



Microsoft

# POWER BI



## BEST PRACTISES



Holger Gubbels

© 2024

# Inhaltsverzeichnis

Vorwort . . . . .	4
Den Überblick behalten: Gruppen in Power Query . . . . .	5
Die Aufgabe . . . . .	5
Best Practise: Gruppen nutzen . . . . .	5
Vorteile . . . . .	6
Gruppenstruktur in Power Query . . . . .	7
Die Aufgabe . . . . .	7
Einschub: Duplikate und Verweise . . . . .	7
Best Practise: Wiederverwendbare Gruppen-Struktur nutzen . . . . .	9
Vorteile . . . . .	11
Automatische Typerkennung deaktivieren . . . . .	13
Die Aufgabe . . . . .	13
Best Practise: Deaktiviere die automatische Typerkennung . . . . .	14
Vorteil . . . . .	14
Letzte Berichtsaktualisierung . . . . .	15
Aufgabe . . . . .	15
Best Practise: Ein Aktualisierungsdatum- oder Zeitstempel einfügen . . . . .	15
Vorteile . . . . .	15
Spalten nicht Löschen . . . . .	17
Aufgabe . . . . .	17
Best Practise: Spalten auswählen statt löschen . . . . .	17
OneDrive . . . . .	19
Aufgabe . . . . .	19
Best Practise: OneDrive als Datenquelle nutzen . . . . .	20
Vorteile . . . . .	21
Geiz ist geil (bei Spalten) . . . . .	23
Aufgabe . . . . .	23
Best Practise: Sei geizig mit Spalten . . . . .	23
Vorteile . . . . .	25
Automatische Kalendertabellen . . . . .	26
Aufgabe . . . . .	26
Best Practise: Automatische Kalendertabellen deaktivieren . . . . .	29
Vorteile . . . . .	31
Wann ist "Heute" . . . . .	32
Aufgabe . . . . .	32

Best Practise: Mein eigenes TODAY() . . . . .	33
Vorteile . . . . .	34
Keine bidirektionalen Kreuzfilter! . . . . .	35
Aufgabe . . . . .	35
Best Practise: Keine bidirektionalen Kreuzfilter . . . . .	42
Vorteil . . . . .	42
Keine impliziten Measures . . . . .	44
Aufgabe . . . . .	44
Best Practise: Nur explizite Measures schreiben! . . . . .	48
Vorteile . . . . .	48
Measure Tabellen . . . . .	51
Aufgabe . . . . .	51
Best Practise: Anlage einer <i>Measure-Tabelle</i> . . . . .	51
Vorteile . . . . .	54
DAX Styleguide . . . . .	55
Aufgabe . . . . .	55
Best Practise: Einen Styleguide formulieren . . . . .	55

## Vorwort

Hallo und herzlich willkommen zu “Power BI - (Meine) Best Practises”!

Wir leben in einer Zeit, in der Datenanalysen und Business Intelligence immer wichtiger werden, um erfolgreich zu sein. Power BI ist dabei ein unglaublich mächtiges Tool, und ich freue mich, dir in diesem Buch meine besten Tipps und Tricks zu zeigen – Erfahrungen und Werkzeuge, die für mich besonders gut funktionieren.

In diesem Buch werde ich dich *duzen* - auch wenn wir uns im echten Leben persönlich nicht kennen oder uns respektvoll mit *Sie* ansprechen.

**Best Practice**, was ist das eigentlich? Es sind bewährte Methoden und Ansätze, die helfen, Prozesse zu optimieren, Fehler zu vermeiden und letztlich bessere Ergebnisse zu erzielen. Sie sind das Ergebnis von Erfahrung und ständigem Lernen. Es gibt oft nicht den einen richtigen Weg, aber diese Vorgehensweisen haben sich für mich als besonders nützlich erwiesen.

Ich möchte betonen, dass es sich hier um **meine** Best Practices handelt. Jeder hat seine eigene Art, Dinge anzugehen, und das ist auch gut so. Ich lade dich ein, meine Methoden auszuprobieren und herauszufinden, ob sie auch für dich funktionieren. Passe sie gerne an deine oder eure Bedürfnisse an.

Für Anfänger mögen manche Themen und Techniken auf den ersten Blick komplex oder umständlich wirken. Das ist normal und kein Grund zur Sorge. Der wahre Wert dieser Best Practices zeigt sich oft erst, wenn man sie in der Praxis anwendet. Es kann sein, dass die Vorteile nicht sofort ersichtlich sind, aber mit der Zeit und nach ein paar Datenmodellen wirst du feststellen, warum diese Ansätze sinnvoll sind.

Weder die Themen noch dieses Buch ist in Stein gemeißelt – es wird wachsen und sich weiterentwickeln. Jedes neue Projekt und jede neue Erkenntnis wird neue Best Practices mit sich bringen, die dann versuche zu ergänzen oder vorhandene zu ändern.

Ich freue mich sehr, dich auf deine Reise durch die Welt der Datenanalyse mit Power BI ein Stück zu begleiten. Ich wünsche dir viel Erfolg und vor allem Freude beim Entdecken und Anwenden der Best Practices, die ich hier zusammengestellt habe.

Mit den besten Grüßen

Holger Gubbels

# Den Überblick behalten: Gruppen in Power Query

## Die Aufgabe

In Power Query habe ich Abfragen, Funktionen und Parameter. Neben Abfragen auf “echte” externe Datenquellen, kommen noch interne Verweise hinzu.



Wenn du nicht weißt, was ein Verweis ist oder was der Unterschied zwischen *Duplikat* und *Verweis* ist, dann lies unbedingt meinen Artikel dazu:

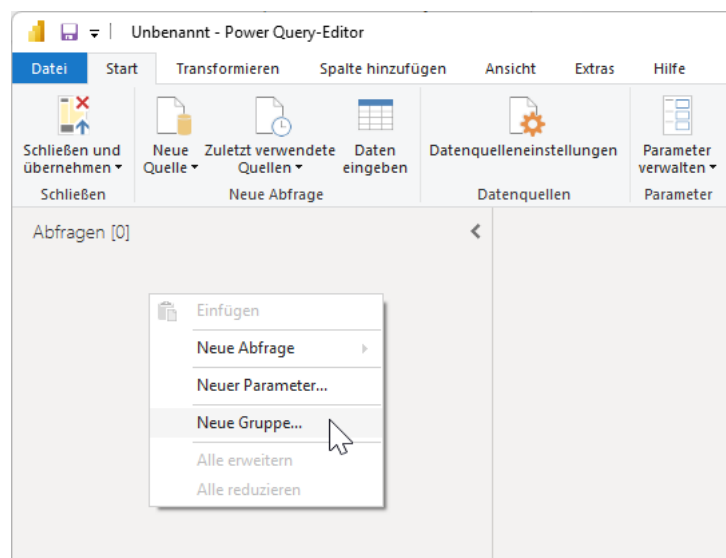
<https://www.durchblick-durch-daten.de/power-query/verweise-und-duplikate/>

Dort gibt es auch eine Erklärung per Video. Im nächsten Abschnitt verliere ich aber noch ein paar Worte dazu.

Um der Vielzahl an Artefakten Herr zu werden, benötige ich eine Struktur. Und die gibt es: *Gruppen*.

## Best Practise: Gruppen nutzen

Im Abfragebereich auf der linken Seite kann ich via Kontextmenü Gruppen anlegen.



**Abbildung 1:** Gruppen anlegen in Power Query

Gruppen können beliebige Namen haben und können beliebig ineinander verschachtelt werden. Wichtig zu wissen ist, dass Gruppen nur der Strukturierung dienen. Sonst haben sie keinerlei Einfluss

auf die Abfragen, Funktionen oder Parameter.



Sobald man eine Gruppe angelegt hat, müssen alle Elemente einer Gruppe zugeordnet sein. Damit das funktioniert legt Power Query kurzerhand eine Gruppe *Andere Abfragen* an, in der alle bisher nicht zugeordneten Abfragen einsortiert werden.

## Vorteile

Gerade bei Änderungen stören mich die vielen Abfragen oder Parameter in Power Query. Wenn ich Änderungen durchführen will, benötige ich ohne Gruppen zu lange, um die richtigen Abfragen oder Funktionen zu finden.

- Gruppen erhöhen die Lesbarkeit
- dadurch die Wartbarkeit
- und erhöhen die Verständlichkeit für Dritte.

# Gruppenstruktur in Power Query

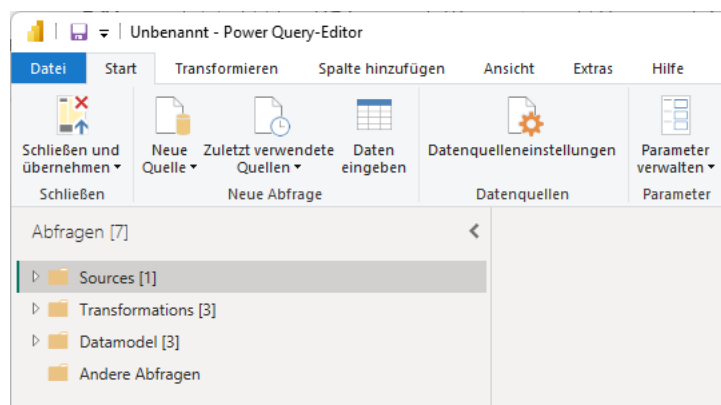
## Die Aufgabe

Ich möchte mir nicht bei jedem Projekt überlegen müssen

- wie viele Gruppen benötige ich?
- wie heißen die Gruppen?
- was sortiere ich in die jeweiligen Gruppen ein?

Darüber hinaus muss ich immer wieder in eigenen, älteren Datenmodellen arbeiten und freue mich, wenn ich mich sofort zurecht finde. Und wenn man viele Datenmodelle aufbaut, bemerkt man, dass sich alle Projekte (auf einer gewissen Flughöhe) ähneln.

Ich bin selbst nicht 100% konsequent - sonst verliert man auch notwendige Flexibilität. Eine Gruppe gibt es bei mir aber tatsächlich immer: eine Gruppe mit allen Basisabfragen.



**Abbildung 2:** Gruppe für alle Basisabfragen in Power Query

## Einschub: Duplikate und Verweise

Eine Abfrage basiert auf einer Datenquelle. Das kann eine Datenbank sein oder eine Excel- oder CSV-Datei. Was viele nicht wissen: Die Datenquelle kann auch eine andere Abfrage sein. Man spricht dann in Power Query von einem *Verweis*: Abfrage A verweist auf Abfrage B.



In meinem Blog gibt es dazu einen ausführlichen Artikel unter <https://www.durchblick-durch-daten.de/power-query/verweise-und-duplikate/> sowie ein kurzes Video bei YouTube unter <https://www.youtube.com/watch?v=FkNk8psAb9c>

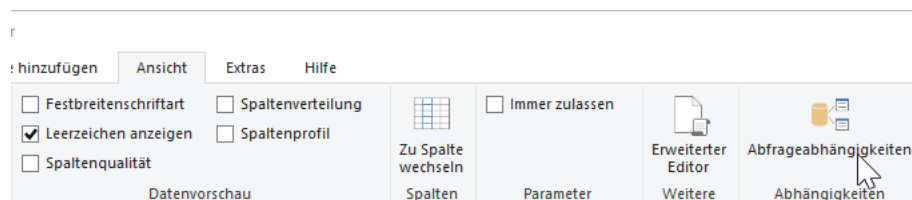
Verweise sind genau dann nützlich, wenn man aus den selben Daten mehrere Tabellen in Power Query erzeugen muss. Ein übliches Beispiel sind Abfragen auf Daten, die von externen Lieferanten oder Kunden kommen. Fast nie wird man separate Daten für Artikel und Bestelldaten bekommen. Sondern du bekommst die Artikelnummer und -beschreibung in jeder Zeile der Bestellposition. Aus Sicht einer Excel-Auswertung ist das genau richtig. Wenn man in Power BI ein vernünftiges Star-Schema aufbauen möchte, sollte man die Daten trennen.

Eine *Dimensionstabelle* erzeuge ich, in dem ich

- auf die Basistabelle eine Verweis erzeuge,
- alle Spalten außer *ArtikelNr* und *Beschreibung* weglasse
- und dann Duplikate lösche.

Die Faktentabelle erzeuge ich ebenfalls über ein Verweis auf die Basistabelle. Hier lasse ich jetzt aber die Artikelbeschreibung weg. Die Artikelnummer dagegen benötigen wird noch als Schlüssel für die spätere Verknüpfung.

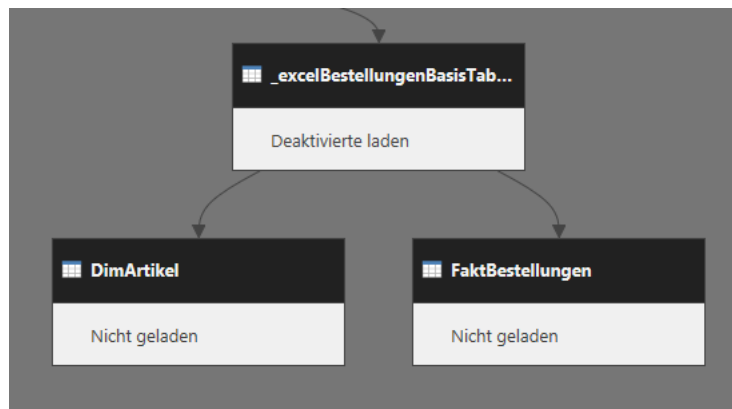
In Power Query im Menü unter *Ansicht* gibt es die Funktion *Abfrageabhängigkeiten*:



**Abbildung 3:** Menüeintrag Abfrageabhängigkeiten in Power Query

Hier stellt Power Query die Abhängigkeiten zwischen den Abfragen dar - was für eine kurze Nachforschung (insbesondere in fremden Datenmodellen) wirklich hilfreich ist:

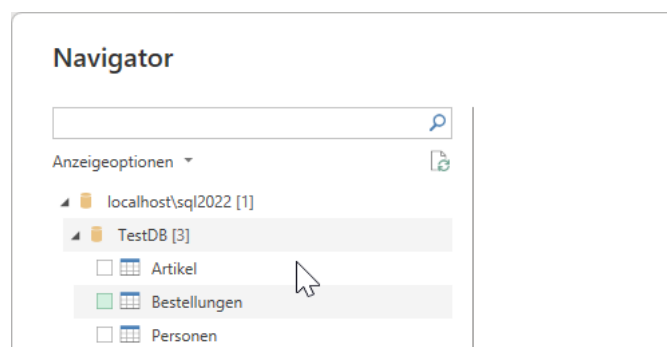




**Abbildung 4:** Abfrageabhängigkeiten in Power Query darstellen

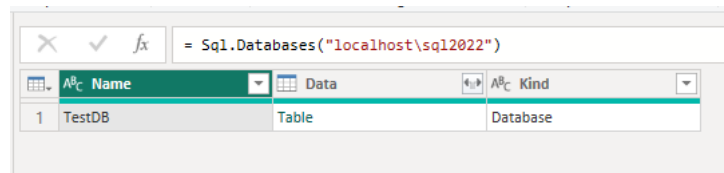
## Best Practise: Wiederverwendbare Gruppen-Struktur nutzen

In meiner Gruppe *Sources* sind alle Quellen enthalten. Bei einem SQL-Server beispielsweise erzeugt der Verbindungs-Assistent beim Anlegen der Abfrage neben dem reinen Zugriff auf die Datenbank immer auch einen Navigationsschritt. Denn man wählt immer eine oder mehrere Tabellen aus.



**Abbildung 5:** SQL Server Verbindungsassistent

Bei einer Excel-Datei wird beispielsweise ein Arbeitsblatt oder eine Tabelle ausgewählt. Löscht man den Navigationsschritt in den *angewendeten Schritten* aus der Abfrage, erhält man das Ergebnis den reinen Verbindungsaufbaus. Bei Excel also eine Liste mit allen Arbeitsblättern und Tabellen beim SQL-Server eine Liste mit allen Katalogen oder *Datenbanken*.



The screenshot shows the Power Query editor with the following content:

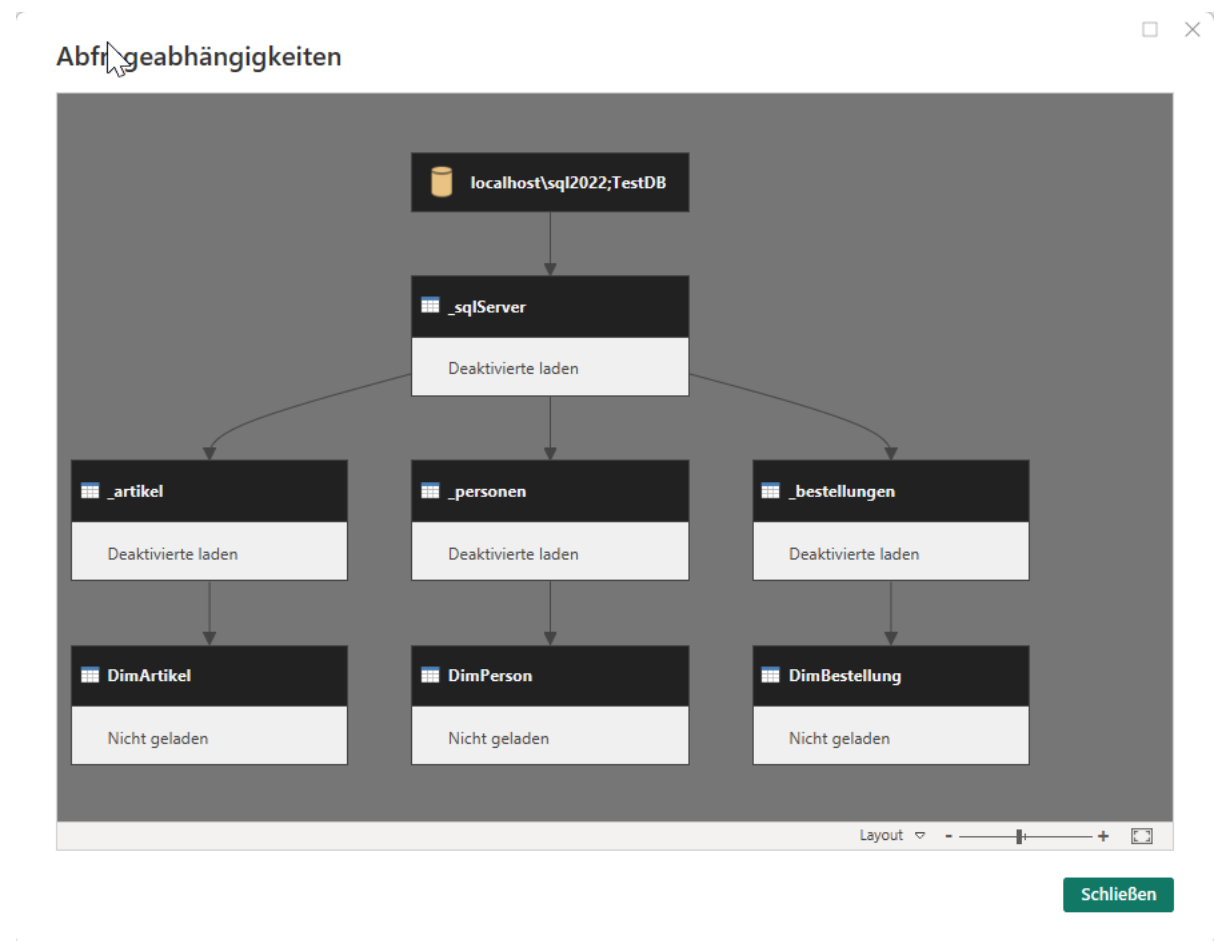
	Name	Data	Kind
1	TestDB	Table	Database

The formula bar at the top contains the query: `= Sql.Databases("localhost\\sql2022")`

**Abbildung 6:** SQL-Server Abfrage ohne Navigationsschritt

Beim SQL-Server ergibt es jetzt Sinn, in der Basisabfrage noch den Navigationsschritt in den jeweiligen Katalog durchzuführen, falls sich alle Tabellen, die benötigt werden im selben Katalog befinden.

Lade ich jetzt die Tabelle *Artikel*, erzeuge ich einen Verweis auf die SQL-Sever-Basisabfrage und erzeuge im Verweis dann den Navigationsschritt auf die Tabelle *Artikel*. Die Abhängigkeiten sehen dann am Ende bei mir so aus:

**Abbildung 7:** Abfrageabhängigkeiten in Power Query

Ich nutze drei Hauptgruppen:

- **Sources:** Hier sind nur Quell-Abfragen
- **Transformations:** Hier sind Nutzdaten und ggf. weitere abhängige Abfragen und Transformationen drin. Häufig werden hier Abfragen nochmals untereinander verknüpft, getrennt etc.
- **Datamodel:** Das sind die Abfragen gegenüber dem Datenmodell. Diese ändere ich später nicht mehr.

Dabei gilt außerdem:

- Abfragen außerhalb der Gruppe *Datamodel* werden bei mir nicht in das Datenmodell geladen.
- Das Laden der Abfrage deaktiviere ich via Kontextmenü auf der Abfrage *Laden aktivieren* - dort darf kein Haken mehr sein. Die Abfrage wird dann kursiv dargestellt.
- Jede Abfrage, die ich nicht lade, fängt bei mir mit einem Unterstrich im Namen an. Zwar sieht man durch die kursive Darstellung, dass die Abfrage nicht geladen wird. Nicht aber, wenn man in Formeln arbeitet und man auf andere Abfragen im M-Quellcode zugreifen möchte.

## Vorteile

### Gruppe Sources

Informationen auf Datenquellen müssen häufiger geändert werden:

- vom Testsystem zum Produktivsystem
- beim Ausliefern von meinem Entwicklungsrechner zu meinem Kunden
- Dateipfade werden umgezogen

Stehen alle Datenquellen an einer Stelle, in meinem Fall in einer *Gruppe*, kann ich diese Änderungen sehr schnell vornehmen. Vorzugsweise werden natürlich noch Parameter verwendet.

### Gruppe Transformations

Wenn ich Daten laden, muss ich diese normalerweise noch transformieren. Wie bereits beschrieben, kann es beliebig viele Verweise auf meine Tabelle geben. Oder ich muss Abfragen miteinander kombinieren. Vielleicht muss ich sogar bereits als Abfrage angelegte Abfragen, die bereits Transformationen besitzen, miteinander verknüpfen.

Transformationen mache ich so früh wie möglich, damit ich die gleichen Aufgaben in abhängigen Abfragen nicht mehrfach und wiederholt durchführen muss.

Abfragen, in denen Transformationen durchgeführt werden, finden sich bei mir in der Gruppe *Transformations*.



Nutze ich einen SQL-Server als Datenquelle, habe ich eine Basisabfrage `*_sqlServer`. Diese Abfrage gibt mir alle Tabellen zurück. Alle weiteren Abfragen leite ich von dieser Abfrage ab - ich erstelle also Verweise auf diese Abfrage.

Diese Vorgehensweise mutet Anfängern umständlich an. Aber: Wenn ich 30 Tabellen aus dem SQL-Server importiere und eines Tages den SQL-Server wechseln möchte, müsste ich anderenfalls in 30 Abfragen die Zugriffsdaten ändern und dabei möglichst keine vergessen. In meinem Fall basieren alle Abfragen auf der SQL-Server Basisabfrage. Damit habe ich nur noch eine Stelle, die ich ändern muss.

## Gruppe *Datenmodell*

Abfragen in dieser Gruppe sind das *Gesicht zum Datenmodell*:

- Alles was vor dieser Abfrage passiert “weiß” das Datenmodell nicht. Wenn die Daten heute aus Excel und morgen aus dem SQL Server kommen, ändert das nichts an dieser Abfrage - nur an den vorangegangenen.
- Berechnete Spalten und *Measures* in Power BI “hängen” an einer Tabelle (*Measures* hänge ich allerdings in eine eigene Tabelle) - korrekter: an einer Abfrage. Lösche ich diese Abfrage und ersetze sie durch eine neue, werden alle diese Spalten und *Measures* ebenfalls gelöscht. Da ich das nicht will, bleibt diese Abfrage in der Gruppe “Datenmodell” bestehen.
- Die einzigen Transformationen, die hier noch vorkommen, sind
  - die Auswahl von Feldern
  - selten auch noch hinzugefügte, benutzerdefinierte Spalten

**Mein wichtigster Vorteil ist aber:** Wenn ich selbst meine Datenmodell nach einiger Zeit anschau, verstehe ich dieses schneller. Halten sich im Unternehmen Power BI Entwickler immer an die gleiche oder zumindest ähnliche Struktur, können sich Entwickler leichter gegenseitig helfen.

Meine Erfahrung ist, dass die Zeitersparnis und Wartbarkeit mit einer durchgängigen Struktur enorm sind.

# Automatische Typerkennung deaktivieren

## Die Aufgabe

Wenn ich etwas nicht mag, dann die automatische Erkennung von Datentypen in Power Query. Wenn ich Excel- oder CSV Dateien importiere, dann nutzt Power Query die erste Zeile als Kopfzeile - und dann wird für jede Spalte automatisch der Datentyp ermittelt. Das mag scheinbar angenehm sein, führt aber zu blöden Fehlern, nach denen man lange suchen muss.

## Unstrukturierte vs. strukturierte Datenquellen

Die Entwickler von Power Query haben eine Einteilung von Datenquellen vorgenommen:

- unstrukturierte und
- strukturierte Datenquellen.

Unstrukturierte sind beispielsweise *Excel-* oder *CSV-Dateien*. Strukturierte dagegen sind Daten beispielsweise aus dem SQL-Server. *Un-/strukturiert* hört sich wertend an. Korrekt müsste es heißen: *Datenquellen mit oder ohne Datentyp-Informationen*: Eine CSV-Datei besteht nur aus Text. Der Empfänger der Datei muss "Wissen", dass eine Spalte keine Zahl sondern ein Datum ist. Oder das die PZN (Pharmazentralnummer) zwar wie eine Zahl aussieht, aber ein Text ist (weil sie führende Nullen beinhaltet).

Eine Datenbank hingegen liefert in den sogenannten *Metadaten* (die von Power Query abgefragt werden) eine exakte Typdefinition mit. Daher muss der Datentyp hier nicht abgeleitet werden, sondern wird direkt von der Quelle übernommen.

## Am Beispiel von Dateien

Das Importieren von Dateien ist *daily business*. Oftmals Dateien, auf deren Format ich leider keinen Einfluss habe. Diese Dateien werden fast immer manuell erzeugt und dann abgelegt oder verschickt. Aus diesen Dateien benötige ich meist nur eine handvoll Spalten. Alle anderen interessieren mich nicht. Meine Lieblingsspalten haben Namen wie

- *Umsatz 2022 YtD*
- *oder Absatz Q1 2024*

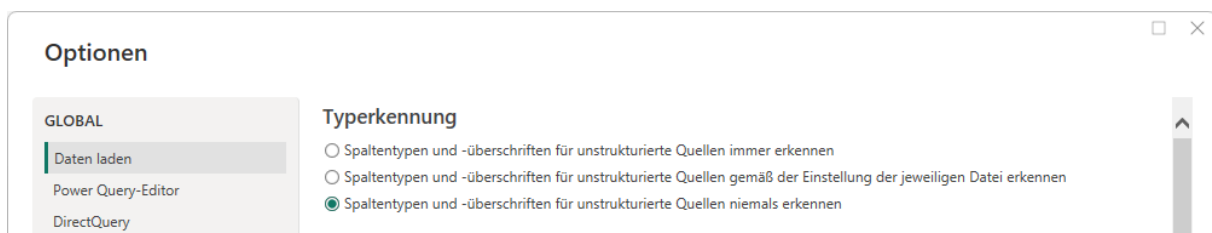
Bei diesen Spalten weiß ich sofort, dass diese Spalten nicht wiederkommen werden. Spätestens im nächsten Quartal oder im nächsten Jahr.

Wenn Power Query jetzt diese Datentypen ableitet, dann nutzt er die Spaltennamen der Spalten. Also auch die Spaltennamen *Umsatz 2022 YtD* oder *Absatz Q1 2024*. Wenn die Datei dann aktualisiert wird und diese Spalten nicht mehr da sind (weil die heißen jetzt eben *Umsatz 2023 YtD* oder *Absatz Q2 2024*), schlägt die Aktualisierung fehl. Und das gesamte Datenmodell wird nicht mehr aktualisiert.

Gerade Anfänger importieren die Datei, merken diese Ableitung der Datentypen nicht, löschen dann in den weiteren Transformationen die Spalten, da sie diese für die Auswertung nicht benötigen und merken erst mit der Datenlieferung ein Quartal später, dass plötzlich Fehler enthalten sind.

## Best Practise: Deaktiviere die automatische Typerkennung

Wie gut, dass man Power Query beibringen kann, diesen Automatismus nicht mehr anzuwenden. Dazu wechselt man in Power Query oder in Power BI (das ist der gleiche Dialog) m Menü unter *Datei* in die *Optionen und Einstellungen* und dort in die *Optionen*. Gleich als erste Option kann man das Verhalten ändern:



**Abbildung 8:** Typerkennung deaktivieren in Power Query

## Vorteil

Nun, der ist schnell erklärt: Man muss diesen Schritt nicht immer manuell entfernen.

# Letzte Berichtsaktualisierung

## Aufgabe

Mein Chef schaut auf den Bericht und sagt: Der Report ist falsch. Wir hatten doch gestern eine Großbestellung!

Das Problem: Die Daten, die er sieht, sind zwei Wochen alt. Er weiß es aber nicht.

## Best Practise: Ein Aktualisierungsdatum- oder Zeitstempel einfügen

In mittlerweile allen meinen Projekten füge ich eine *Last Updated Info* ein. Damit der Leser sofort weiß, wann der Bericht das letzte mal aktualisiert wurde. Dazu gibt es auch einen Blog-Post in meinem Blog unter

<https://www.durchblick-durch-daten.de/aktualisierung-anzeigen/>

Power BI teilt uns das letzte Aktualisierungsdatum nicht mit. Das geht nur mit einem Trick: In Power Query erzeuge ich eine Abfrage, die das aktuelle Systemdatum ausliest. Dazu erzeuge ich eine Tabelle mit genau einer Zeile und einer Spalte *TS* (für *TimeStamp*). Diese Information gebe ich dann an das Datenmodell weiter. Die Abfrage nenne ich *LastUpdateInfo*

Wenn alle Daten aktualisiert werden, wird auch diese *LastUpdateInfo* Abfrage aktualisiert. Mit einer geeigneten Measure in DAX kann ich diesen Zeitstempel beispielsweise in einem *Card Visual* nutzen:

```
1 Last Update Info = MAX>LastUpdatedInfo[TS])
```

Oder als *Display-Measure* um es etwas lesbarer zu machen:

```
1 Last Update Info Display = FORMAT(  
2     MAX>LastUpdatedInfo[TS]), "dd.MM.yyyy")  
3 )
```

Dieses Measure nutze ich dann in einem *Card-Visual*, stelle die Schriftgröße auf 8, deaktiviere die Legende und setze die Information in die Kopf- oder Fußzeile. Oftmals auch nur auf der ersten Seite.

## Vorteile

Ein Bericht-Betrachter weiß sofort, welchen Stand der aktuelle Bericht hat. Gerade bei Kennzahlen, wie *Year-To-Date*, ist es wichtig zu wissen, wann dieses *to-Date* denn eigentlich ist.



Wenn die Daten, die importiert werden, nicht täglich aktualisiert werden (weil es sich beispielsweise um Dateien handelt, die nur wöchentlich ausgetauscht werden), bringt eine tägliche Aktualisierung und das aktuelle *Last Update Date* wenig. Dann ist es zusätzlich sinnvoll, beispielsweise das letzte Rechnungsdatum oder das letzte Auftragsdatum anzuzeigen. Achtung: zusätzlich!



# Spalten nicht Löschen

## Aufgabe

Für meine Auswertungen benötige ich meistens nur sehr wenige Spalten. Greift man auf produktive Systeme zu, haben Tabellen gerne mal 200 oder mehr Spalten. Oder Excel- und CSV-Dateien bringen viele Spalten mit, die ich nicht benötige.

*Mein Grundsatz ist:* Nur das importieren, was ich nach aktuellem Kenntnisstand benötige. Sonst nichts!

In Power Query verleitet die Oberfläche dazu, unnötige Spalten zu markieren - und dann einfach auf [ENTF] drücken. Weg sind die Spalten. Das wird teilweise auch in YouTube-Videos oder sogar von Schulungsunternehmen sogar gelehrt...

## Warum [ENTF] keine gute Idee ist

Stellen wir uns folgendes Szenario vor:

Ich bekomme jeden Monat eine Auswertung von einem Lieferanten oder Dienstleister. Immer als CSV-Datei. Die Datei enthielt schon immer viel Ballast, den niemand bisher benötigt hat. Oder die Datei hat variable Spalten, wie *Umsatz Q1 QtD 2024* die dann irgendwann *Umsatz Q2 QtD 2024* heißen.

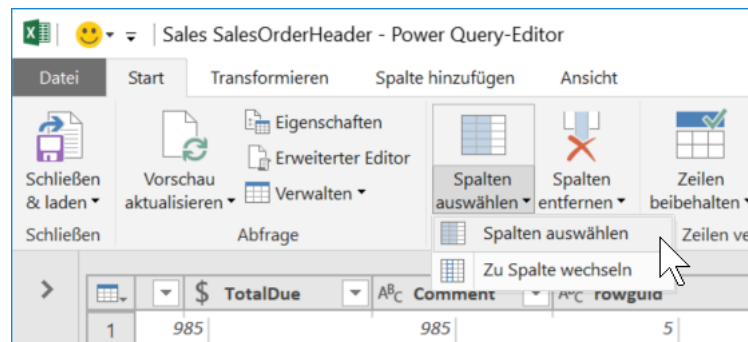
Ich brauche nur die Spalten *Jan, Feb, Mar, Apr, ....* Also markiere ich alle anderen und lösche diese mit [ENTF]

Im nächsten Monat aber werden in der neuen Datei viele dieser gelöschten Spalten nicht mehr geliefert oder sind eben umbenannt, weil sich das Quartal oder das Jahr geändert haben. Auf Basis dieser Daten aktualisiere ich aber jetzt meine Auswertung. Die Aktualisierung schlägt fehl. Denn Power Query versucht Spalten zu löschen, die nicht mehr da sind.

*Anders ausgedrückt:* Ich kann nicht Aktualisieren, weil Spalten in der Datenquelle fehlen, die Power Query versucht zu löschen, weil ich die gar nicht benötige... Bisschen paradox, oder?

## Best Practise: Spalten auswählen statt löschen

Das *scheint* ein nur kleiner Unterschied zu sein. Ich teile Power Query mit, welche Spalten ich will. *Alle anderen* soll er nicht importieren. Fehlen Spalten, die ich für die Auswertung benötige, dann ist es in Ordnung, wenn die Aktualisierung fehlschlägt:



**Abbildung 9:** Funktion Spalten wählen in Power Query

Mit dieser Funktion wählen ich nur Spalten aus, die ich weiterverwenden will. Alle anderen Spalten sind nach dieser Transformation nicht mehr enthalten. Eine Änderung in der Datenquelle hat dann folgende Auswirkungen:

- **unnötige Spalte wurden umbenannt:** Macht nichts. Da die Spalte nicht explizit gelöscht wird, ist der Name nicht relevant
- **unnötige Spalte sind nicht mehr vorhanden:** Macht auch nichts. Ich greife auf die Spalte nicht zu
- **es gibt neue, nicht benötigte Spalten:** Die Daten werden nicht importiert. Der Lieferant kann immer mehr Spalten liefern. Mein Datenmodell wird nicht größer dadurch

Wenn natürlich relevante Spalten umbenannt oder geändert werden, dann muss ich natürlich eingreifen.

Hier kommt auch der Punkt mit der automatischen Typerkennung ins Spiel, der in einem anderen Kapitel beschrieben wird: Werden alle Spalten automatisch als erstes erkannt, wird dadurch auf die Spaltennamen zugegriffen. Fehlen die Spalten bei der nächsten Aktualisierung, führt auch das zu einem Fehler. Bei Neulingen ein absoluter Standardfehler.

# OneDrive

## Aufgabe

Ich nutze (mittlerweile) viel den Online Speicher *OneDrive*. Nicht weil er für mich die technologisch beste Lösung ist, sondern weil OneDrive so ziemlich überall vorhanden ist. Wo lege ich die Dateien aber ab, damit ich diese nachher im Power BI Service vernünftig aktualisieren kann?

OneDrive? SharePoint? Teams? - ein kurzer Einschub:

## Einschub: OneDrive vs. SharePoint vs. Teams

Das Marketing und die Produktnamen bei Microsoft sind verwirrend. Die *Cloud* ermöglicht es Anbietern, wie auch Microsoft, immer mehr und immer schneller Produkte auf den Markt zu werfen. Man fühlt sich schnell überfordert. Die daraus resultierende Unsicherheit spüre ich bei vielen meiner Kunden.

Es war einmal SharePoint. Tatsächlich ein altes Produkt. Die absoluten Ursprünge gehen wohl bis 1996 zurück. SharePoint wurde entwickelt, um Dateiablage und das WEB zusammenzubringen und "echte" Zusammenarbeit (außerhalb des Firmennetzwerks) zu ermöglichen. Als zentrale Dokumentablage wurde SharePoint als Intranet-System genutzt, mit dem man neben der Dateiablage auch Webseiten erstellen kann. Auch hier mit hierarchischem Gedanken. Also Webseiten für das gesamte Unternehmen, für Abteilungen bis hin zu speziellen Projekten.

Es kamen SharePoint-Applikationen hinzu. Beispielsweise Zeiterfassungen oder Zugriff auf SAP über zentrale Schnittstellen. Das Herz bleibt aber bestehen: Die Dokumentablage oder im Fachjargon *Document library*. Der Zugriff auf Dateien in SharePoint erfolgte über den Browser. Dabei ist man als Benutzer den Datei-Explorer gewohnt. Also haben wir hier eine Lücke in der sog. *Benutzererfahrung*.

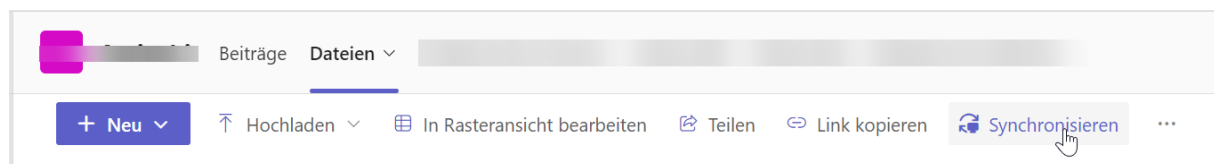
Hier kommt *OneDrive* ins Spiel: OneDrive ist kein eigenes Produkt. Auch wenn es so aussieht. OneDrive ist die *Benutzererfahrung* auf eine *SharePoint DocumentLibrary*\* zugreifen zu können, wie auf ein lokales Laufwerk. Das bedeutet, wenn ich Zugriff auf eine *SharePoint DocumentLibrary* habe, kann ich diese mit dem OneDrive Client automatisch mit meiner Festplatte synchronisieren, kann mit den Dateien lokal arbeiten und muss mir über die "Rückspeicherung" in die *Cloud* keine Gedanken mehr machen.

Im Privatleben gibt es OneDrive. Dazu benötige ich eine private Microsoft-Adresse (via *Live*) und dann steht mir der Speicher zur Verfügung. Praktisch wird im Hintergrund eine *SharePoint DocumentLibrary* erzeugt, auf die nur ich Zugriff habe.

Nutzt mein Unternehmen Office365, dann ist SharePoint Teil davon. Ganz automatisch erhalte ich mit meinem Business Account einen eigenen OneDrive-Bereich. Der heißt dann *OneDrive4Business*. Letztlich auch eine SharePoint *DocumentLibrary*. Auf die habe nur ich, also mein Unternehmensaccount Zugriff.

Jetzt kann mein Unternehmen beliebig viele SharePoint-Instanzen oder *Sites* anlegen. Und Benutzern darauf Lese/Schreibrechte geben. In diesen *Sites* kann ich wiederum *DocumentLibraries* anlegen, so dass Benutzer Dateien ablegen und gemeinsam daran arbeiten können.

Und dann kommt *TEAMS* ins Spiel. Lege ich ein neues *Team* an, wird automatisch eine SharePoint *Team-Site* erzeugt. Und die hat immer eine *DocumentLibrary*. Damit können alle im Team automatisch auf Dateien zugreifen. In der Desktop Applikation von *Teams* finde ich für jedes Team auch *Dateien* - und das ist eben nichts anderes als die SharePoint Webseite auf diese *DocumentLibrary*. Und wenn ich diese Dateien mit meiner lokalen Festplatte synchronisieren möchte (wie beim persönlichen OneDrive eben auch), dann wähle ich in *Teams*, wenn ich im Reiter *Dateien* bin, die Funktion *Synchronisieren* aus.



**Abbildung 10:** Dateien in Teams via OneDrive synchronisieren

Dann wird in meinem Explorer automatisch ein neuer Knoten mit dem Namen meines Unternehmens angelegt. Darunter finde ich das *TEAM*, dass ich synchronisiert habe.

Bin ich Teil mehrerer Teams, dann werden diese alle unter diesen Unternehmensknoten angelegt. Bin ich - so wie ich als Berater - Teil von Teams in mehreren Unternehmen, kann ich diese *DocumentLibraries* auch synchronisieren. Für jedes verschiedene Unternehmen erhalte ich einen entsprechenden Unternehmensknoten. Eigentlich ganz schön praktisch, oder?

## Best Practise: OneDrive als Datenquelle nutzen

Ich komme bei meinen Datenauswertungen bei meinen Kunden selten um Dateien als Datenquellen herum. Es gibt eben doch immer Daten aus Systemen oder von Lieferanten oder Drittanbietern (Marktdaten), die eben nicht in lokalen Systemen direkt auslesbar sind.

Wenn ich Datenmodelle aufbaue habe ich immer im Blick, wie diese später im Power BI Server - also online in der *Cloud* - automatisch aktualisiert werden können. Dateien im Firmennetzwerk gestalten

sich dabei immer als schwierig: Man benötigt den *On Premise Data Connector* von Microsoft, der zentral auf einem Server installiert sein muss (wir müssen also die IT involvieren). Selbst wenn es diesen Connector schon gibt, weil ich ohnehin auf lokale Datenbanken zugreifen muss, sind Dateien immer nochmal besonders zu behandeln. Denn nur weil ein Benutzer ein Netzlaufwerk "Z" besitzt, hat der Dienst-Benutzer, unter dem dieser *On Premise Data Connector* läuft, dieses Netzlaufwerk noch lange nicht. Und ob der Dienst-Benutzer überhaupt Zugriffsrechte hat auf die Netzlaufwerk... Lange Rede kurzer Sinn: Es ist anstrengend.

Daher: Warum nicht einfach Dateien in SharePoint/Teams/OneDrive ablegen. Wenn die Nutzung seitens des Unternehmens erlaubt ist, können notwendige dort abgelegt werden. Möglichst nicht in das eigene OneDrive-Verzeichnis, sondern in die *Document Library* eines Teams. Gibt es beispielsweise ein Team *Controller*, kann ich für die DATEV-Buchungsdateien einfach die dahinter liegende *DocumentLibrary* nutzen. Der administrative Vorteil: Die Dateien müssen meist manuell aktualisiert werden. Das kann jetzt aber jeder Benutzer, der Teil des *Teams* ist.

Liegt die Datei in SharePoint, ist sie nicht mehr im Firmennetzwerk, sondern in Office365 und damit in *derselben* Cloud, in der sich auch der Power BI Service befindet.

Wie das genau geht, steht in diesen beiden Artikeln:

- <https://www.durchblick-durch-daten.de/power-bi/teamarbeit-power-bi-und-onedrive/>
- <https://www.durchblick-durch-daten.de/power-bi/teamarbeit-power-bi-und-onedrive-nachtrag/>



Wichtig zu beachten ist: Bei dieser Vorgehensweise kann ich prima mit CSV-Dateien oder Excel-Dateien, im XLSX oder XLSM Format arbeiten. Das alte xls-Datenformat geht leider nicht. Bitte auch keine Access-Datenbanken o.ä. nutzen. Das geht mit der OneDrive Variante nicht.

## Vorteile

Wichtigster Vorteil ist sicherlich, dass wenn man um Dateien als Datenquelle nicht herum kommt, die notwendigen Dateien und die leicht hinterlegen und später aktualisieren kann. Der Power BI Service kann diese Dateien ohne große administrative Eingriffe nutzen. Also ohne die IT-Infrastruktur zu involvieren. Denn das Firmennetzwerk stellt keine Grenze mehr dar.

Ein weiterer Vorteil ist, dass mehrere Team-Mitglieder auf die Quelldateien zugreifen können. Gerade im Vertretungsfall sind solche Themen ohnehin relevant. Natürlich geht das auch via Netzlaufwerk. Allerdings ist die Rollen/Rechte Administration dort deutlich komplexer.

Und der Zugriff auf die Dateien wird auch außerhalb des Firmennetzwerks und ohne Aufbau eines VPN-Tunnels ermöglicht. Das ist deutlich angenehmer für die Benutzer, die Aktualisierungen durchführen oder Datenmodelle entwickeln. Denn VPNs ins Firmennetzwerk sind nicht für ihre Geschwindigkeit bekannt.

## Geiz ist geil (bei Spalten)

### Aufgabe

Meine Kunden verdrehen meistens die Augen, wenn ich frage: "Wozu genau brauchen Sie diese Spalte denn?".

Datenmodelle werden schnell größer und damit unübersichtlicher. Gleichzeitig wächst die Größe der Datei. Und dann werden die Visuals auf der Oberfläche immer langsamer. Und plötzlich bekomme ich nach der Veröffentlichung im Power BI Service einen Fehler, dass die Daten nicht mehr angezeigt werden können.

### Best Practise: Sei geizig mit Spalten

Wenn ich Abfragen erstelle kommt immer die Stelle, an der ich *Andere Spalten entferne* - ein Transformationsschritt in Power Query. Das bedeutet, ich lösche keine Spalten, sondern wähle die Spalten, die ich benötige. Dabei bin ich möglichst geizig (warum man keine Spalten "löschen" sollte, erläutere ich in einem anderen Abschnitt).

Zur Erläuterung, warum das so wichtig ist:

### Spaltenbasierte Datenbank

"Herkömmliche" Datenbanken sind Zeilen-orientiert. Möchte ich den Vornamen einer Person haben, dann muss die Datenbank die Zeile der Person suchen, alle Felder laden, um dann den Wert der Spalte *Vorname* zurückzugeben.

Das ergibt in sogenannten OLTP (Online Transactional Processing) Datenbanken Sinn: Hier werden Adressen angelegt, gesucht, geändert oder gelöscht. Abfragen, wie *wie viele Personen, die männlich und unter 20 Jahren sind, haben pro Monat wie viele Abos für Online Spiele ausgegeben* sind in solchen Datenbanken nicht optimal. Weil dazu muss die Datenbank sehr viele Felder laden, die wir für die Auswertung gar nicht benötigen.

Power BI nutzt eine andere Datenbankstruktur: eine Spalten-basierte Struktur. Das bedeutet praktisch, dass die Spalten getrennt voneinander abgelegt werden. Natürlich ist der Zusammenhang einer Zeile wieder herstellbar. Aber stellt man sich vor, man hat eine Tabelle mit 100.000 Zeilen mit folgende Spalten:

- Vorname

- Nachname
- Geburtsdatum
- Geschlecht
- Straße
- PLZ
- Ort
- Land

Betrachten wir nur die Spalte *Geschlecht*. Und speichern nicht 100.000 mal das Geschlecht, sondern sortieren die Werte und legen folgende Information ab:

Geschlecht	Zeile von	Zeile bis	Anzahl
M	1	53.102	53.102
W	53.103	98.324	45.222
D	98.325	100.000	1.676

Wenn ich jetzt wissen will, wie viele Personen das Geschlecht *M* haben, muss ich nicht 100.000 Zeilen und alle unnötigen Feldern laden, sondern kann direkt in Spalte *Anzahl* schauen. Alle anderen Spalten sind für mich nicht relevant.

Es gibt noch viele weitere *Komprimierungsvorgänge* in Power BI - dieses Beispiel soll nur dem grundsätzlichen Verständnis dienen.

## Unnötige Felder

Unnötige Felder nehmen unnötigen Speicherplatz weg. Insbesondere Primärschlüssel, die natürlich pro Zeile eindeutig sind, können nicht komprimiert werden. Arbeitet man mit sogenannten *GUIDs*, also "Global Unique Identifier" wird das Problem schnell deutlich.

Eine GUID sieht beispielsweise so aus: 550e8400-e29b-11d4-a716-446655440000

Die GUID aus dem Beispiel besteht aus 36 Zeichen. Power BI speichert die Daten im Unicode Format, damit auch kyrillische oder chinesische Schriftzeichen kein Problem darstellen. Damit benötigt ein Zeichen 2 Byte. Oder unsere GUID aus dem Beispiel oben 72 Byte. Bei 1.000.000 Zeilen, die bei Aufträgen oder Journaleinträgen in einem Buchhaltungssystem durchaus vorkommen können, sind das  $1.000.000 * 72 \text{ Byte} = 72.000.000 \text{ Byte}$  oder knapp 66 MB (zweimal durch 1.024 teilen). 66MB nur für ein Feld, das wir nicht mal benötigen.



## Übersichtlichkeit

Viele Felder auf der rechten Seite von Power BI machen das Datenmodell sehr schnell sehr unübersichtlich. Kollegen oder auch ich selbst finden sich irgendwann nicht mehr zurecht.

## Performance

Nicht alle Nutzer sind fähig, guten, performanten DAX Code zu schreiben. Aber einem gewissen Punkt ist das auch verständlich: Nicht jeder, der im Unternehmen nebenher Power BI Auswertungen macht, will Power BI Berater werden.

Schnell lernt man mit sog. *Iteratoren* zu arbeiten. Also Formeln zu nutzen, die beispielsweise so aussehen:

```
1 Umsatz = SUMX(  
2     FILTER(  
3         Orders,  
4         Orders[Cancelled] = "N"  
5     )  
6 )
```

Hier wird die Tabelle *Orders* genutzt und intern *materialisiert*. *Materialisieren* bedeutet, dass alle Felder der Tabelle als Zeile geladen werden, also die jeweiligen Zeilen wiederhergestellt werden. Das ist in Power BI aufwändig. Dieser “Fehler” kommt häufig vor - und wäre verschmerzbar (da die Datenmengen nicht immer so groß sind), wenn denn nicht zahllose **nicht genutzt Spalten** in den jeweiligen Tabellen vorkommen würden.

## Vorteile

Die Vorteile wurden schon aufgezählt:

- Dateigröße
- Performance
- Übersichtlichkeit des Modells

# Automatische Kalendertabellen

“...und wenn Sie keine haben, dann wird Ihnen eine gestellt”\*.

## Aufgabe

So wie es mit dem Verteidiger in amerikanischen Filmen der Fall ist, verhält es sich in der Standard-einstellung auch in Power BI: Nur, dass hier eine Kalendertabelle und kein Verteidiger gestellt wird.

Ein Kunde berichtet, er habe eine CSV-Datei mit sagen wir mal 10 KB Größe in Power BI importiert. Es hat nichts weiter getan und die Power BI-Datei als *PBIX-Datei* gespeichert. Diese hat jetzt aber über 100 KB. Wie kann das sein? Hatte ich nicht erzählt, wie gut die Komprimierung in Power BI funktioniert?

## Kalendertabellen

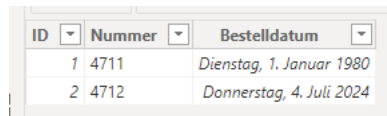
Eine Kalendertabelle kommt in fast jedem Datenmodell vor. Nahezu alle Auswertungen haben eine zeitliche Dimension. Wie man damit umgeht, wie man diese aufbaut, wie man Feiertagen umgeht, beschreibe ich in meinem Blog in den Artikeln

- <https://www.durchblick-durch-daten.de/wozu-eine-Kalendertabelle>
- <https://www.durchblick-durch-daten.de/eine-Kalendertabelle-erzeugen>
- <https://www.durchblick-durch-daten.de/kalendertabelle-mit-feiertagen-in-power-query-erzeugen>
- <https://www.durchblick-durch-daten.de/kalendertabelle-mit-feiertagen-in-dax-erzeugen>

## Datumsfeld ohne Kalenderdimension

Microsoft hat versucht die Einstiegshürde bei Power BI gering zu halten. Das ist prinzipiell auch gut so. Leider nutzen insbesondere Anfänger dann aber Funktionen, die in ernsthaften Auswertungen zu Performance- oder Größenproblemen führen. Oder machen Measures nicht nutzbar. Eine dieser Funktionen ist die automatische Anlage von Datums-Tabellen.

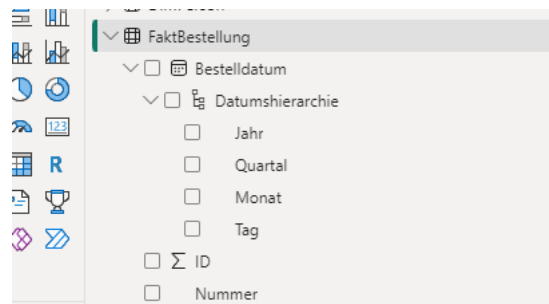
Mein Kunde in obigem Beispiel hat mehrere Datumsfelder in seiner CSV-Datei. Habe ich aber in meinen Quelldaten ein Datumsfeld, erzeugt Power BI im Hintergrund eine unsichtbare Datumstabelle. Nehmen wir eine Bestelltabelle mit nur zwei Bestellungen:



ID	Nummer	Bestelldatum
1	4711	Dienstag, 1. Januar 1980
2	4712	Donnerstag, 4. Juli 2024

**Abbildung 11:** Beispiel-Bestellungen in Power BI

Ich importiere die Tabelle als *FaktBestellung*. Power BI erzeugt aus dem Datum automatisch eine Datumshierarchie, die ich für Auswertungen nutzen könnte.



**Abbildung 12:** Automatische Datumshierarchie in Power BI

Was hat es mit dieser Hierarchie auf sich, kann man die beeinflussen und welche Auswirkungen hat das, wenn ich so arbeite? Um das sichtbar zu machen brauchen wir das *Dax Studio*.

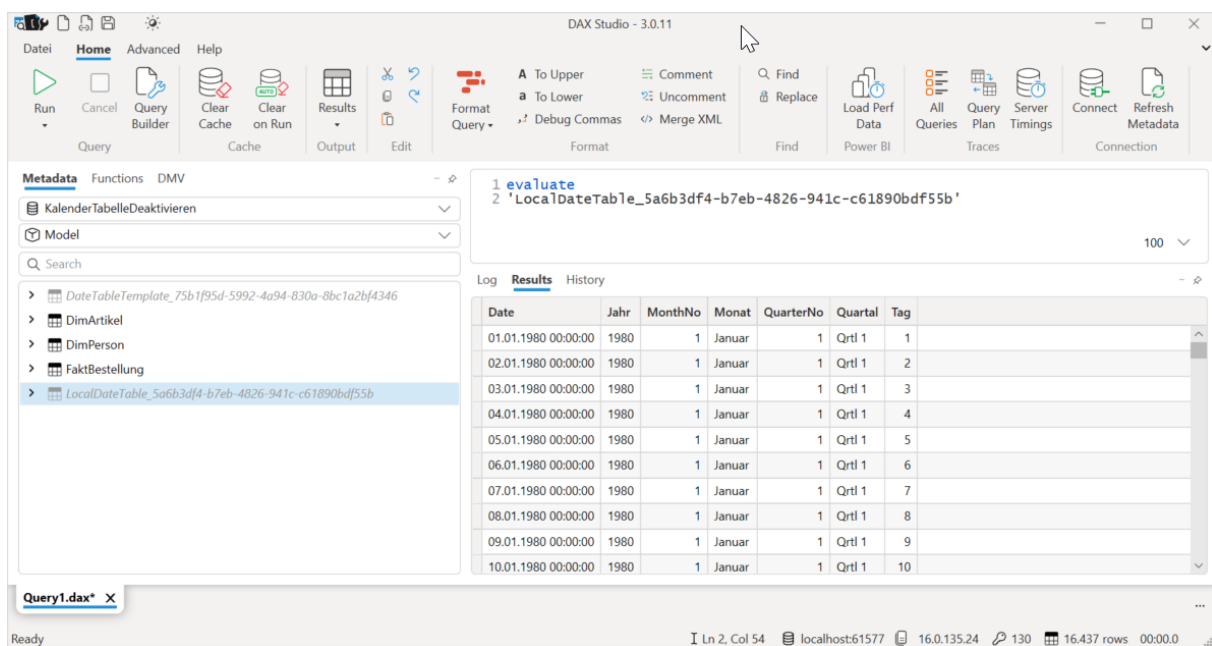
## DAX Studio

Mit dem Werkzeug *Dax Studio* (kostenfrei erhältlich unter <https://daxstudio.org/>) kann man das Datenmodell technisch betrachten. Nach dem Öffnen von DAX-Studio verbinde ich mich mit meinem Datenmodell (es wird eine Liste mit allen geöffneten PBIX-Dateien angeboten). Die Informationen aller meiner Tabellen im Datenmodell werden angezeigt. Hier erwarte ich eigentlich drei Tabellen, nämlich *DimArtikel*, *DimPerson* und *FaktBestellung*. Da ist aber eine vierte:



**Abbildung 13:** Lokale Datumstabelle im DAX-Studio

Ich kann mir in Dax Studio auch die Inhalte der Tabelle ansehen:



**Abbildung 14:** Inhalt der lokalen Datumstabelle in DAX Studio

Power BI erzeugt also für mein Datumsfeld automatisch eine Kalendertabelle. Jeweils immer vom kleinsten bis zum größten Datum und jeweils vom zum 1.1. bis zum 31.12. Ich habe mal absichtlich das Jahr 1980 genommen. Damit haben wir 44 Jahre ( $2024 - 1980 = 44$ ) und damit 16.437 Zeile (Schaltjahre beachten!).

Wenn ich eine zweite Datumsspalte einfüge, wird freundlicherweise auch eine zweite Datumstabelle angelegt. Bei einer dritten Datumsspalte eine dritte. Und so weiter. Kommt ein Zeitstempel hinzu, wird aus dem Datumsanteil des Zeitstempels ebenfalls eine Datumstabelle erzeugt.

## Ich bin Profi, ich nutze ein Datumstabelle

Datumstabellen haben sich langsam durchgesetzt. Und kaum habe ich das Bestelldatum aus dem vorherigen Abschnitt mit einer Datumstabelle verknüpft, verschwindet die Datumshierarchie aus der *FaktBestellung*

Jetzt werde ich die Zeitverbuchung eines Kunden aus. Diese hat beispielsweise folgende Felder

- CreatedAt (also das Anlagedatum der Verbuchung)
- Start (Anfangs-Zeitstempel der Arbeitszeit)
- Ende (Ende-Zeitstempel der verbuchten Arbeitszeit)
- StartDate (das Datum des Start-Zeitstempels)
- EndDate (das Datum des End-Zeitstempels)

Für die Einordnung der verbuchten Stunden benötige ich nur

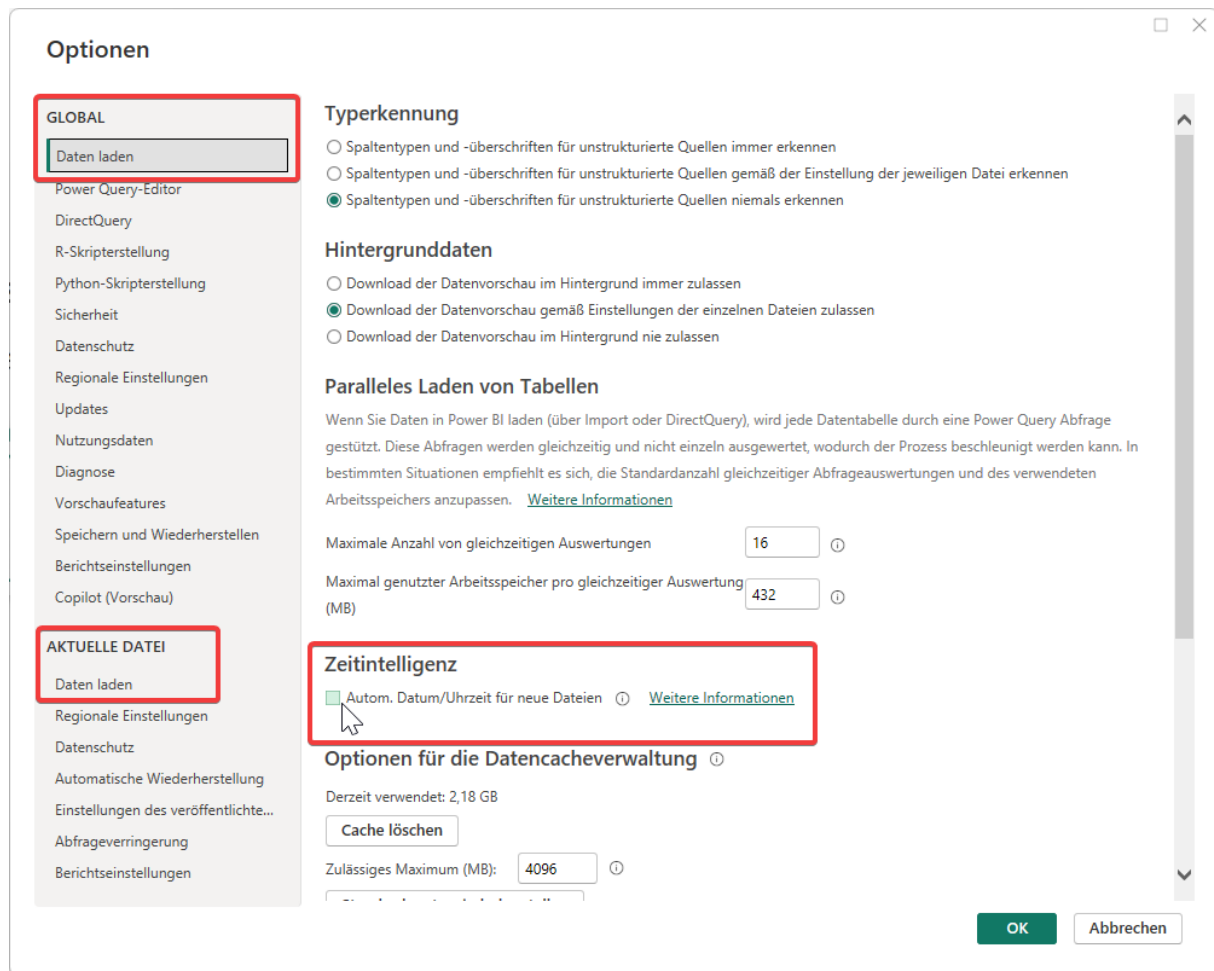
- **EndDate:** Ich ordne die Zeit der geleisteten Arbeit auf das End-Datum ein
- **Dauer in Minuten:** Ich nehme die Differenz aus Start- und Ende-Zeitstempel und lasse mir diese in Minuten ausgeben (mit Power Query ist das kein Problem)
- Im Datenmodell verknüpfe ich die Kalendertabelle mit dem EndDate.

Die anderen Zeitstempel lasse ich aber im Datenmodell bestehen - warum auch immer (das machen leider viele!). Dann erzeugt mir Power BI **VIERT** Datumstabellen, die ich nicht benötige. Es sind übrigens nicht 5, da das *EndDate* mit meiner eigenen Kalendertabelle verknüpft wird und dann erzeugt Power BI keine eigene.

## Best Practise: Automatische Kalendertabellen deaktivieren

**Grundsatz:** Nichts importieren, was man nicht benötigt. Das kann man gar nicht häufig genug sagen.

Manchmal brauche ich aber die anderen Zeitstempel, um beispielsweise über spätere DrillDowns die Quelldaten anzeigen zu lassen. Daher deaktiviere ich zusätzlich die Funktionalität der automatischen Generierung solcher Datumstabellen. Über *Datei/Optionen und Einstellungen/Optionen* finde ich den Dialog für die Einstellungen von Power BI. Dort kann man dieses Verhalten abstellen:

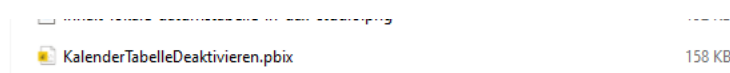


**Abbildung 15:** Automatische Kalendertabelle in Power BI deaktivieren

Hier muss man beachten, dass es diese Einstellung zweifach gibt: Einmal in den globalen Einstellungen, die ab dann für alle neuen Dateien gelten und im unteren Bereich für die aktuelle Datei. Ggf. muss man das eben noch in den Einstellungen der aktuellen Datei abändern.


## Speicherplatz

Ich schaue mir mal die Datei mit den beiden Zeilen in *FaktBestellung* an. Mit den beiden Bestellungen brauchen wir 158KB Speicherplatz.



**Abbildung 16:** Dateigröße mit automatischer Kalendertabelle

Dann deaktiviere ich die automatische Erzeugung der Kalendertabelle und speichere nochmals - 42KB

 KalenderTabelleDeaktivieren.pbix

42 KB

**Abbildung 17:** Dateigröße ohne automatischer Kalendertabelle

## Vorteile

Der Hauptvorteil liegt natürlich im Speicherplatz. Da das Datenmodell immer komplett im Arbeitsspeicher vorhanden sein muss, betrifft das also die nicht nur die Dateigröße, sondern auch die Größe im Hauptspeicher und betrifft damit auch die Performance.

Ganz ohne eigene Kalendertabellen sind viele Zeitfunktionen, wie beispielsweise Vorperiodenvergleiche gar nicht möglich. Mit den automatisch angelegten Kalendertabellen geht das nicht.

# Wann ist “Heute”

## Aufgabe

Die meisten Auswertungen haben einen Zeitbezug. Nehmen wir den Umsatz im aktuellen Jahr. Die Measure für den Umsatz ist einfach, beispielsweise

```
1 Umsatz = SUM(FaktSalesLine[LineTotal])
```

Damit haben wir aber den Umsatz - aber nur den gesamten Umsatz. Nicht den Umsatz im aktuellen Jahr. Nutzen wir ein *Card-Visual*, können wir in den Filtern für das Visual das Datum der Kalendertabelle einfügen, den Filtertyp auf *relatives Datum* stellen und dann das aktuelle Jahr wählen.

Was aber, wenn wir von Aufträgen sprechen? Also von *FaktOrderLine*? Und wenn es bereits angelegte Aufträge in der Zukunft gibt? Dann stellt sich sofort die Frage, ob wir am 3. Juli 2024 die Aufträge vom 20. September 2024 mitzählen sollen. Falls nein, kommt gleich die Anschlussfrage: Bis wann sollen wir denn die Aufträge zählen?

- bis *Heute* - gemäß aktuellem Systemdatum?
- bis *gestern* - gemäß aktuellem Systemdatum (weil die Daten beispielsweise nachts aktualisiert werden)?
- bis zum Ende des letzten Monats?

## TODAY()

In DAX gibt es eine Funktion namens *TODAY()*, die das aktuelle Systemdatum zurückgibt. Wir könnten obige Measure ändern, so dass nur Umsätze in der Vergangenheit, also bis *gestern* beachtet werden:

```
1 Umsatz =  
2 VAR t = Today() // Variable deswegen, weil CALCULATE() an der Stelle  
3                 // unten keine Funktion erlaubt  
4 VAR curYear = YEAR(t)  
5 RETURN  
6     CALCULATE(  
7         SUM(FaktSalesLine[LineTotal]),  
8         DimCalendar[Date] < t  
9         && DimCalendar[Year] = curYear  
10    )
```

## YtD - Year to Date

Unter *YearToDate* (*YtD*) versteht man die Umsätze vom 1. Januar des aktuellen Jahres bis...tja: *Heute*. Die Formel oben nutzt *TODAY()* und gibt damit bereits die Werte *Year-To Date* zurück.



Eine Standard-Berechnung ist jetzt der Vorjahresvergleich. Wir wollen also wissen, wie hoch der Umsatz im gleichen Zeitraum im Vorjahr war. Normalerweise gibt es dafür Funktionen, wie *SAMEPERIOD-LASTYEAR()*, die wir in diesem Fall aber nicht nutzen können. Also lese ich häufig:

```
1 Umsatz =  
2 VAR t = Today()  
3 VAR lastYear = YEAR(t) - 1  
4 VAR endDateLastYear = DATE(lastYear, MONTH(t), DAY(t))  
5 RETURN  
6     CALCULATE(  
7         SUM(FaktSalesLine[LineTotal]),  
8         DimCalender[Date] < endDateLastYear  
9         && DimCalendar[Year] = lastYear  
10    )
```

Ich persönlich bevorzuge eine berechnete Spalte auf der Datumstabelle (wobei ich *TODAY()* nicht nutze - dazu aber gleich):

```
1 isYtD = DimCalender[Date] < TODAY() && DimCalender[Year] = YEAR(TODAY())
```

Dann kann ich in der Formel schreiben:

```
1 CALCULATE(  
2     SUM(FaktSalesLine[LineTotal]),  
3     DimCalender[isYtD] = TRUE  
4 )
```

## “Heute” ändern

Während des Projekts wird dann immer klarer, das “Heute” vielleicht doch nicht “Heute” ist. Sondern eigentlich “gestern”, weil nur die Daten von gestern aktuell sind. Oder aber der aktuelle Monat ist noch nicht relevant und “Heute” ist der letzte Tag des letzten Monats. Womit die Year-to-Date Definition eben auch nicht mehr bis heute, sondern bis zum letzten vollständigen Monat geht.

Habe ich jetzt die Funktion *TODAY()* in vielen Measures genutzt, habe ich reichlich Aufwand, dass wieder zu korrigieren.

## Best Practise: Mein eigenes TODAY()

Die Definition von *HEUTE* frage ich in Projekten meistens früh ab. Die meisten meiner Kunden sind zu diesem Zeitpunkt mit einer Antwort aber überfordert - weil die Auswirkungen noch völlig unklar sind. Und das ist verständlich. In jedem Datenmodell findet sich bei mir eine Measure mit Namen *MYTODAY*. Überall, wo ich in den Formeln oben auf *TODAY()* zurückgegriffen hätte, nutze ich konsequent *MYTODAY*. Damit habe ich eine einzige Stelle, an der ich die Definition ändern kann. Typische Definitionen sind:

```
1 // ich nehme das letzte Aktualisierungsdatum als "Heute"
2 MyToday = MAX(LastUpdateInfo[Date])
3
4 // Oder den letzten Tag des letzten Monats - dabei ist zu beachten,
5 // dass "-1" einfach einen Tag abzieht.
6 MyToday = DATE(YEAR(TODAY()), MONTH(TODAY()), 1) - 1
7
8 // Oder das größte Rechnungsdatum - ich mache "Heute" also an den
9 // verfügbaren Faktendaten fest. Wenn es mehrere Faktentabellen gibt,
10 // muss man natürlich gut überlegen, welche die bessere ist
11 MyToday = CALCULATE(MAX(FaktSales[InvoiceDate]), REMOVEFILTERS(FaktSales))
12
13 // Oder ich nehme tatsächlich das Systemdatum
14 MyToday = TODAY()
```

## Vorteile

Der klare Vorteil ist die Wartbarkeit. Wenn ich das YtD-Datum in der Kalendertabelle hinterlege, dann basiert die Formel auf *MyToday*. Wenn ich eine Measure schreibe mit aktuellem Zeitbezug, dann nutze ich ebenfalls *MyToday*. Ändere ich die Definition von *Heute*, dann kann ich das zentral in der Measure *MyToday* machen und alles ändert sich automatisch mit.

Für Testzwecke ist das Vorgehen ebenfalls relevant. Habe ich Kundendaten, die schon älter sind, dann kann ich *MyToday* zu Testzwecken auch auf ein fixes Datum setzen:

```
1 // Heute fix auf den 23. April 2024 setzen
2 MyToday = DATE(2024,23,4)
```

Oder ich so auch eine *Zeitreise* anbieten: Ich könnte eine völlig unabhängige Kalendertabelle *DimCalendarTimeMachine* einfügen, die ich dem Leser als Slicer anbiete. Und *MyToday* definiere ich als

```
1 MyToday = MIN(DimCalendarTimeMachine[Date])
```

Dann kann der Benutzer selbst wählen, wann *heute* zukünftig gewesen sein worden ist.

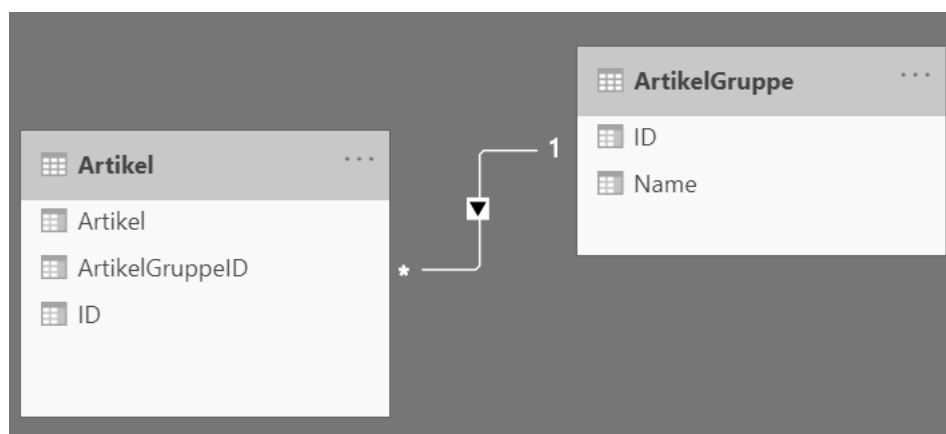
## Keine bidirektionalen Kreuzfilter!

Der folgende Artikel ist nicht leicht zu verstehen - aber ich glaube: er lohnt sich! Es lohnt sich zu verstehen, warum das mit der bidirektionalen Kreuzfilter-Richtung einfach keine so gute Idee ist. Du hast andere Gründe, warum du diese Funktionalität trotzdem benötigst? Ich freuen uns sehr über Kommentare!

### Aufgabe

#### Kreuzfilter

Wenn man in Power BI ein Datenmodell aufbaut, sollte man verstanden haben, wie eine Tabelle auf der 1-Seite eine Tabelle auf der \*-Seite filtert. In den verschiedenen Versionen von Excel Power Pivot und später Power BI wurde die Darstellung einer Beziehung immer ein wenig angepasst. In Power BI wird jetzt neben der Kardinalität auch die Richtung dargestellt, wie Filter propagiert werden:



**Abbildung 18:** Beziehung mit Kreuzfilterrichtung in Power BI

Diese Richtung kann man in Power BI (anders als in Excel Power Pivot) in beide Richtungen, also bidirektional konfigurieren. Dazu öffnet man via Doppelklick (oder Kontextmenü) den Eigenschaftsdialog der Beziehung und passt unten rechts die Kreuzfilter-Richtung auf *Beide*.

×

## Beziehung bearbeiten

Wählen Sie Tabellen und Spalten aus, die aufeinander bezogen sind.

Auftrag

ID	ArtikelID	Wert	StandortID	Jahr
1	1	10	1	2020
2	2	15	1	2020
3	1	20	2	2020

Artikel

ID	Artikel	ArtikelGruppelID
1	Artikel 1	1
2	Artikel 2	1
3	Artikel 3	2

Kardinalität

Viele-zu-Eins (\*:1)

☒ Diese Beziehung aktivieren
 ☐ Referenzielle Integrität voraussetzen

Kreuzfilterrichtung

Einfach

Einfach

Beide

OK

Abbrechen

Abbildung 19: Kreuzfilter-Richtung konfigurieren

Die Filter-Richtung auf der Beziehung wird jetzt in beide Richtungen dargestellt.

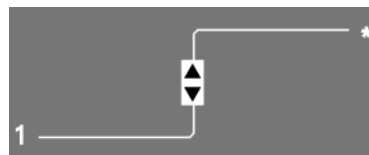


Abbildung 20: Bidirektionaler Kreuzfilter in Power BI

## Wozu ändert man überhaupt die Kreuzfilter-Richtung?

Dafür gibt es sicher viele Gründe - zwei sind mir am geläufigsten:

- Power BI hat die Beziehung automatisch angelegt und hat entschieden, die Kreuzfilter-Richtung bidirektional anzulegen (im Normalfall mit vielen anderen fehlerhaften Beziehungen, die dann nach und nach gelöscht werden müssen)
- Der Benutzer hat eine DrillThrough-Page für Faktendaten angelegt und möchte Dimensionsdaten als Zusatzinformation anzeigen - und das war eben der einfachste Weg
- Es gibt eine n:m Verbindung in den Daten: Ein Artikel kommt in vielen Positionen vor, die wiederum von vielen Kunde bestellt wurden. Möchte man jetzt wissen, von wie vielen verschiedenen Kunden ein Artikel bestellt wurde, kann man die Kreuzfilterrichtung zwischen Position und Kunde bidirektional einstellen. Für jeden Artikel kann man dann die Anzahl Zeilen in der Tabelle *Kunde* auswerten. Das geht aber auch alles anders - ohne bidirektionale Filter!

## Zum zweiten Grund: die *DrillThrough-Page*

Ich möchte hier nicht detailliert auf DrillThrough-Pages eingehen. Aber stellen wir uns vor, in einem Report sollen Auftrags-Detaildaten angezeigt werden. In der Detail-Seite sollen also Auftragsnummer, Umfang und beispielsweise ein Artikel angezeigt werden. Die Faktentabelle kennt aber nur die Artikel-ID, also den Schlüssel auf die entsprechende Dimensionstabelle. Natürlich möchte man zum Artikel mindestens die Bezeichnung anzeigen. Das klappt aber nicht, da eine Faktentabelle seine Dimensionstabellen nicht filtert. Oder konkret: die Tabelle *Auftrag* filtert die Tabelle *Artikel* oder *Standort* nicht, Nur mit Power BI Bordmitteln (also ohne DAX) erreicht man das nur durch *bidirektionale Kreuzfilterung*.

## Beispiel-Dashboard

Betrachten wir folgendes Beispiel-Dashboard:

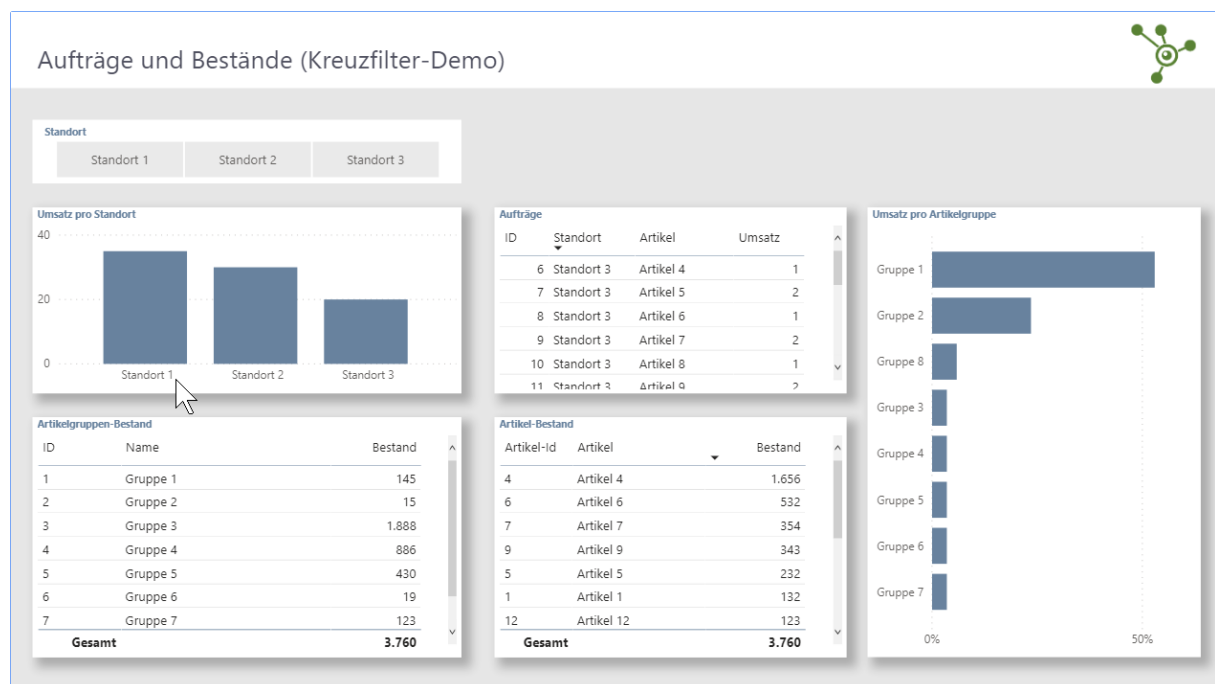


Abbildung 21: Power BI Beispiel Dashboard

Ein Unternehmen hat mehrere Standorte. Die Standorte verkaufen Artikel, die zu Artikelgruppen zusammengefasst wurden. Jeder Standort hat einen lokalen Lagerbestand. Das Dashboard zeigt den Umsatz pro Standort, die Aufträge, Bestände pro Artikelgruppe und pro Artikel sowie den Umsatz pro Artikelgruppe.

Bestände pro Artikelgruppe und Bestände pro Artikel darzustellen hat den Charme, dass via Cross-Filterung durch Auswahl einer Artikelgruppe in der linken Tabelle die zugehörigen Artikel und deren Bestände in der rechten Tabelle *Artikel-Bestand* dargestellt werden. Die Gesamtsummen beider Tabellen sollten übereinstimmen. Wird eine Artikelgruppe selektiert, sollte die Gesamtsumme von *Artikel-Bestand* mit dem Bestand der gewählten Artikelgruppe übereinstimmen.

## Der Fehler

Wählen wir die Artikelgruppe 1, erhalten wir dieses Ergebnis:

ID	Name	Bestand
1	Gruppe 1	145
2	Gruppe 2	15
3	Gruppe 3	1.888
4	Gruppe 4	886
5	Gruppe 5	430
6	Gruppe 6	19
7	Gruppe 7	123
Gesamt		3.760

Artikel-Id	Artikel	Bestand
1	Artikel 1	132
Gesamt		145

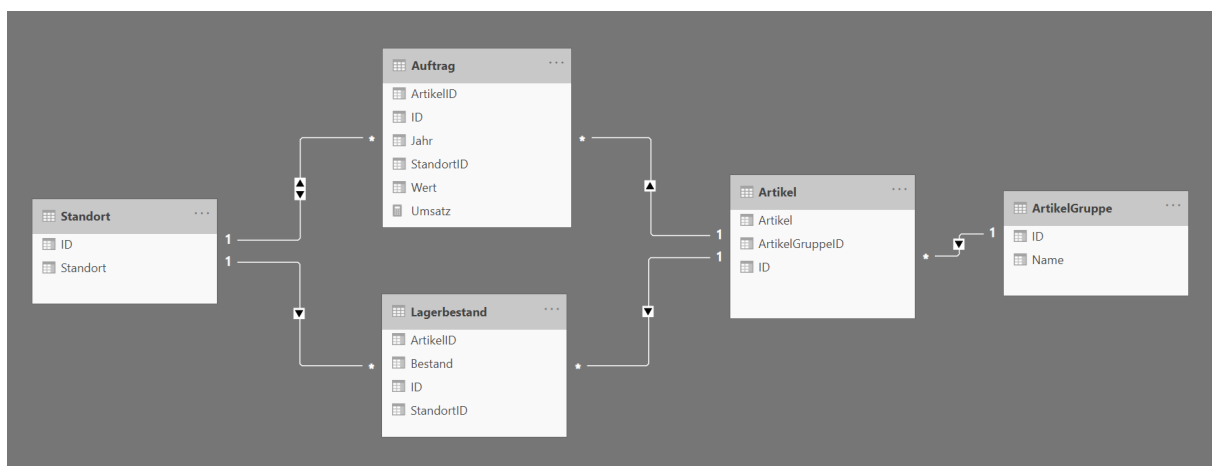
**Abbildung 22:** Beispiel einer Kreuzfilter-Anomalie

Der Bestand der Artikelgruppe beträgt 145, der Gesamtbestand der einzelnen Artikel ebenfalls. Nur ergibt die Zeilensumme eigentlich 132 und nicht 145!

## Willkommen in der Kreuzfilter-Hölle

Ich kann nicht genug betonen, dass man die Kreuzfilter-Richtung nur ändern sollte, wenn man genau weiß, was man tut und dies bei späteren Änderungen am Datenmodell auch im Blick behält. Meist ist es so, dass man wusste, was man tut - und dann ist das Datenmodell gewachsen. Daher: Am besten tut man es einfach nicht!

Das Datenmodell dieses Beispiels sieht folgendermaßen aus:



**Abbildung 23:** Power Bi Datenmodell mit bidirektionaler Kreuzfilter-Richtung

Zwischen *Auftrag* und *Standort* ist die Kreuzfilter-Richtung bidirektional. Vielleicht, weil man eben eine DrillThrough-Page für Aufträge hatte. Und dort wollte man den Standortnamen darstellen.

Und später hat man dann Lagerbestände eingeführt - und mit dem Standort verbunden. Ohne zu beachten, dass die Kreuzfilter-Richtung hier Einfluss auf die Ergebnisse hat.

## Wie entsteht dieser Fehler

Wenn die Artikelgruppe gefiltert wird in der Tabelle *Artikelgruppen-Bestand*, dann filtert Power BI alle Artikel, die zur Gruppe gehören. Dieser Filter wird in die Tabelle *Lagerbestand* propagiert. Der Filter wird aber auch in die Tabelle *Auftrag* propagiert. Dort werden alle Aufträge mit Artikeln dieser Gruppe gefiltert. Konkret sind das folgende Aufträge:

ID	ArtikelID	Wert	StandortID	Jahr
1	1	10	1	2020
2	2	15	1	2020
3	1	20	2	2020

**Abbildung 24:** Power Bi Auftraege in Demo Dashboard

Alle anderen Aufträge sind nur Verschleierung und haben mit unserem konkreten Problem nichts zu tun. Es werden also die Artikelnummern 1 und 2 gefiltert. Diese wurden von Standort 1 und 2 bestellt. Dieser Filter wird an die Tabelle *Standort* weitergereicht. Denn wir haben eine bidirektionale Filterung eingestellt.

Die Tabelle *Standort* filtert jetzt gleichzeitig mit der Tabelle *Artikelgruppe* und *Artikel* die Tabelle *Lagerbestand*. Zusätzlich werden also die Standorten 1 und 2 gefiltert. Obwohl das gar nicht gewollt war. Die Tabelle *Lagerbestand* hat folgende relevanten Einträge:

ID	StandortID	ArtikelID	Bestand
1	1	1	132
2	2	2	13
3	1	3	15

**Abbildung 25:** Power BI Lagerbestand im Datenmodell

Gefiltert sind die Artikel 1 und 2 sowie die Standorte 1 und 2. Damit erhalten wir den Bestand 145. Ein Fehler tritt hier also nicht auf.

In der Tabelle *Artikel-Bestand* werden die Bestände der Artikel der gewählten Gruppe einzeln dargestellt. Was passiert hier: Für jeden Artikel in der Tabelle *Artikel* die zur Gruppe 1 gehören wird eine Zeile erzeugt. Für jede Zeile wird jetzt ein *Artikel-Filter* propagiert. Artikel 1 wird korrekt dargestellt. Hier scheinen wir kein Problem zu haben. Es fehlt aber Artikel 2.



## Wo ist Artikel 2?

In der zweiten Zeile der Tabelle haben wir Artikel 2 “in der Hand”. Artikel 2 filtert die Tabelle *Lagerbestand*. Als Ergebnis erhalten wir eine Zeile mit folgenden Werten:

ID	StandortID	ArtikelID	Bestand
2	2	2	13

**Abbildung 26:** Tabelle Lagerbestand mit ArtikelID=2

**Gleichzeitig** filtert der Artikel (also Artikel 2) die Auftrags-tabelle. Dort erhalten wir mit dem Artikel 2 einen Auftrag:

ID	ArtikelID	Wert	StandortID	Jahr
2	2	15	1	2020

**Abbildung 27:** Tabelle Auftrags-tabelle mit ArtikelID=2

In der Auftrags-tabelle bleibt also genau ein Auftrag übrig. Und zwar mit **Standort=1**. Dieser Standort wird über die bidirektionale Kreuzfilter-Richtung an die *Standort*-Tabelle propagiert. **Und dadurch auch an die Tabelle Lagerbestand!**

Die Tabelle *Lagerbestand* ist aber bereits durch den Artikel gefiltert und es bleibt nur eine Zeile übrig. Mit anderem Standort, nämlich *Standort 2*. Jetzt kommt ein zusätzlicher Filter **Standort = 2** aus der Tabelle *Standort* hinzu. Das Ergebnis ist damit *leer*. Der Artikel wird in der Tabelle nicht mehr aufgeführt.

## Warum stimmt dann die Gesamtsumme?

Die Zeilen der Tabelle werden mit den Filtern der jeweiligen Zeile berechnet, also in unserem Fall die Artikelnummer. Die Gesamtsumme wird *ohne* diesen Filter gebildet. Da *ohne* den Artikel-Einzelfilter beide Standorte in der Auftrags-tabelle vorkommen, wird der Bestand korrekt angezeigt. Sonst wäre der Bestand in der linken Tabelle *Artikelgruppen-Bestand* auch schon fehlerhaft.



In diesem Fall sieht man die Inkonsistenz. Wäre aber der Bestand der Artikelgruppe schon fehlerhaft, weil beide Artikel von anderen Standorten verkauft wurden, als sie aktuell gelagert sind (und das ist ja fachlich absolut vorstellbar), dann sind die Zahlen im Dashboard konsistent. **Konsistent aber falsch!**

## Best Practise: Keine bidirektionalen Kreuzfilter

Bidirektionale Filter einfach zu verbieten kommt etwas zu einfach daher. Weil, wie löst man dann die Probleme, die man mit bidirektionalen Filtern gelöst hätte?

Wenn ich in einer DrillThrough-Page einen Auftrag gefiltert habe und beispielsweise dessen Artikelbezeichnung haben möchte. Die befindet sich aber in der Tabelle *DimArtikel*. In einem Tabellen-Visual muss ich gar nichts machen: Ich schreibe einfach die Bezeichnung aus *DimArtikel* in die Tabelle. Denn die Bezeichnung wirkt dann wie eine Dimensions-Information zu dem Auftrag - also zeige mir alle Artikelnamen und dann den zugehörigen Auftrag. Wobei der Auftrag schon eindeutig gefiltert ist in der DrillThrough-Page.

Oder man versteckt den bidirektionalen Filter in einem solchen Kontext in einer Measure:

```
1 ArtikelBezeichnung = CALCULATE(  
2     MIN(DimArtikel[Bezeichnung]),  
3     CROSSFILTER(DimArtikel[ArtikelNr], FaktSalesLine[ArtikelNr], BOTH)  
4 )
```

Mit *CROSSFILTER* und *BOTH* ändere ich die Kreuzfilterrichtung nur im Kontext dieser einen Measure. Was immer viel besser ist, als das im Datenmodell als Standard zu hinterlegen.

Zur Not geht auch ein *Lookup*:

```
1 ArtikelBezeichnung =  
2 VAR curArtikelNr = MAX(FaktSalesLine[ArtikelNr])  
3 RETURN  
4     LOOKUPVALUE(  
5         DimArtikel[Bezeichnung],  
6         DimArtikel[ArtikelNr],  
7         curArtikelNr  
8     )
```

Dadurch, dass die Auftragsposition eindeutig ist, darf ich mit *VALUES* auf die ArtikelNr des Auftrags zugreifen. Über ein *LOOKUPVALUE* hole ich mir dann den entsprechenden Wert aus *DimArtikel*

## Vorteil

Datenmodelle, die nur 1:n Verbindungen mit einfachen Kreuzfilterrichtungen sind robuster gegen Änderungen und besser zu verstehen. Soweit ich weiß, sind die auch performanter, da durch die doppelte Kreuzfilter-Richtung bei allen Measures DAX-Code "injiziert" wird.

Der Hauptvorteil liegt aber woanders: bei der gesparten Zeit, sehr unangenehme Fehler zu finden. Denn schon hier in meinen Beispieldaten ist der Umstand schwierig nachzuvollziehen. In einem echten Datenmodell kann diese Konstellation entstehen, indem Aufträge durch Filterung des Auftragsdatums eingeschränkt werden. Und durch diese Konstellation gibt es dann vielleicht keinen Auftrag

für einen Artikel mit einem bestimmten Standort - und nur in dieser Konstellation kommt es dann zu Fehlern.

Die Fehler sind insbesondere mit vielen Daten schwierig zu finden oder nachzuvollziehen, da alle Standorte meist kontinuierlich Waren verkaufen und die fehlerhaften Konstellationen nicht sofort sichtbar sind.

## Keine impliziten Measures

Implizite Measures gibt es sowohl in Excel Power Pivot als auch in Power BI. Darüber gibt es einen Artikel in meinem Blog - unter <https://www.durchblick-durch-daten.de/implizite-measures/>.

Aber er passt sehr gut in die Best Practises rein.

## Aufgabe

### Power Pivot in Excel

Sicherlich hast du in Excel schon *herkömmliche* Pivot-Tabellen angelegt. Man markiert einen Datenbereich und wählt dann *Pivot-Tabelle einfügen*. Dann erscheint ein Platzhalter, in dem die Pivot-Tabelle angezeigt werden soll. Außerdem erscheint eine Feldliste mit den Spalten aus dem gewählten Datenbereich.

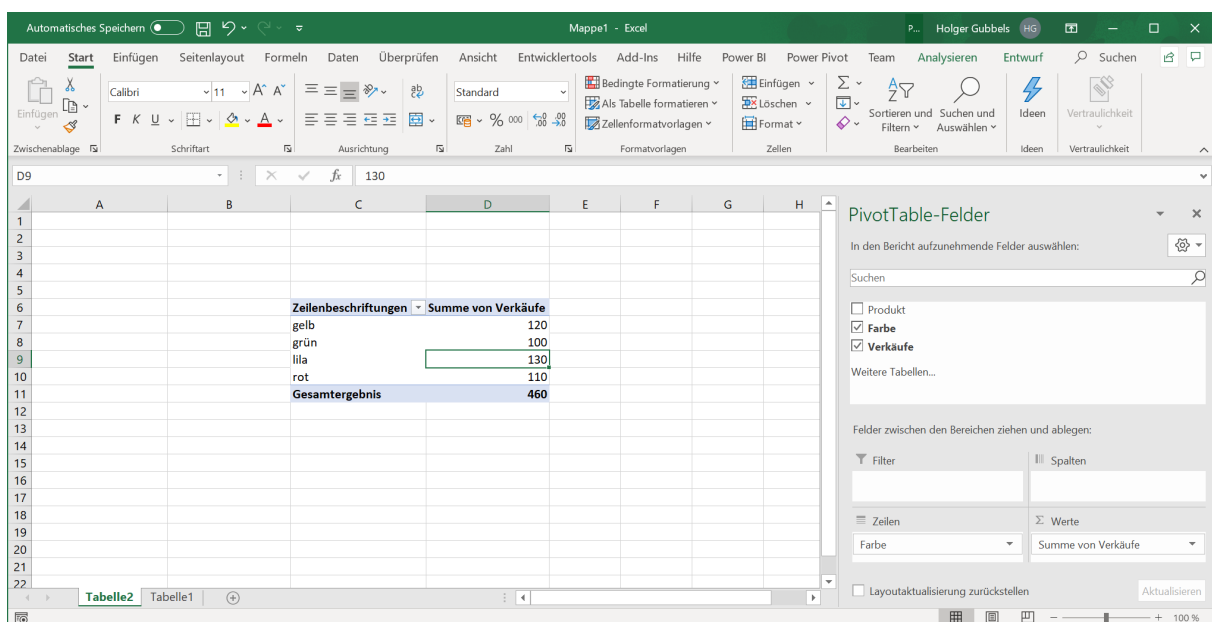


Abbildung 28: Herkömmliche Pivot Tabelle in Excel

Betrachtet man das Feld *Verkäufe* sieht man, dass Excel entschieden hat die Werte zu summieren. Das kann man leicht über die Wertfeldeinstellungen ändern.

Kommen die Daten nicht aus einem Datenbereich, sondern aus dem Excel-Datenmodell, sieht die Feldliste ein wenig anders aus. Die Feldliste besteht aus Tabellen, die man aufklappen kann, um an die

Felder zu gelangen. Man erhält aber weiterhin einen Platzhalter für die Pivot-Tabelle und kann auch hier Felder direkt in den Bereich *Werte* einfügen. Die Werte werden nach wie vor summiert, sofern es sich um summierbare Datentypen handelt.

## Power BI

In Power BI verhält sich das ähnlich. Die Oberfläche ist anders als in Excel Power Pivot. Auf der rechten Seite findet man eine Feldliste mit allen Feldern. In Power BI arbeitet man mit *Visuals* - um beim aktuellen Beispiel zu bleiben nehmen wir das *Matrix* Visual. Auch in Power BI kann ich ein summierbares Feld direkt in die *Werte* einfügen und Power BI summiert das Ergebnis.

## Wozu dann DAX?

In meinen Grundlagen-Trainings kommen wir irgendwann zu Einführung in DAX. Natürlich mit einfachen Funktionen. Beispielsweise:

```
1 Summe Verkäufe Total = SUM(Sales[Total])
```

Dieses neue Measure kann ich jetzt in einer Pivot-Tabelle verwenden. Als Ergebnis erhält man exakt das gleiche Ergebnis, wie wenn man das Feld *Sales[Total]* in der Pivot-Tabelle verwendet hätte. Wozu der umständliche Weg über *Measures*?

## Nähern wir uns dem Problem über das Format

Möchte man in Power BI das Measure *Summe Verkäufe Total* als Währung ohne Nachkommastellen formatieren, wählt man das Feld *Sales[Total]* und formatiert dieses. Irgendwie geht man bei Excel auch so vor. Man wählt die Wertfeldeinstellungen und kann dort das Format ändern.

Möchten wir aber nicht die Summe der Aufträge, sondern die Anzahl, nehmen wir das Feld *Sales[SalesOrderId]* und stellen die Aggregation auf *Anzahl*. Du kannst übrigens auch jedes andere Feld verwenden. Es ist völlig unerheblich, ob du die Auftragsnummer, das Datum oder auch hier *Sales[Total]* verwendest - solange du die Aggregation auf *Anzahl* stellst erhältst du immer das gleiche Ergebnis. In Power BI sieht das dann beispielsweise so aus:

Jahr	SalesOrderID
2011	1607
2012	3915
2013	14182
2014	11761
<b>Gesamt</b>	<b>31465</b>

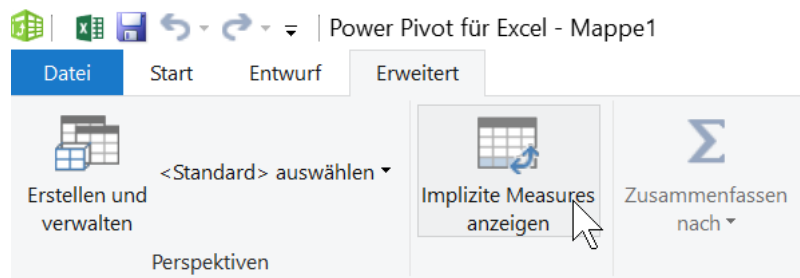
**Abbildung 29:** Implizites Measure *Anzahl Aufträge* in Power BI

Möchten wir jetzt Tausender-Trennzeichen zur einfacheren Lesbarkeit haben, haben wir keine Chance. Man kann zwar dem Feld *SalesOrderID* ein Tausender-Trennzeichen verpassen - es hat aber keine Auswirkung. Wenn die Auftragsnummer alphanumerisch wäre (also Text und Zahlen), dann geht das so ohnehin nicht.

## Implizite Measures

In beiden Fällen, also der Summe der Verkäufe als auch der Anzahl Aufträge handelt es sich um *Implizite Measures*. In Wirklichkeit wird nicht das Feld verwendet, sondern es wird intern ein Measure angelegt, das dem Benutzer verborgen bleibt. Das Format dieses Measures wird abgeleitet aus dem zugrundeliegenden Feld. Im Fall der *Summe der Verkäufe* passt das gut. Im Fall der *Anzahl Verkäufe* passt das nicht.

In Excel Power Pivot kann man implizite Measures tatsächlich sichtbar machen. Nachdem man ein Feld in einer Pivot-Tabelle aggregiert (oder auch in Pivot-Charts), wird im Datenmodell ein solches implizites Measure angelegt. Wechselt man ins Datenmodell kann man dort im Menü unter *Erweitert* die Option *Implizite Measures anzeigen* aktivieren:



**Abbildung 30:** Implizites Measure anzeigen in Excel Power Pivot

Im *Measures*-Bereich in Excel Power Pivot wird dadurch ein Measure sichtbar:

The screenshot shows the 'Power Pivot für Excel - Mappe1' ribbon with the 'Erweitert' tab selected. The 'Implizite Measures anzeigen' button is highlighted. Below the ribbon, a data table is visible with columns: SalesOrderID, OrderDate, Total, and Spalte hinzufügen. The 'Total' column contains values like 3.953,99 €. A tooltip is shown over the 'Summe von Total: 123.216.786,12 €' cell, indicating it is an implicit measure.

	SalesOrderID	OrderDate	Total	Spalte hinzufügen
1	43697	31.05.2011 ...	3.953,99 €	
2	43702	01.06.2011 ...	3.953,99 €	
3	43703	01.06.2011 ...	3.953,99 €	
4	43706	02.06.2011 ...	3.953,99 €	
5	43707	02.06.2011 ...	3.953,99 €	
Summe von Total:			123.216.786,12 €	

**Abbildung 31:** Implizites Measure in Excel Power Pivot

Mit einem kleinen Doppelpfeil rechts oben in der Zelle wird das Measure als *implizit* gekennzeichnet. Im Bearbeitungsbereich kann die Measure nicht geändert werden. Auch wenn implizite Measures nicht sichtbar sind, erscheinen diese immer in der Auswahlliste, wenn DAX-Expressions formuliert werden. Und natürlich: das irritiert, denn man findet diese Measures nicht, wenn man implizite Measures nicht anzeigt.

## Implizite Measures in Power BI...

...kann man anders als in Excel Power Pivot nicht sichtbar machen. Man muss einfach wissen, dass jedes Mal, wenn man ein Feld aus dem Datenmodell direkt in einen Bereich eines Visuals einfügt in dem aggregiert wird (Summe, Anzahl, Min, Max etc.), ein implizites Measure angelegt wird.

## Best Practise: Nur explizite Measures schreiben!

Selbst für die einfachsten Summen wird eine Measure geschrieben:

```
1 Umsatz = SUM(FaktSalesLine[LineTotal])
```

Das Feld *LineTotal* benötige ich dann nicht mehr. Und Felder lassen sich in Power BI aus der Feldliste ausblenden. Einfach via rechte Maustaste. Das macht man häufig, damit klar ist, dass diese Felder nicht verwendet werden sollen. Wird auf ausgeblendete Felder in einer Formel referenziert, bekommt man natürlich keinen Fehler.

Selbst Tabellen lassen sich ausblenden, so dass diese in der Feldliste nicht mehr auftauchen. Ebenfalls via rechte Maustaste.

Ausgeblendete Felder oder ganze Tabellen sind in einer Excel-Datei, von der aus man auf Power BI zugreift, auch nicht mehr sichtbar.

Wenn eine Faktentabelle nur aus Schlüsseln besteht und zu jedem Fakt (also Anzahl, Summe Umsatz etc.) eine Measure existiert. Dann würde das bedeuten, dass die Faktentabellen eigentlich gar nicht mehr sichtbar sein müsste.

**Korrekt!**

## Vorteile

### Übersichtlichkeit

In Excel Power Pivot ist eines der Vorteile, dass implizite Measures irritieren. Bei Erstellung von DAX-Formeln tauchen ständig Measures auf, auf die man sich beziehen kann, die man nicht angelegt hat. Anfänger verstehen nicht, wo diese Measures herkommen.

### Wartbarkeit

In Power BI kann man mit der Formatierung argumentieren - aber das ist natürlich plakativ. Ein wichtiger Grund ist die Wartbarkeit: In einem wartbaren Datenmodell gibt es Measures, die auf anderen Measures aufbauen. So definiert man beispielsweise ein Measure für die Gesamtsumme der Verkäufe:

```
1 Total Sales = SUM(FaktSalesOrder[Total])
```

Den Vorjahreswert definiert man dann über das bereits verfügbare Measure:



```

1 Total Sales PY = CALCULATE(
2     [Total Sales],
3     SAMEPERIODLASTYEAR(DimCalendar[Date]))
4 )

```

Viele weitere Measures sind jetzt denkbar, wie *Vorvorjahr*, *Differenzen absolut*, *Vormonat* - oder einfach eine Year-To-Date-Variante.

Habe ich *[Total Sales]* als implizites Measure verwendet und in 20 Visuals eingefügt und jetzt erfahre ich, dass ich noch den Rabatt abziehen muss. Dann muss ich alle Visuals finden, in denen dieses Feld vorkommt, und das entsprechend tauschen.

Nutze ich durchgehend die Measure aus dem Beispiel oben, ändere ich diese einfach ab:

```

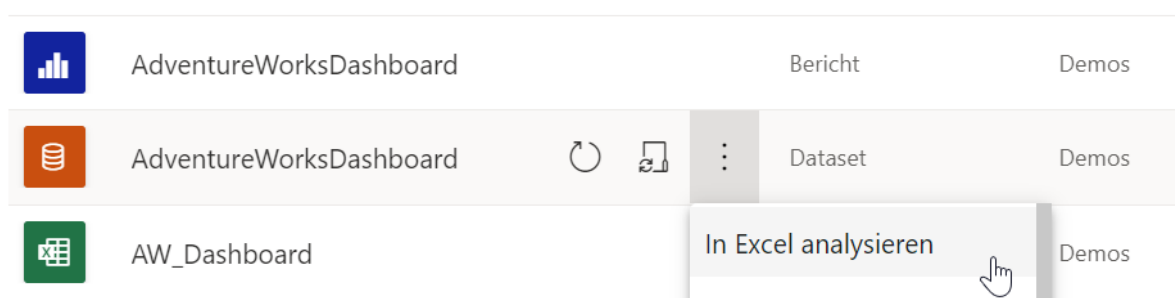
1 Total Sales = SUMX(
2     FaktSalesOrder,
3     FaktSalesOrder[Total Sales] -
4     FaktSalesOrder[Discount] * FaktSalesOrder[Total Sales]
5 )

```

Damit sind alle Visuals korrekt geändert, alle darauf basierenden Measure, wie die Vorjahresbetrachtung, Differenzen etc. ebenfalls.

## Power BI und Excel zusammen nutzen

Power BI Datenmodelle werden im *Power BI Service* online veröffentlicht und die Reporte können in der Organisation und auch außerhalb der Organisation freigegeben werden. Nur wenige wissen, dass via Excel auf dieses Online verfügbare Datenmodelle zugegriffen werden kann. Ausgehend vom Datenmodell im Power BI Service kann man eine Verbindung via Excel herstellen:



**Abbildung 32:** In Power BI veröffentlichte Datenmodelle in Excel analysieren

Über diese Funktion wird eine Excel-Datei im Browser heruntergeladen (oder eine ODT-Datei mit Verbindungsdaten). Die Excel-Datei ist mit dem Datenmodell verknüpft und es kann direkt mit Pivot-Tabellen oder -Charts auf die Daten im Datenmodell zugegriffen werden.

*Implizite Measures* werden hier **nicht** unterstützt. Das bedeutet, dass das Feld *FaktSales[Total Sales]* nicht direkt in die Werte der Pivot-Tabelle eingefügt werden kann. Es muss ein explizites Measure verfügbar sein. Neben allen Wartbarkeitsgründen ist dieser Grund sicherlich ein ausreichender. Um ein zukunftsfähiges Datenmodell zu erstellen muss man auf impliziten Measures verzichten.

## Measure Tabellen

### Aufgabe

Wenn man im Datenmodell eine Tabelle um eine Spalte ergänzen möchte, kann man das mit einer *Berechneten Spalte* lösen. Die *Berechnete Spalte* gehört natürlich zur Tabelle - zu was auch sonst. Eine Measure, also eine Berechnungsformel, gehört immer auch zu einer Tabelle. Betrachten wir folgendes Measure:

```
1 Net Sales = SUM(FaktSalesOrderLine[LineTotal])
```

Es gibt also eine Tabelle *FaktSalesOrderLine*. Ein Power BI Neuling, wenn er denn schon Measures anlegt, wird die Measure dieser Tabelle zuordnen.

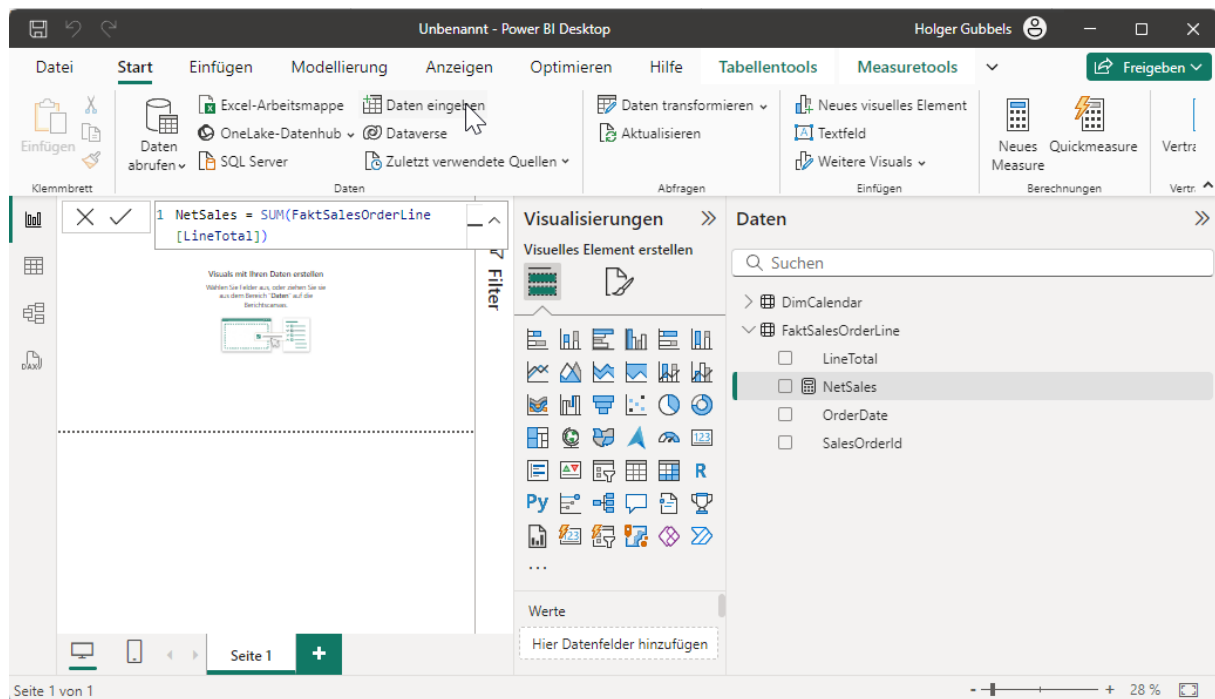
Jetzt habe ich bei einem Projekt Umsatz- und Budgetdaten, die ich auswerten soll. Dann könnte ich die Measure *NetSales* in die *FaktSalesOrderLine* und die Measure *Budget* in der Tabelle *FaktBudget* ablegen. Wo aber lege ich die Differenz und die prozentuale Abweichung, also die Measure *Budget vs. NetSales* und *Budget vs. NetSales%* ab? Und wo würde mein Kollege diese Measures suchen?



Ob sich eine Measure in Tabelle A oder B befindet hat auf die Funktion oder die Formel der Measure keinen Einfluss. So könnte ich einfach alle Measures in die Tabelle *DimCaster* umziehen. Es ändert sich am Berechnungsergebnis nichts.

### Best Practise: Anlage einer Measure-Tabelle

Die Lösung ist einfach: Ich lege eine explizite *Measure-Tabelle* an. Dazu nutze ich in Power BI die Möglichkeit lokale Daten anzulegen.



**Abbildung 33:** Lokale Daten in Power BI anlegen



Streng genommen ist *Daten eingeben* eine Funktion von Power Query. Die Funktion hier ist nur eine Abkürzung. Möchte man an den lokalen Daten etwas ändern, öffnet man einfach Power Query.

Es öffnet sich ein Dialog, in dem man in einer Matrix Spalten und Zeilen eingeben kann. Ich lasse aber alles so stehen - nur den Tabellennamen ändere ich auf *\_measures*.

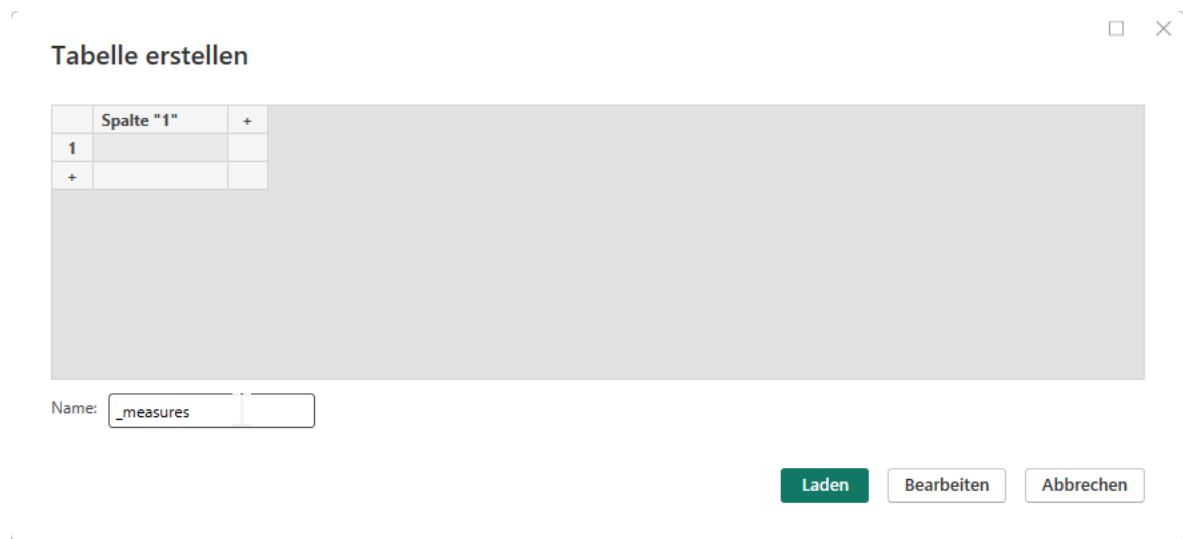


Abbildung 34: Dialog zum eingeben von Daten

Umziehen kann man eine Measure übrigens ganz schnell, in dem man die *Hometabelle* ändert.

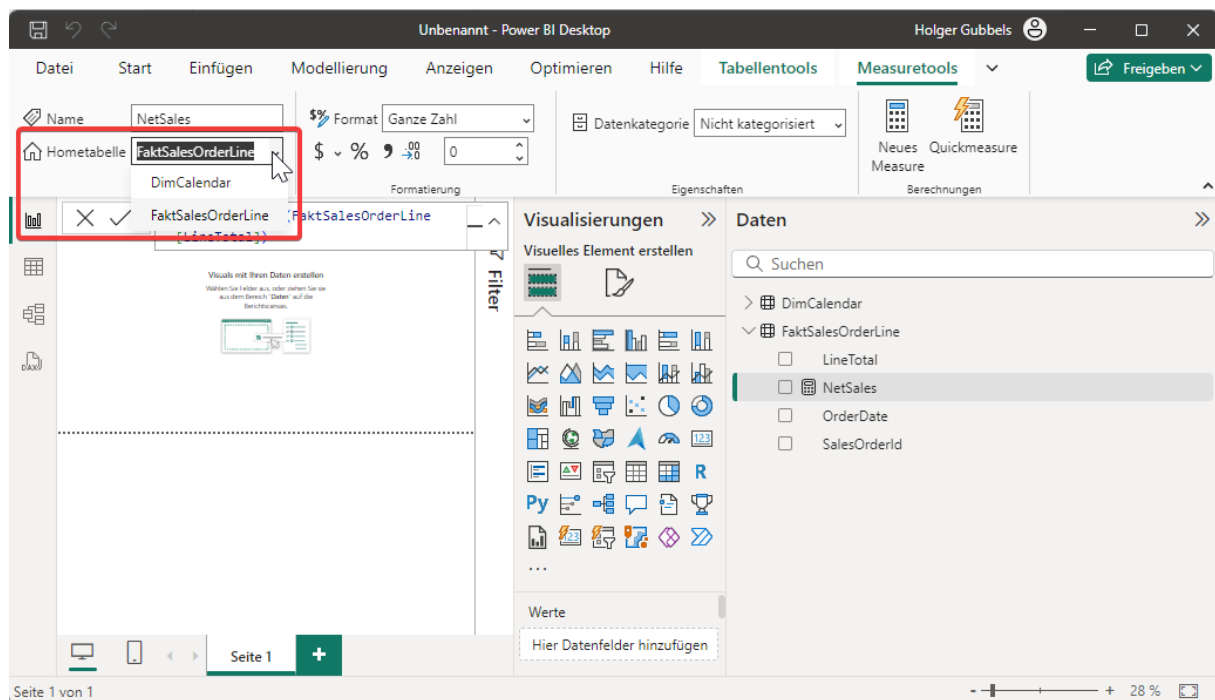
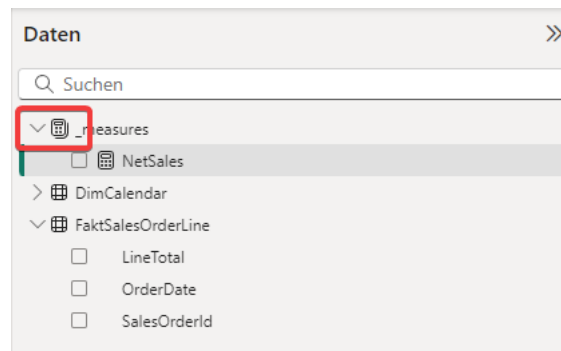


Abbildung 35: Hometabelle einer Measure in Power BI ändern

Natürlich kann man die Tabelle auch anders nennen. Allerdings darf man sie nicht *Measures* nennen, da das ein reserviertes Wort ist. Daher nenne ich meine Measure-Tabellen immer *\_measures*. Manch-

mal bietet es sich auch an, mehr als eine Measure-Tabelle anzulegen.

Jetzt kann ich meine *NetSales* Measure in diese neue Tabelle umziehen. Habe ich das getan, kann ich die *Dummy* Spalte aus der Measures-Tabelle löschen. Diese brauchte ich, um die Tabelle laden zu können. Aber ab dem Zeitpunkt, in dem ein Measure in der Tabelle vorkommt, kann ich die Spalte löschen. Dadurch ändert sich auch das Icon der Tabelle.



**Abbildung 36:** Icon der Measure-Tabelle in Power BI

## Vorteile

Die Vorteile liegen auf der Hand:

- Kollegen und ich selbst suche nur noch in der Measure-Tabelle nach bereits angelegten Formeln.
- Falls eine Tabelle, der ein Measure zugeordnet ist, gelöscht wird, werden auch alle Measures gelöscht. Bei einer expliziten Measure-Tabelle wird das nicht passieren, denn es gibt keinen Grund, diese Tabelle zu löschen.

# DAX Styleguide

## Aufgabe

Für Softwareentwickler sind Styleguides, Kommentare oder defensives Programmieren tägliches Brot. Man programmiert nicht nur, damit etwas funktioniert. Sondern man hat immer den Entwickler im Kopf, der verstehen muss, was man sich selbst gedacht hat.

Das gleiche sollte auch für DAX gelten - weil prinzipiell machen wir in dem Moment, in dem wir DAX schreiben, das gleiche wie ein Softwareentwickler.

## Best Practise: Einen Styleguide formulieren

Am besten man formuliert in einem Unternehmen einen Styleguide aus. Als Dokument oder als Folien. Ein solcher Stylguide lebt. Er wird also kontinuierlich überprüft und erweitert. Nachfolgend ein paar Gedanken für einen Styleguide:

## Namensgebung

Es gibt diesen guten Satz, den jeder Informatiker kennt:

There are two hard things in computer science: cache invalidation, naming things and off by one errors.

*Cache invalidation* lassen wir mal außen vor. *Off by one errors* sind Fehler bei Schleifen, die einmal zu oft oder zu wenig durchlaufen werden (das ist auch der Witz, warum statt zwei eben drei Probleme aufgezählt werden).

**Naming Things:** Das ist das eigentlich problematische. Variablen, Modulen, Funktionen, Prozeduren. Oder in Power Query bzw. DAX: Abfragen, Parameter, Funktionen, Measures oder berechneten Spalten. Wie vergibt man Namen, damit sie auch von Dritten verstanden wird.

Dabei gibt es absolute "Don't s":

- In Abfragen, die in das Datenmodell geladen werden, die Herkunft zu codieren: Bei manchen Kunden lese ich Tabellennamen, wie `SQLSERVERTABLE_dbo_salesordertable`. Die Herkunft der Daten spielt aus Sicht des Datenmodells keine Rolle. In Power Query kann die Quellabfrage, auf die später verwiesen wird, natürlich die Herkunft im Namen beinhalten. Im Datenmodell heißt diese Tabelle bei mir aber `FaktSalesOrder`. Denn belasse ich den Namen der Quelle in der Abfrage, darf ich die Quelle nicht mehr umziehen ohne alle Abfragen umbenennen zu müssen. Und

nebenbei: Wenn man in Power BI in der Modellansicht nur noch *SQLSERVERTABLE\_dbo\_s...* auf der Oberfläche sieht, macht man sich selbst das Leben nur unnötig schwer.

- Measures mit Jahreszahlen im Namen: Heißt die Measure *Umsatz2024* weiß ich, dass jemand das Prinzip nicht verstanden hat. Weil hier muss der Datenmodell-Ersteller jedes Jahr einen neuen Bericht machen und die Measure austauschen oder ändern und umbenennen. Es gibt nur eine Measure *Umsatz*, welches in den Kontext des Jahres 2024 eingesetzt wird.
- Measures verstreuen: Measures werden immer an eine Tabelle angegliedert. Man kann jetzt also in jede Tabelle Measures verstecken. Wenn jemand eine Measure sucht, muss er in jede Tabelle schauen. Besser: eine Measure-Tabelle. Dazu gibt es hier einen eigenen Abschnitt

Man sollte sich überlegen, vergebe ich deutsche oder englische Namen. Ich selbst bin leider nicht konsequent und ich arbeite mehr mit *denglischen* Begriffen. So was wie *Umsatz PY* kommt bei mir leider doch häufiger vor (PY für *Prio Year*). Aber ich bessere mich. Ich muss aber auch sagen, wenn die Tabelle *FaktOrders* statt *FactOrders* heißt, dann empfinde ich das überhaupt nicht schlimm. Ich bin ha oft schon froh, wenn sie nicht “\_inv323\_net\_export\_dgrv\_00900.csv” heißt...

## Dimensionen und Fakten

Das Datenmodell sollte so streng wie möglich einem Star- oder Snowflake-Schema entsprechen. In diesen Schemata sprechen wir von Dimensionen und Fakten. Das bedeutet, eine Tabelle ist entweder Dimensions- oder Faktentabelle. Es gibt im Netz einige Diskussionen, ob man die Tabellen in Power BI entsprechend benennen soll. Ich habe das für mich mit *Ja* beantwortet und gebe das auch so weiter:

- **DimCustomer**: Eine Dimensionstabelle mit Kundeninformationen
- **DimProdukt**: Eine Dimensionstabelle mit Artikelinformationen
- **FaktSalesOrder**: Eine Faktentabelle mit Kundenaufträgen

Dabei bediene ich mich noch weiterer Typen und verletze das Starschema manchmal:

- **AssocProductGroup\_Product**: Eine Assoziationstabelle, um eine n:m Verbindung aufzulösen
- **LookupCurrency**: Eine Tabelle, in der ich Währungen hinterlege und dort ähnlich einem *SVERWEIS()* in Excel an den entsprechenden Stellen ermittle.

Die Objekte werden normalerweise im Singular benannt. Also statt *FaktDimOrders* eben *FaktSalesOrder*. Das ist aber Geschmackssache.

Bei der Benennung nutze ich nur dann Unterstriche, wenn ich Wortteile eindeutig trennen muss. Ich persönlich würde nicht *Fakt\_SalesOrder* schreiben. Aber durchaus *AssocProductGroup\_Product* - weil bei einer Assoziationstabelle die Beziehung zwischen zwei Tabellen relevant sind (*ProductGroup* und *Product*). Auch das: Geschmackssache! Wichtig ist: Konsequenz.



## Referenzen auf Spalten und Measures

Bei Referenzen auf Spalten oder bei der Wiederverwendung von Measures achte ich darauf, dass bei berechneten Spalten immer der Tabellennamen vorangestellt ist, bei der Measures nie. Beispiel für eine berechnete Spalte

```
1 FaktSalesOrderLine[LineTotal] = FaktSalesOrderLine[Quantity] * FaktSalesOrderLine[Price]
```

Jetzt schreiben wir eine Measure, die den Umsatz summiert und eine weitere Measure, die den Umsatz des Vorjahres summiert:

```
1 NetSales = SUM(FaktSalesOrder[LineTotal])
2
3 NetSales PY = CALCULATE(
4     [NetSales],
5     SAMEPERIODLASTYEAR(DimCalendar[Date])
6 )
```

Speichern wir die Measure *NetSales* in der Tabelle *FaktSalesOrderLine* könnten wir auf die Angabe der Tabelle verzichten und könnten schreiben:

```
1 NetSalesShort = SUM([LineTotal])
```

Schaut man sich aber *NetSalesShort* an und *NetSales PY* sieht man, dass beide ein anderes Feld referenzieren, nämlich einmal *[NetSales]* und einmal *[LineTotal]*. Bei *[LineTotal]* handelt es sich aber um eine Spalte während es sich bei *[NetSales]* um eine andere Measure handelt.

Daher:

- Bei Referenzen auf Spalten wird immer der Tabellennamen mit angegeben.
- Bei Referenzen auf Measures wird der Tabellennamen nicht mitgegeben

## Einrückungen und Klammerungen

Mir liegt folgende Formel vor:

```
1 SelectedCurrency = IF(HASONEVALUE(DimCurrency[ID]),SWITCH(
2     SELECTEDVALUE(DimCurrency[ID]),1,"USD",2,"EUR",3,"CHF","NONE"))
```

Die Formel ist schlecht lesbar. Wenn ich mich beim Schreiben vertippt habe, dann suche ich auch lange nach einem Fehler. Und diese Formel ist nicht mal besonders lang... Ich schreibe die Formel anders:

```
1 SelectedCurrency = IF(
2     HASONEVALUE(DimCurrency[ID]),
3     SWITCH(
4         SELECTEDVALUE(DimCurrency[ID]),
5         1,"USD",
6         2,"EUR",
```

```
7         3, "CHF",  
8         "NONE"  
9     )  
10 )
```

Schon sieht man viel schneller, was dort definiert ist: Wenn der Wert in *DimCurrency[ID]* eindeutig ist, dann schaue, welcher es ist und gebe mir entsprechend das Währungskürzel zurück, oder "NONE".

Jede Formel in DAX ist eine Funktion, die Parameter übergeben bekommt. Das bedeutet, dass es immer einen Funktionsnamen gibt, eine öffnende und eine zugehörige schließende Klammer. Wenn man eine DAX-Formel schreibt kann man via *[ALT] + [ENTER]* eine neue Zeile beginnen (geht bei Excel übrigens auch). Via *[TAB]* kann man einrücken. Power BI hilft einem mit der *[TAB]* Taste, damit man nicht jedes mal Leerzeichen tippen muss. Der Funktionsname ist immer auf der selben Einrückungstiefe, wie die zugehörige schließende Klammer, die zur Funktion gehört. Am Schluss meiner Formel muss die letzte Klammer also wieder am Anfang der Zeile stehen. Abstrakt:

```
1 Measure 1 = Funktion1(  
2     Funktion2(  
3         Funktion3(  
4             Parameter1,  
5             Parameter2  
6         )  
7     ),  
8     Parameter2  
9 )
```

Meine Kunden verdrehen schon die Augen, wenn ich ihren Quellcode formatiere, bevor ich Fehler suche oder Erweiterungen vornehme. Es wirkt am Anfang pedantisch und man muss sich daran gewöhnen. Aber es spart sehr viel Zeit bei der Fehlersuche oder bei Anpassungen. Außerdem kann ich viel schneller erfassen, was der Autor bei einer Formeln gedacht hat. Schreibfehler (Klammern, Kommas) erkennt kann man sofort und kann sie schneller eliminieren.

## Variable

Seit ein paar Jahren erlaubt DAX die Nutzung von Variablen in einer Formel. Das vereinfacht nochmals die Lesbarkeit. Schauen wir uns den Umsatz des laufenden Jahres an. *YtD* steht für *Year to date* also der Summe des aktuellen Jahres bis heute. Schaut man sich das Measure monatlich an, handelt es sich einfach um den kumulierten Umsatz im aktuellen Jahr:

```
1 Umsatz YtD = CALCULATE(  
2     SUM(SalesOrderLine[LineTotal]),  
3     FILTER(  
4         ALL(DimCalendar),  
5         DimCalendar[Jahr] = YEAR(MAX(DimKalender[Date]))  
6         && DimKalender[Date] <= MAX(DimKalender[Date])  
7     )  
8 )
```

Hier muss man sich schon ganz gut mit Kontexten auskennen, um zu verstehen, warum man in der Filter-Funktion mit *MAX(DimKalender[Date])* arbeiten darf. Oder aber man formuliert die Formel um:

```
1 Umsatz YtD
2 VAR maxVisibleDate = MAX(DimCalender[Date])
3 VAR maxVisibleYear = YEAR(maxVisibleDate)
4 RETURN
5 CALCULATE(
6     SUM(SalesOrderLine[LineTotal]),
7     FILTER(
8         ALL(DimCalendar),
9         DimCalendar[Jahr] = maxVisibleYear
10         && DimCalender[Date] <= maxVisibleDate
11     )
12 )
```

Oder man formuliert es noch einfacher:

```
1 Umsatz YtD
2 VAR maxVisibleDate = MAX(DimCalender[Date])
3 VAR maxVisibleYear = YEAR(maxVisibleDate)
4 RETURN
5 CALCULATE(
6     SUM(SalesOrderLine[LineTotal]),
7     REMOVEFILTERS(DimCalender),
8     DimKalender[Jahr] = maxVisibleYear
9     && DimKalender[Date] <= maxVisibleDate
10 )
```

Variable haben darüber hinaus einen unschätzbaren Mehrwert: Wenn *Umsatz YtD* falsch ist und ich mir nicht sicher bin, ob der Wert in *maxVisibleDate* richtig berechnet wird, dann kann ich die Formel kurzerhand umschreiben:

```
1 Umsatz YtD =
2 VAR maxVisibleDate = MAX(DimCalender[Date])
3 VAR maxVisibleYear = YEAR(maxVisibleDate)
4 RETURN
5 maxVisibleDate
6 /*
7 CALCULATE(
8     SUM(SalesOrderLine[LineTotal]),
9     REMOVEFILTERS(DimCalender),
10     DimKalender[Jahr] = maxVisibleYear
11     && DimKalender[Date] <= maxVisibleDate
12 )
13 */
```

Alles zwischen */\** und *\*/* gilt als Kommentar - ist also nicht mehr Teil der Formel.

Bei mehrzeiligen Bedingungen, wie *DimCalender[Jahr]=XXX && DimCalendar[Monat]=XY*, wird üblicherweise mit einer neuen Zeile begonnen, wobei der verbindende Operator (*&&*) immer an den Anfang der nächsten Zeile gestellt wird und die Zeile nochmals eingerückt wird. Die nächste Zeile mit einer weiteren Bedingung wird dann aber nicht weiter eingerückt. Beispiel:

```
1 FILTER(
2     DimCalender,
3     DimCalender[Jahr] = maxVisibleJahr
```

```
4      && DimCalender[Date] < maxVisibleDate
5      && DimCalender[Weekday] <> 6
6      && DimCalender[Weekday] <> 7
7  )
```



Manche Styleguides sehen vor, dass Variable in einer Formel mit einem Unterstrich beginnen. Ich mache das nicht, da mir der Mehrwert nicht klar ist

## Spaltennamen in temporären Tabellen

Auch wenn es sich hier um fortgeschrittene Formeln handelt, sei dennoch erwähnt:

In Formeln ist es manchmal notwendig, temporäre Tabellen aufzubauen, die man in der nachfolgenden Berechnung benötigt. Beispielsweise möchte ich die Anzahl Kunden wissen, die mindestens einen grünen und einen roten Artikel gekauft haben. Dazu erzeuge ich eine temporäre Tabelle, in der ich für jeden Kunden berechne, wie viele rote und wie viele grüne Artikel er gekauft hat:

```
1  Anzahl Kunden mit roten und grünen Artikeln =
2  var tempTable = ADDCOLUMNS(
3      DISTINCT(DimCustomer[ID]),
4      "@Anzahl rote Artikel", CALCULATE(
5          COUNTROWS(FaktSalesOrderLine),
6          DimProduct[Color] = "red"
7      ),
8      "@Anzahl grüne Artikel", CALCULATE(
9          COUNTROWS(FaktSalesOrderLine),
10         DimProduct[Color] = "grün"
11     ),
12 )
13 RETURN
14 COUNTROWS(
15     FILTER(
16         tempTable,
17         [@Anzahl rote Artikel] > 0
18         && [@Anzahl rote Artikel] > 0
19     )
20 )
```

In der temporären Tabelle *tempTable* lege ich zwei berechnete Spalten an: die Anzahl Verkaufspositionen mit roten und mit grünen Artikeln. Weiter unten filtere ich diese Tabelle. Da es sich bei den beiden temporären Spalten um berechnete Spalten handelt, müsste ich eigentlich den Tabellennamen davorstellen. Es gibt aber keinen Tabellennamen, da es sich um eine temporäre Tabelle handelt und damit nur eine Variable *tempTable* in der die Tabelle gespeichert wird. Also muss ich im Filter auf den Namen ohne Tabelle zugreifen: *[@Anzahl rote Artikel]*. Damit könnte es sich aber auch ein Measure handeln. Damit der Leser das sofort unterscheiden kann, stelle ich bei solchen temporären Spalten einfach ein *\*@\** im Namen voran.



In der obigen Formeln sieht man eine weitere Einrückungsregel: Nach dem *RETURN* rücke ich üblicherweise eine Stufe ein. Damit grenze ich optisch den Teil vor dem *RETURN* zu dem Teil nach dem *RETURN* ab. Das heißt die *Basislinie* (wenn man so will), also die Linie, bei der Formel aufhört und am Schluss die letzte Klammer steht, ist eine Stufe eingerückt.

Obwohl ich bei Parametern oft eine neue Zeile beginne, wie beispielsweise bei einem Filter:

```
1 Filter(
2     FaktOrderLine,
3     FaktOrderLine[Type] = "I"
4 )
```

mache ich das bei bestimmten Funktionen nicht. Dazu gehören beispielsweise *ADDCOLUMNS* oder auch *SWITCH*: Bei *SWITCH* hat man einen Parameter, den es zu überprüfen gilt. Anschließend folgen die Mapping-Regeln, wobei immer ein Paar (oder *Tupel*) zusammengehören. Außer der letzte Parameter: das ist der *default*-Fall, der wieder einzeln steht.

```
1 SWITCH(
2     SELECTEDVALUE(DimCurrency[LocalCurrencyId]),
3     1, "EUR",
4     2, "USD",
5     3, "CHF",
6     "unknown"
7 )
```

Bei *ADDCOLUMNS* ist es ähnlich. Erst wird die Basistabelle angegeben, an die neue Spalten angefügt werden sollen. Dann werden beliebig viele Tupel angegeben, bestehend aus *Name* und *Formel*:

```
1 ADDCOLUMNS(
2     DimCustomer,
3     "@Auftragspositionen", COUNTROWS(FaktSalesOrderLine),
4     "@Umsatz", CALCULATE(SUM(FaktSalesOrderLine[LineTotal])),
5     "@Umsatz > 50, grüne Produkte", CALCULATE(
6         SUM(FaktSalesOrderLine[LineTotal]),
7         FILTER(
8             FaktSalesOrderLine,
9             FaktSalesOrderLine[LineTotal] > 50
10        )
11     ),
12     DimProduct[Color] = "grün"
13 )
```

Durch das *Hintereinanderschreiben* der Paare, sieht man den Zusammenhang zwischen Namen und Formel besser. In meinem Beispiel sieht man aber auch, dass ich mir die Freiheit nehme, kurze Formeln nicht umzuberechnen (*@Umsatz*), längere aber schon.



Unter [www.daxformatter.com](http://www.daxformatter.com) findet man auch einen online DAX-Formatierer. Dort einfach die Formeln einfügen und formatieren lassen. Die dort hinterlegten Regeln sind nicht 100% identisch mit meinen - aber eine Automatismus kommt mit Freiheitsgraden auch nicht zurecht :-)