POWER QUERY

Tradutor Vitor Klein

O que é o **Power Query**?

"Ambiente integrado de desenvolvimento em linguagem M"

Componentes

› **Menu Superior –** um menu de configurações e etapas pré-configuradas em linguagem M pelo Power Query para conveniência do usuário.
› **Consultas –** uma expressão em linguagem M. Consultas podem ser organizadas em grupos
› **Primitivas –** um valor primitivo, que pode ser numérico, lógico, data, texto ou **nulo**. Um valor nulo pode ser utilizado para indicar a ausência de dados.
› **Listas –** a lista é uma sequência ordenada de valores. M suporta infinitas listas. Os caracteres **"{"** e **"}"** indicam o início e fim de uma lista.
› **Registro –** um registro é um conjunto de campos, composto por um par de valores que formam o nome e o dado. O nome é um texto único dentro do registro.
› **Tabela –** uma tabela é um conjunto de valores organizados em linhas e colunas nomeadas. Tabelas podem ser operadas como uma lista de registros, ou como um registro de listas. Tabela[campos] (sintaxe de referência à campos nos registros) retorna uma lista de valores naquele campo. 'Tabela{i}' (sintaxe de acesso ao índice de lista) retorna um registro representando a linha da tabela.
› **Função –** uma função é uma expressão que, quando chamada utilizando argumentos, cria um novo valor. Funções são escritas com seus parâmetros entre parênteses, seguidos do símbolo de ir para "=>", seguido da expressão que define a função. Essa expressão normalmente se refere à parâmetros por nomes. Também há funções sem parâmetros.
› **Parâmetros –** os parâmetros armazenam um valor que pode ser utilizado nas transformações. Além do nome do parâmetro e do valor que ele armazena, ele também tem outras propriedades que fornecem metadados. A grande vantagem dos parâmetros é que eles podem ser alterados do Power BI Service, sem a necessidade de intervenção direta no conjunto de dados. A sintaxe dos parâmetros é igual a de uma consulta regular, sendo a única diferença que os metadados seguem um formato específico.
› **Barra de fórmulas –** demonstra a etapa atualmente carregada e permite que você a edite. Para visualizá-la, ela deve estar ativa no menu superior em **Exibição**.
› **Configurações de consulta –** configurações que incluem a possibilidade de editar o nome e a descrição da consulta. Também contém uma visão geral de todas as etapas aplicadas. Essas etapas são variáveis definidas em uma expressão "let" e são representadas por nomes de variáveis
› **Pré-visualização dos dados –** um componente que demonstra uma amostra dos dados na etapa da transformação selecionada.
› **Barra de Status –** É a barra localizada no canto inferior da tela. Contém informações sobre o status das linhas, colunas e data da última visualização. Além disso, há informações sobre o perfil das colunas. Aqui temos a possiblidade de alterar as informações baseadas nas 1000 primeiras linhas para toda a tabela.

Funções no Power Query

Saber utilizar funções é fundamental quando trabalhamos com linguagens funcionais como a linguagem **M**. Funções são chamadas com parentêses.
› **Shared –** É uma palavra-chave que carrega todas as funções (incluindo ajuda e exemplos) e enumeradores no resultado. A chamada de uma função é realizada dentro de uma consulta vazia usando **= #shared**

```
= #shared
```

Funções podem ser divididas em duas categorias:
› Pré fabricadas – por exemplo: Date.From()
› Customizadas – são funções preparadas pelo usuário através da extensão e notação "**()=>** ", onde os argumentos que serão requisitados para validação da função podem ser colocados entre parênteses. Quando utilizar múltiplos argumentos, é necessário separá-los através de um delimitador.

Valores

Cada tipo de valor é associado a uma sintaxe literal, um conjunto de valores do mesmo tipo, um cojunto de operadores definidos sobre aquele conjunto de valores, e um tipo intrínseco atribuído a valores recém-criados.
› **Nulo –** null
› **Lógico –** verdadeiro (true) ou falso (false)
› **Numérico –** 1, 2, 3, ...
› **Hora –** #time(HH,MM,SS)
› **Data –** #date(yyyy,mm,ss)
› **DateTime –** #datetime(yyyy,mm,dd,HH,MM,SS)
› **DateTimeZone –** #datetimezone(yyyy,mm,dd,HH,MM,SS,9,00)
› **Duração –** #duration(DD,HH,MM,SS)
› **Texto –** "texto"
› **Binário –** #binary("link")
› **Lista –** {1, 2, 3}
› **Registro –** [A = 1, B = 2]
› **Tabela –** #table({columns},{{first row contenct},{}...})*
› **Função –** (x) => x + 1
› **Tipo –** type [number], type table [A = any, B = text]
* O índice da primeira linha da tabela é o mesmo para os registros na folha

Operadores

Há vários operadores na linguagem M, mas nem todos podem ser utilizados para qualquer tipo de dado.
› **Operadores primários**
› **(x)** – Expressão entre parênteses
› **x[i]** – Referência de campo. Retorna o valor de um registro, lista de valores de uma tabela.
› **x{i}** – Acesso do item*. Retorna valor de lista, registro de tabela.
"Colocando "**?**" depois do operador retorna nulo se o índice não estiver na lista"
› **x(...)** – Chamada da função
› **{1 .. 10}** – Criação automática de uma lista de 1 a 10
› **...** – Não implementado
› **Operadores matemáticos** – **+, -, *, /**
› **Operadores relacionais**
› **>** , **>=** – maior que, maior que ou igual a
› **<** , **<=** – menor que, menor que ou igual a
› **=** , **<>** – é igual, é diferente de. O operador de igual retorna verdadeiro quando **null =null**
› **Operadores lógicos**
› **and** – operador de interseção
› **or** – operador de união
› **not** – negação lógica
› **Operadores de tipo**
› **as** – É um tipo primitivo que permite valor nulo ou então resulta em erro
› **is** – Testar se o tipo primitivo que permite valor nulo é compatível
› **Metadata** – a palavra **meta** determina metadados de um valor. Exemplo de metadados para uma variável **x:**
"x meta y" ou **"x meta [name = x, value = 123,...]"**
Dentro do Power Query, a prioridade dos operadores é respeitada. Por exemplo, "**X + Y * Z**" será calculado como "**X + (Y*Z)**"

Comentários

A linguagem M suporta **dois** tipos de comentários:
› Comentário de linha única – podem ser criados com **//** antes do código
› Atalho: **CTRL + ´**
› Comentários de múltiplas linhas – podem ser criados com **/*** antes do código e ***/** depois do código
› Atalho: **ALT + SHIFT + A**

Expressão let

A expressão let é utilizada para capturar o valor de um cálculo intermediário em uma variável. Essas variáveis são locais, dentro do escopo da expressão let. A construção da expressão se parece com:

```
let
  Nome_da_variavel = <expressão>,
  VariavelRetorno = <função>(Nome_da_variavel )
in
VariavelRetorno
```

Quando avaliada, as seguintes regras são aplicadas:

› Expressões em variáveis definem um novo escopo contendo identificadores da produção de lista de variáveis e devem estar presentes quando validando termos dentro da lista de variáveis. A expressão na lista de variáveis podem ser referenciadas entre elas
› Todas as variáveis devem ser avaliadas antes que a expressão let.
› Se expressões em variáveis não estiverem disponíveis, a expressão let não será avaliada
› Erros que ocorrerem durante a validação da consulta propagam como um erro para consultas relacionadas.

Condições

No Power Query também há uma expressão '**if**', que, com base nas condições inseridas, decide se o resultado será uma expressão da condição verdadeira ou falsa.

Sintaxe para a expressão if:
if <predicate> **then** < true-expression > **else** < false-expression >
"**Else** é obrigatório na expressão de condição em M."

Condição de entrada:
If x > 2 **then** 1 **else** 0
If [Month] > [Fiscal_Month] **then** true **else** false

A expressão if é a única condicional em M. Se você tem múltiplas condições para testar, você deve incorporá-las assim:

```
if <predicate>
  then < true-expression >
  else if <predicate>
    then < false-true-expression >
    else  < false-false-expression >
```

Para validação das condições, as seguinte regras são aplicadas:

› Se o valor criado pela validação da condição dentro do if não for um valor lógico, um erro "**Expression.Error**" aparecerá.
› A expressão caso verdadeiro é executada apenas se a condição if for verdadeira. Caso contrário, a expressão caso **falso** é executada**.**
› Se expressões em variáveis não estiverem disponíveis, elas não devem ser avaliadas
› O erro que ocorre durante a avaliação da condição será repassado adiante resultando em uma falha da consulta toda ou com valores "**Error**" nos registros.

A expressão try ... otherwise

Capturar erros é possível, por exemplo, utilizando a expressão try. É realizada uma tentativa de validar a expressão depois da palavra **try**. Se um erro acontecer durante essa validação, a expressão depois da palavra **otherwise** é aplicada
Exemplo da sintaxe:
try Date.From([textDate]) **otherwise** null

Função customizada

Exemplos de funções customizadas:
(x, y) => Number.From(**x**) + Number.From(**y**)

```
(x) =>
let
   out = Number.From(x) +
   Number.From(Date.From(DateTime.LocalNow()))
in
   out
```

Existem **dois** tipos de argumento de entrada para a função:
› **Obrigatórios –** Todos os argumentos comuns escritos em (). Sem esses argumentos, a função não pode ser chamada.
› **Opcionais –** Pode ou não estar na entrada da função. Marque o parâmetro como **opcional** colocando a palavra "**Optional**" antes do nome do argumento. Por Exemplo **(optional x)**. Se não houver o preenchimento de um argumento opcional, o mesmo acontecerá para os cálculos, mas seu valor será nulo.
Os argumentos opcionais devem estar depois dos obrigatórios.

Argumentos podem ser escritos com '**as <type>**' para indicar o tipo requisitado para aquele argumento. Essa função resultará em erro caso sejam determinados argumentos com o tipo errado. Funções também podem ter seus resultados renomeados. Isso pode ser feito assim:
(x as number, y as text) as logical => <expressão>

Os resultados das funções podem ser diferentes. A saída pode ser uma sheet, uma tabela, um valor e também outras funções. Isso significa que uma função pode criar outra função. Podem ser escritas assim:

let first = **(x)=> () =>** let out = {1..x} in out in first

Quando as funções são validadas, deve-se considerar:

› Erros causados pela avaliação das expressões em uma lista de expressões irão propagar tanto como uma falha ou um valor "**Error**"
› O número de argumentos criados na lista de argumentos deve ser compatível com o argumento formal da função, ou um erro "**Expression.Error**" acontecerá

Funções recursivas

Para funções recursivas, é necessário utilizar o caractere "**@**", que refere-se a função dentro do cálculo. Uma típica função recursiva é a fatorial, que pode ser escrita assim:

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

Each

Funções podem ser chamadas com argumentos específicos. Porém, se a função necessita ser executada para cada linha, tabela inteira, ou uma coluna inteira em uma tabela, é necessário anexar a palavra each ao código. Como o nome sugere, para cada registro, ele aplica as etapas chamadas. Each não é obrigatório. Simplesmente torna-se mais fácil de definir a função linha a linha para funções que requerem uma função como argumento

Simplificação Sintática

› Each é essenssialmente uma abreviação para declarar funções sem tipo, utilizando um único parâmetro formal. Por isso, as seguintes notações são semanticamente equivalentes:

```
let
  Source = ...,
  addColumn = Table.AddColumn(Source, „NewName", each [field1] + 1)
in
  addColumn
```

```
let
  Source = ...,
  add1ToField1 = (_) => [field1] + 1,
  addColumn(Source,"NewName",add1ToField1)
in
```

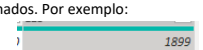
A segunda parte do código utiliza os colchetes para simplificar o acesso de um campo de um registro denominado '_'.

Cruzamento de consulta

Também chamado de query folding. Quando utilizada, as etapas no Power Query são compostas em uma consulta única, que é implementada diretamente na base de dados. As bases que suportam o query folding são bases que suportam o conceito de linguagem de consulta, como os bancos relacionais. Isso significa que, por exemplo, aquivos CSV ou XML, chamados flat files, não suportam o query folding. Sendo assim, a transformação não é executada até que os dados sejam completamente carregados, mas ainda é possível transformá-los. Infelizmente, não são todos os bancos que permitem essa funcionalidade.
› Funções válidas
› Remover e renomear colunas
› Filtrar linhas
› Agrupar, sumarizar, linhas em colunas (pivot e unpivot)
› Combinar e extrair dados de consulta
› Conectar consultas com a mesma base de dados
› Adicionar colunas customizadas com lógica simples
› Funções inválidas
› Combinar consultas de bases de dados diferentes
› Adicionar coluna com índice
› Alterar o tipo dos dados de uma coluna

Demonstração

› Operadores podem ser combinados. Por exemplo:
› LastStep[Year]{[ID]}
*Isso significa que você pode pegar um valor de outra etapa baseado no índice da coluna



› Produção de uma chave primária para dimensão tempo:

```
#table(
   type table [Date=date, Day=Int64.Type, Month=Int64.Type,
MonthName=text, Year=Int64.Type,Quarter=Int64.Type],
   List.Transform(
      List.Dates(start_date, (start_date-endd_ate),
                 #duration(1, 0, 0 ,0)),
      each {_, Date.Day(_), Date.Month(_),
Date.MonthName(_), Date.Year(_), Date.QuarterOfYear(_)}
))
```

Palavras-chave

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