sobre

Cria um registro que representa um ponto geográfico de suas partes constituintes, como longitude, latitude e, se houver, elevação (Z) e medida (M). Um SRID (identificador de referência espacial opcional) poderá ser fornecido se for diferente do valor padrão (4326).

Geometry.FromWellKnownText

30/09/2020 • 2 minutes to read

Sintaxe

```
Geometry.FromWellKnownText(input as nullable text) as nullable record
```

Sobre

Converte texto representando um valor geométrico no formato WKT (Texto Bem Conhecido) em um registro estruturado. WKT é um formato padrão definido pelo OGC (Open Geospatial Consortium) e é o formato de serialização típico usado por bancos de dados, incluindo o SQL Server.

Geometry.ToWellKnownText

30/09/2020 • 2 minutes to read

Sintaxe

```
Geometry.ToWellKnownText(input as nullable record, optional omitSRID as nullable logical) as
nullable text
```

Sobre

Converte um valor de ponto geométrico estruturado em sua representação WKT (Texto Bem Conhecido), formato de serialização usado por muitos bancos de dados, incluindo SQL Server, conforme definido pelo OGC (Open Geospatial Consortium).

GeometryPoint.From

30/09/2020 • 2 minutes to read

Sintaxe

```
GeometryPoint.From(x as number, y as number, optional z as nullable number, optional m as nullable number, optional srid as nullable number) as record
```

Sobre

Cria um registro que representa um ponto geométrico de suas partes constituintes, como coordenada X, coordenada Y e, se houver, coordenada Z e medida (M). Um SRID (identificador de referência espacial opcional) poderá ser fornecido se for diferente do valor padrão (0).

MissingField.Error

09/05/2020 • 2 minutes to read

Sobre

Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes devem resultar em erro. (Esse é o valor do parâmetro padrão.)

MissingField.Ignore

09/05/2020 • 2 minutes to read

Sobre

Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes serão ignorados.

MissingField.UseNull

09/05/2020 • 2 minutes to read

Sobre

Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes serão incluídos como valores nulos.

Record.AddField

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.AddField(record as record, fieldName as text, value as any, optional delayed as nullable
logical) as record
```

Sobre

Adiciona um campo a um registro `record` do nome do campo `fieldName` e do valor `value`.

Exemplo 1

Adicionar o campo Endereço ao registro.

```
Record.AddField([CustomerID = 1, Name = "Bob", Phone = "123-4567"], "Address", "123 Main St.")
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567
ENDEREÇO	123 Main St.

Record.Combine

08/05/2020 • 2 minutes to read

Sintaxe

```
Record.Combine(records as list) as record
```

Sobre

Combina os registros na lista especificada `records`. Se o `records` contiver valores que não são de registro, um erro será retornado.

Exemplo 1

Criar um registro combinado dos registros.

```
Record.Combine({
    [CustomerID = 1, Name = "Bob"],
    [Phone = "123-4567"]
})
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Record.Field

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.Field(record as record, field as text) as any
```

Sobre

Retorna o valor do `field` especificado no `record`. Se o campo não for encontrado, uma exceção será gerada.

Exemplo 1

Localizar o valor do campo " CustomerID " no registro.

```
Record.Field([CustomerID = 1, Name = "Bob", Phone = "123-4567"], "CustomerID")
```

`1`

Record.FieldCount

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.FieldCount(record as record) as number
```

Sobre

Retorna o número de campos no registro `record`.

Exemplo 1

Localizar o número de campos no registro.

```
Record.FieldCount([CustomerID = 1, Name = "Bob"])
```

`2`

Record.FieldNames

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.FieldNames(record as record) as list
```

Sobre

Retorna os nomes dos campos no registro `record` como texto.

Exemplo 1

Localizar os nomes dos campos no registro.

```
Record.FieldNames([OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0])
```

OrderID
CustomerID
Item
Preço

Record.FieldOrDefault

21/05/2020 • 2 minutes to read

Sintaxe

```
Record.FieldOrDefault(record as nullable record, field as text, optional defaultValue as any) as any
```

Sobre

Retorna o valor do campo especificado `field` em um registro `record`. Se o campo não for encontrado, o `defaultValue` opcional será retornado.

Exemplo 1

Localizar o valor do campo "Phone" no registro ou retornar nulo, caso não exista.

```
Record.FieldOrDefault([CustomerID = 1, Name = "Bob"], "Phone")
```

`null`

Exemplo 2

Localizar o valor do campo "Phone" no registro ou retornar o padrão, caso não exista.

```
Record.FieldOrDefault([CustomerID = 1, Name = "Bob"], "Phone", "123-4567")
```

`"123-4567"`

Record.FieldValues

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.FieldValues(record as record) as list
```

Sobre

Retorna uma lista dos valores de campo no registro `record`.

Exemplo 1

Localizar os valores de campo no registro.

```
Record.FieldValues([CustomerID = 1, Name = "Bob", Phone = "123-4567"])
```

1
Bob
123-4567

Record.FromList

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.FromList(list as list, fields as any) as record
```

Sobre

Retorna um registro de acordo com uma `list` de valores de campos e um conjunto de campos. O `fields` pode ser especificado por uma lista de valores de texto ou um tipo de registro. Um erro será gerado se os campos não forem exclusivos.

Exemplo 1

Crie um registro com base em uma lista de valores de campos e uma lista de nomes de campos.

```
Record.FromList({1, "Bob", "123-4567"}, {"CustomerID", "Name", "Phone"})
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Exemplo 2

Crie um registro com base em uma lista de valores de campos e um tipo de registro.

```
Record.FromList({1, "Bob", "123-4567"}, type [CustomerID = number, Name = text, Phone = number])
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Record.FromTable

08/05/2020 • 2 minutes to read

Sintaxe

```
Record.FromTable(table as table) as record
```

Sobre

Retorna um registro de uma tabela de registros `table` contendo nomes de campo e nomes de valores `{[Name = name, Value = value]}`. Uma exceção será gerada se os nomes de campo não forem exclusivos.

Exemplo 1

Criar um registro da tabela no formato Table.FromRecords({[Name = "CustomerID", Value = 1], [Name = "Name", Value = "Bob"], [Name = "Phone", Value = "123-4567"]}).

```
Record.FromTable(
    Table.FromRecords({
        [Name = "CustomerID", Value = 1],
        [Name = "Name", Value = "Bob"],
        [Name = "Phone", Value = "123-4567"]
    })
)
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Record.HasFields

08/05/2020 • 2 minutes to read

Sintaxe

```
Record.HasFields(record as record, fields as any) as logical
```

Sobre

Indica se o registro `record` tem os campos especificados em `fields`, retornando um valor lógico (true ou false). Diversos valores de campo podem ser especificados usando uma lista.

Exemplo 1

Verificar se o registro tem o campo "CustomerID".

```
Record.HasFields([CustomerID = 1, Name = "Bob", Phone = "123-4567"], "CustomerID")
```

`true`

Exemplo 2

Verificar se o registro tem os campos "CustomerID" e "Address".

```
Record.HasFields([CustomerID = 1, Name = "Bob", Phone = "123-4567"], {"CustomerID", "Address"})
```

`false`

Record.RemoveFields

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.RemoveFields(record as record, fields as any, optional missingField as nullable number) as record
```

Sobre

Retorna um registro que remove todos os campos especificados na lista `fields` da entrada `record`. Se o campo especificado não existir, uma exceção será gerada.

Exemplo 1

Remover o campo "Price" do registro.

```
Record.RemoveFields([CustomerID = 1, Item = "Fishing rod", Price = 18.00], "Price")
```

CUSTOMERID	1
ITEM	Vara de pescar

Exemplo 2

Remover os campos "Price" e "Item" do registro.

```
Record.RemoveFields([CustomerID = 1, Item = "Fishing rod", Price = 18.00], {"Price", "Item"})
```

CUSTOMERID	1

Record.RenameFields

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.RenameFields(record as record, renames as list, optional missingField as nullable number)
as record
```

Sobre

Retorna um registro após alterar o nome dos campos no `record` de entrada para os novos nomes de campo especificados na lista `renames`. Para várias alterações de nome, uma lista aninhada pode ser usada ({ {old1, new1}, {old2, new2} }.

Exemplo 1

Renomear o campo "UnitPrice" para "Price" no registro.

```
Record.RenameFields(
    [OrderID = 1, CustomerID = 1, Item = "Fishing rod", UnitPrice = 100.0],
    {"UnitPrice", "Price"}
)
```

ORDERID	1
CUSTOMERID	1
ITEM	Vara de pescar
PREÇO	100

Exemplo 2

Renomear o campo "UnitPrice" para "Price" e "OrderNum" para "OrderID" no registro.

```
Record.RenameFields(
    [OrderNum = 1, CustomerID = 1, Item = "Fishing rod", UnitPrice = 100.0],
    {
        {"UnitPrice", "Price"},
        {"OrderNum", "OrderID"}
    }
)
```

ORDERID	1
CUSTOMERID	1
ITEM	Vara de pescar

PREÇO	100

Record.ReorderFields

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.ReorderFields(record as record, fieldOrder as list, optional missingField as nullable
number) as record
```

Sobre

Retorna um registro após reordenar os campos em `record` na ordem dos campos especificados na lista `fieldOrder`. Os valores de campo são mantidos e os campos não listados no `fieldOrder` são deixados em sua posição original.

Exemplo 1

Reordene alguns dos campos do registro.

```
Record.ReorderFields(
    [CustomerID = 1, OrderID = 1, Item = "Fishing rod", Price = 100.0],
    {"OrderID", "CustomerID"}
)
```

ORDERID	1
CUSTOMERID	1
ITEM	Vara de pescar
PREÇO	100

Record.SelectFields

08/05/2020 • 2 minutes to read

Sintaxe

```
Record.SelectFields(record as record, fields as any, optional missingField as nullable number) as
record
```

Sobre

Retorna um registro que inclui somente os campos especificados na lista `fields` da entrada `record`.

Exemplo 1

Selecionar os campos "Item" e "Price" no registro.

```
Record.SelectFields(
    [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
    {"Item", "Price"}
)
```

ITEM	Vara de pescar
PREÇO	100

Record.ToList

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.ToList(record as record) as list
```

Sobre

Retorna uma lista de valores que contêm valores de campo da entrada `record`.

Exemplo 1

Extraia os valores de campo de um registro.

```
Record.ToList([A = 1, B = 2, C = 3])
```

1
2
3

Record.ToTable

09/05/2020 • 2 minutes to read

Sintaxe

```
Record.ToTable(record as record) as table
```

Sobre

Retorna uma tabela que contém as colunas `Name` e `Value` com uma linha para cada campo em `record`.

Exemplo 1

Retornar a tabela do registro.

```
Record.ToTable([OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0])
```

NOME	VALOR
OrderID	1
CustomerID	1
Item	Vara de pescar
Preço	100

Record.TransformFields

21/05/2020 • 2 minutes to read

Sintaxe

```
Record.TransformFields(record as record, transformOperations as list, optional missingField as
nullable number) as record
```

Sobre

Retorna um registro após aplicar as transformações especificadas na lista `transformOperations` para `record`. Um ou mais campos podem ser transformados em um determinado momento.

Caso um único campo esteja sendo transformado, espera-se que `transformOperations` seja uma lista com dois itens. O primeiro item em `transformOperations` especifica um nome de campo e o segundo item em `transformOperations` especifica a função a ser usada para a transformação. Por exemplo, `{"Quantity", Number.FromText}`
No caso de vários campos estarem sendo transformados, espera-se que `transformOperations` seja uma lista de listas, em que cada lista interna é um par de nome de campo e operação de transformação. Por exemplo, `{{"Quantity",Number.FromText},{"UnitPrice", Number.FromText}}`

Exemplo 1

Converte o campo "Price" em número.

```
Record.TransformFields(
    [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = "100.0"],
    {"Price", Number.FromText}
)
```

ORDERID	1
CUSTOMERID	1
ITEM	Vara de pescar
PREÇO	100

Exemplo 2

Converte os campos "OrderID" e "Price" em números.

```
Record.TransformFields(
    [OrderID = "1", CustomerID = 1, Item = "Fishing rod", Price = "100.0"],
    {{"OrderID", Number.FromText}, {"Price", Number.FromText}}
)
```

ORDERID	1

CUSTOMERID	1
ITEM	Vara de pescar
PREÇO	100

Funções de substituto

08/05/2020 • 2 minutes to read

Essas funções são usadas por outras funções na biblioteca para substituir um determinado valor.

Substituto

FUNÇÃO	DESCRIÇÃO
Replacer.ReplaceText	Essa função é fornecida para List.ReplaceValue ou Table.ReplaceValue para substituir valores de texto em valores de lista e tabela, respectivamente.
Replacer.ReplaceValue	Essa função é fornecida para List.ReplaceValue ou Table.ReplaceValue para substituir valores em valores de lista e tabela, respectivamente.

Replacer.ReplaceText

09/05/2020 • 2 minutes to read

Sintaxe

```
Replacer.ReplaceText(text as nullable text, old as text, new as text) as nullable text
```

Sobre

Substitui o texto de `old` no `text` original pelo texto de `new`. Essa função de substituto pode ser usada em `List.ReplaceValue` e `Table.ReplaceValue`.

Exemplo 1

Substitua o texto "oL" por "Ol" na cadeia de caracteres "oLá, mundo".

```
Replacer.ReplaceText("hEllo world", "hE", "He")
```

`"Hello world"`

Replacer.ReplaceValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Replacer.ReplaceValue(value as any, old as any, new as any) as any
```

Sobre

Substitui o valor `old` no `value` original pelo valor `new`. Essa função de substituto pode ser usada em `List.ReplaceValue` e `Table.ReplaceValue`.

Exemplo 1

Substitua o valor 11 pelo valor 10.

```
Replacer.ReplaceValue(11, 11, 10)
```

`10`

Funções de divisor

08/05/2020 • 2 minutes to read

Essas funções dividem o texto.

Divisor

FUNÇÃO	DESCRIÇÃO
Splitter.SplitByNothing	Retorna uma função que não oferece divisão, retornando seu argumento como uma única lista de elementos.
Splitter.SplitTextByCharacterTransition	Retorna uma função que divide o texto em uma lista de texto de acordo com uma transição de um tipo de caractere para outro.
Splitter.SplitTextByAnyDelimiter	Retorna uma função que divide o texto por qualquer delimitador com suporte.
Splitter.SplitTextByDelimiter	Retorna uma função que dividirá o texto de acordo com um delimitador.
Splitter.SplitTextByEachDelimiter	Retorna uma função que divide o texto por cada delimitador por vez.
Splitter.SplitTextByLengths	Retorna uma função que divide o texto de acordo com os comprimentos especificados.
Splitter.SplitTextByPositions	Retorna uma função que divide o texto de acordo com as posições especificadas.
Splitter.SplitTextByRanges	Retorna uma função que divide o texto de acordo com os intervalos especificados.
Splitter.SplitTextByRepeatedLengths	Retorna uma função que divide o texto em uma lista de texto após o comprimento especificado repetidamente.
Splitter.SplitTextByWhitespace	Retorna uma função que divide o texto de acordo com o espaço em branco.

VALORES DE PARÂMETROS	DESCRIÇÃO
QuoteStyle.Csv	Caracteres de aspas indicam o início de uma cadeia de caracteres entre aspas. Aspas aninhadas são indicadas por dois caracteres de aspas.
QuoteStyle.None	Os caracteres de aspas não têm significado.

QuoteStyle.Csv

09/05/2020 • 2 minutes to read

Sobre

Caracteres de aspas indicam o início de uma cadeia de caracteres entre aspas. Aspas aninhadas são indicadas por dois caracteres de aspas.

QuoteStyle.None

09/05/2020 • 2 minutes to read

Sobre

Os caracteres de aspas não têm significado.

Splitter.SplitByNothing

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitByNothing() as function
```

Sobre

Retorna uma função que não oferece divisão, retornando seu argumento como uma única lista de elementos.

Splitter.SplitTextByAnyDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByAnyDelimiter(delimiters as list, optional quoteStyle as nullable number,
optional startAtEnd as nullable logical) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto em qualquer delimitador especificado.

Splitter.SplitTextByCharacterTransition

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByCharacterTransition(before as anynonnull, after as anynonnull) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto de acordo com uma transição de um tipo de caractere para outro. Os parâmetros `before` e `after` podem ser uma lista de caracteres ou uma função que usa um caractere e retorna true/false.

Splitter.SplitTextByDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByDelimiter(delimiter as text, optional quoteStyle as nullable number) as
function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto de acordo com o delimitador especificado.

Splitter.SplitTextByEachDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number,
optional startAtEnd as nullable logical) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto em cada delimitador especificado sequencialmente.

Splitter.SplitTextByLengths

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByLengths(lengths as list, optional startAtEnd as nullable logical) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto por cada comprimento especificado.

Splitter.SplitTextByPositions

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByPositions(positions as list, optional startAtEnd as nullable logical) as
function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto em cada posição especificada.

Splitter.SplitTextByRanges

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByRanges(ranges as list, optional startAtEnd as nullable logical) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto de acordo com os deslocamentos e comprimentos especificados.

Splitter.SplitTextByRepeatedLengths

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByRepeatedLengths(length as number, optional startAtEnd as nullable logical) as
function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto após o comprimento especificado repetidamente.

Splitter.SplitTextByWhitespace

09/05/2020 • 2 minutes to read

Sintaxe

```
Splitter.SplitTextByWhitespace(optional quoteStyle as nullable number) as function
```

Sobre

Retorna uma função que divide o texto em uma lista de texto em espaço em branco.

Funções de tabela

30/09/2020 • 28 minutes to read

Essas funções criam e manipulam valores de tabela.

Construção de tabela

FUNÇÃO	DESCRIÇÃO
ItemExpression.From	Retorna o AST para o corpo de uma função.
ItemExpression.Item	Um nó AST que representa o item em uma expressão de item.
RowExpression.Column	Retorna um AST que representa acesso a uma coluna em uma expressão de linha.
RowExpression.From	Retorna o AST para o corpo de uma função.
RowExpression.Row	Um nó AST representando a linha em uma expressão de linha.
Table.FromColumns	Retorna uma tabela de uma lista contendo listas aninhadas com os nomes e os valores de coluna.
Table.FromList	Converte uma lista em tabela aplicando a função de divisão especificada a cada item da lista.
Table.FromRecords	Retorna uma tabela de uma lista de registros.
Table.FromRows	Cria uma tabela da lista em que cada elemento da lista é uma lista que contém os valores de coluna para uma única linha.
Table.FromValue	Retorna uma tabela com uma coluna que contém o valor fornecido ou a lista de valores.
Table.FuzzyGroup	Agrupa as linhas de uma tabela pelos valores com correspondência difusa na coluna especificada para cada linha.
Table.FuzzyJoin	Une as linhas das duas tabelas com correspondência difusa com base nas chaves fornecidas.
Table.FuzzyNestedJoin	Executa uma junção difusa entre as tabelas nas colunas fornecidas e insere o resultado da junção em uma nova coluna.
Table.Split	Divide a tabela especificada em uma lista de tabelas usando o tamanho da página especificado.
Table.View	Cria ou estende uma tabela com manipuladores definidos pelo usuário para operações de consulta e ação.

FUNÇÃO	DESCRIÇÃO
Table.ViewFunction	Cria uma função que pode ser interceptada por um manipulador definido em uma exibição (por meio de `Table.View`).

Conversões

FUNÇÃO	DESCRIÇÃO
Table.ToColumns	Retorna uma lista das listas aninhadas, cada uma representando uma coluna de valores na tabela de entrada.
Table.ToList	Retorna uma tabela em lista aplicando a função de combinação especificada a cada linha de valores em uma tabela.
Table.ToRecords	Retorna uma lista de registros de uma tabela de entrada.
Table.ToRows	Retorna uma lista aninhada de valores de linha de uma tabela de entrada.

Informações

FUNÇÃO	DESCRIÇÃO
Table.ColumnCount	Retorna o número de colunas em uma tabela.
Table.IsEmpty	Retornará true se a tabela não contiver nenhuma linha.
Table.Profile	Retorna um perfil das colunas de uma tabela.
Table.RowCount	Retorna o número de linhas em uma tabela.
Table.Schema	Retorna uma tabela contendo uma descrição das colunas (ou seja, o esquema) da tabela especificada.
Tables.GetRelationships	Retorna os relacionamentos entre um conjunto de tabelas.

Operações de linha

FUNÇÃO	DESCRIÇÃO
Table.AlternateRows	Retorna uma tabela que contém um padrão alternado das linhas de uma tabela.
Table.Combine	Retorna uma tabela que é o resultado da mesclagem de uma lista de tabelas. Todas as tabelas devem ter a mesma estrutura de tipo de linha.

FUNÇÃO	DESCRIÇÃO
Table.FindText	Retorna uma tabela que contém apenas as linhas com o texto especificado dentro de uma de suas células ou de qualquer parte dela.
Table.First	Retorna a primeira linha de uma tabela.
Table.FirstN	Retorna as primeiras linhas de uma tabela, dependendo do parâmetro countOrCondition.
Table.FirstValue	Retorna a primeira coluna da primeira linha da tabela ou um valor padrão especificado.
Table.FromPartitions	Retorna uma tabela que é o resultado da combinação de um conjunto de tabelas particionadas em novas colunas. O tipo da coluna pode, opcionalmente, ser especificado; o padrão é Qualquer.
Table.InsertRows	Retorna uma tabela com a lista de linhas inseridas na tabela em um índice. Cada linha a ser inserida deve corresponder ao tipo de linha da tabela.
Table.Last	Retorna a última linha de uma tabela.
Table.LastN	Retorna as últimas linhas de uma tabela, dependendo do parâmetro countOrCondition.
Table.MatchesAllRows	Retornará true se todas as linhas em uma tabela cumprirem uma condição.
Table.MatchesAnyRows	Retornará true se alguma das linhas em uma tabela cumprirem uma condição.
Table.Partition	Particiona a tabela em uma lista de grupos de número de tabelas com base no valor da coluna de cada linha e uma função de hash. A função de hash é aplicada ao valor da coluna de uma linha para obter um valor de hash para a linha. Os grupos de módulos de valor de hash determinam em quais tabelas retornadas a linha será colocada.
Table.Range	Retorna o número especificado de linhas de uma tabela que começa em um deslocamento.
Table.RemoveFirstN	Retorna uma tabela com o número especificado de linhas removidas da tabela começando na primeira linha. O número de linhas removidas depende do parâmetro opcional countOrCondition.
Table.RemoveLastN	Retorna uma tabela com o número especificado de linhas removidas da tabela começando na última linha. O número de linhas removidas depende do parâmetro opcional countOrCondition.
Table.RemoveRows	Retorna uma tabela com o número especificado de linhas removidas da tabela começando em um deslocamento.

FUNÇÃO	DESCRIÇÃO
Table.RemoveRowsWithErrors	Retorna uma tabela com todas as linhas removidas da tabela que contêm um erro em pelo menos uma das células em uma linha.
Table.Repeat	Retorna uma tabela que contém as linhas da tabela repetidas o número de vezes da contagem.
Table.ReplaceRows	Retorna uma tabela na qual as linhas que começam em um deslocamento e continuam para a contagem são substituídas pelas linhas fornecidas.
Table.ReverseRows	Retorna uma tabela com as linhas na ordem inversa.
Table.SelectRows	Retorna uma tabela que contém apenas as linhas que correspondem a uma condição.
Table.SelectRowsWithErrors	Retorna uma tabela apenas com as linhas da tabela que contêm um erro em pelo menos uma das células em uma linha.
Table.SingleRow	Retorna uma única linha de uma tabela.
Table.Skip	Retorna uma tabela que não contém a primeira linha ou as primeiras linhas da tabela.

Operações de coluna

FUNÇÃO	DESCRIÇÃO
Table.Column	Retorna os valores de uma coluna em uma tabela.
Table.ColumnNames	Retorna os nomes das colunas de uma tabela.
Table.ColumnsOfType	Retorna uma lista com os nomes das colunas que correspondem aos tipos especificados.
Table.DemoteHeaders	Rebaixa a linha de cabeçalho para a primeira linha de uma tabela.
Table.DuplicateColumn	Duplica uma coluna com o nome especificado. Os valores e o tipo são copiados da coluna de origem.
Table.HasColumns	Retornará true se uma tabela tiver a coluna ou as colunas especificadas.
Table.Pivot	Dada uma tabela e uma coluna de atributo contendo pivotValues, cria novas colunas para cada um dos valores dinâmicos e atribui a eles valores de valueColumn. Uma aggregationFunction opcional pode ser fornecida para lidar com várias ocorrências do mesmo valor de chave na coluna de atributo.

FUNÇÃO	DESCRIÇÃO
Table.PrefixColumns	Retorna uma tabela em que todas as colunas sejam prefixadas um valor de texto.
Table.PromoteHeaders	Promove a primeira linha da tabela em seus nomes de cabeçalho ou coluna.
Table.RemoveColumns	Retorna uma tabela sem uma ou mais colunas específicas.
Table.ReorderColumns	Retorna uma tabela com colunas específicas em uma ordem relativa umas às outras.
Table.RenameColumns	Retorna uma tabela com as colunas renomeadas conforme especificado.
Table.SelectColumns	Retorna uma tabela que contém apenas colunas específicas.
Table.TransformColumnNames	Transforma nomes de colunas ao usar a função fornecida.
Table.Unpivot	Dada uma lista de colunas de tabela, transforma essas colunas em pares de atributo/valor.
Table.UnpivotOtherColumns	Converte todas as colunas que não sejam um conjunto especificado em pares de atributo/valor, combinados com o restante dos valores em cada linha.

Parâmetros

VALORES DE PARÂMETROS	DESCRIÇÃO
JoinKind.Inner	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. A tabela resultante de uma junção interna contém uma linha para cada par de linhas das tabelas especificadas que foram determinadas para correspondência com base nas colunas de chave especificadas.
JoinKind.LeftOuter	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa esquerda garante que todas as linhas da primeira tabela sejam exibidas no resultado.
JoinKind.RightOuter	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa direita garante que todas as linhas da segunda tabela sejam exibidas no resultado.
JoinKind.FullOuter	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa completa garante que todas as linhas de ambas as tabelas sejam exibidas no resultado. As linhas que não tiveram uma correspondência na outra tabela são unidas com uma linha padrão que contém valores nulos para todas as colunas.

VALORES DE PARÂMETROS	DESCRIÇÃO
JoinKind.LeftAnti	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma antijunção esquerda retorna todas as linhas da primeira tabela que não têm uma correspondência na segunda tabela.
JoinKind.RightAnti	Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma antijunção direita retorna todas as linhas da segunda tabela que não têm uma correspondência na primeira tabela.
MissingField.Error	Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes devem resultar em erro. (Esse é o valor do parâmetro padrão.)
MissingField.Ignore	Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes serão ignorados.
MissingField.UseNull	Um parâmetro opcional nas funções de registro e de tabela indicando que os campos ausentes serão incluídos como valores nulos.
GroupKind.Global	GroupKind.Global
GroupKind.Local	GroupKind.Local
ExtraValues.List	Se a função de divisor retornar mais colunas do que o esperado pela tabela, elas serão coletadas em uma lista.
ExtraValues.Ignore	Se a função de divisor retornar mais colunas do que o esperado pela tabela, elas serão ignoradas.
ExtraValues.Error	Se a função de divisor retornar mais colunas do que o esperado pela tabela, um erro será gerado.
JoinAlgorithm.Dynamic	JoinAlgorithm.Dynamic
JoinAlgorithm.PairwiseHash	JoinAlgorithm.PairwiseHash
JoinAlgorithm.SortMerge	JoinAlgorithm.SortMerge
JoinAlgorithm.LeftHash	JoinAlgorithm.LeftHash
JoinAlgorithm.RightHash	JoinAlgorithm.RightHash
JoinAlgorithm.LeftIndex	JoinAlgorithm.LeftIndex
JoinAlgorithm.RightIndex	JoinAlgorithm.RightIndex
JoinSide.Left	Especifica a tabela à esquerda de uma junção.
JoinSide.Right	Especifica a tabela à direita de uma junção.

Transformation

Parâmetros para opções de Grupo

- GroupKind.Global = 0;
- GroupKind.Local = 1;

Parâmetros para tipos de Junção

- JoinKind.Inner = 0;
- JoinKind.LeftOuter = 1;
- JoinKind.RightOuter = 2;
- JoinKind.FullOuter = 3;
- JoinKind.LeftAnti = 4;
- JoinKind.RightAnti = 5

Algoritmo de Junção

Os seguintes valores de JoinAlgorithm podem ser especificados para Table.Join

```
JoinAlgorithm.Dynamic        0,
```

```
JoinAlgorithm.PairwiseHash   1,
```

```
JoinAlgorithm.SortMerge      2,
```

```
JoinAlgorithm.LeftHash       3,
```

```
JoinAlgorithm.RightHash      4,
```

```
JoinAlgorithm.LeftIndex      5,
```

```
JoinAlgorithm.RightIndex     6,
```

VALORES DE PARÂMETROS	DESCRIÇÃO
JoinSide.Left	Especifica a tabela à esquerda de uma junção.
JoinSide.Right	Especifica a tabela à direita de uma junção.

Dados de exemplo

Os exemplos nesta seção usam as tabelas a seguir.

Tabela de clientes

```
Customers = Table.FromRecords({
  [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
  [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
  [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
  [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}
```

Tabela de pedidos

```
Orders = Table.FromRecords({
  [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
  [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
  [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
  [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
  [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
  [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
  [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
  [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
  [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
})
```

FUNÇÃO	DESCRIÇÃO
Table.AddColumn	Adiciona uma coluna chamada newColumnName a uma tabela.
Table.AddFuzzyClusterColumn	Adiciona uma nova coluna com valores representativos obtidos pelo agrupamento difuso dos valores da coluna especificada na tabela.
Table.AddIndexColumn	Retorna uma tabela com uma nova coluna com um nome específico que, para cada linha, contém um índice da linha na tabela.
Table.AddJoinColumn	Executa uma junção aninhada entre table1 e table2 de colunas específicas e produz o resultado de junção como uma coluna newColumnName para cada linha da table1.
Table.AddKey	Adicione uma chave à tabela.
Table.AggregateTableColumn	Agrega tabelas aninhadas em uma coluna específica em várias colunas que contêm valores de agregação para essas tabelas.

FUNÇÃO	DESCRIÇÃO
Table.CombineColumns	Table.CombineColumns mescla colunas usando uma função de combinação para produzir uma nova coluna. Table.CombineColumns é o inverso de Table.SplitColumns.
Table.CombineColumnsToRecord	Combina as colunas especificadas em uma nova coluna com valor de registro, na qual cada registro tem nomes e valores de campo correspondentes aos nomes e valores de coluna das colunas combinadas.
Table.ConformToPageReader	Esta função destina-se somente a uso interno.
Table.ExpandListColumn	Dada uma coluna de listas em uma tabela, crie a cópia de uma linha para cada valor na lista.
Table.ExpandRecordColumn	Expande uma coluna de registros em colunas com cada um dos valores.
Table.ExpandTableColumn	Expande uma coluna de registros ou uma coluna de tabelas em várias colunas na tabela que as contém.
Table.FillDown	Substitui valores nulos na coluna ou nas colunas especificadas da tabela com o valor não nulo mais recente na coluna.
Table.FillUp	Retorna uma tabela da tabela especificada em que o valor da próxima célula é propagado para as células de valor nulo acima na coluna especificada.
Table.FilterWithDataTable	
Table.Group	Agrupa linhas de tabela pelos valores de colunas de chave para cada linha.
Table.Join	Une as linhas da tabela1 com as linhas da tabela2 com base na igualdade dos valores das colunas de chave selecionadas por tabela1, chave1 e tabela2, chave2.
Table.Keys	Retorna uma lista de nomes de coluna de chave de uma tabela.
Table.NestedJoin	Une as linhas das tabelas com base na igualdade das chaves. Os resultados são inseridos em uma nova coluna.
Table.ReplaceErrorValues	Substitui os valores de erro nas colunas especificadas com o valor especificado correspondente.
Table.ReplaceKeys	Retorna uma nova tabela com o novo conjunto de informações de chave no argumento keys.
Table.ReplaceRelationshipIdentity	
Table.ReplaceValue	Substitui oldValue por newValue em colunas específicas de uma tabela, usando a função de substituto fornecida, como text.Replace ou Value.Replace.

FUNÇÃO	DESCRIÇÃO
Table.SplitColumn	Retorna um novo conjunto de colunas de uma única coluna aplicando uma função de divisor a cada valor.
Table.TransformColumns	Transforma colunas de uma tabela usando uma função.
Table.TransformColumnTypes	Transforma os tipos de coluna de uma tabela usando um tipo.
Table.TransformRows	Transforma as linhas de uma tabela usando uma função de transformação.
Table.Transpose	Retorna uma tabela com colunas convertidas em linhas e linhas convertidas em colunas da tabela de entrada.

Associação

Parâmetros para verificações de associação

Especificação de ocorrência

```
Occurrence.First  = 0
```

```
Occurrence.Last   = 1
```

```
Occurrence.All    = 2
```

FUNÇÃO	DESCRIÇÃO
Table.Contains	Determina se o registro a aparece como uma linha na tabela.
Table.ContainsAll	Determina se todos os registros especificados serão exibidos como linhas na tabela.
Table.ContainsAny	Determina se algum dos registros especificados será exibido como linhas na tabela.
Table.Distinct	Remove linhas duplicadas de uma tabela, garantindo que todas as linhas restantes sejam distintas.
Table.IsDistinct	Determina se uma tabela contém apenas linhas distintas.
Table.PositionOf	Determina a posição ou as posições de uma linha em uma tabela.
Table.PositionOfAny	Determina uma ou mais posições de qualquer linha especificada na tabela.
Table.RemoveMatchingRows	Remove todas as ocorrências de linhas de uma tabela.

FUNÇÃO	DESCRIÇÃO
Table.ReplaceMatchingRows	Substitui linhas específicas de uma tabela pelas novas linhas.

Ordenando

Dados de exemplo

Os exemplos nesta seção usam as tabelas a seguir.

Tabela Funcionários

```
Employees = Table.FromRecords(

    {[Name="Bill",   Level=7,  Salary=100000],

     [Name="Barb",   Level=8,  Salary=150000],

     [Name="Andrew", Level=6,  Salary=85000],

     [Name="Nikki",  Level=5,  Salary=75000],

     [Name="Margo",  Level=3,  Salary=45000],

     [Name="Jeff",   Level=10, Salary=200000]},

type table [

    Name = text,

    Level = number,

    Salary = number

])
```

FUNÇÃO	DESCRIÇÃO
Table.Max	Retorna a maior linha ou as maiores linhas de uma tabela usando um comparisonCriteria.
Table.MaxN	Retorna as N maiores linhas de uma tabela. Depois que as linhas são classificadas, o parâmetro countOrCondition deve ser especificado para filtrar melhor o resultado.
Table.Min	Retorna a menor linha ou as menores linhas de uma tabela usando um comparisonCriteria.
Table.MinN	Retorna as N menores linhas na tabela especificada. Depois que as linhas são classificadas, o parâmetro countOrCondition deve ser especificado para filtrar melhor o resultado.
Table.Sort	Classifica as linhas em uma tabela usando um comparisonCriteria ou uma ordenação padrão se uma não foi especificada.

Outro

FUNÇÃO	DESCRIÇÃO
Table.Buffer	Armazena uma tabela em buffer na memória, isentando-a de alterações externas durante a avaliação.

Valores dos parâmetros

Como dar nome a colunas de saída

Esse parâmetro é uma lista de valores de texto especificando os nomes de coluna da tabela resultante. Esse parâmetro geralmente é usado nas funções de construção de tabela, como Table.FromRows e Table.FromList.

Critérios de comparação

O critério de comparação pode ser fornecido como um dos seguintes valores:

- Um valor numérico para especificar uma ordem de classificação. Confira ordem de classificação na seção valores de parâmetro acima.
- Para calcular uma chave a ser usada para classificação, uma função de um argumento pode ser usada.
- Para selecionar uma ordem de chave e de controle, o critério de comparação pode ser uma lista que contém a chave e a ordem.
- Para controlar por completo a comparação, é possível usar uma função de dois argumentos que retorna -1, 0 ou 1, considerando a relação entre as entradas esquerda e direita. Value.Compare é um método que pode ser usado para delegar essa lógica.

Para obter exemplos, confira a descrição de Table.Sort.

Critérios de contagem ou condição

Geralmente, esses critérios são usados em operações de ordenação ou de linha. Eles determinam o número de linhas retornadas na tabela e podem usar duas formas, um número ou uma condição:

- Um número indica quantos valores retornar embutidos com a função apropriada
- Se uma condição for especificada, as linhas que contêm valores que atendem inicialmente à condição serão retornadas. Quando um valor falha na condição, não são considerados outros valores.

Confira Table.FirstN ou Table.MaxN.

Manipulação de valores extras

Isso é usado para indicar como a função deve tratar valores adicionais em uma linha. Esse parâmetro é especificado como um número, que é mapeado para as opções abaixo.

```
ExtraValues.List = 0

ExtraValues.Error = 1

ExtraValues.Ignore = 2
```

Para obter mais informações, confira Tabela.FromList.

Manipulação de coluna ausente

Isso é usado para indicar como a função deve tratar colunas ausentes. Esse parâmetro é especificado como um número, que é mapeado para as opções abaixo.

```
MissingField.Error = 0;

MissingField.Ignore = 1;

MissingField.UseNull = 2;
```

Isso é usado em operações de coluna ou transformação. Para obter exemplos, confira Table.TransformColumns.

Ordem de classificação

Isso é usado para indicar como os resultados devem ser classificados. Esse parâmetro é especificado como um número, que é mapeado para as opções abaixo.

```
Order.Ascending = 0

    Order.Descending = 1
```

Critérios de equação

Os critérios de equação para tabelas podem ser especificados como

- Um valor de função que é
  - Um seletor de chave que determina a coluna na tabela a aplicar os critérios de igualdade ou
  - Uma função de comparador que é usada para especificar o tipo de comparação a ser aplicado. As funções de comparador internas podem ser especificadas; confira a seção Funções de comparação.
- Uma lista das colunas na tabela para aplicar os critérios de igualdade

Para obter exemplos, confira a descrição de Table.Distinct.

ExtraValues.Error

09/05/2020 • 2 minutes to read

Sobre

Se a função de divisor retornar mais colunas do que o esperado pela tabela, um erro será gerado.

ExtraValues.Ignore

09/05/2020 • 2 minutes to read

Sobre

Se a função de divisor retornar mais colunas do que o esperado pela tabela, elas serão ignoradas.

ExtraValues.List

09/05/2020 • 2 minutes to read

Sobre

Se a função de divisor retornar mais colunas do que o esperado pela tabela, elas serão coletadas em uma lista.

GroupKind.Global

09/05/2020 • 2 minutes to read

Sobre

Sintaxe

```
GroupKind.Global
```

GroupKind.Local

09/05/2020 • 2 minutes to read

Sobre

Sintaxe

```
GroupKind.Local
```

Sobre

GroupKind.Local

ItemExpression.From

09/05/2020 • 2 minutes to read

Sintaxe

```
ItemExpression.From(function as function) as record
```

Sobre

Retorna a AST do corpo de `function`, normalizado em uma *expressão de item*:

- A função deve ser um lambda de 1 argumento.
- Todas as referências ao parâmetro de função são substituídas por `ItemExpression.Item`.
- A AST será simplificada para conter apenas nós dos tipos:
  - `Constant`
  - `Invocation`
  - `Unary`
  - `Binary`
  - `If`
  - `FieldAccess`
  - `NotImplemented`

Um erro será gerado se uma AST de expressão de item não puder ser retornada para o corpo de `function`.

Exemplo 1

Retorna o AST do corpo da função `each _ <> null`

```
ItemExpression.From(each _ <> null)
```

TIPO	Binário
OPERADOR	NotEquals
ESQUERDA	[Registro]
DIREITA	[Registro]

ItemExpression.Item

09/05/2020 • 2 minutes to read

Sobre

Um nó AST que representa o item em uma expressão de item.

JoinAlgorithm.Dynamic

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.Dynamic

JoinAlgorithm.LeftHash

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.LeftHash

JoinAlgorithm.LeftIndex

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.LeftIndex

JoinAlgorithm.PairwiseHash

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.PairwiseHash

JoinAlgorithm.RightHash

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.RightHash

JoinAlgorithm.RightIndex

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.RightIndex

JoinAlgorithm.SortMerge

09/05/2020 • 2 minutes to read

Sobre

JoinAlgorithm.SortMerge

JoinKind.FullOuter

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa completa garante que todas as linhas de ambas as tabelas sejam exibidas no resultado. As linhas que não tiveram uma correspondência na outra tabela são unidas com uma linha padrão que contém valores nulos para todas as colunas.

JoinKind.Inner

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. A tabela resultante de uma junção interna contém uma linha para cada par de linhas das tabelas especificadas que foram determinadas para correspondência com base nas colunas de chave especificadas.

JoinKind.LeftAnti

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma antijunção esquerda retorna todas as linhas da primeira tabela que não têm uma correspondência na segunda tabela.

JoinKind.LeftOuter

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa esquerda garante que todas as linhas da primeira tabela sejam exibidas no resultado.

JoinKind.RightAnti

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma antijunção direita retorna todas as linhas da segunda tabela que não têm uma correspondência na primeira tabela.

JoinKind.RightOuter

09/05/2020 • 2 minutes to read

Sobre

Um valor possível para o parâmetro opcional `JoinKind` no `Table.Join`. Uma junção externa direita garante que todas as linhas da segunda tabela sejam exibidas no resultado.

JoinSide.Left

09/05/2020 • 2 minutes to read

Sobre

Especifica a tabela à esquerda de uma junção.

JoinSide.Right

09/05/2020 • 2 minutes to read

Sobre

Especifica a tabela à direita de uma junção.

Occurrence.All

09/05/2020 • 2 minutes to read

Sobre

Uma lista de posições de todas as ocorrências dos valores encontrados é retornada.

Occurrence.First

09/05/2020 • 2 minutes to read

Sobre

A posição da primeira ocorrência do valor encontrado é retornada.

Occurrence.Last

09/05/2020 • 2 minutes to read

Sobre

A posição da última ocorrência do valor encontrado é retornada.

Order.Ascending

09/05/2020 • 2 minutes to read

Sobre

Tipo de função que classifica a lista em ordem crescente.

Order.Descending

09/05/2020 • 2 minutes to read

Sobre

Tipo de função que classifica a lista em ordem decrescente.

RowExpression.Column

09/05/2020 • 2 minutes to read

Sintaxe

```
RowExpression.Column(columnName as text) as record
```

Sobre

Retorna um AST que representa acesso à coluna columnName da linha em uma expressão de linha.

Exemplo 1

Cria um AST representando o acesso da coluna "CustomerName".

```
RowExpression.Column("CustomerName")
```

TIPO	FieldAccess
EXPRESSÃO	[Registro]
MEMBERNAME	CustomerName

RowExpression.From

09/05/2020 • 2 minutes to read

Sintaxe

```
RowExpression.From(function as function) as record
```

Sobre

Retorna a AST do corpo de `function`, normalizado em uma *expressão de linha*:

- A função deve ser um lambda de 1 argumento.
- Todas as referências ao parâmetro de função são substituídas por `RowExpression.Row`.
- Todas as referências a colunas são substituídas por `RowExpression.Column(columnName)`.
- A AST será simplificada para conter apenas nós dos tipos:
  - `Constant`
  - `Invocation`
  - `Unary`
  - `Binary`
  - `If`
  - `FieldAccess`
  - `NotImplemented`

Um erro será gerado se uma AST de expressão de linha não puder ser retornada para o corpo de `function`.

Exemplo 1

Retorna o AST para o corpo da função each [CustomerID] = "ALFKI"

```
RowExpression.From(each [CustomerName] = "ALFKI")
```

TIPO	Binário
OPERADOR	Igual a
ESQUERDA	[Registro]
DIREITA	[Registro]

RowExpression.Row

09/05/2020 • 2 minutes to read

Sobre

Um nó AST representando a linha em uma expressão de linha.

Table.AddColumn

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.AddColumn(table as table, newColumnName as text, columnGenerator as function, optional
columnType as nullable type) as table
```

Sobre

Adiciona uma coluna denominada `newColumnName` à tabela `table`. Os valores da coluna são computados por meio da função de seleção especificada `columnGenerator` com cada linha assumida como entrada.

Exemplo 1

Adicionar uma coluna denominada "TotalPrice" à tabela, com cada valor sendo a soma da coluna [Price] e da coluna [Shipping].

```
Table.AddColumn(
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0, Shipping = 10.00],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0, Shipping = 15.00],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0, Shipping = 10.00]
    }),
    "TotalPrice",
    each [Price] + [Shipping]
)
```

ORDERID	CUSTOMERID	ITEM	PREÇO	REMESSA	TOTALPRICE
1	1	Vara de pescar	100	10	110
2	1	1 lb de minhocas	5	15	20
3	2	Rede de pesca	25	10	35

Table.AddFuzzyClusterColumn

19/10/2020 • 4 minutes to read

Sintaxe

```
Table.AddFuzzyClusterColumn(table as table, columnName as text, newColumnName as text, optional
options as nullable record) as table
```

Sobre

Adiciona uma nova coluna `newColumnName` a `table` com valores representativos de `columnName`. Os valores representativos são obtidos por meio da correspondência difusa dos valores em `columnName` para cada linha.

Um conjunto opcional de `options` pode ser incluído para especificar como comparar as colunas de chave. As opções incluem:

- `Culture`: permite agrupar registros com base em regras específicas da cultura. Pode ser qualquer nome de cultura válido. Por exemplo, a opção de Culture "ja-JP" agrupa registros com base no idioma japonês. O valor padrão é "", que faz o agrupamento com base na cultura inglês invariável.
- `IgnoreCase`: um valor lógico (true/false) que permite fazer o agrupamento de chaves sem diferenciar maiúsculas de minúsculas. Por exemplo, quando true, "Grapes" é agrupado com "grapes". O valor padrão é true.
- `IgnoreSpace`: um valor lógico (true/false) que permite a combinação de partes de textos a fim de localizar grupos. Por exemplo, quando true, "Gra pes" é agrupado com "Grapes". O valor padrão é true.
- `SimilarityColumnName`: um nome para a coluna que mostra a similaridade entre um valor de entrada e o valor representativo dessa entrada. O valor padrão é nulo e, nesse caso, não será adicionada uma nova coluna de similaridades.
- `Threshold`: um número entre 0.00 e 1.00 que especifica a pontuação de similaridade com a qual dois valores serão agrupados. Por exemplo, "Grapes" e "Graes" (sem o "p") serão agrupados somente se essa opção estiver configurada como inferior a 0.90. Um limite de 1.00 é o mesmo que especificar o critério de correspondência exata para o agrupamento. O valor padrão é 0.80.
- `TransformationTable`: uma tabela que permite agrupar registros com base em mapeamentos de valor personalizados. Deve conter as colunas "From" e "To". Por exemplo, "Grapes" será agrupado com "Raisins" se for fornecida uma tabela de transformação com a coluna "From" contendo "Grapes" e a coluna "To" contendo "Raisins". Observe que a transformação será aplicada a todas as ocorrências do texto na tabela de transformação. Com a tabela de transformação acima, a frase "Grapes are sweet" também será agrupada com a frase "Raisins are sweet".

Exemplo 1

Localize os valores representativos da localização dos funcionários.

```
Table.AddFuzzyClusterColumn(
    Table.FromRecords(
        {
            [EmployeeID = 1, Location = "Seattle"],
            [EmployeeID = 2, Location = "seattl"],
            [EmployeeID = 3, Location = "Vancouver"],
            [EmployeeID = 4, Location = "Seatle"],
            [EmployeeID = 5, Location = "vancover"],
            [EmployeeID = 6, Location = "Seattle"],
            [EmployeeID = 7, Location = "Vancouver"]
        },
        type table [EmployeeID = nullable number, Location = nullable text]
    ),
    "Location",
    "Location_Cleaned",
    [IgnoreCase = true, IgnoreSpace = true]
)
```

```
Table.FromRecords(
    {
        [EmployeeID = 1, Location = "Seattle", Location_Cleaned = "Seattle"],
        [EmployeeID = 2, Location = "seattl", Location_Cleaned = "Seattle"],
        [EmployeeID = 3, Location = "Vancouver", Location_Cleaned = "Vancouver"],
        [EmployeeID = 4, Location = "Seatle", Location_Cleaned = "Seattle"],
        [EmployeeID = 5, Location = "vancover", Location_Cleaned = "Vancouver"],
        [EmployeeID = 6, Location = "Seattle", Location_Cleaned = "Seattle"],
        [EmployeeID = 7, Location = "Vancouver", Location_Cleaned = "Vancouver"]
    },
    type table [EmployeeID = nullable number, Location = nullable text, Location_Cleaned = nullable text]
)
```

Table.AddIndexColumn

21/09/2020 • 2 minutes to read

Sintaxe

```
Table.AddIndexColumn(table as table, newColumnName as text, optional initialValue as nullable
number, optional increment as nullable number, optional columnType as nullable type) as table
```

Sobre

Anexa uma coluna chamada `newColumnName` à `table` com valores de posição explícitos. Um valor opcional, `initialValue`, o valor do índice inicial. Um valor opcional, `increment`, especifica o quanto incrementar cada valor de índice.

Exemplo 1

Adicionar uma coluna de índice denominada "Index" à tabela.

```
Table.AddIndexColumn(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Index"
)
```

CUSTOMERID	NOME	TELEFONE	ÍNDICE
1	Bob	123-4567	0
2	Jim	987-6543	1
3	Paul	543-7890	2
4	Ringo	232-1550	3

Exemplo 2

Adicionar uma coluna de índice denominada "Index", que começa no valor 10 e incrementando em 5, à tabela.

```
Table.AddIndexColumn(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Index",
    10,
    5
)
```

CUSTOMERID	NOME	TELEFONE	ÍNDICE
1	Bob	123-4567	10
2	Jim	987-6543	15
3	Paul	543-7890	20
4	Ringo	232-1550	25

Table.AddJoinColumn

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.AddJoinColumn(table1 as table, key1 as any, table2 as function, key2 as any, newColumnName
as text) as table
```

Sobre

Une as linhas de `table1` com as linhas de `table2` com base na igualdade dos valores das colunas de chave selecionadas por `key1` (para `table1`) e `key2` (para `table2`). Os resultados são inseridos na coluna chamada `newColumnName`. Essa função se comporta de forma semelhante à Table.Join com um JoinKind de LeftOuter, exceto que os resultados da junção são apresentados em um modo aninhado, e não nivelado.

Exemplo 1

Adicione uma coluna de junção a ({[saleID = 1, item = "Shirt"], [saleID = 2, item = "Hat"]}) denominada "preço/estoque" da tabela ({[saleID = 1, price = 20], [saleID = 2, price = 10]}) associada em [saleID].

```
Table.AddJoinColumn(
    Table.FromRecords({
        [saleID = 1, item = "Shirt"],
        [saleID = 2, item = "Hat"]
    }),
    "saleID",
    () => Table.FromRecords({
        [saleID = 1, price = 20, stock = 1234],
        [saleID = 2, price = 10, stock = 5643]
    }),
    "saleID",
    "price"
)
```

SALEID	ITEM	PREÇO
1	Camiseta	[Tabela]
2	Hat	[Tabela]

Table.AddKey

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.AddKey(table as table, columns as list, isPrimary as logical) as table
```

Sobre

Adicione uma chave a `table`, considerando que `columns` é o subconjunto dos nomes de coluna de `table` que define a chave e `isPrimary` especifica se a chave é primária.

Exemplo 1

Adicione uma chave a {[Id = 1, Name = "Hello There"], [Id = 2, Name = "Good Bye"]} composta por {"Id"} e torne-a primária.

```
let
    tableType = type table [Id = Int32.Type, Name = text],
    table = Table.FromRecords({
        [Id = 1, Name = "Hello There"],
        [Id = 2, Name = "Good Bye"]
    }),
    resultTable = Table.AddKey(table, {"Id"}, true)
in
    resultTable
```

ID	NOME
1	Olá
2	Até logo

Table.AggregateTableColumn

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.AggregateTableColumn(table as table, column as text, aggregations as list) as table
```

Sobre

Agrega tabelas na `table` [`column`] em várias colunas que contêm valores de agregação para as tabelas. `aggregations` é usado para especificar as colunas que contêm as tabelas a serem agregadas, as funções de agregação a serem aplicadas às tabelas para gerar seus valores e os nomes das colunas de agregação a serem criadas.

Exemplo 1

As colunas de tabela de agregação em `[t]` na tabela `{[t = {[a=1, b=2, c=3], [a=2,b=4,c=6]}, b = 2]}` na soma de `[t.a]`, o mínimo e o máximo de `[t.b]` e a contagem de valores em `[t.a]`.

```
Table.AggregateTableColumn(
    Table.FromRecords(
        {
            [
                t = Table.FromRecords({
                    [a = 1, b = 2, c = 3],
                    [a = 2, b = 4, c = 6]
                }),
                b = 2
            ]
        },
        type table [t = table [a = number, b = number, c = number], b = number]
    ),
    "t",
    {
        {"a", List.Sum, "sum of t.a"},
        {"b", List.Min, "min of t.b"},
        {"b", List.Max, "max of t.b"},
        {"a", List.Count, "count of t.a"}
    }
)
```

SOMA DE T.A	MÍNIMO DE T.B	MÁXIMO DE T.B	CONTAGEM DE T.A	B
3	2	4	2	2

Table.AlternateRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.AlternateRows(table as table, offset as number, skip as number, take as number) as table
```

Sobre

Mantém o deslocamento inicial e alterna o uso e não uso das linhas a seguir.

- `table`: A tabela de entrada.
- `offset`: O número de linhas a serem mantidas antes de iniciar as iterações.
- `skip`: O número de linhas a serem removidas em cada iteração.
- `take`: O número de linhas a serem mantidas em cada iteração.

Exemplo 1

Retornar uma tabela com base na tabela que, começando na primeira linha, ignora 1 valor e depois mantém 1 valor.

```
Table.AlternateRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    }),
    1,
    1,
    1
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
3	Paul	543-7890

Table.Buffer

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Buffer(table as table) as table
```

Sobre

Armazena uma tabela em buffer na memória, isentando-a de alterações externas durante a avaliação.

Table.Column

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Column(table as table, column as text) as list
```

Sobre

Retorna a coluna de dados especificada por `column` da tabela `table` como uma lista.

Exemplo 1

Retornar os valores da coluna [Name] da tabela.

```
Table.Column(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Name"
)
```

Bob
Jim
Paul
Ringo

Table.ColumnCount

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ColumnCount(table as table) as number
```

Sobre

Retorna o número de colunas na tabela `table`.

Exemplo 1

Localizar o número de colunas na tabela.

```
Table.ColumnCount(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

`3`

Table.ColumnNames

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ColumnNames(table as table) as list
```

Sobre

Retorna os nomes de coluna na tabela `table` como uma lista de texto.

Exemplo 1

Localizar os nomes de coluna da tabela.

```
Table.ColumnNames(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    })
)
```

CustomerID
Nome
Telefone

Table.ColumnsOfType

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ColumnsOfType(table as table, listOfTypes as list) as list
```

Sobre

Retorna uma lista com os nomes das colunas na tabela `table` que correspondem aos tipos especificados no `listOfTypes`.

Exemplo 1

Retornar os nomes de colunas do tipo Number.Type da tabela.

```
Table.ColumnsOfType(
    Table.FromRecords(
        {[a = 1, b = "hello"]},
        type table[a = Number.Type, b = Text.Type]
    ),
    {type number}
)
```

um

Table.Combine

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Combine(tables as list, optional columns as any) as table
```

Sobre

Retorna uma tabela que é o resultado da mesclagem de uma lista de tabelas, `tables`. A tabela resultante terá uma estrutura de tipo de linha definida por `columns` ou por uma União dos tipos de entrada se `columns` não for especificado.

Exemplo 1

Mesclar as três tabelas.

```
Table.Combine({
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    Table.FromRecords({[CustomerID = 2, Name = "Jim", Phone = "987-6543"]}),
    Table.FromRecords({[CustomerID = 3, Name = "Paul", Phone = "543-7890"]})
})
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
2	Jim	987-6543
3	Paul	543-7890

Exemplo 2

Mescle três tabelas com estruturas diferentes.

```
Table.Combine({
    Table.FromRecords({[Name = "Bob", Phone = "123-4567"]}),
    Table.FromRecords({[Fax = "987-6543", Phone = "838-7171"]}),
    Table.FromRecords({[Cell = "543-7890"]})
})
```

NOME	TELEFONE	FAX	CÉLULA
Bob	123-4567		
	838-7171	987-6543	
			543-7890

Exemplo 3

Mescle duas tabelas e o projeto para o tipo fornecido.

```
Table.Combine(
    {
        Table.FromRecords({[Name = "Bob", Phone = "123-4567"]}),
        Table.FromRecords({[Fax = "987-6543", Phone = "838-7171"]}),
        Table.FromRecords({[Cell = "543-7890"]})
    },
    {"CustomerID", "Name"}
)
```

CUSTOMERID	NOME
	Bob

Table.CombineColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.CombineColumns(table as table, sourceColumns as list, combiner as function, column as text)
as table
```

Sobre

Combina as colunas especificadas em uma nova coluna usando a função de combinação especificada.

Table.CombineColumnsToRecord

20/10/2020 • 2 minutes to read

Sintaxe

```
Table.CombineColumnsToRecord(table as table, newColumnName as text, sourceColumns as list,
optional options as nullable record) as table
```

Sobre

Combina as colunas especificadas de `table` em uma nova coluna com valor de registro chamada `newColumnName`, na qual cada registro tem nomes e valores de campo correspondentes aos nomes e valores de coluna das colunas combinadas. Se um registro for especificado para `options`, as seguintes opções poderão ser fornecidas:

- `DisplayNameColumn`: quando especificada como texto, indica que o nome de coluna fornecido deve ser tratado como o nome de exibição do registro. Isso não precisa ser uma das colunas no próprio registro.
- `TypeName`: quando especificada como texto, fornece um nome de tipo lógico para o registro resultante que pode ser usado durante o carregamento de dados para direcionar o comportamento pelo ambiente de carregamento.

Table.ConformToPageReader

30/09/2020 • 2 minutes to read

Sintaxe

```
Table.ConformToPageReader(table as table, shapingFunction as function) as table
```

Sobre

Esta função destina-se somente a uso interno.

Table.Contains

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Contains(table as table, row as record, optional equationCriteria as any) as logical
```

Sobre

Indica se o registro especificado, `row`, é exibido como uma linha na `table`. Um parâmetro `equationCriteria` opcional pode ser especificado para controlar a comparação entre as linhas da tabela.

Exemplo 1

Determinar se a tabela contém a linha.

```
Table.Contains(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    [Name = "Bob"]
)
```

`true`

Exemplo 2

Determinar se a tabela contém a linha.

```
Table.Contains(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    [Name = "Ted"]
)
```

`false`

Exemplo 3

Determinar se a tabela contém a linha comparando apenas a coluna [Name].

```
Table.Contains(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    [CustomerID = 4, Name = "Bob"],
    "Name"
)
```

`true`

Table.ContainsAll

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ContainsAll(table as table, rows as list, optional equationCriteria as any) as logical
```

Sobre

Indica se todos os registros especificados na lista de registros `rows` aparecem como linhas na `table`. Um parâmetro `equationCriteria` opcional pode ser especificado para controlar a comparação entre as linhas da tabela.

Exemplo 1

Determinar se a tabela contém todas as linhas comparando apenas a coluna [CustomerID].

```
Table.ContainsAll(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    {
        [CustomerID = 1, Name = "Bill"],
        [CustomerID = 2, Name = "Fred"]
    },
    "CustomerID"
)
```

`true`

Exemplo 2

Determinar se a tabela contém todas as linhas.

```
Table.ContainsAll(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    {
        [CustomerID = 1, Name = "Bill"],
        [CustomerID = 2, Name = "Fred"]
    }
)
```

`false`

Table.ContainsAny

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ContainsAny(table as table, rows as list, optional equationCriteria as any) as logical
```

Sobre

Indica se algum dos registros especificados na lista de registros `rows` aparece como linhas na `table`. Um parâmetro `equationCriteria` opcional pode ser especificado para controlar a comparação entre as linhas da tabela.

Exemplo 1

Determine se a tabela `({[a = 1, b = 2], [a = 3, b = 4]})` contém as linhas `[a = 1, b = 2]` e `[a = 3, b = 5]`.

```
Table.ContainsAny(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4]
    }),
    {
        [a = 1, b = 2],
        [a = 3, b = 5]
    }
)
```

`true`

Exemplo 2

Determine se a tabela `({[a = 1, b = 2], [a = 3, b = 4]})` contém as linhas `[a = 1, b = 3]` e `[a = 3, b = 5]`.

```
Table.ContainsAny(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4]
    }),
    {
        [a = 1, b = 3],
        [a = 3, b = 5]
    }
)
```

`false`

Exemplo 3

Determine se a tabela `(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}))` contém as linhas `[a = 1, b = 3]` ou `[a = 3, b = 5]` comparando apenas a coluna [a].

```
Table.ContainsAny(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4]
    }),
    {
        [a = 1, b = 3],
        [a = 3, b = 5]
    },
    "a"
)
```

`true`

Table.DemoteHeaders

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.DemoteHeaders(table as table) as table
```

Sobre

Rebaixa os cabeçalhos de coluna (ou seja, nomes de coluna) para a primeira linha de valores. Os nomes de coluna padrão são "Coluna1", "Coluna2" e assim por diante.

Exemplo 1

Rebaixar a primeira linha de valores da tabela.

```
Table.DemoteHeaders(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"]
    })
)
```

COLUNA1	COLUNA2	COLUMN3
CustomerID	Nome	Telefone
1	Bob	123-4567
2	Jim	987-6543

Table.Distinct

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Distinct(table as table, optional equationCriteria as any) as table
```

Sobre

Remove linhas duplicadas da tabela `table`. Um parâmetro opcional, `equationCriteria`, especifica quais colunas da tabela são testadas para duplicação. Se `equationCriteria` não for especificado, todas as colunas serão testadas.

Exemplo 1

Remover as linhas duplicadas da tabela.

```
Table.Distinct(
    Table.FromRecords({
        [a = "A", b = "a"],
        [a = "B", b = "b"],
        [a = "A", b = "a"]
    })
)
```

UM	B
A	um
B	b

Exemplo 2

Remova as linhas duplicadas da coluna [b] na tabela `({[a = "A", b = "a"], [a = "B", b = "a"], [a = "A", b = "b"]})`.

```
Table.Distinct(
    Table.FromRecords({
        [a = "A", b = "a"],
        [a = "B", b = "a"],
        [a = "A", b = "b"]
    }),
    "b"
)
```

UM	B
A	um
A	b

Table.DuplicateColumn

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.DuplicateColumn(table as table, columnName as text, newColumnName as text, optional
columnType as nullable type) as table
```

Sobre

Duplique a coluna denominada `columnName` para a tabela `table`. Os valores e o tipo da coluna `newColumnName` são copiados da coluna `columnName`.

Exemplo

Duplique a coluna "a" para uma coluna denominada "copied column" na tabela `({[a = 1, b = 2], [a = 3, b = 4]})`.

```
Table.DuplicateColumn(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4]
    }),
    "a",
    "copied column"
)
```

UM	B	COLUNA COPIADA
1	2	1
3	4	3

Table.ExpandListColumn

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.ExpandListColumn(table as table, column as text) as table
```

Sobre

Dado um `table`, em que um `column` é uma lista de valores, divide a lista em uma linha para cada valor. Os valores nas outras colunas são duplicados em cada nova linha criada.

Exemplo 1

Dividir a coluna de lista [Name] na tabela.

```
Table.ExpandListColumn(
    Table.FromRecords({[Name = {"Bob", "Jim", "Paul"}, Discount = .15]}),
    "Name"
)
```

NOME	DESCONTO
Bob	0,15
Jim	0,15
Paul	0,15

Table.ExpandRecordColumn

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ExpandRecordColumn(table as table, column as text, fieldNames as list, optional
newColumnNames as nullable list) as table
```

Sobre

Devido à `column` de registros na entrada `table`, cria uma tabela com uma coluna para cada campo no registro. Opcionalmente, `newColumnNames` pode ser especificado para garantir nomes exclusivos para as colunas na nova tabela.

- `table`: A tabela original com a coluna de registro a ser expandida.
- `column`: A coluna a ser expandida.
- `fieldNames`: A lista de campos a serem expandidos em colunas na tabela.
- `newColumnNames`: A lista de nomes de coluna para dar às novas colunas. Os novos nomes de coluna não podem duplicar nenhuma coluna na nova tabela.

Exemplo 1

Expanda a coluna [a] na tabela `({[a = [aa = 1, bb = 2, cc = 3], b = 2]})` em 3 colunas "aa", "bb" e "cc".

```
Table.ExpandRecordColumn(
    Table.FromRecords({
        [
            a = [aa = 1, bb = 2, cc = 3],
            b = 2
        ]
    }),
    "a",
    {"aa", "bb", "cc"}
)
```

AA	BB	CC	B
1	2	3	2

Table.ExpandTableColumn

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.ExpandTableColumn(table as table, column as text, columnNames as list, optional
newColumnNames as nullable list) as table
```

Sobre

Expande tabelas no `table` [`column`] em várias linhas e colunas. `columnNames` é usado para selecionar as colunas a serem expandidas da tabela interna. Especifique `newColumnNames` para evitar conflitos entre colunas existentes e novas colunas.

Exemplo 1

Expanda colunas de tabela em `[a]` na tabela `({[t = {[a=1, b=2, c=3], [a=2,b=4,c=6]}, b = 2]})` em três colunas `[t.a]`, `[t.b]` e `[t.c]`.

```
Table.ExpandTableColumn(
    Table.FromRecords({
        [
            t = Table.FromRecords({
                [a = 1, b = 2, c = 3],
                [a = 2, b = 4, c = 6]
            }),
            b = 2
        ]
    }),
    "t",
    {"a", "b", "c"},
    {"t.a", "t.b", "t.c"}
)
```

T.A	T.B	T.C	B
1	2	3	2
2	4	6	2

Table.FillDown

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.FillDown(table as table, columns as list) as table
```

Sobre

Retorna uma tabela do `table` especificado em que o valor de uma célula anterior é propagado para as células de valor nulo abaixo no `columns` especificado.

Exemplo 1

Retornar uma tabela com os valores nulos da coluna [Place] preenchidos com o valor acima deles na tabela.

```
Table.FillDown(
    Table.FromRecords({
        [Place = 1, Name = "Bob"],
        [Place = null, Name = "John"],
        [Place = 2, Name = "Brad"],
        [Place = 3, Name = "Mark"],
        [Place = null, Name = "Tom"],
        [Place = null, Name = "Adam"]
    }),
    {"Place"}
)
```

LOCAL	NOME
1	Bob
1	John
2	Brad
3	Marcar
3	Tom
3	Luís

Table.FillUp

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.FillUp(table as table, columns as list) as table
```

Sobre

Retorna uma tabela do `table` especificado em que o valor da próxima célula é propagado para as células de valor nulo acima no `columns` especificado.

Exemplo 1

Retornar uma tabela com os valores nulos da coluna [Column2] preenchidos com o valor abaixo deles na tabela.

```
Table.FillUp(
    Table.FromRecords({
        [Column1 = 1, Column2 = 2],
        [Column1 = 3, Column2 = null],
        [Column1 = 5, Column2 = 3]
    }),
    {"Column2"}
)
```

COLUNA1	COLUNA2
1	2
3	3
5	3

Table.FilterWithDataTable

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FilterWithDataTable(**table** as table, **dataTableIdentifier** as text) as any
```

Sobre

Table.FilterWithDataTable

Table.FindText

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FindText(table as table, text as text) as table
```

Sobre

Retorna as linhas na tabela `table` que contêm o texto `text`. Se o texto não for encontrado, uma tabela vazia será retornada.

Exemplo 1

Localizar as linhas da tabela que contém "Bob".

```
Table.FindText(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Bob"
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Table.First

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.First(table as table, optional default as any) as any
```

Sobre

Retorna a primeira linha da `table` ou um valor padrão `default` opcional, se a tabela está vazia.

Exemplo 1

Localizar a primeira linha da tabela.

```
Table.First(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Exemplo 2

Localize a primeira linha da tabela `({})` ou retorne [a = 0, b = 0] se ela estiver vazia.

```
Table.First(Table.FromRecords({}), [a = 0, b = 0])
```

UM	0
B	0

Table.FirstN

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FirstN(table as table, countOrCondition as any) as table
```

Sobre

Retorna as primeiras linhas da tabela `table`, dependendo do valor de `countOrCondition`:

- Se `countOrCondition` for um número, serão retornadas muitas linhas (começando na parte superior).
- Se `countOrCondition` for uma condição, as linhas que atendem à condição serão retornadas até que uma linha não atenda à condição.

Exemplo 1

Localizar as duas primeiras linhas da tabela.

```
Table.FirstN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    }),
    2
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
2	Jim	987-6543

Exemplo 2

Localizar as primeiras linhas em que [a] > 0 na tabela.

```
Table.FirstN(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4],
        [a = -5, b = -6]
    }),
    each [a] > 0
)
```

UM	B
1	2

3	4

Table.FirstValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FirstValue(table as table, optional default as any) as any
```

Sobre

Retorna a primeira coluna da primeira linha da tabela `table` ou um valor padrão especificado.

Table.FromColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromColumns(lists as list, optional columns as any) as table
```

Sobre

Cria uma tabela do tipo `columns` de uma lista `lists` contendo listas aninhadas com os nomes e os valores de coluna. Se algumas colunas tiverem mais valores que outras, os valores ausentes serão preenchidos com o valor padrão, 'null', se as colunas forem anuláveis.

Exemplo 1

Retornar uma tabela de uma lista de nomes de clientes em uma lista. Cada valor no item de lista de clientes torna-se um valor de linha e cada lista torna-se uma coluna.

```
Table.FromColumns({
    {1, "Bob", "123-4567"},
    {2, "Jim", "987-6543"},
    {3, "Paul", "543-7890"}
})
```

COLUNA1	COLUNA2	COLUMN3
1	2	3
Bob	Jim	Paul
123-4567	987-6543	543-7890

Exemplo 2

Crie uma tabela com base em uma lista de colunas especificada e de uma lista de nomes de coluna.

```
Table.FromColumns(
    {
        {1, "Bob", "123-4567"},
        {2, "Jim", "987-6543"},
        {3, "Paul", "543-7890"}
    },
    {"CustomerID", "Name", "Phone"}
)
```

CUSTOMERID	NOME	TELEFONE
1	2	3

Bob	Jim	Paul
123-4567	987-6543	543-7890

Exemplo 3

Crie uma tabela com um número diferente de colunas por linha. O valor da linha ausente é nulo.

```
Table.FromColumns(
    {
        {1, 2, 3},
        {4, 5},
        {6, 7, 8, 9}
    },
    {"column1", "column2", "column3"}
)
```

COLUMN1	COLUMN2	COLUMN3
1	4	6
2	5	7
3		8
		9

Table.FromList

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromList(list as list, optional splitter as nullable function, optional columns as any,
optional default as any, optional extraValues as nullable number) as table
```

Sobre

Converte uma lista, `list`, em tabela aplicando a função de divisão opcional, `splitter`, a cada item da lista. Por padrão, a lista é considerada uma lista de valores de texto dividida por vírgulas. O `columns` opcional pode ser o número de colunas, uma lista de colunas ou um TableType. Também é possível especificar `default` e `extraValues` opcionais.

Exemplo 1

Criar uma tabela com base na lista com uma coluna chamada "Letters" usando o divisor padrão.

```
Table.FromList({"a", "b", "c", "d"}, null, {"Letters"})
```

LETRAS
um
b
c
d

Exemplo 2

Criar uma tabela com base na lista usando o divisor Record.FieldValues em que a tabela resultante tem "CustomerID" e "Name" como nomes de coluna.

```
Table.FromList(
    {
        [CustomerID = 1, Name = "Bob"],
        [CustomerID = 2, Name = "Jim"]
    },
    Record.FieldValues,
    {"CustomerID", "Name"}
)
```

CUSTOMERID	NOME
1	Bob

2	Jim

Table.FromPartitions

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromPartitions(partitionColumn as text, partitions as list, optional partitionColumnType as nullable type) as table
```

Sobre

Retorna uma tabela que é o resultado da combinação de um conjunto de tabelas particionadas, `partitions`. `partitionColumn` é o nome da coluna a ser adicionada. O tipo da coluna é padronizado para `any`, mas pode ser especificado por `partitionColumnType`.

Exemplo 1

Encontre o tipo de item na lista `{number}`.

```
Table.FromPartitions(
    "Year",
    {
        {
            1994,
            Table.FromPartitions(
                "Month",
                {
                    {
                        "Jan",
                        Table.FromPartitions(
                            "Day",
                            {
                                {1, #table({"Foo"}, {{"Bar"}})},
                                {2, #table({"Foo"}, {{"Bar"}})}
                            }
                        )
                    },
                    {
                        "Feb",
                        Table.FromPartitions(
                            "Day",
                            {
                                {3, #table({"Foo"}, {{"Bar"}})},
                                {4, #table({"Foo"}, {{"Bar"}})}
                            }
                        )
                    }
                }
            )
        }
    }
)
```

FOO	DIA	MÊS	ANO
Barras	1	Jan	1994

Barras	2	Jan	1994
Barras	3	Fev	1994
Barras	4	Fev	1994

Table.FromRecords

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromRecords(records as list, optional columns as any, optional missingField as nullable
number) as table
```

Sobre

Converte `records`, uma lista de registros, em uma tabela.

Exemplo 1

Cria uma tabela com base nos registros usando os nomes de campo dos registros como nomes de coluna.

```
Table.FromRecords({
    [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
    [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
    [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
})
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
2	Jim	987-6543
3	Paul	543-7890

Exemplo 2

Cria uma tabela com base nos registros com colunas de tipo e seleciona as colunas de número.

```
Table.ColumnsOfType(
    Table.FromRecords(
        {[CustomerID = 1, Name = "Bob"]},
        type table[CustomerID = Number.Type, Name = Text.Type]
    ),
    {type number}
)
```

CustomerID

Table.FromRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromRows(rows as list, optional columns as any) as table
```

Sobre

Cria uma tabela da lista `rows` em que cada elemento da lista é uma lista interna que contém os valores de coluna para uma única linha. Uma lista opcional de nomes de coluna, um tipo de tabela ou um número de colunas pode ser fornecida para `columns`.

Exemplo 1

Retornar uma tabela com a coluna [CustomerID] com valores {1, 2}, a coluna [Name] com valores {"Bob", "Jim"} e a coluna [Phone] com valores {"123-4567", "987-6543"}.

```
Table.FromRows(
    {
        {1, "Bob", "123-4567"},
        {2, "Jim", "987-6543"}
    },
    {"CustomerID", "Name", "Phone"}
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
2	Jim	987-6543

Exemplo 2

Retornar uma tabela com a coluna [CustomerID] com valores {1, 2}, a coluna [Name] com valores {"Bob", "Jim"} e a coluna [Phone] com valores {"123-4567", "987-6543"}, em que [CustomerID] é um tipo de número e [Name] e [Phone] são tipos de texto.

```
Table.FromRows(
    {
        {1, "Bob", "123-4567"},
        {2, "Jim", "987-6543"}
    },
    type table [CustomerID = number, Name = text, Phone = text]
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

2	Jim	987-6543

Table.FromValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.FromValue(value as any, optional options as nullable record) as table
```

Sobre

Cria uma tabela com uma coluna que contém o valor fornecido ou a lista de valores, `value`. Um parâmetro de registro opcional, `options`, pode ser especificado para controlar as seguintes opções:

- `DefaultColumnName`: O nome da coluna usada ao construir uma tabela com base em uma lista ou em um valor escalar.

Exemplo 1

Crie uma tabela a com base no valor 1.

```
Table.FromValue(1)
```

VALOR
1

Exemplo 2

Criar uma tabela com base na lista.

```
Table.FromValue({1, "Bob", "123-4567"})
```

VALOR
1
Bob
123-4567

Exemplo 3

Criar uma coluna do valor 1, com um nome de coluna personalizado.

```
Table.FromValue(1, [DefaultColumnName = "MyValue"])
```

MYVALUE

1

Table.FuzzyGroup

19/10/2020 • 4 minutes to read

Sintaxe

```
Table.FuzzyGroup(table as table, key as any, aggregatedColumns as list, optional options as
nullable record) as table
```

Sobre

Agrupa as linhas de `table` pelos valores com correspondência difusa na coluna especificada, `key`, para cada linha. Para cada grupo, é construído um registro contendo as colunas de chave (e seus valores) com as colunas agregadas especificadas por `aggregatedColumns`. Essa função não pode garantir o retorno de uma ordem fixa de linhas.

Um conjunto opcional de `options` pode ser incluído para especificar como comparar as colunas de chave. As opções incluem:

- `Culture`: permite agrupar registros com base em regras específicas da cultura. Pode ser qualquer nome de cultura válido. Por exemplo, a opção de Culture "ja-JP" agrupa registros com base no idioma japonês. O valor padrão é "", que faz o agrupamento com base na cultura inglês invariável.
- `IgnoreCase`: um valor lógico (true/false) que permite fazer o agrupamento de chaves sem diferenciar maiúsculas de minúsculas. Por exemplo, quando true, "Grapes" é agrupado com "grapes". O valor padrão é true.
- `IgnoreSpace`: um valor lógico (true/false) que permite a combinação de partes de textos a fim de localizar grupos. Por exemplo, quando true, "Gra pes" é agrupado com "Grapes". O valor padrão é true.
- `SimilarityColumnName`: um nome para a coluna que mostra a similaridade entre um valor de entrada e o valor representativo dessa entrada. O valor padrão é nulo e, nesse caso, não será adicionada uma nova coluna de similaridades.
- `Threshold`: um número entre 0.00 e 1.00 que especifica a pontuação de similaridade com a qual dois valores serão agrupados. Por exemplo, "Grapes" e "Graes" (sem o "p") serão agrupados somente se essa opção estiver configurada como inferior a 0.90. Um limite de 1.00 é o mesmo que especificar o critério de correspondência exata para o agrupamento. O valor padrão é 0.80.
- `TransformationTable`: uma tabela que permite agrupar registros com base em mapeamentos de valor personalizados. Deve conter as colunas "From" e "To". Por exemplo, "Grapes" será agrupado com "Raisins" se for fornecida uma tabela de transformação com a coluna "From" contendo "Grapes" e a coluna "To" contendo "Raisins". Observe que a transformação será aplicada a todas as ocorrências do texto na tabela de transformação. Com a tabela de transformação acima, a frase "Grapes are sweet" também será agrupada com a frase "Raisins are sweet".

Exemplo

Agrupe a tabela adicionando uma coluna de agregação [Count] que contém o número de funcionários em cada local (`each Table.RowCount(_)`).

```
Table.FuzzyGroup(
    Table.FromRecords(
        {
            [EmployeeID = 1, Location = "Seattle"],
            [EmployeeID = 2, Location = "seattl"],
            [EmployeeID = 3, Location = "Vancouver"],
            [EmployeeID = 4, Location = "Seatle"],
            [EmployeeID = 5, Location = "vancover"],
            [EmployeeID = 6, Location = "Seattle"],
            [EmployeeID = 7, Location = "Vancouver"]
        },
        type table [EmployeeID = nullable number, Location = nullable text]
    ),
    "Location",
    {"Count", each Table.RowCount(_)},
    [IgnoreCase = true, IgnoreSpace = true]
)
```

LOCATION	CONTAGEM
Seattle	4
Vancouver	3

Table.FuzzyJoin

30/09/2020 • 4 minutes to read

Sintaxe

```
Table.FuzzyJoin(table1 as table, key1 as any, table2 as table, key2 as any, optional joinKind as nullable number, optional joinOptions as nullable record) as table
```

Sobre

Une as linhas de `table1` com as linhas de `table2` com base na correspondência difusa dos valores das colunas de chave selecionadas por `key1` (para `table1`) e `key2` (para `table2`).

A correspondência difusa é uma comparação baseada na similaridade de texto, em vez de igualdade de texto.

Por padrão, uma junção interna é executada, no entanto, um `joinKind` opcional pode ser incluído para especificar o tipo de junção. As opções incluem:

- `JoinKind.Inner`
- `JoinKind.LeftOuter`
- `JoinKind.RightOuter`
- `JoinKind.FullOuter`
- `JoinKind.LeftAnti`
- `JoinKind.RightAnti`

Um conjunto opcional de `joinOptions` pode ser incluído para especificar como comparar as colunas de chave. As opções incluem:

- `ConcurrentRequests`: um número entre 1 e 8 que especifica o número de threads paralelos a serem usados para correspondência difusa. O valor padrão é 1.
- `Culture`: permite combinar registros com base em regras específicas da cultura. Pode ser qualquer nome de cultura válido. Por exemplo, a opção de Culture "ja-JP" faz a correspondência com registros com base no idioma japonês. O valor padrão é "", que faz a correspondência com base na cultura inglês invariável.
- `IgnoreCase`: um valor lógico (true/false) que permite fazer a correspondência de chaves sem diferenciar maiúsculas de minúsculas. Por exemplo, quando true, "Grapes" corresponde a "grapes". O valor padrão é true.
- `IgnoreSpace`: um valor lógico (true/false) que permite a combinação de partes de textos a fim de localizar correspondências. Por exemplo, quando true, "Gra pes" corresponde a "Grapes". O valor padrão é true.
- `NumberOfMatches`: um número inteiro que especifica o número máximo de linhas correspondentes que podem ser retornadas. Por exemplo, um valor de 1 retornará no máximo uma linha correspondente para cada linha de entrada. Se essa opção não for informada, todas as linhas correspondentes serão retornadas.
- `Threshold`: um número entre 0.0 e 1.0 que especifica a pontuação de similaridade com a qual dois valores serão correspondidos. Por exemplo, "Grapes" e "Graes" (sem o "p") serão correspondentes somente se essa opção estiver configurada como inferior a 0.90. Um limite de 1,00 é o mesmo que especificar um critério de correspondência exato. O valor padrão é 0.80.
- `TransformationTable`: uma tabela que permite fazer a correspondência de registros com base em mapeamentos de valor personalizados. Deve conter as colunas "From" e "To". Por exemplo, "Grapes" corresponderá a "Raisins" se for fornecida uma tabela de transformação com a coluna "From" contendo "Grapes" e a coluna "To" contendo "Raisins". Observe que a transformação será aplicada a todas as ocorrências do texto na tabela de

transformação. Com a tabela de transformação acima, a frase "Grapes are sweet" também corresponderá à frase "Raisins are sweet".

Exemplo

Junção difusa interna esquerda das duas tabelas com base em [FirstName]

```
Table.FuzzyJoin(
      Table.FromRecords(
        {
                [CustomerID = 1, FirstName1 = "Bob", Phone = "555-1234"],
                [CustomerID = 2, FirstName1 = "Robert", Phone = "555-4567"]
          },
        type table [CustomerID = nullable number, FirstName1 = nullable text, Phone = nullable text]
    ),
      {"FirstName1"},
      Table.FromRecords(
        {
                [CustomerStateID = 1, FirstName2 = "Bob", State = "TX"],
                [CustomerStateID = 2, FirstName2 = "bOB", State = "CA"]
          },
        type table [CustomerStateID = nullable number, FirstName2 = nullable text, State = nullable text]
    ),
      {"FirstName2"},
      JoinKind.LeftOuter,
      [IgnoreCase = true, IgnoreSpace = false]
)
```

CUSTOMERID	FIRSTNAME1	TELEFONE	CUSTOMERSTATEID	FIRSTNAME2	ESTADO
1	Roberto	555-1234	1	Roberto	TX
1	Roberto	555-1234	2	bOB	AC
2	Robert	555-4567			

Table.FuzzyNestedJoin

30/09/2020 • 4 minutes to read

Sintaxe

```
Table.FuzzyNestedJoin(table1 as table, key1 as any, table2 as table, key2 as any, newColumnName as text, optional joinKind as nullable number, optional joinOptions as nullable record) as table
```

Sobre

Une as linhas de `table1` com as linhas de `table2` com base na correspondência difusa dos valores das colunas de chave selecionadas por `key1` (para `table1`) e `key2` (para `table2`). Os resultados são retornados em uma nova coluna chamada `newColumnName`.

A correspondência difusa é uma comparação baseada na similaridade de texto, em vez de igualdade de texto.

O `joinKind` opcional especifica o tipo de junção a ser executada. Por padrão, uma junção externa esquerda será executada se uma `joinKind` não for especificada. As opções incluem:

- `JoinKind.Inner`
- `JoinKind.LeftOuter`
- `JoinKind.RightOuter`
- `JoinKind.FullOuter`
- `JoinKind.LeftAnti`
- `JoinKind.RightAnti`

Um conjunto opcional de `joinOptions` pode ser incluído para especificar como comparar as colunas de chave. As opções incluem:

- `ConcurrentRequests`: um número entre 1 e 8 que especifica o número de threads paralelos a serem usados para correspondência difusa. O valor padrão é 1.
- `Culture`: permite combinar registros com base em regras específicas da cultura. Pode ser qualquer nome de cultura válido. Por exemplo, a opção de Culture "ja-JP" faz a correspondência com registros com base no idioma japonês. O valor padrão é "", que faz a correspondência com base na cultura inglês invariável.
- `IgnoreCase`: um valor lógico (true/false) que permite fazer a correspondência de chaves sem diferenciar maiúsculas de minúsculas. Por exemplo, quando true, "Grapes" corresponde a "grapes". O valor padrão é true.
- `IgnoreSpace`: um valor lógico (true/false) que permite a combinação de partes de textos a fim de localizar correspondências. Por exemplo, quando true, "Gra pes" corresponde a "Grapes". O valor padrão é true.
- `NumberOfMatches`: um número inteiro que especifica o número máximo de linhas correspondentes que podem ser retornadas. Por exemplo, um valor de 1 retornará no máximo uma linha correspondente para cada linha de entrada. Se essa opção não for informada, todas as linhas correspondentes serão retornadas.
- `Threshold`: um número entre 0.0 e 1.0 que especifica a pontuação de similaridade com a qual dois valores serão correspondidos. Por exemplo, "Grapes" e "Graes" (sem o "p") serão correspondentes somente se essa opção estiver configurada como inferior a 0.90. Um limite de 1,00 é o mesmo que especificar um critério de correspondência exato. O valor padrão é 0.80.
- `TransformationTable`: uma tabela que permite fazer a correspondência de registros com base em mapeamentos de valor personalizados. Deve conter as colunas "From" e "To". Por exemplo, "Grapes" corresponderá a "Raisins" se for fornecida uma tabela de transformação com a coluna "From" contendo "Grapes" e a coluna "To" contendo

"Raisins". Observe que a transformação será aplicada a todas as ocorrências do texto na tabela de transformação. Com a tabela de transformação acima, a frase "Grapes are sweet" também corresponderá à frase "Raisins are sweet".

Exemplo

Junção difusa interna esquerda das duas tabelas com base em [FirstName]

```
Table.FuzzyNestedJoin(
      Table.FromRecords(
        {
                [CustomerID = 1, FirstName1 = "Bob", Phone = "555-1234"],
                [CustomerID = 2, FirstName1 = "Robert", Phone = "555-4567"]
          },
        type table [CustomerID = nullable number, FirstName1 = nullable text, Phone = nullable text]
    ),
      {"FirstName1"},
      Table.FromRecords(
        {
                [CustomerStateID = 1, FirstName2 = "Bob", State = "TX"],
                [CustomerStateID = 2, FirstName2 = "bOB", State = "CA"]
          },
        type table [CustomerStateID = nullable number, FirstName2 = nullable text, State = nullable text]
    ),
      {"FirstName2"},
      "NestedTable",
      JoinKind.LeftOuter,
      [IgnoreCase = true, IgnoreSpace = false]
)
```

CUSTOMERID	FIRSTNAME1	TELEFONE	NESTEDTABLE
1	Roberto	555-1234	[Tabela]
2	Robert	555-4567	[Tabela]

Table.Group

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Group(table as table, key as any, aggregatedColumns as list, optional groupKind as nullable number, optional comparer as nullable function) as table
```

Sobre

Agrupa as linhas de `table` pelos valores na coluna especificada, `key`, para cada linha. Para cada grupo, é construído um registro contendo as colunas de chave (e seus valores) com as colunas agregadas especificadas por `aggregatedColumns`. Observe que, se várias chaves corresponderem ao comparador, chaves diferentes poderão ser retornadas. Essa função não pode garantir o retorno de uma ordem fixa de linhas. Opcionalmente, também é possível especificar `groupKind` e `comparer`.

Exemplo 1

Agrupar a tabela adicionando uma coluna agregada [total] que contém a soma de preços ("each List.Sum([price])").

```
Table.Group(
    Table.FromRecords({
        [CustomerID = 1, price = 20],
        [CustomerID = 2, price = 10],
        [CustomerID = 2, price = 20],
        [CustomerID = 1, price = 10],
        [CustomerID = 3, price = 20],
        [CustomerID = 3, price = 5]
    }),
    "CustomerID",
    {"total", each List.Sum([price])}
)
```

CUSTOMERID	TOTAL
1	30
2	30
3	25

Table.HasColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.HasColumns(table as table, columns as any) as logical
```

Sobre

indica se o `table` contém as colunas especificadas, `columns`. Retorna `true` se a tabela contém as colunas; caso contrário, `false`.

Exemplo 1

Determine se a tabela tem a coluna [Name].

```
TTable.HasColumns(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Name"
)
```

`true`

Exemplo 2

Descubra se a tabela tem a coluna [Name] e [PhoneNumber].

```
Table.HasColumns(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    {"Name", "PhoneNumber"}
)
```

`false`

Table.InsertRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.InsertRows(table as table, offset as number, rows as list) as table
```

Sobre

Retorna uma tabela com a lista de linhas, `rows`, inserida no `table` na posição especificada, `offset`. Cada coluna na linha a ser inserida deve corresponder aos tipos de coluna da tabela.

Exemplo 1

Inserir a linha na tabela na posição 1.

```
Table.InsertRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"]
    }),
    1,
    {[CustomerID = 3, Name = "Paul", Phone = "543-7890"]}
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
3	Paul	543-7890
2	Jim	987-6543

Exemplo 2

Inserir duas linhas na tabela na posição 1.

```
Table.InsertRows(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    1,
    {
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    }
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

2	Jim	987-6543
3	Paul	543-7890

Table.IsDistinct

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.IsDistinct(table as table, optional comparisonCriteria as any) as logical
```

Sobre

Indica se a `table` contém somente linhas distintas (sem duplicatas). Retorna `true` se as linhas são distintas; caso contrário, `false`. Um parâmetro opcional, `comparisonCriteria`, especifica quais colunas da tabela são testadas para duplicação. Se `comparisonCriteria` não for especificado, todas as colunas serão testadas.

Exemplo 1

Determine se a tabela é distinta.

```
Table.IsDistinct(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    })
)
```

`true`

Exemplo 2

Determine se a tabela é distinta na coluna.

```
Table.IsDistinct(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 5, Name = "Bob", Phone = "232-1550"]
    }),
    "Name"
)
```

`false`

Table.IsEmpty

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.IsEmpty(table as table) as logical
```

Sobre

Indica se `table` contém linhas. Retorna `true` se não há linhas (ou seja, a tabela está vazia); caso contrário, `false`.

Exemplo 1

Determine se a tabela está vazia.

```
Table.IsEmpty(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

`false`

Exemplo 2

Determine se a tabela `({})` está vazia.

```
Table.IsEmpty(Table.FromRecords({}))
```

`true`

Table.Join

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Join(table1 as table, key1 as any, table2 as table, key2 as any, optional joinKind as nullable number, optional joinAlgorithm as nullable number, optional keyEqualityComparers as nullable list) as table
```

Sobre

Une as linhas de `table1` com as linhas de `table2` com base na igualdade dos valores das colunas de chave selecionadas por `key1` (para `table1`) e `key2` (para `table2`).

Por padrão, uma junção interna é executada, no entanto, um `joinKind` opcional pode ser incluído para especificar o tipo de junção. As opções incluem:

- `JoinKind.Inner`
- `JoinKind.LeftOuter`
- `JoinKind.RightOuter`
- `JoinKind.FullOuter`
- `JoinKind.LeftAnti`
- `JoinKind.RightAnti`

Um conjunto opcional de `keyEqualityComparers` pode ser incluído para especificar como comparar as colunas de chave.

Exemplo 1

Junção interna das duas tabelas em [CustomerID]

```
Table.Join(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "CustomerID",
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
        [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
        [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
        [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
        [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25]
    }),
    "CustomerID"
)
```

CUSTOMERID	NOME	TELEFONE	ORDERID	ITEM	PREÇO

1	Bob	123-4567	1	Vara de pescar	100
1	Bob	123-4567	2	1 lb de minhocas	5
2	Jim	987-6543	3	Rede de pesca	25
3	Paul	543-7890	4	Taser de peixe	200
3	Paul	543-7890	5	Bandaids	2
1	Bob	123-4567	6	Caixa de material de pesca	20

Table.Keys

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Keys(table as table) as list
```

Sobre

Table.Keys

Table.Last

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Last(table as table, optional default as any) as any
```

Sobre

Retorna a última linha da `table` ou um valor padrão `default` opcional, se a tabela está vazia.

Exemplo 1

Localize a última linha da tabela.

```
Table.Last(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

CUSTOMERID	3
NOME	Paul
TELEFONE	543-7890

Exemplo 2

Localize a última linha da tabela `({})` ou retorne [a = 0, b = 0] se ela estiver vazia.

```
Table.Last(Table.FromRecords({}), [a = 0, b = 0])
```

UM	0
B	0

Table.LastN

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.LastN(table as table, countOrCondition as any) as table
```

Sobre

Retorna as últimas linhas da tabela, `table`, dependendo do valor de `countOrCondition`:

- Se `countOrCondition` for um número, essa quantidade de linhas será retornada começando na posição (final - `countOrCondition`).
- Se `countOrCondition` for uma condição, as linhas que atendem à condição serão retornadas na posição crescente até que uma linha não atenda à condição.

Exemplo 1

Localizar as duas últimas linhas da tabela.

```
Table.LastN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    }),
    2
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890

Exemplo 2

Localizar as últimas linhas em que [a] > 0 na tabela.

```
Table.LastN(
    Table.FromRecords({
        [a = -1, b = -2],
        [a = 3, b = 4],
        [a = 5, b = 6]
    }),
    each _ [a] > 0
```

UM	B

3	4
5	6

Table.MatchesAllRows

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.MatchesAllRows(table as table, condition as function) as logical
```

Sobre

Indica se todas as linhas na `table` atendem à `condition` especificada. Retorna `true` se todas as linhas correspondem; caso contrário, `false`.

Exemplo 1

Determinar se todos os valores de linha na coluna [a] são pares na tabela.

```
Table.MatchesAllRows(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 6, b = 8]
    }),
    each Number.Mod([a], 2) = 0
)
```

`true`

Exemplo 2

Localize se todos os valores de linha são [a = 1, b = 2], na tabela `({[a = 1, b = 2], [a = 3, b = 4]})`.

```
Table.MatchesAllRows(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = -3, b = 4]
    }),
    each _ = [a = 1, b = 2]
)
```

`false`

Table.MatchesAnyRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.MatchesAnyRows(table as table, condition as function) as logical
```

Sobre

Indica se alguma das linhas na `table` atende à `condition` especificada. Retorna `true` se alguma das linhas corresponde; caso contrário, `false`.

Exemplo 1

Determinar se algum dos valores de linha na coluna [a] são pares na tabela `({[a = 2, b = 4], [a = 6, b = 8]})`.

```
Table.MatchesAnyRows(
    Table.FromRecords({
        [a = 1, b = 4],
        [a = 3, b = 8]
    }),
    each Number.Mod([a], 2) = 0
)
```

`false`

Exemplo 2

Determinar se algum dos valores de linha são [a = 1, b = 2] na tabela `({[a = 1, b = 2], [a = 3, b = 4]})`.

```
Table.MatchesAnyRows(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = -3, b = 4]
    }),
    each _ = [a = 1, b = 2]
)
```

`true`

Table.Max

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Max(table as table, comparisonCriteria as any, optional default as any) as any
```

Sobre

Retorna a maior linha do `table`, dado o `comparisonCriteria`. Se a tabela estiver vazia, o valor opcional de `default` será retornado.

Exemplo 1

Localize a linha com o maior valor na coluna [a] da tabela `({[a = 2, b = 4], [a = 6, b = 8]})`.

```
Table.Max(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 6, b = 8]
    }),
    "a"
)
```

UM	6
B	8

Exemplo 2

Localize a linha com o maior valor na coluna [a] da tabela `({})`. Retornará -1 se estiver vazia.

```
Table.Max(#table({"a"}, {}), "a", -1)
```

`-1`

Table.MaxN

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.MaxN(table as table, comparisonCriteria as any, countOrCondition as any) as table
```

Sobre

Retorna a maior linha do `table`, dado o `comparisonCriteria`. Depois que as linhas são classificadas, o parâmetro `countOrCondition` deve ser especificado para filtrar melhor o resultado. Observe que o algoritmo de classificação não pode garantir um resultado de classificação fixo. O parâmetro `countOrCondition` pode assumir várias formas:

- Se um número for especificado, uma lista de até `countOrCondition` itens em ordem crescente será retornada.
- Se uma condição for especificada, uma lista de itens que atendam inicialmente à condição será retornada. Quando um item falha na condição, nenhum outro item é considerado.

Exemplo 1

Localizar a linha com o maior valor na coluna [a] com a condição [a] > 0 na tabela. As linhas são classificadas antes da aplicação do filtro.

```
Table.MaxN(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 0, b = 0],
        [a = 6, b = 2]
    }),
    "a",
    each [a] > 0
)
```

UM	B
6	2
2	4

Exemplo 2

Localizar a linha com o maior valor na coluna [a] com a condição [b] > 0 na tabela. As linhas são classificadas antes da aplicação do filtro.

```
Table.MaxN(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 8, b = 0],
        [a = 6, b = 2]
    }),
    "a",
    each [b] > 0
)
```

Table.Min

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Min(table as table, comparisonCriteria as any, optional default as any) as any
```

Sobre

Retorna a menor linha do `table`, dado o `comparisonCriteria`. Se a tabela estiver vazia, o valor opcional de `default` será retornado.

Exemplo 1

Localizar a linha com o menor valor na coluna [a] da tabela.

```
Table.Min(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 6, b = 8]
    }),
    "a"
)
```

UM	2
B	4

Exemplo 2

Localizar a linha com o menor valor na coluna [a] da tabela. Retornará -1 se estiver vazia.

```
Table.Min(#table({"a"}, {}), "a", -1)
```

`-1`

Table.MinN

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.MinN(table as table, comparisonCriteria as any, countOrCondition as any) as table
```

Sobre

Retorna as menores linhas do `table`, dado o `comparisonCriteria`. Depois que as linhas são classificadas, o parâmetro `countOrCondition` deve ser especificado para filtrar melhor o resultado. Observe que o algoritmo de classificação não pode garantir um resultado de classificação fixo. O parâmetro `countOrCondition` pode assumir várias formas:

- Se um número for especificado, uma lista de até `countOrCondition` itens em ordem crescente será retornada.
- Se uma condição for especificada, uma lista de itens que atendam inicialmente à condição será retornada. Quando um item falha na condição, nenhum outro item é considerado.

Exemplo 1

Localizar a linha com o menor valor na coluna [a] com a condição [a] < 3 na tabela. As linhas são classificadas antes da aplicação do filtro.

```
Table.MinN(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 0, b = 0],
        [a = 6, b = 4]
    }),
    "a",
    each [a] < 3
)
```

UM	B
0	0
2	4

Exemplo 2

Localizar a linha com o menor valor na coluna [a] com a condição [a] < 0 da tabela. As linhas são classificadas antes da aplicação do filtro.

```
Table.MinN(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 8, b = 0],
        [a = 6, b = 2]
    }),
    "a",
    each [b] < 0
)
```

Table.NestedJoin

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.NestedJoin(table1 as table, key1 as any, table2 as any, key2 as any, newColumnName as text,
optional joinKind as nullable number, optional keyEqualityComparers as nullable list) as table
```

Sobre

Une as linhas de `table1` com as linhas de `table2` com base na igualdade dos valores das colunas de chave selecionadas por `key1` (para `table1`) e `key2` (para `table2`). Os resultados são inseridos na coluna chamada `newColumnName`.

O `joinKind` opcional especifica o tipo de junção a ser executada. Por padrão, uma junção externa esquerda será executada se uma `joinKind` não for especificada.

Um conjunto opcional de `keyEqualityComparers` pode ser incluído para especificar como comparar as colunas de chave.

Table.Partition

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Partition(table as table, column as text, groups as number, hash as function) as list
```

Sobre

Particiona `table` em uma lista de números `groups` de tabelas, com base no valor da `column` e uma função `hash`. A função `hash` é aplicada ao valor da linha `column` para obter um valor de hash para a linha. O módulo de valor de hash `groups` determina em qual das tabelas retornadas a linha será colocada.

- `table`: a tabela a ser particionada.
- `column`: a coluna cujo hash será efetuado para determinar em qual tabela retornada a linha está.
- `groups`: o número de tabelas nas quais a tabela de entrada será particionada.
- `hash`: a função aplicada para obter um valor de hash.

Exemplo

Particione a tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})` em duas tabelas na coluna [a], usando o valor das colunas como função de hash.

```
Table.Partition(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 1, b = 4],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    "a",
    2,
    each _
)
```

```
{
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 2, b = 4]
    }),
    Table.FromRecords({
        [a = 1, b = 4],
        [a = 1, b = 4]
    })
}
```

Table.PartitionValues

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Partition(table as table, column as text, groups as number, hash as function) as list
```

Sobre

Particiona `table` em uma lista de números `groups` de tabelas, com base no valor da `column` e uma função `hash`. A função `hash` é aplicada ao valor da linha `column` para obter um valor de hash para a linha. O módulo de valor de hash `groups` determina em qual das tabelas retornadas a linha será colocada.

- `table`: a tabela a ser particionada.
- `column`: a coluna cujo hash será efetuado para determinar em qual tabela retornada a linha está.
- `groups`: o número de tabelas nas quais a tabela de entrada será particionada.
- `hash`: a função aplicada para obter um valor de hash.

Exemplo 1

Particione a tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})` em duas tabelas na coluna [a], usando o valor das colunas como função de hash.

```
Table.Partition(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), "a", 2,
each _)
```

[Tabela]
[Tabela]

Table.Pivot

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Pivot(table as table, pivotValues as list, attributeColumn as text, valueColumn as text,
optional aggregationFunction as nullable function) as table
```

Sobre

Dado um par de colunas que representam pares de atributo/valor, gira os dados na coluna de atributo para um título de coluna.

Exemplo 1

Pegue os valores "a", "b" e "c" na coluna de atributo da tabela

```
({ [ key = "x", attribute = "a", value = 1 ], [ key = "x", attribute = "c", value = 3 ], [ key = "y", attribute
= "a", value = 2 ], [ key = "y", attribute = "b", value = 4 ] })
```

e dinamize-os em sua própria coluna.

```
Table.Pivot(
    Table.FromRecords({
        [key = "x", attribute = "a", value = 1],
        [key = "x", attribute = "c", value = 3],
        [key = "y", attribute = "a", value = 2],
        [key = "y", attribute = "b", value = 4]
    }),
    {"a", "b", "c"},
    "attribute",
    "value"
)
```

CHAVE	UM	B	C
x	1		3
a	2	4	

Exemplo 2

Pegue os valores "a", "b" e "c" na coluna de atributo da tabela

```
({ [ key = "x", attribute = "a", value = 1 ], [ key = "x", attribute = "c", value = 3 ], [ key = "x", attribute
= "c", value = 5 ], [ key = "y", attribute = "a", value = 2 ], [ key = "y", attribute = "b", value = 4 ] })
```

e dinamize-os em sua própria coluna. O atributo "c" para a chave "x" tem vários valores associados a ele. Portanto, use a função List.Max para resolver o conflito.

```
Table.Pivot(
    Table.FromRecords({
        [key = "x", attribute = "a", value = 1],
        [key = "x", attribute = "c", value = 3],
        [key = "x", attribute = "c", value = 5],
        [key = "y", attribute = "a", value = 2],
        [key = "y", attribute = "b", value = 4]
    }),
    {"a", "b", "c"},
    "attribute",
    "value",
    List.Max
)
```

CHAVE	UM	B	C
x	1		5
a	2	4	

Table.PositionOf

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.PositionOf(table as table, row as record, optional occurrence as any, optional
equationCriteria as any) as any
```

Sobre

Retorna a posição da linha da primeira ocorrência do `row` no `table` especificado. Retornará -1 se nenhuma ocorrência for encontrada.

- `table`: A tabela de entrada.
- `row`: A linha na tabela cuja posição será localizada.
- `occurrence`: *[Opcional]* Especifica as ocorrências da linha a retornar.
- `equationCriteria`: *[Opcional]* Controla a comparação entre as linhas da tabela.

Exemplo 1

Localize a posição da primeira ocorrência de [a = 2, b = 4] na tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})`.

```
Table.PositionOf(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 1, b = 4],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    [a = 2, b = 4]
)
```

`0`

Exemplo 2

Localize a posição da segunda ocorrência de [a = 2, b = 4] na tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})`.

```
Table.PositionOf(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 1, b = 4],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    [a = 2, b = 4],
    1
)
```

2

Exemplo 3

Localize a posição de todas as ocorrências de [a = 2, b = 4] na tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})`.

```
Table.PositionOf(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 1, b = 4],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    [a = 2, b = 4],
    Occurrence.All
)
```

0
2

Table.PositionOfAny

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.PositionOfAny(table as table, rows as list, optional occurrence as nullable number, optional equationCriteria as any) as any
```

Sobre

Retorna as posições das linhas da `table` da primeira ocorrência da lista de `rows`. Retornará -1 se nenhuma ocorrência for encontrada.

- `table`: A tabela de entrada.
- `rows`: A lista de linhas na tabela cujas posições deverão ser encontradas.
- `occurrence`: *[Opcional]* Especifica as ocorrências da linha a retornar.
- `equationCriteria`: *[Opcional]* Controla a comparação entre as linhas da tabela.

Exemplo 1

Localize a posição da primeira ocorrência de [a = 2, b = 4] ou [a = 6, b = 8] na tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]})`.

```
Table.PositionOfAny(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 1, b = 4],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    {
        [a = 2, b = 4],
        [a = 6, b = 8]
    }
)
```

`0`

Exemplo 2

Localize a posição de todas as ocorrências de [a = 2, b = 4] ou [a = 6, b = 8] na tabela `({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}`.

```
Table.PositionOfAny(
    Table.FromRecords({
        [a = 2, b = 4],
        [a = 6, b = 8],
        [a = 2, b = 4],
        [a = 1, b = 4]
    }),
    {
        [a = 2, b = 4],
        [a = 6, b = 8]
    },
    Occurrence.All
)
```

0
1
2

Table.PrefixColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.PrefixColumns(table as table, prefix as text) as table
```

Sobre

Retorna uma tabela em que todos os nomes de coluna da `table` fornecida são prefixados com o texto fornecido, `prefix`, mais um ponto no formulário `prefix` `.ColumnName`.

Exemplo 1

Prefixe as colunas com "MyTable" na tabela.

```
Table.PrefixColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    "MyTable"
)
```

MYTABLE.CUSTOMERID	MYTABLE.NAME	MYTABLE.PHONE
1	Bob	123-4567

Table.Profile

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Profile(table as table, optional additionalAggregates as nullable list) as table
```

Sobre

Retorna um perfil para as colunas em `table`.

As informações a seguir são retornadas para cada coluna (quando aplicável):

- mínimo
- máximo
- média
- desvio padrão
- contagem
- contagem de nulos
- contagem distinta

Table.PromoteHeaders

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.PromoteHeaders(table as table, optional options as nullable record) as table
```

Sobre

Promove a primeira linha de valores como novos cabeçalhos de coluna (por exemplo, nomes de coluna). Por padrão, apenas valores de texto ou número são promovidos para cabeçalhos. Opções válidas:

`PromoteAllScalars`: Se definido como `true`, todos os valores escalares na primeira linha serão promovidos para os cabeçalhos usando a `Culture`, se especificado (ou a localidade do documento atual). Para valores que não podem ser convertidos em texto, será usado um nome da coluna padrão.
`Culture`: Um nome de cultura especificando a cultura dos dados.

Exemplo 1

Promover a primeira linha de valores na tabela.

```
Table.PromoteHeaders(
    Table.FromRecords({
        [Column1 = "CustomerID", Column2 = "Name", Column3 = #date(1980, 1, 1)],
        [Column1 = 1, Column2 = "Bob", Column3 = #date(1980, 1, 1)]
    })
)
```

CUSTOMERID	NOME	COLUMN3
1	Bob	1/1/1980 12:00:00 AM

Exemplo 2

Promova todos os escalares na primeira linha da tabela a cabeçalhos.

```
Table.PromoteHeaders(
    Table.FromRecords({
        [Rank = 1, Name = "Name", Date = #date(1980, 1, 1)],
        [Rank = 1, Name = "Bob", Date = #date(1980, 1, 1)]}
    ),
    [PromoteAllScalars = true, Culture = "en-US"]
)
```

1	NOME	1/1/1980
1	Bob	1/1/1980 12:00:00 AM

Table.Range

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Range(table as table, offset as number, optional count as nullable number) as table
```

Sobre

Retorna as linhas da `table` começando no `offset` especificado. Um parâmetro opcional, `count`, especifica quantas linhas retornar. Por padrão, todas as linhas após o deslocamento são retornadas.

Exemplo 1

Retornar todas as linhas que começam no deslocamento 1 da tabela.

```
Table.Range(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 2

Retornar uma linha que começa no deslocamento 1 da tabela.

```
Table.Range(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1,
    1
)
```

CUSTOMERID	NOME	TELEFONE

2	Jim	987-6543

Table.RemoveColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveColumns(table as table, columns as any, optional missingField as nullable number) as table
```

Sobre

Remove o `columns` especificado do `table` fornecido. Se a coluna não existir, uma exceção será gerada, a menos que o parâmetro opcional `missingField` especifique uma alternativa (por exemplo, `MissingField.UseNull` ou `MissingField.Ignore`).

Exemplo 1

Remova a coluna [Phone] da tabela.

```
Table.RemoveColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    "Phone"
)
```

CUSTOMERID	NOME
1	Bob

Exemplo 2

Remova a coluna [Address] da tabela. Gera um erro se ele não existe.

```
Table.RemoveColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    "Address"
)
```

```
[Expression.Error] The field 'Address' of the record was not found.
```

Table.RemoveFirstN

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveFirstN(table as table, optional countOrCondition as any) as table
```

Sobre

Retorna uma tabela que não contém o primeiro número especificado de linhas, `countOrCondition`, da tabela `table`. O número de linhas removidas depende do parâmetro opcional `countOrCondition`.

- Se `countOrCondition` for omitido, somente a primeira linha será removida.
- Se `countOrCondition` for um número, serão removidas muitas linhas (começando na parte superior).
- Se `countOrCondition` for uma condição, as linhas que atendem à condição serão removidas até que uma linha não atenda à condição.

Exemplo 1

Remover a primeira linha da tabela.

```
Table.RemoveFirstN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 2

Remover as duas primeiras linhas da tabela.

```
Table.RemoveFirstN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    2
)
```

CUSTOMERID	NOME	TELEFONE
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 3

Remover da tabela as primeiras linhas em que [CustomerID] <=2.

```
Table.RemoveFirstN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    each [CustomerID] <= 2
)
```

CUSTOMERID	NOME	TELEFONE
3	Paul	543-7890
4	Ringo	232-1550

Table.RemoveLastN

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveLastN(table as table, optional countOrCondition as any) as table
```

Sobre

Retorna uma tabela que não contém as `countOrCondition` últimas linhas da tabela `table`. O número de linhas removidas depende do parâmetro opcional `countOrCondition`.

- Se `countOrCondition` for omitido, somente a última linha será removida.
- Se `countOrCondition` for um número, essa quantidade de linhas (começando na parte inferior) será removida.
- Se `countOrCondition` for uma condição, as linhas que atendem à condição serão removidas até que uma linha não atenda à condição.

Exemplo 1

Remover a última linha da tabela.

```
Table.RemoveLastN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
2	Jim	987-6543
3	Paul	543-7890

Exemplo 2

Remover da tabela as últimas linhas em que [CustomerID] > 2.

```
Table.RemoveLastN(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    each [CustomerID] >= 2
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Table.RemoveMatchingRows

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveMatchingRows(table as table, rows as list, optional equationCriteria as any) as table
```

Sobre

Remove todas as ocorrências das `rows` especificadas da `table`. Um parâmetro `equationCriteria` opcional pode ser especificado para controlar a comparação entre as linhas da tabela.

Exemplo 1

Remova todas as linhas em que [a = 1] da tabela `({[a = 1, b = 2], [a = 3, b = 4], [a = 1, b = 6]})`.

```
Table.RemoveMatchingRows(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4],
        [a = 1, b = 6]
    }),
    {[a = 1]},
    "a"
)
```

UM	B
3	4

Table.RemoveRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveRows(table as table, offset as number, optional count as nullable number) as table
```

Sobre

Remove `count` de linhas do início do `table`, começando pelo `offset` especificado. Uma contagem padrão de 1 será usada se o parâmetro `count` não for fornecido.

Exemplo 1

Remover a primeira linha da tabela.

```
Table.RemoveRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    0
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 2

Remover a linha na posição 1 da tabela.

```
Table.RemoveRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1
)
```

CUSTOMERID	NOME	TELEFONE

1	Bob	123-4567
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 3

Remover duas linhas começando na posição 1 da tabela.

```
Table.RemoveRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1,
    2
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567
4	Ringo	232-1550

Table.RemoveRowsWithErrors

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.RemoveRowsWithErrors(table as table, optional columns as nullable list) as table
```

Sobre

Retorna uma tabela com as linhas removidas da tabela de entrada que contêm um erro em pelo menos uma das células. Se uma lista de colunas for especificada, somente as células nas colunas especificadas serão inspecionadas quanto a erros.

Exemplo 1

Remove um valor de erro da primeira linha.

```
Table.RemoveRowsWithErrors(
    Table.FromRecords({
        [Column1 = ...],
        [Column1 = 2],
        [Column1 = 3]
    })
)
```

COLUNA1
2
3

Table.RenameColumns

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.RenameColumns(table as table, renames as list, optional missingField as nullable number) as table
```

Sobre

Executa os renomeações fornecidas para as colunas na tabela `table`. Uma operação de substituição `renames` consiste em uma lista de dois valores, o nome de coluna antigo e o novo, fornecidos em uma lista. Se a coluna não existir, uma exceção será gerada, a menos que o parâmetro opcional `missingField` especifique uma alternativa (por exemplo, `MissingField.UseNull` ou `MissingField.Ignore`).

Exemplo 1

Substituir o nome de coluna "CustomerNum" por "CustomerID" na tabela.

```
Table.RenameColumns(
    Table.FromRecords({[CustomerNum = 1, Name = "Bob", Phone = "123-4567"]}),
    {"CustomerNum", "CustomerID"}
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Exemplo 2

Substituir o nome de coluna "CustomerNum" por "CustomerID" e "PhoneNum" por "Phone" na tabela.

```
Table.RenameColumns(
    Table.FromRecords({[CustomerNum = 1, Name = "Bob", PhoneNum = "123-4567"]}),
    {
        {"CustomerNum", "CustomerID"},
        {"PhoneNum", "Phone"}
    }
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Exemplo 3

Substituir o nome de coluna "NewCol" por "NewColumn" na tabela e ignorar se a coluna não existir.

```
Table.RenameColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    {"NewCol", "NewColumn"},
    MissingField.Ignore
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Table.ReorderColumns

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReorderColumns(table as table, columnOrder as list, optional missingField as nullable
number) as table
```

Sobre

Retorna uma tabela da entrada `table`, com as colunas na ordem especificada por `columnOrder`. As colunas que não forem especificadas na lista não serão reordenadas. Se a coluna não existir, uma exceção será gerada, a menos que o parâmetro opcional `missingField` especifique uma alternativa (por exemplo, `MissingField.UseNull` ou `MissingField.Ignore`).

Exemplo 1

Alternar a ordem das colunas [Phone] e [Name] na tabela.

```
Table.ReorderColumns(
    Table.FromRecords({[CustomerID = 1, Phone = "123-4567", Name = "Bob"]}),
    {"Name", "Phone"}
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Exemplo 2

Alterne a ordem das colunas [Phone] e [Address] ou use "MissingField.Ignore" na tabela. Ele não altera a tabela porque a coluna [Address] não existe.

```
Table.ReorderColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    {"Phone", "Address"},
    MissingField.Ignore
)
```

CUSTOMERID	NOME	TELEFONE
1	Bob	123-4567

Table.Repeat

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Repeat(table as table, count as number) as table
```

Sobre

Retorna uma tabela com as linhas da entrada `table` repetidas as `count` vezes especificadas.

Exemplo 1

Repetir as linhas da tabela duas vezes.

```
Table.Repeat(
    Table.FromRecords({
        [a = 1, b = "hello"],
        [a = 3, b = "world"]
    }),
    2
)
```

UM	B
1	olá
3	mundo
1	olá
3	mundo

Table.ReplaceErrorValues

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceErrorValues(table as table, errorReplacement as list) as table
```

Sobre

Substitui os valores de erro nas colunas especificadas do `table` pelos novos valores na lista `errorReplacement`. O formato da lista é {{coluna1, valor1},...}. Pode haver apenas um valor de substituição por coluna; especificar a coluna mais de uma vez resultará em um erro.

Exemplo 1

Substituir o valor de erro pelo texto "mundo" na tabela.

```
Table.ReplaceErrorValues(
    Table.FromRows({{1, "hello"}, {3, ...}}, {"A", "B"}),
    {"B", "world"}
)
```

A	B
1	olá
3	mundo

Exemplo 2

Substituir o valor de erro na coluna A pelo texto "olá" e na coluna B pelo texto "mundo" na tabela.

```
Table.ReplaceErrorValues(
    Table.FromRows({{..., ...}, {1, 2}}, {"A", "B"}),
    {{"A", "hello"}, {"B", "world"}}
)
```

A	B
olá	mundo
1	2

Table.ReplaceKeys

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceKeys(table as table, keys as list) as table
```

Sobre

Table.ReplaceKeys

Table.ReplaceMatchingRows

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceMatchingRows(table as table, replacements as list, optional equationCriteria as any)
as table
```

Sobre

Substitui todas as linhas especificadas na `table` pelas linhas fornecidas. As linhas a serem substituídas e as substituições são especificadas em `replacements`, usando a formatação {old, new}. Um parâmetro `equationCriteria` opcional pode ser especificado para controlar a comparação entre as linhas da tabela.

Exemplo 1

Substituir as linhas [a = 1, b = 2] e [a = 2, b = 3] por [a = -1, b = -2],[a = -2, b = -3] na tabela.

```
Table.ReplaceMatchingRows(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 2, b = 3],
        [a = 3, b = 4],
        [a = 1, b = 2]
    }),
    {
        {[a = 1, b = 2], [a = -1, b = -2]},
        {[a = 2, b = 3], [a = -2, b = -3]}
    }
)
```

UM	B
-1	-2
-2	-3
3	4
-1	-2

Table.ReplaceRelationshipIdentity

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceRelationshipIdentity(value as any, identity as text) as any
```

Sobre

Table.ReplaceRelationshipIdentity

Table.ReplaceRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceRows(table as table, offset as number, count as number, rows as list) as table
```

Sobre

Substitui um número especificado de linhas, `count`, no `table` de entrada pelo `rows` especificado, começando após o `offset`. O parâmetro `rows` é uma lista de registros.

- `table`: A tabela em que a substituição é executada.
- `offset`: O número de linhas a serem ignoradas antes de fazer a substituição.
- `count`: Número de linhas a ser substituídas.
- `rows`: A lista de registros de linha a serem inseridos no `table` na localização especificada pelo `offset`.

Exemplo 1

Começando na posição 1, substituir 3 linhas.

```
Table.ReplaceRows(
    Table.FromRecords({
        [Column1 = 1],
        [Column1 = 2],
        [Column1 = 3],
        [Column1 = 4],
        [Column1 = 5]
    }),
    1,
    3,
    {[Column1 = 6], [Column1 = 7]}
)
```

COLUNA1
1
6
7
5

Table.ReplaceValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReplaceValue(table as table, oldValue as any, newValue as any, replacer as function,
columnsToSearch as list) as table
```

Sobre

Substitui `oldValue` por `newValue` nas colunas especificadas da `table`.

Exemplo 1

Substituir o texto "adeus" pelo texto "mundo" na tabela.

```
Table.ReplaceValue(
    Table.FromRecords({
        [a = 1, b = "hello"],
        [a = 3, b = "goodbye"]
    }),
    "goodbye",
    "world",
    Replacer.ReplaceText,
    {"b"}
)
```

UM	B
1	olá
3	mundo

Exemplo 2

Substitua o texto "uo" pelo texto "ou" na tabela.

```
Table.ReplaceValue(
    Table.FromRecords({
        [a = 1, b = "hello"],
        [a = 3, b = "wurld"]
    }),
    "ur",
    "or",
    Replacer.ReplaceText,
    {"b"}
)
```

UM	B
1	olá

3	mundo

Table.Reverse

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Reverse(text as nullable text) as nullable text
```

Sobre

Inverte o `text` fornecido.

Exemplo 1

Inverta o texto "123".

```
Text.Reverse("123")
```

`"321"`

Table.ReverseRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ReverseRows(table as table) as table
```

Sobre

Retorna uma tabela com as linhas da entrada `table` na ordem inversa.

Exemplo 1

Inverter as linhas da tabela.

```
Table.ReverseRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    })
)
```

CUSTOMERID	NOME	TELEFONE
4	Ringo	232-1550
3	Paul	543-7890
2	Jim	987-6543
1	Bob	123-4567

Table.RowCount

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.RowCount(table as table) as number
```

Sobre

Retorna o número de linhas na `table`.

Exemplo 1

Encontre o número de linhas da tabela.

```
Table.RowCount(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

`3`

Table.Schema

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Schema(table as table) as table
```

Sobre

Retorna uma tabela que descreve as colunas de `table`.

Cada linha na tabela descreve as propriedades de uma coluna de `table`:

Nome de coluna	Descrição
`Name`	O nome da coluna.
`Position`	A posição baseada em 0 da coluna em `table`.
`TypeName`	O nome do tipo da coluna.
`Kind`	A espécie de tipo da coluna.
`IsNullable`	Se a coluna pode conter valores `null`.
`NumericPrecisionBase`	A base numérica (por exemplo, base-2, base-10) dos campos `NumericPrecision` e `NumericScale`.
`NumericPrecision`	A precisão de uma coluna numérica na base especificada por `NumericPrecisionBase`. Este é o número máximo de dígitos que podem ser representados por um valor desse tipo (incluindo dígitos fracionários).
`NumericScale`	A escala de uma coluna numérica na base especificada por `NumericPrecisionBase`. Esse é o número de dígitos na parte fracionária de um valor desse tipo. Um valor de `0` indica uma escala fixa sem dígitos fracionários. Um valor de `null` indica que a escala não é conhecida (seja porque é flutuante ou não está definida).
`DateTimePrecision`	O número máximo de dígitos fracionários com suporte na parte de segundos de um valor de data ou hora.
`MaxLength`	O número máximo de caracteres permitidos em uma coluna `text` ou o número máximo de bytes permitidos em uma coluna `binary`.
`IsVariableLength`	Indica se essa coluna pode variar de comprimento (até `MaxLength`) ou se é de tamanho fixo.

`NativeTypeName`	O nome do tipo da coluna no sistema de tipos nativo da fonte (por exemplo, `nvarchar` para SQL Server).
`NativeDefaultExpression`	A expressão padrão para um valor dessa coluna na linguagem de expressão nativa da fonte (por exemplo, `42` ou `newid()` para SQL Server).
`Description`	A descrição da coluna.

Table.SelectColumns

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.SelectColumns(table as table, columns as any, optional missingField as nullable number) as table
```

Sobre

Retorna o `table` apenas com o `columns` especificado.

- `table`: A tabela fornecida.
- `columns`: A lista de colunas da tabela `table` a ser retornada. As colunas na tabela retornada estão na ordem listada em `columns`.
- `missingField`: *(Opcional)* o que fazer se a coluna não existir. Exemplo: `MissingField.UseNull` ou `MissingField.Ignore`.

Exemplo 1

Inclui apenas a coluna [Name].

```
Table.SelectColumns(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    "Name"
)
```

NOME
Bob
Jim
Paul
Ringo

Exemplo 2

Inclui apenas as colunas [CustomerID] e [Name].

```
Table.SelectColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    {"CustomerID", "Name"}
)
```

CUSTOMERID	NOME
1	Bob

Exemplo 3

Se a coluna incluída não sair, o resultado padrão será um erro.

```
Table.SelectColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    "NewColumn"
)
```

```
[Expression.Error] The field 'NewColumn' of the record wasn't found.
```

Exemplo 4

Se a coluna incluída não sair, a opção `MissingField.UseNull` criará uma coluna de valores nulos.

```
Table.SelectColumns(
    Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}),
    {"CustomerID", "NewColumn"},
    MissingField.UseNull
)
```

CUSTOMERID	NOVA COLUNA
1	

Table.SelectRows

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.SelectRows(table as table, condition as function) as table
```

Sobre

Retorna uma tabela de linhas da `table` que corresponde à seleção `condition`.

Exemplo 1

Seleciona as linhas da tabela em que os valores da coluna [CustomerID] são maiores que 2.

```
Table.SelectRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    each [CustomerID] > 2
)
```

CUSTOMERID	NOME	TELEFONE
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 2

Seleciona as linhas da tabela em que os nomes não contêm um "B".

```
Table.SelectRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    each not Text.Contains([Name], "B")
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890

4	Ringo	232-1550

Table.SelectRowsWithErrors

21/05/2020 • 2 minutes to read

Sintaxe

```
Table.SelectRowsWithErrors(table as table, optional columns as nullable list) as table
```

Sobre

Retorna uma tabela apenas com as linhas da tabela de entrada que contêm um erro em pelo menos uma das células. Se uma lista de colunas for especificada, somente as células nas colunas especificadas serão inspecionadas quanto a erros.

Exemplo 1

Seleciona os nomes de clientes com erros nas linhas.

```
Table.SelectRowsWithErrors(
    Table.FromRecords({
        [CustomerID = ..., Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    })
)[Name]
```

Bob

Table.SingleRow

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.SingleRow(table as table) as record
```

Sobre

Retorna a única linha da linha `table`. Se `table` tiver mais de uma linha, uma exceção será lançada.

Exemplo 1

Retornar a única linha da tabela.

```
Table.SingleRow(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}))
```

CUSTOMERID	1
NOME	Bob
TELEFONE	123-4567

Table.Skip

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Skip(table as table, optional countOrCondition as any) as table
```

Sobre

Retorna uma tabela que não contém o primeiro número especificado de linhas, `countOrCondition`, da tabela `table`. O número de linhas ignoradas depende do parâmetro opcional `countOrCondition`.

- Se `countOrCondition` for omitido, somente a primeira linha será ignorada.
- Se `countOrCondition` for um número, serão ignoradas muitas linhas (começando na parte superior).
- Se `countOrCondition` for uma condição, as linhas que atendem à condição serão ignoradas até que uma linha não atenda à condição.

Exemplo 1

Ignorar a primeira linha da tabela.

```
Table.Skip(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    1
)
```

CUSTOMERID	NOME	TELEFONE
2	Jim	987-6543
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 2

Ignorar as duas primeiras linhas da tabela.

```
Table.Skip(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
    }),
    2
)
```

CUSTOMERID	NOME	TELEFONE
3	Paul	543-7890
4	Ringo	232-1550

Exemplo 3

Ignorar as primeiras linhas em que [Price] > 25 na tabela.

```
Table.Skip(
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
        [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
        [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
        [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
        [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
        [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
        [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
    }),
    each [Price] > 25
)
```

ORDERID	CUSTOMERID	ITEM	PREÇO
2	1	1 lb de minhocas	5
3	2	Rede de pesca	25
4	3	Taser de peixe	200
5	3	Bandaids	2
6	1	Caixa de material de pesca	20
7	5	Isca	3,25
8	5	Vara de pescar	100
9	6	Isca	3,25

Table.Sort

08/05/2020 • 4 minutes to read

Sintaxe

```
Table.Sort(table as table, comparisonCriteria as any) as table
```

Sobre

Classifica a `table` usando a lista de um ou mais nomes de coluna e o `comparisonCriteria` opcional no formato {{ {{ col1, comparisonCriteria }}, {{col2}} }}.

Exemplo 1

Classificar a tabela na coluna "OrderID".

```
Table.Sort(
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
        [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
        [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
        [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
        [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
        [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
        [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
    }),
    {"OrderID"}
)
```

ORDERID	CUSTOMERID	ITEM	PREÇO
1	1	Vara de pescar	100
2	1	1 lb de minhocas	5
3	2	Rede de pesca	25
4	3	Taser de peixe	200
5	3	Bandaids	2
6	1	Caixa de material de pesca	20
7	5	Isca	3,25
8	5	Vara de pescar	100
9	6	Isca	3,25

Exemplo 2

Classificar a tabela na coluna "OrderID" em ordem decrescente.

```
Table.Sort(
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
        [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
        [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
        [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
        [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
        [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
        [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
    }),
    {"OrderID", Order.Descending}
)
```

ORDERID	CUSTOMERID	ITEM	PREÇO
9	6	Isca	3,25
8	5	Vara de pescar	100
7	5	Isca	3,25
6	1	Caixa de material de pesca	20
5	3	Bandaids	2
4	3	Taser de peixe	200
3	2	Rede de pesca	25
2	1	1 lb de minhocas	5
1	1	Vara de pescar	100

Exemplo 3

Classificar a tabela na coluna "CustomerID" e depois em "OrderID", com "CustomerID" em ordem crescente.

```
Table.Sort(
    Table.FromRecords({
        [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
        [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
        [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
        [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
        [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
        [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
        [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
        [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
        [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
    }),
    {
        {"CustomerID", Order.Ascending},
        "OrderID"
    }
)
```

ORDERID	CUSTOMERID	ITEM	PREÇO
1	1	Vara de pescar	100
2	1	1 lb de minhocas	5
6	1	Caixa de material de pesca	20
3	2	Rede de pesca	25
4	3	Taser de peixe	200
5	3	Bandaids	2
7	5	Isca	3,25
8	5	Vara de pescar	100
9	6	Isca	3,25

Table.Split

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.Split(table as table, pageSize as number) as list
```

Sobre

Divide o `table` em uma lista de tabelas em que o primeiro elemento da lista é uma tabela que contém as primeiras `pageSize` linhas da tabela de origem, o próximo elemento da lista é uma tabela que contém as próximas `pageSize` linhas da tabela de origem etc.

Exemplo 1

Divida uma tabela de cinco registros em tabelas com dois registros cada.

```
let
    Customers = Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Cristina", Phone = "232-1550"],
        [CustomerID = 5, Name = "Anita", Phone = "530-1459"]
    })
in
    Table.Split(Customers, 2)
```

[Tabela]
[Tabela]
[Tabela]

Table.SplitColumn

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.SplitColumn(table as table, sourceColumn as text, splitter as function, optional
columnNamesOrNumber as any, optional default as any, optional extraColumns as any) as table
```

Sobre

Divide as colunas especificadas em um conjunto de colunas adicionais usando a função de divisão especificada.

Exemplo 1

Divida a coluna [Name] em duas na posição do "i"

```
let
    Customers = Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
        [CustomerID = 4, Name = "Cristina", Phone = "232-1550"]
    })
in
    Table.SplitColumn(Customers, "Name", Splitter.SplitTextByDelimiter("i"), 2
```

CUSTOMERID	NAME.1	NAME.2	TELEFONE
1	Bob		123-4567
2	J	m	987-6543
3	Paul		543-7890
4	Cr	ST	232-1550

Table.ToColumns

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ToColumns(table as table) as list
```

Sobre

Cria uma lista de listas aninhadas da tabela, `table`. Cada item de lista é uma lista interna que contém os valores de coluna.

Exemplo

Criar uma lista dos valores de coluna da tabela.

```
Table.ToColumns(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"]
    })
)
```

[Lista]
[Lista]
[Lista]

Table.ToList

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ToList(table as table, optional combiner as nullable function) as list
```

Sobre

Converte uma tabela em lista aplicando a função de combinação especificada a cada linha de valores na tabela.

Exemplo 1

Combina o texto de todas as linhas com uma vírgula.

```
Table.ToList(
    Table.FromRows({
        {Number.ToText(1), "Bob", "123-4567"},
        {Number.ToText(2), "Jim", "987-6543"},
        {Number.ToText(3), "Paul", "543-7890"}
    }),
    Combiner.CombineTextByDelimiter(",")
)
```

1,Bob,123-4567
2,Jim,987-6543
3,Paul,543-7890

Table.ToRecords

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ToRecords(table as table) as list
```

Sobre

Converte uma tabela, `table`, em uma lista de registros.

Exemplo

Converta a tabela em uma lista de registros.

```
Table.ToRecords(
    Table.FromRows(
        {
            {1, "Bob", "123-4567"},
            {2, "Jim", "987-6543"},
            {3, "Paul", "543-7890"}
        },
        {"CustomerID", "Name", "Phone"}
    )
)
```

[Registro]
[Registro]
[Registro]

Table.ToRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.ToRows(table as table) as list
```

Sobre

Cria uma lista de listas aninhadas da tabela, `table`. Cada item de lista é uma lista interna que contém os valores de linha.

Exemplo

Crie uma lista dos valores de linha da tabela.

```
Table.ToRows(
    Table.FromRecords({
        [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
        [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
        [CustomerID = 3, Name = "Paul", Phone = "543-7890"]
    })
)
```

[Lista]
[Lista]
[Lista]

Table.TransformColumnNames

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.TransformColumnNames(table as table, nameGenerator as function, optional options as nullable record) as table
```

Sobre

Transforma nomes de colunas ao usar a função `nameGenerator` fornecida. Opções válidas:

`MaxLength` especifica o comprimento máximo de novos nomes de coluna. Se a função fornecida resultar em um nome de coluna maior, o nome longo será cortado.
`Comparer` é usado para controlar a comparação ao gerar novos nomes de coluna. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Remover o caractere de `#(tab)` dos nomes de coluna

```
Table.TransformColumnNames(Table.FromRecords({[#"Col#(tab)umn" = 1]}), Text.Clean)
```

COLUNA
1

Exemplo 2

Transforme nomes de coluna para gerar nomes de comprimento 6 sem diferenciação de maiúsculas e minúsculas.

```
Table.TransformColumnNames(
    Table.FromRecords({[ColumnNum = 1, cOlumnnum = 2, coLumnNUM = 3]}),
    Text.Clean,
    [MaxLength = 6, Comparer = Comparer.OrdinalIgnoreCase]
)
```

COLUNA	COLUM1	COLUM2
1	2	3

Table.TransformColumns

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.TransformColumns(table as table, transformOperations as list, optional defaultTransformation as nullable function, optional missingField as nullable number) as table
```

Sobre

Retorna uma tabela da entrada `table` aplicando a operação de transformação às colunas especificadas no parâmetro `transformOperations` (cujo formato é { nome da coluna, transformação}). Se a coluna não existir, uma exceção será gerada, a menos que o parâmetro opcional `defaultTransformation` especifique uma alternativa (por exemplo, `MissingField.UseNull` ou `MissingField.Ignore`).

Exemplo 1

Transforma os valores numéricos da coluna [A] em valores numéricos.

```
Table.TransformColumns(
    Table.FromRecords({
        [A = "1", B = 2],
        [A = "5", B = 10]
    }),
    {"A", Number.FromText}
)
```

A	B
1	2
5	10

Exemplo 2

Transforma os valores numéricos da coluna ausente [X] em valores de texto, ignorando colunas inexistentes.

```
Table.TransformColumns(
    Table.FromRecords({
        [A = "1", B = 2],
        [A = "5", B = 10]
    }),
    {"X", Number.FromText},
    null,
    MissingField.Ignore
)
```

A	B
1	2

5	10

Exemplo 3

Transforma os valores numéricos da coluna ausente [X] em valores de texto, usando nulo como padrão em colunas inexistentes.

```
Table.TransformColumns(
    Table.FromRecords({
        [A = "1", B = 2],
        [A = "5", B = 10]
    }),
    {"X", Number.FromText},
    null,
    MissingField.UseNull
)
```

A	B	X
1	2	
5	10	

Exemplo 4

Transforma os valores numéricos da coluna ausente [X] em valores de texto, retornando um erro em colunas inexistentes.

```
Table.TransformColumns(
    Table.FromRecords({
        [A = "1", B = 2],
        [A = "5", B = 10]
    }),
    {"X", Number.FromText}
)
```

```
[Expression.Error] The column 'X' of the table wasn't found.
```

Table.TransformColumnTypes

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.TransformColumnTypes(table as table, typeTransformations as list, optional culture as
nullable text) as table
```

Sobre

Retorna uma tabela da entrada `table` aplicando a operação de transformação às colunas especificadas no parâmetro `typeTransformations` (em que formato é {nome da coluna, nome do tipo}), usando a cultura especificada no parâmetro opcional `culture` (por exemplo, "pt-BR"). Se a coluna não existir, uma exceção será gerada.

Exemplo 1

Transforme os valores numéricos da coluna [a] em valores de texto da tabela `({[a = 1, b = 2], [a = 3, b = 4]})`.

```
Table.TransformColumnTypes(
    Table.FromRecords({
        [a = 1, b = 2],
        [a = 3, b = 4]
    }),
    {"a", type text},
    "en-US"
)
```

UM	B
1	2
3	4

Table.TransformRows

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.TransformRows(table as table, transform as function) as list
```

Sobre

Cria uma tabela de `table` aplicando a operação `transform` às linhas. Se o tipo de retorno da função `transform` for especificado, o resultado será uma tabela com esse tipo de linha. Em todos os outros casos, o resultado dessa função será uma lista com um tipo de item do tipo de retorno da função de transformação.

Exemplo 1

Transforme as linhas em uma lista de números com base na tabela `({[A = 1], [A = 2], [A = 3], [A = 4], [A = 5]})`.

```
Table.TransformRows(
    Table.FromRecords({
        [a = 1],
        [a = 2],
        [a = 3],
        [a = 4],
        [a = 5]
    }),
    each [a]
)
```

1
2
3
4
5

Exemplo 2

Transforme as linhas da coluna [A] em valores de texto em uma coluna [B] com base na tabela `({[A = 1], [A = 2], [A = 3], [A = 4], [A = 5])`.

```
Table.TransformRows(
    Table.FromRecords({
        [a = 1],
        [a = 2],
        [a = 3],
        [a = 4],
        [a = 5]
    }),
    (row) as record => [B = Number.ToText(row[a])]
)
```

[Registro]
[Registro]
[Registro]
[Registro]
[Registro]

Table.Transpose

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Transpose(table as table, optional columns as any) as table
```

Sobre

Transforma colunas em linhas e linhas em colunas.

Exemplo 1

Transforme as linhas da tabela de pares de nome/valor em colunas.

```
Table.Transpose(
    Table.FromRecords({
        [Name = "Full Name", Value = "Fred"],
        [Name = "Age", Value = 42],
        [Name = "Country", Value = "UK"]
    })
)
```

COLUNA1	COLUNA2	COLUMN3
Nome completo	Idade	País
Fred	42	REINO UNIDO

Table.Unpivot

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.Unpivot(table as table, pivotColumns as list, attributeColumn as text, valueColumn as text)
as table
```

Sobre

Converte um conjunto de colunas de uma tabela em pares de atributo/valor, combinados com o restante dos valores em cada linha.

Exemplo 1

Transforme as colunas "a", "b" e "c" da tabela `({[ key = "x", a = 1, b = null, c = 3 ], [ key = "y", a = 2, b = 4, c = null ]})` em pares de atributo/valor.

```
Table.Unpivot(
    Table.FromRecords({
        [key = "x", a = 1, b = null, c = 3],
        [key = "y", a = 2, b = 4, c = null]
    }),
    {"a", "b", "c"},
    "attribute",
    "value"
)
```

CHAVE	ATRIBUTO	VALOR
x	um	1
x	c	3
a	um	2
a	b	4

Table.UnpivotOtherColumns

08/05/2020 • 2 minutes to read

Sintaxe

```
Table.UnpivotOtherColumns(table as table, pivotColumns as list, attributeColumn as text,
valueColumn as text) as table
```

Sobre

Converte todas as colunas que não sejam um conjunto especificado em pares de atributo/valor, combinados com o restante dos valores em cada linha.

Exemplo 1

Converte todas as colunas que não sejam um conjunto especificado em pares de atributo/valor, combinados com o restante dos valores em cada linha.

```
Table.UnpivotOtherColumns(
    Table.FromRecords({
        [key = "key1", attribute1 = 1, attribute2 = 2, attribute3 = 3],
        [key = "key2", attribute1 = 4, attribute2 = 5, attribute3 = 6]
    }),
    {"key"},
    "column1",
    "column2"
)
```

CHAVE	COLUMN1	COLUMN2
key1	attribute1	1
key1	attribute2	2
key1	attribute3	3
key2	attribute1	4
key2	attribute2	5
key2	attribute3	6

Table.View

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.View(table as nullable table, handlers as record) as table
```

Sobre

Retorna uma exibição de `table` em que as funções especificadas em `handlers` são usadas no lugar do comportamento padrão de uma operação quando a operação é aplicada à exibição. Funções de manipulador são opcionais. Se uma função de manipulador não for especificada para uma operação, o comportamento padrão da operação será aplicado a `table` em vez disso (exceto no caso de `GetExpression`).

As funções de manipulador devem retornar um valor que seja semanticamente equivalente ao resultado da aplicação da operação em relação a `table` (ou à exibição resultante, no caso de `GetExpression`).

Se uma função de manipulador gerar um erro, o comportamento padrão da operação será aplicado à exibição.

`Table.View` pode ser usado para implementar a dobragem em uma fonte de dados – a conversão de consultas M em consultas específicas da fonte (por exemplo, para criar instruções T-SQL com base em consultas M).

Confira a documentação publicada para obter uma descrição mais completa de `Table.View`.

Table.ViewFunction

09/05/2020 • 2 minutes to read

Sintaxe

```
Table.ViewFunction(function as function) as function
```

Sobre

Cria uma função de exibição baseada em `function` que pode ser manipulada em uma exibição criada por `Table.View`.

O manipulador de `OnInvoke` de `Table.View` pode ser usado para definir um manipulador para a função de exibição.

Assim como os manipuladores para operações internas, se nenhum manipulador de `OnInvoke` for especificado, se ele não tratar a função de exibição ou se um erro for gerado pelo manipulador, `function` será aplicado na parte superior da exibição.

Confira a documentação publicada para obter uma descrição mais completa das funções de `Table.View` e de exibição personalizada.

Tables.GetRelationships

09/05/2020 • 2 minutes to read

Sintaxe

```
Tables.GetRelationships(tables as table, optional dataColumn as nullable text) as table
```

Sobre

Obtém os relacionamentos entre um conjunto de tabelas. Presume-se que o conjunto `tables` tenha uma estrutura semelhante à de uma tabela de navegação. A coluna definida por `dataColumn` contém as tabelas de dados reais.

#table

08/05/2020 • 2 minutes to read

Sintaxe

```
#table(columns as any, rows as any) as any
```

Sobre

Cria um valor de tabela de colunas `columns` e a lista `rows` em que cada elemento da lista é uma lista interna que contém os valores de coluna para uma única linha. `columns` pode ser uma lista de nomes de coluna, um tipo de tabela, um número de colunas ou nulo.

Funções de texto

08/05/2020 • 8 minutes to read

Essas funções criam e manipulam valores de texto.

Texto

Informações

FUNÇÃO	DESCRIÇÃO
Text.InferNumberType	Infere o tipo de número granular (Int64.Type, Double.Type etc.) de `text` usando `culture`.
Text.Length	Retorna o número de caracteres em um valor de texto.

Comparações de texto

FUNÇÃO	DESCRIÇÃO
Character.FromNumber	Retorna um número para seu valor de caractere.
Character.ToNumber	Retorna um caractere para seu valor numérico.
Guid.From	Retorna um valor `Guid.Type` do `value` especificado.
Json.FromValue	Produz uma representação JSON de um determinado valor.
Text.From	Retorna a representação de texto de um valor de número, data, time, datetime, datetimezone, lógico, duração ou binário. Se um valor for nulo, Text.from retornará nulo. O parâmetro opcional Culture é usado para formatar o valor de texto de acordo com a cultura fornecida.
Text.FromBinary	Decodifica dados de um valor binário para um valor de texto usando uma codificação.
Text.NewGuid	Retorna um valor Guid como um valor de texto.
Text.ToBinary	Codifica um valor de texto em um valor binário usando uma codificação.
Text.ToList	Retorna uma lista de caracteres de um valor de texto.
Value.FromText	Decodifica um valor de uma representação textual, um valor e o interpreta como um valor com tipo apropriado. Value.FromText usa um valor de texto e retorna um número, um valor lógico, um valor nulo, um valor DateTime, um valor Duration ou um valor de texto. O valor de texto vazio é interpretado como um valor nulo.

Extrair

FUNÇÃO	DESCRIÇÃO
Text.At	Retorna um caractere que começa em um deslocamento de base zero.
Text.Middle	Retorna a substring até um comprimento específico.
Text.Range	Retorna um número de caracteres de um valor de texto começando em um deslocamento de base zero e também para o número de contagem de caracteres.
Text.Start	Retorna a contagem de caracteres do início de um valor de texto.

FUNÇÃO	DESCRIÇÃO
Text.End	Retorna o número de caracteres do final de um valor de texto.

Modificado

FUNÇÃO	DESCRIÇÃO
Text.Insert	Retorna um valor de texto com newValue inserido em um valor de texto começando em um deslocamento de base zero.
Text.Remove	Remove todas as ocorrências de um caractere ou de uma lista de caracteres de um valor de texto. O parâmetro removeChars pode ser um valor de caractere ou uma lista de valores de caracteres.
Text.RemoveRange	Remove caracteres de contagem em um deslocamento de base zero de um valor de texto.
Text.Replace	Substitui todas as ocorrências de uma substring por um novo valor de texto.
Text.ReplaceRange	Substitui caracteres de comprimento em um valor de texto que começa em um deslocamento de base zero com o novo valor de texto.
Text.Select	Seleciona todas as ocorrências do caractere especificado ou da lista de caracteres do valor de texto de entrada.

Associação

FUNÇÃO	DESCRIÇÃO
Text.Contains	Retorna true se uma substring de valor de texto foi encontrada em uma substring de valor de texto; caso contrário, false.
Text.EndsWith	Retorna um valor lógico que indica se uma substring de valor de texto foi encontrada no final de uma cadeia de caracteres.

FUNÇÃO	DESCRIÇÃO
Text.PositionOf	Retorna a primeira ocorrência de substring em uma cadeia de caracteres e retorna sua posição começando em startOffset.
Text.PositionOfAny	Retorna a primeira ocorrência de um valor de texto na lista e retorna sua posição começando em startOffset.
Text.StartsWith	Retorna um valor lógico que indica se uma substring de valor de texto foi encontrada no início de uma cadeia de caracteres.

Transformações

FUNÇÃO	DESCRIÇÃO
Text.AfterDelimiter	Retorna a parte do texto que está após o delimitador especificado.
Text.BeforeDelimiter	Retorna a parte do texto que está antes do delimitador especificado.
Text.BetweenDelimiters	Retorna a parte do texto que está entre o startDelimiter e o endDelimiter especificados.
Text.Clean	Retorna o valor de texto original com os caracteres não imprimíveis removidos.
Text.Combine	Retorna um valor de texto que é o resultado da junção de todos os valores de texto com cada valor separado por um separador.
Text.Lower	Retorna um valor de texto em minúsculas.
Text.PadEnd	Retorna um valor de texto preenchido no final com o pad para que ele fique no comprimento mínimo de caracteres.
Text.PadStart	Retorna um valor de texto preenchido no início com o pad para que ele fique no comprimento mínimo de caracteres. Se o pad não for especificado, será usado o espaço em branco.
Text.Proper	Retorna um valor de texto com as primeiras letras de todas as palavras convertidas em maiúsculas.
Text.Repeat	Retorna um valor de texto composto pelo valor de texto de entrada repetido várias vezes.
Text.Reverse	Reverte o texto fornecido.
Text.Split	Retorna uma lista contendo partes de um valor de texto delimitadas por um valor de texto separador.
Text.SplitAny	Retorna uma lista contendo partes de um valor de texto delimitadas por quaisquer valores de texto separadores.

FUNÇÃO	DESCRIÇÃO
Text.Trim	Remove todas as ocorrências de caracteres em trimChars do texto.
Text.TrimEnd	Remove todas as ocorrências dos caracteres especificados em trimChars do final do valor de texto original.
Text.TrimStart	Remove todas as ocorrências dos caracteres em trimChars do início do valor de texto original.
Text.Upper	Retorna um valor de texto em maiúsculas.

Parâmetros

VALORES DE PARÂMETROS	DESCRIÇÃO
Occurrence.All	Uma lista de posições de todas as ocorrências dos valores encontrados é retornada.
Occurrence.First	A posição da primeira ocorrência do valor encontrado é retornada.
Occurrence.Last	A posição da última ocorrência do valor encontrado é retornada.
RelativePosition.FromEnd	Indica que a indexação deve ser feita desde o final da entrada.
RelativePosition.FromStart	Indica que a indexação deve ser feita desde o início da entrada.
TextEncoding.Ascii	Use para escolher o formato binário ASCII.
TextEncoding.BigEndianUnicode	Use para escolher o formato binário UTF16 big endian.
TextEncoding.Unicode	Use para escolher o formato binário UTF16 little endian.
TextEncoding.Utf8	Use para escolher o formato binário UTF8.
TextEncoding.Utf16	Use para escolher o formato binário UTF16 little endian.
TextEncoding.Windows	Use para escolher o formato binário do Windows.

Character.FromNumber

09/05/2020 • 2 minutes to read

Sintaxe

```
Character.FromNumber(number as nullable number) as nullable text
```

Sobre

Retorna o equivalente de caracteres do número.

Exemplo 1

Dado o número 9, localize o valor de caractere.

```
Character.FromNumber(9)
```

`"#(tab)"`

Character.ToNumber

09/05/2020 • 2 minutes to read

Sintaxe

```
Character.ToNumber(character as nullable text) as nullable number
```

Sobre

Retorna o equivalente numérico do caractere `character`.

Exemplo 1

Dado o caractere "#(tab)" 9, localize o valor numérico.

```
Character.ToNumber("#(tab)")
```

`9`

Guid.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Guid.From(value as nullable text) as nullable text
```

Sobre

Retorna um valor `Guid.Type` do `value` especificado. Se o `value` fornecido for `null`, `Guid.From` retornará `null`. Uma verificação será executada para ver se o `value` fornecido está em um formato aceitável. Formatos aceitáveis fornecidos nos exemplos.

Exemplo 1

O GUID pode ser fornecido como 32 dígitos hexadecimais contíguos.

```
Guid.From("05FE1DADC8C24F3BA4C2D194116B4967")
```

`"05fe1dad-c8c2-4f3b-a4c2-d194116b4967"`

Exemplo 2

O GUID pode ser fornecido como 32 dígitos hexadecimais separados por hífen em blocos de 8-4-4-4-12.

```
Guid.From("05FE1DAD-C8C2-4F3B-A4C2-D194116B4967")
```

`"05fe1dad-c8c2-4f3b-a4c2-d194116b4967"`

Exemplo 3

O GUID pode ser fornecido como 32 dígitos hexadecimais separados por hífen e delimitados por chaves.

```
Guid.From("{05FE1DAD-C8C2-4F3B-A4C2-D194116B4967}")
```

`"05fe1dad-c8c2-4f3b-a4c2-d194116b4967"`

Exemplo 4

O GUID pode ser fornecido como 32 dígitos hexadecimais separados por hífen e delimitados por parênteses.

```
Guid.From("(05FE1DAD-C8C2-4F3B-A4C2-D194116B4967)")
```

`"05fe1dad-c8c2-4f3b-a4c2-d194116b4967"`

Json.FromValue

09/05/2020 • 2 minutes to read

Sintaxe

```
Json.FromValue(value as any, optional encoding as nullable number) as binary
```

Sobre

Produz uma representação JSON de um valor especificado `value` com uma codificação de texto especificada por `encoding`. Se `encoding` for omitido, UTF8 será usado. Os valores são representados da seguinte maneira:

- Valores nulos, lógicos e de texto são representados como os tipos JSON correspondentes
- Os números são representados como números em JSON, exceto que `#infinity`, `-#infinity` e `#nan` são convertidos em NULL
- As listas são representadas como matrizes JSON
- Os registros são representados como objetos JSON
- As tabelas são representadas como uma matriz de objetos
- Datas, horas, data e hora, fusos horários e durações são representados como texto ISO-8601
- Os valores binários são representados como texto codificado em base64
- Tipos e funções produzem um erro

Exemplo 1

Converter um valor complexo em JSON.

```
Text.FromBinary(Json.FromValue([A = {1, true, "3"}, B = #date(2012, 3, 25)]))
```

`"{""A"":[1,true,""3""],""B"":""2012-03-25""}"`

RelativePosition.FromEnd

08/05/2020 • 2 minutes to read

Sobre

Indica que a indexação deve ser feita desde o final da entrada.

RelativePosition.FromStart

09/05/2020 • 2 minutes to read

Sobre

Indica que a indexação deve ser feita desde o início da entrada.

Text.AfterDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.AfterDelimiter(text as nullable text, delimiter as text, optional index as any) as any
```

Sobre

Retorna a parte de `text` que está após o `delimiter` especificado. Um `index` numérico opcional indica qual ocorrência do `delimiter` deve ser considerada. Uma lista opcional `index` indica qual ocorrência do `delimiter` deve ser considerada, bem como se a indexação deve ser feita do início ou do fim da entrada.

Exemplo 1

Obtenha a porção de "111-222-333" depois do (primeiro) hífen.

```
Text.AfterDelimiter("111-222-333", "-")
```

`"222-333"`

Exemplo 2

Obtenha a porção de "111-222-333" depois do segundo hífen.

```
Text.AfterDelimiter("111-222-333", "-", 1)
```

`"333"`

Exemplo 3

Obtenha a porção de "111-222-333" depois do segundo hífen do fim.

```
Text.AfterDelimiter("111-222-333", "-", {1, RelativePosition.FromEnd})
```

`"222-333"`

Text.At

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.At(text as nullable text, index as number) as nullable text
```

Sobre

Retorna o caractere no valor de texto `text` na posição `index`. O primeiro caractere no texto está na posição 0.

Exemplo 1

Localize o caractere na posição 4 na cadeia de caracteres "Olá, Mundo".

```
Text.At("Hello, World", 4)
```

`"o"`

Text.BeforeDelimiter

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.BeforeDelimiter(text as nullable text, delimiter as text, optional index as any) as any
```

Sobre

Retorna a parte de `text` antes do `delimiter` especificado. Um `index` numérico opcional indica qual ocorrência do `delimiter` deve ser considerada. Uma lista opcional `index` indica qual ocorrência do `delimiter` deve ser considerada, bem como se a indexação deve ser feita do início ou do fim da entrada.

Exemplo 1

Obtenha a porção de "111-222-333" antes do (primeiro) hífen.

```
Text.BeforeDelimiter("111-222-333", "-")
```

`"111"`

Exemplo 2

Obtenha a porção de "111-222-333" antes do segundo hífen.

```
Text.BeforeDelimiter("111-222-333", "-", 1)
```

`"111-222"`

Exemplo 3

Obtenha a porção de "111-222-333" antes do segundo hífen do fim.

```
Text.BeforeDelimiter("111-222-333", "-", {1, RelativePosition.FromEnd})
```

`"111"`

Text.BetweenDelimiters

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.BetweenDelimiters(text as nullable text, startDelimiter as text, endDelimiter as text,
optional startIndex as any, optional endIndex as any) as any
```

Sobre

Retorna a parte de `text` entre o `startDelimiter` especificado e o `endDelimiter`. Um `startIndex` numérico opcional indica qual ocorrência do `startDelimiter` deve ser considerada. Uma lista opcional `startIndex` indica qual ocorrência do `startDelimiter` deve ser considerada, bem como se a indexação deve ser feita do início ou do fim da entrada. O `endIndex` é semelhante, exceto que a indexação é feita com relação a `startIndex`.

Exemplo 1

Obtenha a porção de "111 (222) 333 (444)" entre o (primeiro) parêntese de abertura e o (primeiro) parêntese de fechamento que o segue.

```
Text.BetweenDelimiters("111 (222) 333 (444)", "(", ")")
```

`"222"`

Exemplo 2

Obtenha a porção de "111 (222) 333 (444)" entre o segundo parêntese de abertura e o primeiro parêntese de fechamento que o segue.

```
Text.BetweenDelimiters("111 (222) 333 (444)", "(", ")", 1, 0)
```

`"444"`

Exemplo 3

Obtenha a porção de "111 (222) 333 (444)" entre o segundo parêntese de abertura do final e o segundo parêntese de fechamento que o segue.

```
Text.BetweenDelimiters("111 (222) 333 (444)", "(", ")", {1, RelativePosition.FromEnd}, {1,
RelativePosition.FromStart})
```

`"222) 333 (444"`

Text.Clean

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Clean(text as nullable text) as nullable text
```

Sobre

Retorna um valor de texto com todos os caracteres não imprimíveis de `text` removidos.

Exemplo 1

Remova os feeds de linha e outros caracteres não imprimíveis de um valor de texto.

```
Text.Clean("ABC#(lf)D")
```

`"ABCD"`

Text.Combine

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Combine(texts as list, optional separator as nullable text) as text
```

Sobre

Retorna o resultado da combinação da lista de valores de texto, `texts`, em um único valor de texto. Um separador opcional usado no texto combinado final pode ser especificado, `separator`.

Exemplo 1

Combine os valores "Seattle" e "WA".

```
Text.Combine({"Seattle", "WA"})
```

`"SeattleWA"`

Exemplo 2

Combine os valores de texto "Seattle" e "WA" separados por uma vírgula e um espaço, ", ".

```
Text.Combine({"Seattle", "WA"}, ", ")
```

`"Seattle, WA"`

Text.Contains

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Contains(text as nullable text, substring as text, optional comparer as nullable function) as nullable logical
```

Sobre

Detecta se o texto `text` contém o texto `substring`. Retornará true se o texto for encontrado.

`comparer` é um `Comparer` usado para controlar a comparação. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Descubra se o texto "Olá, Mundo" contém "Olá".

```
Text.Contains("Hello World", "Hello")
```

`true`

Exemplo 2

Descubra se o texto "Olá, Mundo" contém "olá".

```
Text.Contains("Hello World", "hello")
```

`false`

Text.End

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.End(text as nullable text, count as number) as nullable text
```

Sobre

Retorna um valor `text`, que é composto pelos últimos `count` caracteres do valor `text``text`.

Exemplo 1

Obtenha os cinco últimos caracteres do texto "Olá, Mundo".

```
Text.End("Hello, World", 5)
```

`"World"`

Text.EndsWith

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.EndsWith(text as nullable text, substring as text, optional comparer as nullable function) as nullable logical
```

Sobre

Indica se o texto fornecido, `text`, termina com o valor especificado, `substring`. A indicação diferencia maiúsculas de minúsculas.

`comparer` é um `Comparer` usado para controlar a comparação. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Verifique se "Olá, Mundo" termina com "mundo".

```
Text.EndsWith("Hello, World", "world")
```

`false`

Exemplo 2

Verifique se "Olá, Mundo" termina com "Mundo".

```
Text.EndsWith("Hello, World", "World")
```

`true`

Text.Format

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.Format(formatString as text, arguments as any, optional culture as nullable text) as text
```

Sobre

Retorna o texto formatado criado aplicando `arguments` de uma lista ou registro a uma cadeia de caracteres de formato `formatString`. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Formata uma lista de números.

```
Text.Format("#{0}, #{1}, and #{2}.", {17, 7, 22})
```

```
"17, 7, and 22."
```

Exemplo 2

Formata diferentes tipos de dados de um registro de acordo com a cultura do inglês (Estados Unidos).

```
Text.Format(
    "The time for the #[distance] km run held in #[city] on #[date] was #[duration].",
    [
        city = "Seattle",
        date = #date(2015, 3, 10),
        duration = #duration(0, 0, 54, 40),
        distance = 10
    ],
    "en-US"
)
```

```
"The time for the 10 km run held in Seattle on 3/10/2015 was 00:54:40."
```

Text.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.From(value as any, optional culture as nullable text) as nullable text
```

Sobre

Retorna a representação de texto de `value`. O `value` pode ser um valor `number`, `date`, `time`, `datetime`, `datetimezone`, `logical`, `duration` ou `binary`. Se o valor fornecido for nulo, `Text.From` retornará nulo. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Crie um valor de texto com base no número 3.

```
Text.From(3)
```

`"3"`

Text.FromBinary

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.FromBinary(binary as nullable binary, optional encoding as nullable number) as nullable text
```

Sobre

Decodifica os dados, `binary`, de um valor binário em valor de texto, usando o tipo `encoding`.

Text.InferNumberType

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.InferNumberType(text as text, optional culture as nullable text) as type
```

Sobre

Infere o tipo de número granular (Int64.Type, Double.Type etc.) de `text`. Um erro será gerado se `text` não for um número. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Text.Insert

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Insert(text as nullable text, offset as number, newText as text) as nullable text
```

Sobre

Retorna o resultado da inserção do valor de texto `newText` no valor de texto `text` na posição `offset`. As posições começam no número 0.

Exemplo 1

Insira "C" entre "B" e "D" em "ABD".

```
Text.Insert("ABD", 2, "C")
```

`"ABCD"`

Text.Length

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Length(text as nullable text) as nullable number
```

Sobre

Retorna o número de caracteres no texto `text`.

Exemplo 1

Localize quantos caracteres existem no texto "Olá, Mundo".

```
Text.Length("Hello World")
```

```
11
```

Text.Lower

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.Lower(text as nullable text, optional culture as nullable text) as nullable text
```

Sobre

Retorna o resultado da conversão de todos os caracteres de `text` em minúsculas. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha a versão minúscula de "AbCd".

```
Text.Lower("AbCd")
```

`"abcd"`

Text.Middle

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Middle(text as nullable text, start as number, optional count as nullable number) as nullable
text
```

Sobre

Retorna `count` caracteres ou até o final de `text`; no deslocamento `start`.

Exemplo 1

Encontre a substring no texto "Olá, Mundo" iniciando no índice 6 abrangendo 5 caracteres.

```
Text.Middle("Hello World", 6, 5)
```

`"World"`

Exemplo 2

Encontre a substring no texto "Olá, Mundo" iniciando no índice 6 até o final.

```
Text.Middle("Hello World", 6, 20)
```

`"World"`

Text.NewGuid

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.NewGuid() as text
```

Sobre

Retorna um novo identificador global exclusivo (GUID) aleatório.

Text.PadEnd

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.PadEnd(text as nullable text, count as number, optional character as nullable text) as
nullable text
```

Sobre

Retorna um valor de `text` preenchido para comprimento `count` inserindo espaços no fim do valor de texto `text`. Um caractere opcional `character` pode ser usado para especificar o caractere de preenchimento. O caractere de preenchimento padrão é um espaço.

Exemplo 1

Preencha o final de um valor de texto para que tenha 10 caracteres.

```
Text.PadEnd("Name", 10)
```

```
"Name      "
```

Exemplo 2

Preencha o final de um valor de texto com "|" para que tenha 10 caracteres.

```
Text.PadEnd("Name", 10, "|")
```

```
"Name||||||"
```

Text.PadStart

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.PadStart(text as nullable text, count as number, optional character as nullable text) as
nullable text
```

Sobre

Retorna um valor de `text` preenchido para comprimento `count` inserindo espaços no início do valor de texto `text`. Um caractere opcional `character` pode ser usado para especificar o caractere de preenchimento. O caractere de preenchimento padrão é um espaço.

Exemplo 1

Preencha o início de um valor de texto para que tenha 10 caracteres.

```
Text.PadStart("Name", 10)
```

```
"      Name"
```

Exemplo 2

Preencha o início de um valor de texto com "|" para que tenha 10 caracteres.

```
Text.PadStart("Name", 10, "|")
```

```
"||||||Name"
```

Text.PositionOf

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.PositionOf(text as text, substring as text, optional occurrence as nullable number, optional comparer as nullable function) as any
```

Sobre

Retorna a posição da ocorrência especificada do valor de texto `substring` encontrado em `text`. Um parâmetro opcional `occurrence` pode ser usado para especificar qual posição de ocorrência deve ser retornada (primeira ocorrência por padrão). Retorna -1 se `substring` não foi encontrado.

`comparer` é um `Comparer` usado para controlar a comparação. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Obtenha a posição da primeira ocorrência de "Mundo" no texto "Olá, Mundo! Olá, Mundo!".

```
Text.PositionOf("Hello, World! Hello, World!", "World")
```

`7`

Exemplo 2

Obtenha a posição da última ocorrência de "Mundo" no texto "Olá, Mundo! Olá, Mundo!".

```
Text.PositionOf("Hello, World! Hello, World!", "World", Occurrence.Last)
```

`21`

Text.PositionOfAny

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.PositionOfAny(text as text, characters as list, optional occurrence as nullable number) as
any
```

Sobre

Retorna a posição da primeira ocorrência de qualquer um dos caracteres na lista de caracteres `text` encontrada no valor de texto `characters`. Um parâmetro opcional `occurrence` pode ser usado para especificar qual posição de ocorrência deve ser retornada.

Exemplo 1

Localize a posição do "M" no texto "Olá, Mundo!".

```
Text.PositionOfAny("Hello, World!", {"W"})
```

`7`

Exemplo 2

Localize a posição do "M" ou do "O" no texto "Olá, Mundo!".

```
Text.PositionOfAny("Hello, World!", {"H", "W"})
```

`0`

Text.Proper

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Proper(text as nullable text, optional culture as nullable text) as nullable text
```

Sobre

Retorna o resultado do uso de maiúsculas somente da primeira letra de cada palavra no valor de texto `text`. Todas as outras letras são retornadas em minúsculas. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Use `Text.Proper` em uma frase simples.

```
Text.Proper("the QUICK BrOWn fOx jUmPs oVER tHe LAzy DoG")
```

`"The Quick Brown Fox Jumps Over The Lazy Dog"`

Text.Range

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Range(text as nullable text, offset as number, optional count as nullable number) as nullable text
```

Sobre

Retorna a substring do texto `text` encontrado no deslocamento `offset`. Um parâmetro opcional, `count`, pode ser incluído para especificar quantos caracteres serão retornados. Gerará um erro se não houver caracteres suficientes.

Exemplo 1

Localize a substring no texto "Olá, Mundo", começando no índice. 6.

```
Text.Range("Hello World", 6)
```

`"World"`

Exemplo 2

Localize a substring no texto "Olá, Mundo, Olá", começando no índice 6 e abrangendo 5 caracteres.

```
Text.Range("Hello World Hello", 6, 5)
```

`"World"`

Text.Remove

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Remove(text as nullable text, removeChars as any) as nullable text
```

Sobre

Retorna uma cópia do valor de texto `text` com todos os caracteres de `removeChars` removidos.

Exemplo 1

Remova os caracteres , e ; do valor de texto.

```
Text.Remove("a,b;c", {",",";"})
```

`"abc"`

Text.RemoveRange

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.RemoveRange(text as nullable text, offset as number, optional count as nullable number) as
nullable text
```

Sobre

Retorna uma cópia do valor de texto `text` com todos os caracteres da posição `offset` removidos. Um parâmetro opcional, `count`, pode ser usado para especificar o número de caracteres a serem removidos. O valor padrão de `count` é 1. Os valores de posição começam em 0.

Exemplo 1

Remova um caractere do valor de texto "ABEFC" na posição 2.

```
Text.RemoveRange("ABEFC", 2)
```

`"ABFC"`

Exemplo 2

Remova dois caracteres do valor de texto "ABEFC" começando na posição 2.

```
Text.RemoveRange("ABEFC", 2, 2)
```

`"ABC"`

Text.Repeat

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Repeat(text as nullable text, count as number) as nullable text
```

Sobre

Retorna um valor de texto composto pelo texto de entrada `text` repetido `count` vezes.

Exemplo 1

Repita o texto "a" cinco vezes.

```
Text.Repeat("a", 5)
```

`"aaaaa"`

Exemplo 2

Repita o texto "olámundo" três vezes.

```
Text.Repeat("helloworld.", 3)
```

`"helloworld.helloworld.helloworld."`

Text.Replace

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Replace(text as nullable text, old as text, new as text) as nullable text
```

Sobre

Retorna o resultado da substituição de todas as ocorrências do valor de texto `old` no valor de texto `text` pelo valor de texto `new`. Essa função diferencia maiúsculas de minúsculas.

Exemplo 1

Substitua cada ocorrência de "the" em uma frase por "a".

```
Text.Replace("the quick brown fox jumps over the lazy dog", "the", "a")
```

```
"a quick brown fox jumps over a lazy dog"
```

Text.ReplaceRange

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.ReplaceRange(text as nullable text, offset as number, count as number, newText as text) as
nullable text
```

Sobre

Retorna o resultado da remoção de um número de caracteres, `count`, do valor de texto `text` começando na posição `offset` e inserindo o valor de texto `newText` na mesma posição em `text`.

Exemplo 1

Substitua um único caractere na posição 2 do valor de texto "ABGF" pelo novo valor de texto "CDE".

```
Text.ReplaceRange("ABGF", 2, 1, "CDE")
```

`"ABCDEF"`

Text.Reverse

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Reverse(text as nullable text) as nullable text
```

Sobre

Inverte o `text` fornecido.

Exemplo 1

Inverta o texto "123".

```
Text.Reverse("123")
```

`"321"`

Text.Select

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Select(text as nullable text, selectChars as any) as nullable text
```

Sobre

Retorna uma cópia do valor de texto `text` com todos os caracteres que não estão na `selectChars` removida.

Exemplo 1

Selecione todos os caracteres no intervalo de 'a' a 'z' no valor de texto.

```
Text.Select("a,b;c", {"a".."z"})
```

`"abc"`

Text.Split

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Split(text as text, separator as text) as list
```

Sobre

Retorna uma lista de valores de texto resultantes da divisão de um valor de texto `text` com base no delimitador especificado, `separator`.

Exemplo 1

Crie uma lista com base no valor de texto delimitado por "|", "Name|Address|PhoneNumber".

```
Text.Split("Name|Address|PhoneNumber", "|")
```

Nome
Endereço
PhoneNumber

Text.SplitAny

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.SplitAny(text as text, separators as text) as list
```

Sobre

Retorna uma lista de valores de texto resultantes da divisão de um valor de texto `text` com base em qualquer caractere no delimitador especificado, `separators`.

Exemplo 1

Crie uma lista com base no valor de texto "Jamie|Campbell|Admin|Adventure Works| www.adventure-works.com".

```
Text.SplitAny("Jamie|Campbell|Admin|Adventure Works|www.adventure-works.com", "|")
```

Jamie
Campbell
Administrador
Adventure Works
www.adventure-works.com

Text.Start

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.Start(text as nullable text, count as number) as nullable text
```

Sobre

Retorna os `count` primeiros caracteres de `text` como um valor de texto.

Exemplo 1

Obtenha os cinco primeiros caracteres de "Olá, Mundo".

```
Text.Start("Hello, World", 5)
```

`"Hello"`

Text.StartsWith

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.StartsWith(text as nullable text, substring as text, optional comparer as nullable function)
as nullable logical
```

Sobre

Retornará true se o valor de texto `text` começar com o valor de texto `substring`.

- `text`: Um valor `text` que deve ser pesquisado
- `substring`: Um valor de `text` que é a substring a ser pesquisada em `substring`
- `comparer`: *[opcional]* Um `Comparer` usado para controlar a comparação. Por exemplo, `Comparer.OrdinalIgnoreCase` pode ser usado para executar pesquisas que não diferenciam maiúsculas de minúsculas

`comparer` é um `Comparer` usado para controlar a comparação. Os comparadores podem ser usados para fornecer comparações com detecção de localidade e cultura ou sem diferenciação de maiúsculas e minúsculas.
Os seguintes comparadores internos estão disponíveis na linguagem da fórmula:

- `Comparer.Ordinal`: Usado para executar uma comparação ordinal exata
- `Comparer.OrdinalIgnoreCase`: Usado para executar uma comparação ordinal exata sem diferenciação de maiúsculas e minúsculas
- `Comparer.FromCulture`: Usado para executar uma comparação com detecção de cultura

Exemplo 1

Verifique se o texto "Olá, Mundo" começa com o texto "olá".

```
Text.StartsWith("Hello, World", "hello")
```

`false`

Exemplo 2

Verifique se o texto "Olá, Mundo" começa com o texto "Olá".

```
Text.StartsWith("Hello, World", "Hello")
```

`true`

Text.ToBinary

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.ToBinary(text as nullable text, optional encoding as nullable number, optional
includeByteOrderMark as nullable logical) as nullable binary
```

Sobre

Codifica o valor de texto especificado, `text`, em um valor binário usando o `encoding` especificado.

Text.ToList

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.ToList(text as text) as list
```

Sobre

Retorna uma lista de valores de caractere do valor de texto `text` especificado.

Exemplo 1

Crie uma lista de valores de caractere do texto "Olá, Mundo".

```
Text.ToList("Hello World")
```

H
e
l
l
o
W
o
r
l
d

Text.Trim

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.Trim(text as nullable text, optional trim as any) as nullable text
```

Sobre

Retorna o resultado da remoção de todos os espaços em branco à esquerda e à direita do valor de texto `text`.

Exemplo 1

Remova o espaço em branco à esquerda e à direita de "a b c d".

```
Text.Trim("    a b c d   ")
```

`"a b c d"`

Text.TrimEnd

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.TrimEnd(text as nullable text, optional trim as any) as nullable text
```

Sobre

Retorna o resultado da remoção de todos os espaços em branco à direita do valor de texto `text`.

Exemplo 1

Remova o espaço em branco à direita de "a b c d".

```
Text.TrimEnd("    a b c d    ")
```

```
"    a b c d"
```

Text.TrimStart

09/05/2020 • 2 minutes to read

Sintaxe

```
Text.TrimStart(text as nullable text, optional trim as any) as nullable text
```

Sobre

Retorna o resultado da remoção de todos os espaços em branco à esquerda do valor de texto `text`.

Exemplo 1

Remova o espaço em branco à esquerda de "a b c d".

```
Text.TrimStart("   a b c d    ")
```

```
"a b c d    "
```

Text.Upper

08/05/2020 • 2 minutes to read

Sintaxe

```
Text.Upper(text as nullable text, optional culture as nullable text) as nullable text
```

Sobre

Retorna o resultado da conversão de todos os caracteres de `text` em maiúsculas. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha a versão em letras maiúsculas de "\aBcD".

```
Text.Upper("aBcD")
```

`"ABCD"`

TextEncoding.Ascii

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário ASCII.

TextEncoding.BigEndianUnicode

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário UTF16 big endian.

TextEncoding.Unicode

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário UTF16 little endian.

TextEncoding.Utf8

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário UTF8.

TextEncoding.Utf16

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário UTF16 little endian.

TextEncoding.Windows

09/05/2020 • 2 minutes to read

Sobre

Use para escolher o formato binário do Windows.

Funções de hora

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam valores de tempo.

Time

FUNÇÃO	DESCRIÇÃO
Time.EndOfHour	Retorna um valor de DateTime do fim da hora.
Time.From	Retorna um valor temporal de um valor.
Time.FromText	Retorna um valor temporal de um conjunto de formatos de data.
Time.Hour	Retorna o valor de hora de um valor de DateTime.
Time.Minute	Retorna o valor de minuto de um valor de DateTime.
Time.Second	Retorna o valor de segundo de um valor de DateTime
Time.StartOfHour	Retorna o primeiro valor da hora de um valor temporal.
Time.ToRecord	Retorna um registro que contém partes de valor de Date.
Time.ToText	Retorna um valor de texto de um valor temporal.
#time	Cria um valor temporal usando hora, minuto e segundo.

Time.EndOfHour

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.EndOfHour(dateTime as any) as any
```

Sobre

Retorna um valor `time`, `datetime` ou `datetimezone` que representa o fim da hora em `dateTime`, incluindo segundos fracionários. As informações de fuso horário são preservadas.

- `dateTime`: Um valor `time`, `datetime` ou `datetimezone` do qual o fim da fora é calculado.

Exemplo 1

Obtenha o fim da hora de 14/5/2011 05:00: 00 PM.

```
Time.EndOfHour(#datetime(2011, 5, 14, 17, 0, 0))
```

`#datetime(2011, 5, 14, 17, 59, 59.9999999)`

Exemplo 2

Obtenha o fim da hora de 17/5/2011 05:00: 00 PM -7:00.

```
Time.EndOfHour(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
```

`#datetimezone(2011, 5, 17, 5, 59, 59.9999999, -7, 0)`

Time.From

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.From(value as any, optional culture as nullable text) as nullable time
```

Sobre

Retorna um valor `time` do `value` especificado. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR"). Se o `value` fornecido for `null`, `Time.From` retornará `null`. Se o `value` fornecido for `time`, `value` será retornado. Os valores dos seguintes tipos podem ser convertidos em um valor `time`:

- `text`: Um valor `time` da representação textual. Confira `Time.FromText` para obter detalhes.
- `datetime`: O componente de hora do `value`.
- `datetimezone`: O componente de hora do equivalente de datetime local de `value`.
- `number`: Um `time` equivalente ao número de dias fracionários expressos por `value`. Se `value` for negativo ou maior ou igual a 1, um erro será retornado.

Se `value` for de qualquer outro tipo, um erro será retornado.

Exemplo 1

Converter `0.7575` em um valor `time`.

```
Time.From(0.7575)
```

`#time(18, 10, 48)`

Exemplo 2

Converter `#datetime(1899, 12, 30, 06, 45, 12)` em um valor `time`.

```
Time.From(#datetime(1899, 12, 30, 06, 45, 12))
```

`#time(06, 45, 12)`

Time.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.FromText(text as nullable text, optional culture as nullable text) as nullable time
```

Sobre

Cria um valor de `time` com base em uma representação textual, `text`, seguindo o padrão de formato ISO 8601. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

- `Time.FromText("12:34:12")` // Hora, hh:mm:ss
- `Time.FromText("12:34:12.1254425")` // hh:mm:ss.nnnnnnn

Exemplo 1

Converta `"10:12:31am"` em um valor temporal.

```
Time.FromText("10:12:31am")
```

`#time(10, 12, 31)`

Exemplo 2

Converta `"1012"` em um valor temporal.

```
Time.FromText("1012")
```

`#time(10, 12, 00)`

Exemplo 3

Converta `"10"` em um valor temporal.

```
Time.FromText("10")
```

`#time(10, 00, 00)`

Time.Hour

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.Hour(dateTime as any) as nullable number
```

Sobre

Retorna o componente de hora do valor de `time`, `datetime` ou `datetimezone` fornecido, `dateTime`.

Exemplo 1

Localize a hora em #datetime (2011, 12, 31, 9, 15, 36).

```
Time.Hour(#datetime(2011, 12, 31, 9, 15, 36))
```

`9`

Time.Minute

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.Minute(dateTime as any) as nullable number
```

Sobre

Retorna o componente de minuto do valor `time`, `datetime` ou `datetimezone` fornecido, `dateTime`.

Exemplo 1

Localize o minuto em #datetime (2011, 12, 31, 9, 15, 36).

```
Time.Minute(#datetime(2011, 12, 31, 9, 15, 36))
```

`15`

Time.Second

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.Second(dateTime as any) as nullable number`
```

Sobre

Retorna o componente de segundo do valor de `time`, `datetime` ou `datetimezone` fornecido, `dateTime`.

Exemplo 1

Localize o segundo valor por meio de um valor de datetime.

```
Time.Second(#datetime(2011, 12, 31, 9, 15, 36.5))
```

`36.5`

Time.StartOfHour

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.StartOfHour(dateTime as any) as any
```

Sobre

Retorna o primeiro valor da hora dado um tipo `time`, `datetime` ou `datetimezone`.

Exemplo 1

Localizar o início da hora de 10 de outubro de 2011, 8h10min32 (`#datetime(2011, 10, 10, 8, 10, 32)`).

```
Time.StartOfHour(#datetime(2011, 10, 10, 8, 10, 32))
```

`#datetime(2011, 10, 10, 8, 0, 0)`

Time.ToRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Time.ToRecord(time as time) as record
```

Sobre

Retorna um registro que contém as partes do valor temporal especificado, `time`.

- `time`: Um valor `time` com base no qual o registro das partes deve ser calculado.

Exemplo 1

Converta o valor de `#time(11, 56, 2)` em um registro contendo valores de Hora.

```
Time.ToRecord(#time(11, 56, 2))
```

HORA	11
MINUTE	56
SECOND	2

Time.ToText

26/05/2020 • 2 minutes to read

Sintaxe

```
Time.ToText(time as nullable time, optional format as nullable text, optional culture as nullable text) as nullable text
```

Sobre

Retorna uma representação textual de `time`. Um `format` opcional pode ser fornecido para personalizar a formatação do texto. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Exemplo 1

Obtenha uma representação textual #time(11, 56, 2).

```
Time.ToText(#time(11, 56, 2))
```

`"11:56 AM"`

Exemplo 2

Obtenha uma representação textual de #time(11, 56, 2) com opção de formato.

```
Time.ToText(#time(11, 56, 2), "hh:mm")
```

`"11:56"`

#time

09/05/2020 • 2 minutes to read

Sintaxe

```
#time(hour as number, minute as number, second as number) as time
```

Sobre

Cria um valor temporal usando números inteiros de hora `hour`, minuto `minute` e segundo (fracionário) `second`. Gera um erro se isto não é verdadeiro:

- 0 ≤ hora ≤ 24
- 0 ≤ minuto ≤ 59
- 0 ≤ segundo ≤ 59
- se a hora for 24, o minuto e o segundo deverão ser 0

Funções de tipo

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam valores de tipo.

Tipo

FUNÇÃO	DESCRIÇÃO
Type.AddTableKey	Adicione uma chave a um tipo de tabela.
Type.ClosedRecord	O tipo fornecido deve ser um tipo de registro que retorna uma versão fechada do tipo de registro determinado (ou do mesmo tipo, se já está fechado)
Type.Facets	Retorna as facetas de um tipo.
Type.ForFunction	Cria um tipo de função com base no valor especificado de .
Type.ForRecord	Retorna um tipo de Registro de um registro de campos.
Type.FunctionParameters	Retorna um registro com valores de campo definidos para o nome dos parâmetros de um tipo de função e seus valores definidos para seus tipos correspondentes.
Type.FunctionRequiredParameters	Retorna um número que indica o número mínimo de parâmetros necessários para invocar um tipo de função.
Type.FunctionReturn	Retorna um tipo retornado por um tipo de função.
Type.Is	Type.Is
Type.IsNullable	Retorna true se um tipo é um tipo que permite valor nulo; caso contrário, false.
Type.IsOpenRecord	Retorna se um tipo de registro for aberto.
Type.ListItem	Retorna um tipo de item de um tipo de lista.
Type.NonNullable	Retorna o tipo que não permite valor nulo de um tipo.
Type.OpenRecord	Retorna uma versão aberta de um tipo de registro (ou do mesmo tipo, se já está aberto).
Type.RecordFields	Retorna um registro descrevendo os campos de um tipo de registro com cada campo do tipo de registro retornado tendo um nome correspondente e um valor que é o registro do formulário [Tipo = tipo, Opcional = lógico].
Type.ReplaceFacets	Substitui as facetas de um tipo.

FUNÇÃO	DESCRIÇÃO
Type.ReplaceTableKeys	Substitui as chaves em um tipo de tabela.
Type.TableColumn	Retorna o tipo de uma coluna em uma tabela.
Type.TableKeys	Retorna chaves de um tipo de tabela.
Type.TableRow	Retorna um tipo de linha de um tipo de tabela.
Type.TableSchema	Retorna uma tabela que contém uma descrição das colunas (p.ex., o esquema) do tipo de tabela especificado.
Type.Union	Retorna a união de uma lista de tipos.

Type.AddTableKey

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.AddTableKey(table as type, columns as list, isPrimary as logical) as type
```

Sobre

Adiciona uma chave ao tipo de tabela especificado.

Type.ClosedRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ClosedRecord(type as type) as type
```

Sobre

Retorna uma versão fechada do `record` `type` fornecido (ou do mesmo tipo, se já está fechado).

Exemplo 1

Crie uma versão fechada de `type [ A = number,…]`.

```
Type.ClosedRecord(type [A = number, ...])
```

`type [A = number]`

Type.Facets

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.Facets(type as type) as record
```

Sobre

Retorna um registro contendo as facetas de `type`

Type.ForFunction

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ForFunction(signature as record, min as number) as type
```

Sobre

Cria um `function type` de `signature`, um registro de `ReturnType` e `Parameters` e `min`, o número mínimo de argumentos necessários para invocar a função.

Exemplo 1

Cria o tipo para uma função que aceita um parâmetro de número denominado X e retorna um número.

```
Type.ForFunction([ReturnType = type number, Parameters = [X = type number]], 1)
```

```
type function (X as number) as number
```

Type.ForRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ForRecord(fields as record, open as logical) as type
```

Sobre

Retorna um tipo que representa registros com restrições de tipo específicas em campos.

Type.FunctionParameters

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.FunctionParameters(type as type) as record
```

Sobre

Retorna um registro com valores de campo definidos para o nome dos parâmetros de `type`, e seus valores definidos para seus tipos correspondentes.

Exemplo

Localize os tipos dos parâmetros para a função `(x as number, y as text)`.

```
Type.FunctionParameters(type function (x as number, y as text) as any)
```

X	[Type]
A	[Type]

Type.FunctionRequiredParameters

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.FunctionRequiredParameters(type as type) as number
```

Sobre

Retorna um número que indica o número mínimo de parâmetros necessários para invocar a entrada `type` da função.

Exemplo 1

Localize o número de parâmetros necessários para a função `(x as number, optional y as text)`.

```
Type.FunctionRequiredParameters(type function (x as number, optional y as text) as any)
```

```
1
```

Type.FunctionReturn

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.FunctionReturn(type as type) as type
```

Sobre

Retorna um tipo retornado por uma função `type`.

Exemplo 1

Localize o tipo de retorno de `() as any)`.

```
Type.FunctionReturn(type function () as any)
```

`type any`

Type.Is

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.Is(type1 as type, type2 as type) as logical
```

Sobre

Type.Is

Type.IsNullable

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.IsNullable(type as type) as logical
```

Sobre

Retorna `true` se um tipo é `nullable`; caso contrário, `false`.

Exemplo 1

Determine se `number` é anulável.

```
Type.IsNullable(type number)
```

`false`

Exemplo 2

Determine se `type nullable number` é anulável.

```
Type.IsNullable(type nullable number)
```

`true`

Type.IsOpenRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.IsOpenRecord(type as type) as logical
```

Sobre

Retorna um `logical` indicando se um registro `type` está aberto.

Exemplo 1

Determine se o registro `type [ A = number, ...]` está aberto.

```
Type.IsOpenRecord(type [A = number, ...])
```

`true`

Type.ListItem

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ListItem(type as type) as type
```

Sobre

Retorna um tipo de item de uma lista `type`.

Exemplo 1

Encontre o tipo de item na lista `{number}`.

```
Type.ListItem(type {number})
```

`type number`

Type.NonNullable

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.NonNullable(type as type) as type
```

Sobre

Retorna o tipo não `nullable` do `type`.

Exemplo 1

Retornar o tipo que não permite valor nulo de `type nullable number`.

```
Type.NonNullable(type nullable number)
```

`type number`

Type.OpenRecord

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.OpenRecord(type as type) as type
```

Sobre

Retorna uma versão aberta do `record` `type` fornecido (ou do mesmo tipo, se já está aberto).

Exemplo 1

Crie uma versão aberta de `type [ A = number]`.

```
Type.OpenRecord(type [A = number])
```

`type [A = number, ...]`

Type.RecordFields

08/05/2020 • 2 minutes to read

Sintaxe

```
Type.RecordFields(type as type) as record
```

Sobre

Retorna um registro que descreve os campos de um registro `type`. Cada campo do tipo de registro retornado tem um nome e um valor correspondentes na forma de um registro `[ Type = type, Optional = logical ]`.

Exemplo

Localize o nome e o valor do registro `[ A = number, optional B = any]`.

```
Type.RecordFields(type [A = number, optional B = any])
```

A	[Registro]
B	[Registro]

Type.ReplaceFacets

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ReplaceFacets(type as type, facets as record) as type
```

Sobre

Substitui as facetas de `type` pelas facetas contidas no registro `facets`.

Type.ReplaceTableKeys

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.ReplaceTableKeys(tableType as type, keys as list) as type
```

Sobre

Retorna um novo tipo de tabela com todas as chaves substituídas pela lista de chaves especificada.

Type.TableColumn

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.TableColumn(tableType as type, column as text) as type
```

Sobre

Retorna o tipo da coluna `column` no tipo de tabela `tableType`.

Type.TableKeys

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.TableKeys(tableType as type) as list
```

Sobre

Retorna a lista de chaves possivelmente vazias do tipo de tabela especificado.

Type.TableRow

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.TableRow(table as type) as type
```

Sobre

Type.TableRow

Type.TableSchema

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.TableSchema(tableType as type) as table
```

Sobre

Retorna uma tabela que descreve as colunas de `tableType`.

Type.Union

09/05/2020 • 2 minutes to read

Sintaxe

```
Type.Union(types as list) as type
```

Sobre

Retorna a união dos tipos em `types`.

Funções de URI

08/05/2020 • 2 minutes to read

Essas funções criam e manipulam cadeias de caracteres de consulta de URI.

Uri

FUNÇÃO	DESCRIÇÃO
Uri.BuildQueryString	Monte um registro em uma cadeia de caracteres de consulta da URI.
Uri.Combine	Retorna um URI com base na combinação das partes base e relativa.
Uri.EscapeDataString	Codifica caracteres especiais de acordo com o RFC 3986.
Uri.Parts	Retorna um valor de registro com os campos definidos como as partes de um valor de texto do URI.

Uri.BuildQueryString

09/05/2020 • 2 minutes to read

Sintaxe

```
Uri.BuildQueryString(query as record) as text
```

Sobre

Monte o registro `query` em uma cadeia de caracteres de consulta da URI, efetuando o escape dos caracteres conforme necessário.

Exemplo

Codifique uma cadeia de caracteres de consulta que contém alguns caracteres especiais.

```
Uri.BuildQueryString([a = "1", b = "+$"])
```

```
"a=1&b=%2B%24"
```

Uri.Combine

09/05/2020 • 2 minutes to read

Sintaxe

```
Uri.Combine(baseUri as text, relativeUri as text) as text
```

Sobre

Retorna um URI absoluto que é a combinação da entrada `baseUri` e `relativeUri`.

Uri.EscapeDataString

09/05/2020 • 2 minutes to read

Sintaxe

```
Uri.EscapeDataString(data as text) as text
```

Sobre

Codifica caracteres especiais na entrada `data` de acordo com as regras do RFC 3986.

Exemplo

Codifique os caracteres especiais em "+money$".

```
Uri.EscapeDataString("+money$")
```

`"%2Bmoney%24"`

Uri.Parts

08/05/2020 • 2 minutes to read

Sintaxe

```
Uri.Parts(absoluteUri as text) as record
```

Sobre

Retorna as partes do `absoluteUri` de entrada como um registro, contendo valores como Scheme, Host, Port, Path, Query, Fragment, UserName e Password.

Exemplo 1

Localize as partes do URI absoluto "www.adventure-works.com".

```
Uri.Parts("www.adventure-works.com")
```

SCHEME	http
HOST	www.adventure-works.com
PORTA	80
CAMINHO	/
CONSULTAR	[Registro]
FRAGMENT	
USERNAME	
SENHA	

Exemplo 2

Decodifique uma cadeia de caracteres codificada por percentual.

```
let
    UriUnescapeDataString = (data as text) as text => Uri.Parts("http://contoso?a=" & data)[Query][a]
in
    UriUnescapeDataString("%2Bmoney%24")
```

`"+money$"`

Funções de valor

19/10/2020 • 4 minutes to read

Essas funções avaliam e executam operações em valores.

Valores

FUNÇÃO	DESCRIÇÃO
Value.Compare	Retorna 1, 0 ou -1 com base em value1 ser maior que, igual a ou menor que value2. Uma função de comparador opcional pode ser fornecida.
Value.Equals	Especifica se os dois valores são iguais.
Value.Expression	Retorna um AST que representa a expressão do valor.
Value.NativeQuery	Avalia uma consulta em relação a um destino.
Value.NullableEquals	Retorna um valor lógico ou nulo com base em dois valores.
Value.Optimize	Se o valor representa uma consulta que pode ser otimizada, retorna a consulta otimizada. Caso contrário, retorna o valor.
Value.Type	Retorna o tipo do valor especificado.

Operações aritméticas

FUNÇÃO	DESCRIÇÃO
Value.Add	Retorna a soma dos dois valores.
Value.Divide	Retorna o resultado da divisão do primeiro valor pelo segundo.
Value.Multiply	Retorna o produto dos dois valores.
Value.Subtract	Retorna a diferença dos dois valores.

Parâmetros aritméticos

FUNÇÃO	DESCRIÇÃO
Precision.Double	Um parâmetro opcional para que os operadores aritméticos internos especifiquem a precisão dupla.
Precision.Decimal	Um parâmetro opcional para que os operadores aritméticos internos especifiquem a precisão decimal.

Tipos de parâmetro

TIPO	DESCRIÇÃO
Value.As	Value.As é a função correspondente ao operador as na linguagem de fórmula. O valor da expressão como tipo declara que o valor de um argumento de valor é compatível com o tipo, de acordo com o operador is. Se não for compatível, um erro será gerado.
Value.Is	Value.Is é a função correspondente ao operador is na linguagem de fórmula. O valor da expressão é tipo e retornará true se o tipo de valor atribuído for compatível com o tipo e retornará false se o tipo de valor atribuído for incompatível com o tipo.
Value.ReplaceType	Um tipo pode ser atribuído a um valor usando Value.ReplaceType. Value.ReplaceType retorna um novo valor com o tipo atribuído ou gera um erro se o novo tipo for incompatível com o tipo primitivo nativo do valor. Em particular, a função gera um erro quando é feita uma tentativa de atribuir um tipo abstrato, como any. Ao substituir o tipo de um registro, o novo tipo deve ter o mesmo número de campos e os novos campos substituem os campos antigos pela posição ordinal, não pelo nome. Da mesma forma, ao substituir o tipo de uma tabela, o novo tipo deve ter o mesmo número de colunas e as novas colunas substituem as colunas antigas pela posição ordinal.

IMPLEMENTAÇÃO	DESCRIÇÃO
DirectQueryCapabilities.From	DirectQueryCapabilities.From
Embedded.Value	Acessa um valor por nome em um mashup inserido.
Value.Firewall	Value.Firewall
Variable.Value	Variable.Value
SqlExpression.SchemaFrom	SqlExpression.SchemaFrom
SqlExpression.ToExpression	SqlExpression.ToExpression

Metadados

FUNÇÃO	DESCRIÇÃO
Value.Metadata	Retorna um registro contendo os metadados da entrada.
Value.RemoveMetadata	Remove os metadados do valor e retorna o valor original.
Value.ReplaceMetadata	Substitui os metadados em um valor pelo novo registro de metadados fornecido e retorna o valor original com os novos metadados anexados.

Linhagem

FUNÇÃO	DESCRIÇÃO
Graph.Nodes	Esta função destina-se somente a uso interno.
Value.Lineage	Esta função destina-se somente a uso interno.
Value.Traits	Esta função destina-se somente a uso interno.

DirectQueryCapabilities.From

09/05/2020 • 2 minutes to read

Sintaxe

```
DirectQueryCapabilities.From(value as any) as table
```

Sobre

DirectQueryCapabilities.From

Embedded.Value

09/05/2020 • 2 minutes to read

Sintaxe

```
Embedded.Value(value as any, path as text) as any
```

Sobre

Acessa um valor por nome em um mashup inserido.

Graph.Nodes

20/10/2020 • 2 minutes to read

Sintaxe

```
Graph.Nodes(graph as record) as list
```

Sobre

Esta função destina-se somente a uso interno.

Precision.Decimal

09/05/2020 • 2 minutes to read

Sobre

Um parâmetro opcional para que os operadores aritméticos internos especifiquem a precisão decimal.

Precision.Double

09/05/2020 • 2 minutes to read

Sobre

Um parâmetro opcional para que os operadores aritméticos internos especifiquem a precisão dupla.

SqlExpression.SchemaFrom

09/05/2020 • 2 minutes to read

Sintaxe

```
SqlExpression.SchemaFrom(schema as any) as any
```

Sobre

SqlExpression.SchemaFrom

SqlExpression.ToExpression

09/05/2020 • 2 minutes to read

Sintaxe

```
SqlExpression.ToExpression(sql as text, environment as record) as text
```

Sobre

SqlExpression.ToExpression

Value.Add

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Add(value1 as any, value2 as any, optional precision as nullable number) as any
```

Sobre

Retorna a soma de `value1` e `value2`. Um parâmetro `precision` opcional pode ser especificado. Por padrão, `Precision.Double` é usado.

Value.As

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.As(value as any, type as type) as any
```

Sobre

Value.As

Value.Compare

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Compare(value1 as any, value2 as any, optional precision as nullable number) as number
```

Sobre

Retorna -1, 0 ou 1, dependendo da condição do primeiro valor: se ele é menor que, igual a ou maior que o segundo.

Value.Divide

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Divide(value1 as any, value2 as any, optional precision as nullable number) as any
```

Sobre

Retorna o Resultado da divisão de `value1` por `value2`. Um parâmetro `precision` opcional pode ser especificado. Por padrão, `Precision.Double` é usado.

Value.Equals

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Equals(value1 as any, value2 as any, optional precision as nullable number) as logical
```

Sobre

Retornará true se o valor `value1` for igual ao valor `value2`; do contrário, false será retornado.

Value.Expression

30/09/2020 • 2 minutes to read

Sintaxe

```
Value.Expression(value as any) as nullable record
```

Sobre

Retorna um AST que representa a expressão do valor.

Value.Firewall

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Firewall(key as text) as any
```

Sobre

Value.Firewall

Value.FromText

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.FromText(text as any, optional culture as nullable text) as any
```

Sobre

Decodifica um valor de uma representação textual, de um `text` e o interpreta como um valor com tipo apropriado. `Value.FromText` usa um valor de texto e retorna um número, um valor lógico, um valor nulo, um valor de data/hora, um valor de duração ou um valor de texto. O valor de texto vazio é interpretado como um valor nulo. Uma `culture` opcional também pode ser fornecida (por exemplo, "pt-BR").

Value.Is

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Is(value as any, type as type) as logical
```

Sobre

Value.Is

Value.Lineage

20/10/2020 • 2 minutes to read

Sintaxe

```
Value.Lineage(value as any) as any
```

Sobre

Esta função destina-se somente a uso interno.

Value.Metadata

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Metadata(value as any) as any
```

Sobre

Retorna um registro contendo os metadados da entrada.

Value.Multiply

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Multiply(value1 as any, value2 as any, optional precision as nullable number) as any
```

Sobre

Retorna o produto da multiplicação de `value1` por `value2`. Um parâmetro `precision` opcional pode ser especificado. Por padrão, `Precision.Double` é usado.

Value.NativeQuery

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.NativeQuery(target as any, query as text, optional parameters as any, optional options as
nullable record) as any
```

Sobre

Avalia `query` em relação a `target` usando os parâmetros especificados em `parameters` e as opções especificadas em `options`.

A saída da consulta é definida por `target`.

`target` fornece o contexto para a operação descrita por `query`.

`query` descreve a consulta a ser executada em `target`. `query` é expresso de maneira específica para `target` (por exemplo, uma instrução T-SQL).

O valor opcional de `parameters` pode conter uma lista ou um registro, conforme apropriado, para fornecer os valores de parâmetro esperados por `query`.

O registro de `options` opcional pode conter opções que afetam o comportamento de avaliação de `query` em relação a `target`. Essas opções são específicas de `target`.

Value.NullableEquals

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.NullableEquals(value1 as any, value2 as any, optional precision as nullable number) as
nullable logical
```

Sobre

Retornará nulo se um argumento `value1`, `value2` for nulo, caso contrário, equivalente a Value.Equals.

Value.Optimize

30/09/2020 • 2 minutes to read

Sintaxe

```
Value.Optimize(value as any) as any
```

Sobre

Se `value` representar uma consulta que pode ser otimizada, a consulta otimizada será retornada. Caso contrário, retornará `value`.

Value.RemoveMetadata

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.RemoveMetadata(value as any, optional metaValue as any) as any
```

Sobre

Remove a entrada dos metadados.

Value.ReplaceMetadata

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.ReplaceMetadata(value as any, metaValue as any) as any
```

Sobre

Substitui as informações de metadados da entrada.

Value.ReplaceType

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.ReplaceType(value as any, type as type) as any
```

Sobre

Value.ReplaceType

Value.Subtract

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Subtract(value1 as any, value2 as any, optional precision as nullable number) as any
```

Sobre

Retorna a diferença de `value1` e `value2`. Um parâmetro `precision` opcional pode ser especificado. Por padrão, `Precision.Double` é usado.

Value.Traits

20/10/2020 • 2 minutes to read

Sintaxe

```
Value.Traits(value as any) as table
```

Sobre

Esta função destina-se somente a uso interno.

Value.Type

09/05/2020 • 2 minutes to read

Sintaxe

```
Value.Type(value as any) as type
```

Sobre

Retorna o tipo do valor especificado.

Variable.Value

09/05/2020 • 2 minutes to read

Sintaxe

```
Variable.Value(identifier as text) as any
```

Sobre

Variable.Value