Excellent days
12.10. 13.10. 2018

Track #3: Power & Co.

Deep Dive Power Query und „M“

Über die Benutzeroberfläche hinaus

Mit freundlicher Unterstützung von:
dnp dotnetpro
in video2brain
GFU Cyrus AG

Vorstellen

Hans Peter Pfister

- Jahrgang 1973, 2 Kinder, Schweizer
- Partner, CFO und Lead BI / Analytics
  WAGNER AG Informatik Dienstleistungen, Schweiz
- Co-Leader Power BI PUG Switzerland, Speaker
- über 25 Jahren Erfahrung in Datenanalyse & Reporting
- dipl. Experte Controlling & Rechnungslegung
- Dipl. Bankfachexperte
- Microsoft Certified Professional & MCSA “BI Reporting”
- Blog www.PowerBI-Pro.com
- h.p.pfister@powerbi-pro.com

Wer sind die Teilnehmer?

- **BI / Reporting Professional?**
- **Business User (Controlling / Finance, Marketing, anderes?)**
- **Erfahrung mit Power Query?**
  - keine, erste, längere, Profi, Ninja?
- **Erfahrungen mit Power BI?**

Anforderungen an die Teilnehmer

- **Fit für mittelschweres Power Query Trekking**
- **Leidensfähigkeit für lange Monologe notwendig**

Es gibt keine Umkehr. Die Türen sind geschlossen!

Agenda

- Einleitung
- Einführung „M“
- Code Aufbau
- Dynamischer Kalender
- Funktionen erstellen
- Abschluss & Fragen

Einleitung

Einleitung

Power Query vereinfacht den **Datenimport und -transformationen** erheblich!

Vorbei die Zeiten von Importen und Transformationen durch

- SQL Statements,
- VBA Scripts,
- Access oder Exceldateien (?!) als Datenbank
- copy & paste
- usw.

Einsatzgebiete Power Query

- Excel
- Power BI
- SQL Server Analysis Services (SSAS) ab 2017
- Common Data Service (CDS)
- Common Data Services for Apps (CDS-A)
- Power BI Dataflows

Power BI – Design Phasen

Daten importieren

- Quellen anbinden
- Daten importieren

Daten bearbeiten

- Bereinigen
- Transformieren
- Filtern
- Anreichern
- Zusammenfügen
- usw.

Daten modellieren

- Beziehungen
- Berechnete Spalten/Tabellen
- Measures
- Formatieren
- usw.

Daten visualisieren

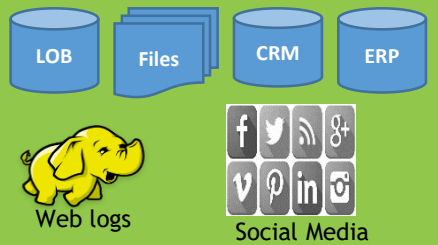
- Charts, Tabellen
- Filtern
- Datenschnitte
- Drill down
- Drill through
- usw.

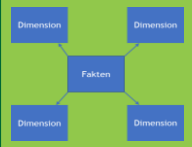
Reports veröffentlichen

- Power BI Service
- Power BI Mobil App
- Embed
- Zusammenarbeit
- usw.

Power Query

DAX





Grenzen der Benutzeroberfläche

- Power Query hat 681 Funktionen (Stand Juli 2018)
  => #shared oder http://bit.ly/PQ_Code

- nicht alle Funktionen über die Benutzeroberfläche erreichbar

- Es gibt Anforderungen, die «M» benötigen, wie z.B.

  => Spalten hinzufügen mit eigener Logik

  => «if then else» Befehle mit mehreren Bedingungen

  => Fehlerroutinen

  => wiederverwendbare, eigene Funktionen – sehr nützlich!

Einführung «M»

Was ist «M»?

- Sprache hinter Power Query
- Alle Arbeitsschritte = «M» Code
- funktionale Sprache
- «M» / Power Query ist faul resp. effizient
  => nicht benötigte Schritte werden ignoriert!
- Arbeitsschritte wiederverwendbar
- case sensitive => gRoSs- & kLeInScHrEiBuNg beachten!

Demo – einfache Beispiel

- Einführungsbeispiele
- Statischer Kalender ab Exceltabelle

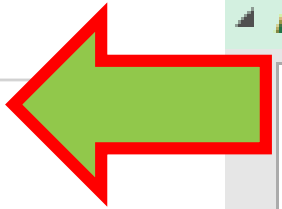
Code Aufbau

Aufzeichnung der Arbeitsschritte

Erweiterter Editor

_4_Zeitreihe

```
let
    Quelle = Excel.CurrentWorkbook(){[Name="Zeitreihe"]}[Content],
    #"Geänderter Typ" = Table.TransformColumnTypes(Quelle,{{"Datum", type date}}),
    #"Jahr eingefügt" = Table.AddColumn(#"Geänderter Typ", "Jahr", each Date.Year([Datum]), type number),
    #"Monat eingefügt" = Table.AddColumn(#"Jahr eingefügt", "Monat", each Date.Month([Datum]), type number),
    #"Quartal eingefügt" = Table.AddColumn(#"Monat eingefügt", "Quartal", each Date.QuarterOfYear([Datum]), type number)
in
    #"Quartal eingefügt"
```



ANGEWENDETE SCHRITTE

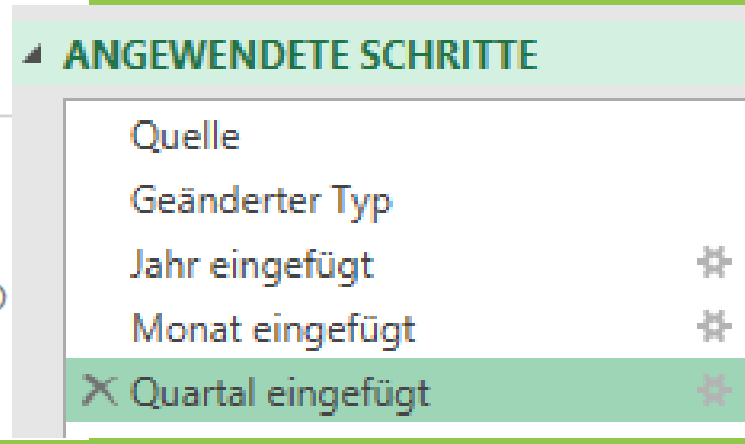
- Quelle
- Geänderter Typ
- Jahr eingefügt
- Monat eingefügt
- Quartal eingefügt

Code Aufbau

Erweiterter Editor

_4_Zeitreihe

```
let
    Quelle = Excel.CurrentWorkbook(){[Name="Zeitreihe"]}[Content],
    #"Geänderter Typ" = Table.TransformColumnTypes(Quelle,{{"Datum", type date}}),
    #"Jahr eingefügt" = Table.AddColumn(#"Geänderter Typ", "Jahr", each Date.Year([Datum]), type number),
    #"Monat eingefügt" = Table.AddColumn(#"Jahr eingefügt", "Monat", each Date.Month([Datum]), type number),
    #"Quartal eingefügt" = Table.AddColumn(#"Monat eingefügt", "Quartal", each Date.QuarterOfYear([Datum]), type number)
in
    #"Quartal eingefügt"
```



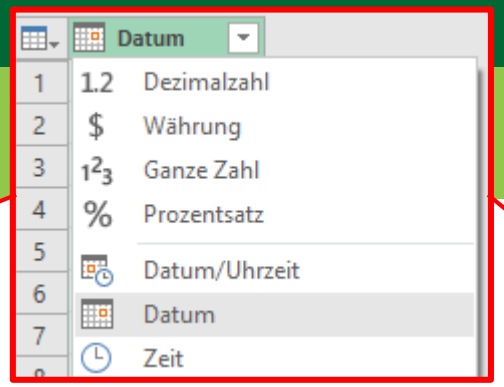
- let => Beginn einer normalen Abfrage
- in => kennzeichnet Ende der Abfrage

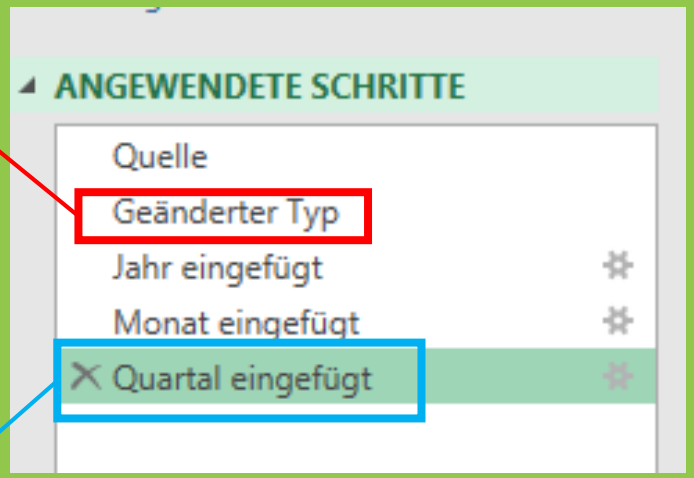
Code Aufbau - Variablen

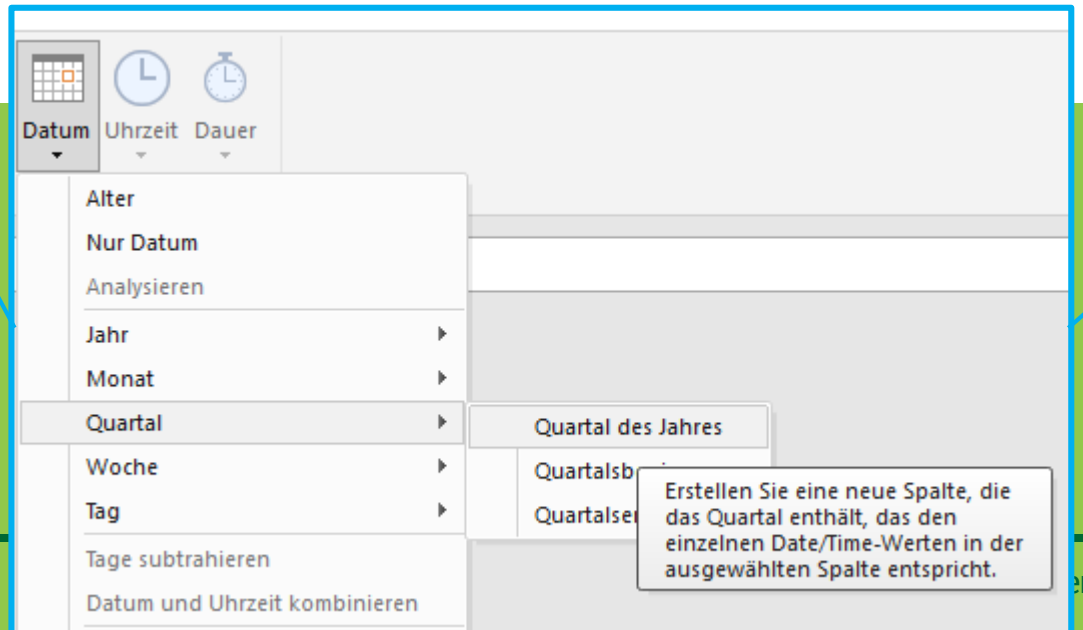
Erweiterter Editor

_4_Zeitreihe

```
let
    Quelle = Excel.CurrentWorkbook(){[Name="Zeitreihe"]}[Content],
    #"Geänderter Typ" = Table.TransformColumnTypes(Quelle,{{"Datum", type date}}),
    #"Jahr eingefügt" = Table.AddColumn(#"Geänderter Typ", "Jahr", each Date.Year([Datum]), type number),
    #"Monat eingefügt" = Table.AddColumn(#"Jahr eingefügt", "Monat", each Date.Month([Datum]), type number),
    #"Quartal eingefügt" = Table.AddColumn(#"Monat eingefügt", "Quartal", each Date.QuarterOfYear([Datum]), type number)
in
    #"Quartal eingefügt"
```







eroberfläche hinaus

Code Aufbau - Variablen

- Variablen = Arbeitsschritte
- Gültige Variablennamen – keine Leerzeichen, keine Zahlen zu Beginn

Folgende Variablennamen sind gültig:

Name	Begründung
Verkaufsberechnung	Ein zusammengeschriebenes Wort ohne Leerschlag
#"Verkaufsberechnung 2018"	Bei Leerschlag muss dem Namen ein # vorangestellt und die Wörter in Anführungszeichen geschrieben werden
_Verkaufsberechnung	Ein Unterstrich darf vorangestellt werden
_Verkaufsberechnung_2018	Ebenso dürfen im Namen Unterstriche verwendet werden
#""	Gültig, da # vorangestellt und "" verwendet

- «Heilung» durch # " "

Folgende Variablennamen sind hingegen nicht gültig:

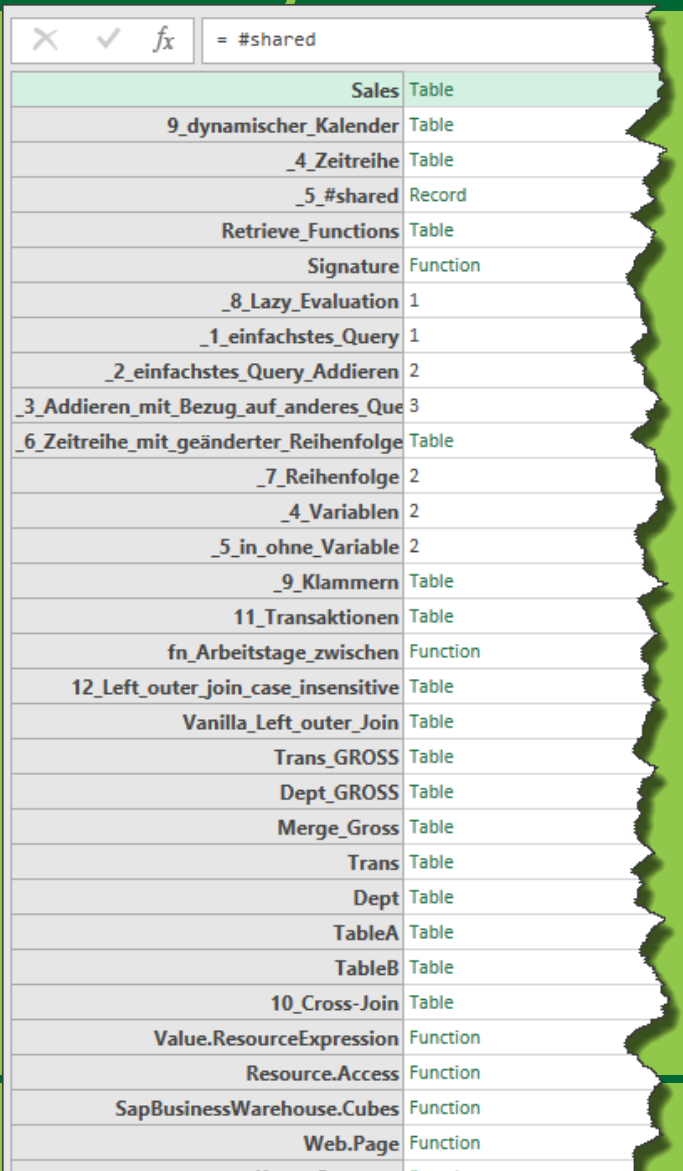
Name	Begründung
2018_Verkaufsberechnung	Der Namen darf nicht mit einer Zahl beginnen. # und "" verwenden!
Verkaufsberechnung 2018	Leerschläge sind nicht erlaubt ohne # und ""

«M» Code Beispiel

```
let
 Quelle =                  Excel.CurrentWorkbook(){[Name="Zeitreihe"]}[Content],

 #"Geänderter Typ" =       Table.TransformColumnTypes(Quelle,{{"Datum", type date}}),

#"Jahr eingefügt" =        Table.AddColumn(#"Geänderter Typ", "Jahr“,
                                each Date.Year([Datum]), type number),
#"Monat eingefügt" =       Table.AddColumn(#"Jahr eingefügt", "Monat“,
                                each Date.Month([Datum]), type number),
#"Quartal eingefügt" =     Table.AddColumn(#"Monat eingefügt", "Quartal“,
                                each Date.QuarterOfYear([Datum]), type number)

in
#"Quartal eingefügt"
```

1. Beginn Abfrage
2. Variablen
3. Funktionen
4. Zeilenende
5. Bezug auf vorherige Variable (Arbeitsschritt)
6. Parameter
7. Ende der Abfrage

Verfügbare Funktionen



= #shared

Sales	Table
9_dynamischer_Kalender	Table
_4_Zeitreihe	Table
_5_#shared	Record
Retrieve_Functions	Table
Signature	Function
_8_Lazy_Evaluation	1
_1_einfachstes_Query	1
_2_einfachstes_Query_Addieren	2
_3_Addieren_mit_Bezug_auf_anderes_Que	3
_6_Zeitreihe_mit_geänderter_Reihenfolge	Table
_7_Reihenfolge	2
_4_Variablen	2
_5_in_ohne_Variable	2
_9_Klammern	Table
11_Transaktionen	Table
fn_Arbeitstage_zwischen	Function
12_Left_outer_join_case_insensitive	Table
Vanilla_Left_outer_Join	Table
Trans_GROSS	Table
Dept_GROSS	Table
Merge_Gross	Table
Trans	Table
Dept	Table
TableA	Table
TableB	Table
10_Cross-Join	Table
Value.ResourceExpression	Function
Resource.Access	Function
SapBusinessWarehouse.Cubes	Function
Web.Page	Function

- Power Query hat 681 Funktionen (Stand Juli 2018)
  => #shared oder http://bit.ly/PQ_Code

Reihenfolge der Arbeitsschritte

Erweiterter Editor

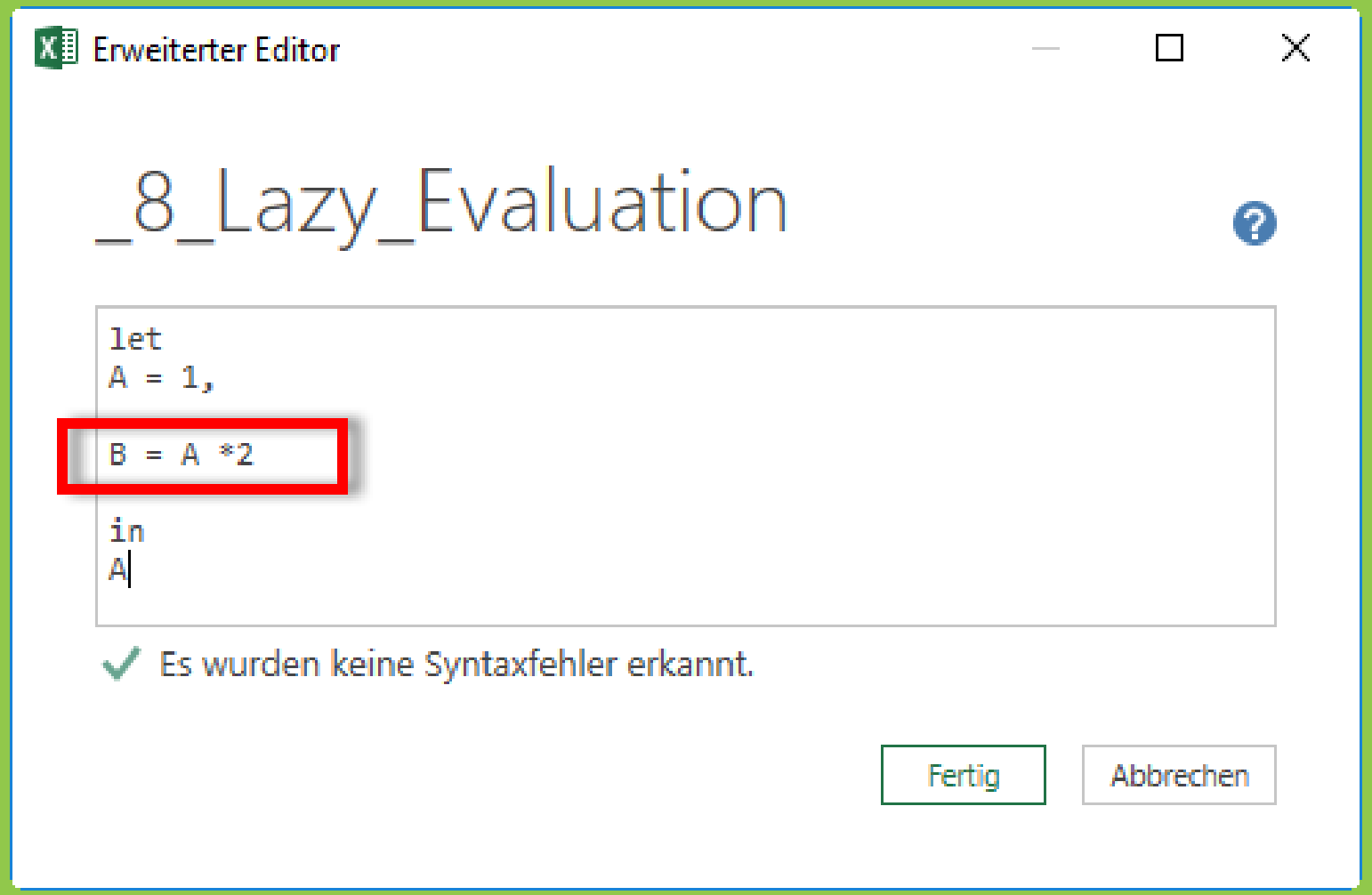
_6_Zeitreihe_mit_geänderter_Reihenfolge

```
let
    Quelle = Excel.CurrentWorkbook(){[Name="Zeitreihe"]}[Content],
    #"Quartal eingefügt" = Table.AddColumn(#"Monat eingefügt", "Quartal", each Date.QuarterOfYear([Datum]), type number),
    #"Monat eingefügt" = Table.AddColumn(#"Jahr eingefügt", "Monat", each Date.Month([Datum]), type number),
    #"Geänderter Typ" = Table.TransformColumnTypes(Quelle,{{"Datum", type datetime}}),
    #"Jahr eingefügt" = Table.AddColumn(#"Geänderter Typ", "Jahr", each Date.Year([Datum]), type number)

in
    #"Quartal eingefügt"
```

Es wurden keine Syntaxfehler erkannt.

Fertig | Abbrechen

Lazy Evaluation



Code Kommentierung

Erweiterter Editor

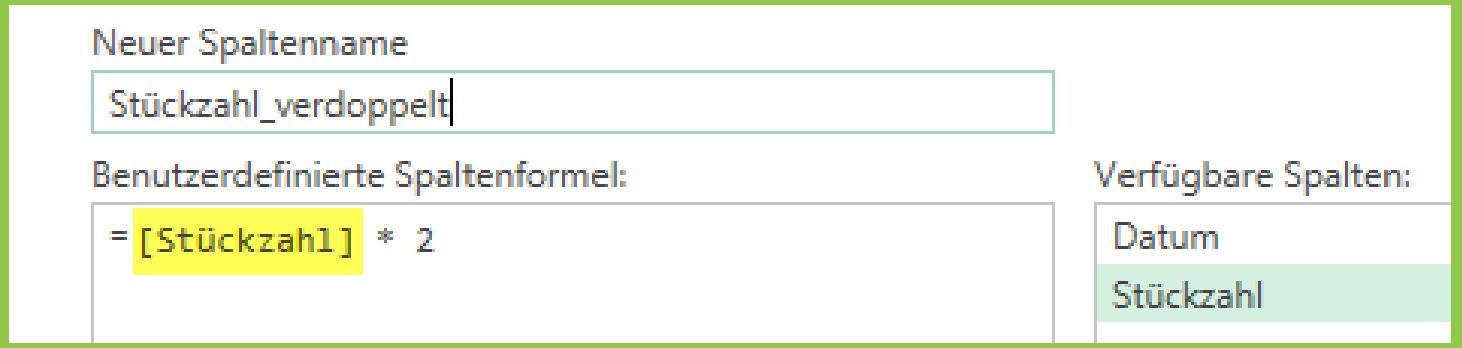
12_Left_outer_join_case_insensitive

```
let
/*  Table.AddColumn ergänzt die Tabelle "Trans" um eine neue Spalte "Merge"
    Die Werte für die neue Spalte werden aus der neuen Funktion namens "case_insensitive"
    befüllt.
    Die Funktion wandelt die Spalte "Name" aus der Tabelle "Dept" in Grossbuchstaben
    (Text.Upper) um
    Das Resultat wird mit der ebenfalls in Grossbuchstaben umgewandelte Spalte "Employee"
   aus der Tabelle "Dept" verglichen.
    Table.SelectRows befüllt die neue Spalte nur dann mit Werten, falls die in
    Grossbuchstaben umgewandelten Einträge übereinstimmen.
    Der Aufruf Text.Upper(case_insensitive[Employee]) ruft innerhalb der Funktion die
    Funktion auf!
    Es handelt sich daher um eine rekursive Funktion!
*/

    Quelle = Table.AddColumn(Trans, "Merge",
            (case_insensitive) => Table.SelectRows(Dept,
                        each Text.Upper([Name])=Text.Upper(case_insensitive[Name]))),

// Diese Codezeile erweitert die Spalte
    #"Erweiterte Merge" = Table.ExpandTableColumn(Quelle, "Merge", {"Name", "Dept"}, {"Emp.Name", "Emp.Dept"})
in
    #"Erweiterte Merge"
```

✓ Es wurden keine Syntaxfehler erkannt.

Spezialzeichen

- Eckige Klammer => Bezug auf Spalten



- Geschweifte Klammern => Bezug auf Felder

```
let
    Quelle = Excel.CurrentWorkbook(){[Name="Sales"]}[Content],
    #"Geänderter Typ" = Table.TransformColumnTypes(Quelle,{{"Datum", type datetime}, {"Stückzah
    #"Hinzugefügte benutzerdefinierte Spalte" = Table.AddColumn(#"Geänderter Typ", "Stückzahl_v
    Stückzahl_verdoppelt = #"Hinzugefügte benutzerdefinierte Spalte"{1}[Stückzahl_verdoppelt]
```

Dynamischer Kalender

Warum ein Kalender?

- Jede Auswertung, welche einen Zeitbezug hat, muss über chronologisch fortlaufende Kalenderdaten verfügen

- Der Ansatz, in der Transaktionstabelle für jeden Eintrag die entsprechenden Spalten (wie Jahr, Monat, Quartal usw.) anzufügen, funktioniert nur für Modelle mit 1 Tabelle!

Grundsatz:

Alle beschreibende Daten (wie Kunden, Kalender, Produkte, Ortschaften usw.) erhalten eine eigene Tabelle (Dimension)!

Funktionen

Funktionen

- Netto-Arbeitstage

Zusammenfassung

Repetition Code Aufbau

- Jede Abfrage (Query) ist ein **einzelner Ausdruck**, der einen Wert zurück gibt => kann auch eine Tabelle oder Liste sein
- Jede Abfrage beginnt mit «**let**», gefolgt von einer Reihe von **Variablen**
- Die Variablen sind die benannten Arbeitsschritte auf der Oberfläche
- Jede Variablendeklaration endet mit einem **Komma**
- Vor «in» **darf kein Komma** stehen!
- Der Ausdruck hört mit «**in**» und dem gewünschten Output (meistens letzter Arbeitsschritt) auf
- Innerhalb der Abfrage ist jede Variablen **wiederholt ansprechbar**

Vielen Dank für Ihre Aufmerksamkeit!

Deep Dive Power Query und "M" - Über die Benutzeroberfläche hinaus

Mit freundlicher Unterstützung von:
dnp dotnetpro
in® | video2brain
GFU Cyrus AG