POWER QUERY

Übersetzer Kathrin Borchert

Was ist **Power Query**?

"Eine intigrierte Entwicklungsumgebung für M-Skripte"

Components

›**Benutzeroberfläche** - Die Transformations-Engine in Power Query enthält viele vordefinierte Transformationsfunktionen, die über die grafische Benutzeroberfläche des Power Query-Editor verwendet werden können.

›**Abfragen** – Bei der Nutzung der vordefinierten Transformationsfunktionen wird zeitgleich ein M-Skript im Erweiterten Editor geschrieben.

› **Primitive Typen – Beispiele:** binary, date, datetimezone, datetime, duration, etc.

› **Listen –** ist eine geordnete Folge von Werten. M unterstützt endlose Listen. Bei Listen kennzeichnen die Zeichen "{" und "}" den Anfang und das Ende der Liste.

› **Record –** Ist ein Datensatz, eine Menge von Feldern, wobei das Feld ein Paar aus Name und Wert besteht. Der Name ist ein Textwert, der im Feld Datensatz eindeutig ist.

› **Tabelle –** Eine Tabelle ist eine Menge von Werten, die in benannten Spalten und Zeilen angeordnet sind. Eine Tabelle kann so bearbeitet werden, als ob es sich um eine Liste von Datensätzen handelt, oder als ob es sich um einen Datensatz von Listen handelt. Tabelle [Feld]` (Feldreferenz-Syntax für Datensätze) gibt eine Liste von Werten in diesem Feld zurück. Tabelle {i}" (Syntax für Listenindex-Zugriff) gibt einen Datensatz zurück, der eine Zeile der Tabelle darstellt.

› **Funktion –** Eine Funktion ist ein Wert, der wenn er mit Argumenten aufgerufen wird, einen neuen Wert erzeugt. Funktionen werden geschrieben, indem die Funktionsargumente in Klammern, gefolgt von dem Übergangssymbol "=>" und dem Ausdruck der die Funktion definiert.

› **Parameter –** Der Parameter speichert einen Wert, der für Transformationen verwendet werden kann. Neben dem Namen des Parameters und dem Wert, den er speichert, hat er auch andere Eigenschaften, die Metadaten liefern. Vorteil eines Parameters ist, dass er aus der Power BI Service-Umgebung geändert werden kann, ohne direkten Eingriff in das pbix. Die Syntax des Parameters ist wie bei einer regulären Abfrage, das einzige, was besonders ist, dass die Metadaten einem bestimmten Format folgen.

› **Bearbeitungsleiste–** Zeigt den aktuell geladenen Schritt an und ermöglicht die Bearbeitung. Um die Formelleiste sehen zu können, muss diese im Menüband unter Ansicht aktiviert werden.

› **Abfrageeinstellungen –** Ermöglicht die Bearbeitung des Namens und der Beschreibung der Abfrage. Enthält eine Übersicht über alle aktuell angewandten Schritte. Angewandte Schritte sind die in einem let-Ausdruck definierten Variablen und sie werden durch Variablennamen dargestellt.

› **Datenvorschau -** Ermöglicht eine Vorschau auf die Daten für den aktuell ausgewählten Transformationsschritt.

› **Status Leiste –**Leiste am unteren Rand des Bildschirms. Enthält Informationen über Zeilen, Spalten und den Zeitpunkt der letzten Überprüfung der Daten. Die Spaltenprofilerstellung ermöglicht die Datenvorschau auf das gesamte Datenset zu erweitern.

Funktionen in Power Query

Ein solider Kenntnisstand über die Funktionen im M erleichtert einem die Arbeit enorm.

› **Shared –** Shlüsselwort um alle befindlichen Funktionen innerhalb Power Query aufzurufen(einschließlich Hilfe und Beispiele). Die Eingabe erfolgt über eine Leere Abfrage**= # shared**

```
= #shared
```

Funktionen können in zwei Kategorien eingeteilt werden:

› Vorgefertigt - Beispiel: Date.From()

› Benutzerdefiniert - das sind Funktionen, die der Anwender selbst für das Modell durch die Erweiterung von "()=> „ vornehmen kann. Wenn mehrere Argumente verwendet werden, müssen diese durch ein Trennzeichen getrennt sein.

Daten Typen

Datentypen in Power Query werden verwendet, um Werte zu klassifizieren, und so ein strukturiertes Datenset zu erhalten. Datentypen werden auf Feldebene definiert.

› **Null –** null
› **Logik –** true, false
› **Nummer –** 1, 2, 3, ...
› **Zeit –** #time(HH,MM,SS)
› **Datum –** #date(yyyy,mm,ss)
› **DatumZeit –** #datetime(yyyy,mm,dd,HH,MM,SS)
› **DatumZeitZone –** #datetimezone(yyyy,mm,dd,HH,MM,SS, 9,00)
› **Dauer –** #duration(DD,HH,MM,SS)
› **Text –** "text"
› **Binary –** #binary("link")
› **List –** {1, 2, 3}
› **Record –** [A = 1, B = 2]
› **Table –** #table({columns},{{first row contenct},{}...})*
› **Funktion –** (x) => x + 1
› **Type –** type { number }, type table [A = any, B = text]
*Der Index der ersten Zeile der Tabelle ist derselbe wie bei den Datensätzen in Blatt 0

Operatoren

In der Sprache M gibt es mehrere Operatoren, aber nicht jeder Operator kann für alle Arten von Werten verwendet werden.

› **Primäre Operatoren**
› **(x)** – In Klammern gesetzter Ausdruck
› **x[i]** – Feldbezug. Rückgabewert aus Datensatz, Liste von Werten aus Tabelle.
› **x{i}** – Elementzugriff. Rückgabewert von Listensatz aus Tabelle. "Platzieren des "**?**"-Zeichens nach dem Operator liefert Null, wenn der Index nicht in der Liste ist „
› **x(...)** – Aufruf einer Funktion
› **{1 .. 10}** – Automatische Erstellung einer Liste von 1 bis 10
› **...** – Nicht verfügbar
› **Mathematische Operatoren** – **+, -, *, /**
› **Vergleichsoperator**
› **>** , **>=** – Größer als, Größer als oder gleich
› **<** , **<=** – Kleiner als, Kleiner als oder gleich
› **=** , **<>** – ist gleich, ist nicht gleich.
› **Logische Operatoren**
› **and** – Verkürzte Konjunktion
› **or** – Verkürzte Disjunktion
› **not** – Logische Negation
› **Type Operatoren**
› **as** – Kompatibler „nullable-primitive"-Typ oder Fehler
› **is** –›Test, ob kompatibler „nullable-primitive"-Typ oder Fehler

Metadata - Das Wort **meta** ordnet einem Wert Metadaten zu. Beispiel für die Zuweisung von Metadaten an die Variable **x**: **"x meta y"** oder **"x meta [Name = x, Wert = 123,...]"** In Power Query gilt die Priorität der Operatoren, so dass z. B**. "X + Y * Z"** als "X + (Y * Z)" ausgewertet wird**.**

Kommentare

M unterstützt zwei Arten von Kommentaren:
› Einzeilige Kommentare– können erstellt werden mit **//** vor dem Code
› Shortcut: **CTRL + ´**
› Mehrzeilige Kommentare– können erstellt werden mit **/*** vor dem Code ***/** nach dem Code
› Shortcut: **ALT + SHIFT + A**

"let" Ausdruck

Der **let**-Ausdruck kapselt Werte, die berechnet, benannt und dann in einem nachfolgenden Ausdruck verwendet werden sollen, der auf die **in**-Anweisung folgt. Ein let-Ausdruck könnte beispielsweise eine Variable **Source** enthalten, die dem Wert von **Text.Proper()** entspricht und im richtigen Fall einen Textwert liefert.

let Source = Text.Proper("hello world") in Source

Wenn der Ausdruck ausgewertet wird, gilt immer Folgendes:
› Ausdrücke in Variablen definieren einen neuen Bereich mit Identifikatoren aus der Variablenliste und müssen bei der Auswertung von Termen innerhalb einer Variablenliste vorhanden sein.
› Die Ausdrücke in der Liste der Variablen können sich aufeinander beziehen.
› Alle Ausdrücke in Variablen müssen ausgewertet werden, bevor der Begriff **let** ausgewertet wird.
› Wenn Ausdrücke in Variablen nicht vorhanden sind, wird **let** nicht ausgewertet.
› Fehler die bei der Abfrageauswertung auftreten, werden als Fehler auf andere verknüpfte Abfragen angewendet.

Bedingungen

Auch in Power Query existiert der "**If**" Ausdruck, die auf der Grundlage der eingefügten Bedingung entscheidet, ob das Ergebnis ein wahrer oder ein falscher Ausdruck sein wird.
Syntaktische Form des **If**-Ausdrucks:
IF <Prädikat> **then** <wahr-Ausdruck > **else** <falsch-Ausdruck >
"**else** ist im bedingten Ausdruck erforderlich"
Bedingung:
If x > 2 **then** 1 **else** 0
If [Month] > [Fiscal_Month] **then** true **else** false

If-Ausdruck ist die einzige Bedingung in M.
Verkettung von mehreren Bedingungen:

```
If <Prädikat>
  then < wahr-Ausdruck >
  else if <Prädikat>
    then < falsch-wahr-Ausdruck >
     else < falsch-falsch-Ausdruck >
```

Bei der Auswertung der Bedingungen gilt folgendes:
' Ist der Wert, der durch die Auswertung der **If** Bedingung entsteht, kein logischer Wert, dann wird ein Fehler mit dem Ursachencode **"Expression.Error"** ausgelöst.
' Ein Wahr-Ausdruck wird nur ausgewertet, wenn die Wenn-Bedingung als wahr gewertet wird. Andernfalls wird ein falsch-Ausdruck ausgewertet.
' Wenn Ausdrücke in Variablen nicht vorhanden sind, dürfen sie nicht ausgewertet werden
' Der Fehler, der bei der Auswertung der Bedingung auftritt, wird entweder in Form eines Fehlers in der gesamten Abfrage oder als "Fehler"-Wert im Datensatz zurück gegeben.

Der Ausdruck try und otherwise

Das Erfassen von Fehlern ist z. B. mit dem **try**-Ausdruck möglich. Es wird versucht, den Ausdruck nach dem Wort **try** auszuwerten. Tritt bei der Auswertung ein Fehler auf, wird der Ausdruck nach dem Wort **otherwise** angewendet.

Syntax-Beispiel:
try Date.From([textDate]) **otherwise** null

Benutzerdef. Funktion

Beispiel:
(x, y) => Number.From(**x**) + Number.From(**y**)

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

Es gibt zwei Arten für die Eingabe der Funktion:
› **Erforderlich** - Alle Argumente in (). Ohne diese Argumente kann die Funktion nicht aufgerufen werden.
› **Optional –** Ein solcher Parameter kann, muss aber nicht zwingend in die Funktion eingegeben werden. Kennzeichnen Sie den Parameter als optional, indem Sie den Text **„Optional"** davor eingeben. Beispiel **(optional x)**.
Optionale Argumente sind nach erforderlichen Argumenten anzugeben.

Argumente können mit <Typ> kommentiert werden, um den erforderlichen Typ des Arguments anzugeben. Die Funktion wird einen Typfehler auslösen, wenn sie mit Argumenten des falschen Typs aufgerufen werden.
(x as number, y as text) as logical => <expression>

Das Ergebnis der Funktionen ist sehr unterschiedlich. Die Ausgabe kann ein Blatt, eine Tabelle, ein Wert, aber auch andere Funktionen sein. Das bedeutet, dass eine Funktion eine andere Funktion erzeugen kann. Eine solche Funktion wird wie folgt geschrieben:

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

Bei der Auswertung von Funktionen gilt:

' Fehler, die durch die Auswertung von Ausdrücken in einer Liste von Ausdrücken entstehen, werden als "Fehler"-Wert weitergegeben.

Recursive Funktionen

Für Recursive Funktionen wir das "**@**"genutzt.
Eine typische rekursive Funktion ist die Fakultät. Die Funktion für die Fakultät kann wie folgt geschrieben werden:

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

Each Funktion

Each kann man sich am besten als eine Iteration über Zeilen in einer Spalte oder Liste vorstellen. Wenn Sie eine Kundenspalte im Power Query einfügen, z. B. [Menge] * [Preis], schreiben Sie eigentlich nur eine einzige Formel, aber Sie erhalten möglicherweise viele verschiedene Antworten auf der Seite in der neuen Spalte. Jede Zeile könnte eine andere Antwort enthalten. In diesem Sinne wird die Formel für jede Zeile in der Spalte (oder jedes Element in einer Liste) ausgewertet.

Syntaktischer Zucker

› Das Schlüsselwort **Each** wird verwendet, um einfache Funktionen zu erstellen. **"each** ..." ist syntaktischer Zucker für eine Funktionssignatur, die den _-Parameter "(_) => ..." enthalten.
Each ist nützlich, wenn es mit dem Suchoperator kombiniert wird, der standardmäßig auf „_" angewendet wird.
Beispielsweise ist **„each** [CustomerID]" dasselbe wie **„each** _[CustomerID]", das wiederum mit „(_) => _[CustomerID]" identisch ist.

```
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"
```

Query Folding

Query Folding ist die Möglichkeit, dass Power Query eine einzelne Abfrage-Anweisung generieren kann, um Quelldaten abzurufen und zu transformieren. Die Power Query Mashup-Engine versucht, aus Effizienzgründen nach Möglichkeit ein Query Folding durchzuführen.
Die meisten Datenquellen unterstützen das Query Folding. Zu diesen Datenquellen zählen beispielsweise relationale Datenbanken, OData-Feeds (einschließlich SharePoint-Listen), Exchange und Active Directory. Flatfiles, Blobs oder Webquellen unterstützen das Query Folding in der Regel nicht.

› **Unterstützt**
› Entfernen und Umbenennen von Zeilen
› Zeilenfilter
› Gruppieren, Zusammenfassen, pivot and unpivot
› Zusammenführen und Extrahieren von Daten
› Verbundene Abfragen mit gleicher Quelle
› Benutzerdefinierte Spalten mit einfacher Logik
› **Nicht unterstützt**
› Zusammengeführte Daten aus unterschiedlichen Quellen
› Hinzufügen von Spalten mit Indexen
› Spalten mit geändertem Datentyp

DEMO

› Operatoren können kombiniert werden.
›Beispiel: LastStep[Year]{[ID]} › 1899
* Dies bedeutet, dass Sie den Wert aus einem anderen Schritt auf der Grundlage des Index der Spalte abrufen können.

› Erstellen einer DateKey Dimension:

```
#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, (Number.From(end_date)-
Number.From(start_date)), #duration(1, 0, 0 ,0)),
      each {_, Date.Day(_), Date.Month(_),
Date.MonthName(_), Date.Year(_), Date.QuarterOfYear(_)}
   ))
```

Schlüsselwörter

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