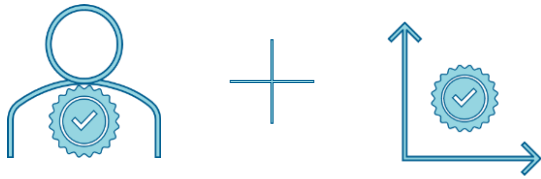


# Excel-Tools auf Power Query-Basis

## Checkliste



## #UserExperience

- User
- Nutzeroberfläche
- Gebrauchstauglichkeit
- Aktualisierungsfehler
- Abfrageperformance
  - Exkurse
- #AnalystExperience



## #DeveloperExperience

- Übersichtlicher Code
- Namenskonventionen - Vorschläge
- Stolperfallen
- Cheats & Hacks



## #SchutzSicherheit

- Schadensszenarien
- Datenschutz – Sicherheitsstufen
  - Exkurs
- Schutz von M-Code
- Power Query sieht alles\*
- Weitere Datenschutz-Überlegungen
- Checkliste



## #Exkurse

- ① Filtern anhand Liste
- ② Transponieren
- ③ Nachschlagen von Werten
- ④ Sicherheitsstufen
- ⑤ Verarbeitung strukturierter Werte
- ⑥ Gruppieren



## #Goldstückchen

- ① Spalten-umbenennung
- ② Spaltensortierung
- ③ Spaltentyp-Festlegung
- ④ Projektion / Spaltenauswahl
- ⑤ Mehrere Spalten auf einmal anlegen
- ⑥ Abfragen dynamisch auswählen
- ⑦ Zahlen/Zeichen extrahieren
- ⑧ Namensliste „putzen“
- ⑨ Erster Wert, der nicht null ist
- ⑩ Pivotieren mit Aggregation von Textwerten
- ⑪ Spalten zusammenführen, Header erhalten



- PQ als ETL-Werkzeug
- Definition ETL
  - „ETL ist ein umfassender Prozess, mit dem Unternehmen all ihre unterschiedlichen Daten – strukturiert oder unstrukturiert ... – in einen Zustand bringen, in dem sie für geschäftliche Zwecke nützlich sind.“  
„...ein bewährtes Verfahren, mit dem Daten aus mehreren Systemen in einer Datenbank, einem Datenspeicher, Data Warehouse oder Data Lake zusammengeführt werden. Mit ETL werden ... Daten aggregiert, um sie zu analysieren und fundiertere Geschäftsentscheidungen zu treffen.“  
<https://cloud.google.com/learn/what-is-etl?hl=de>
- Analyse! Kennzahlen! >> Datenbasierte Entscheidungen

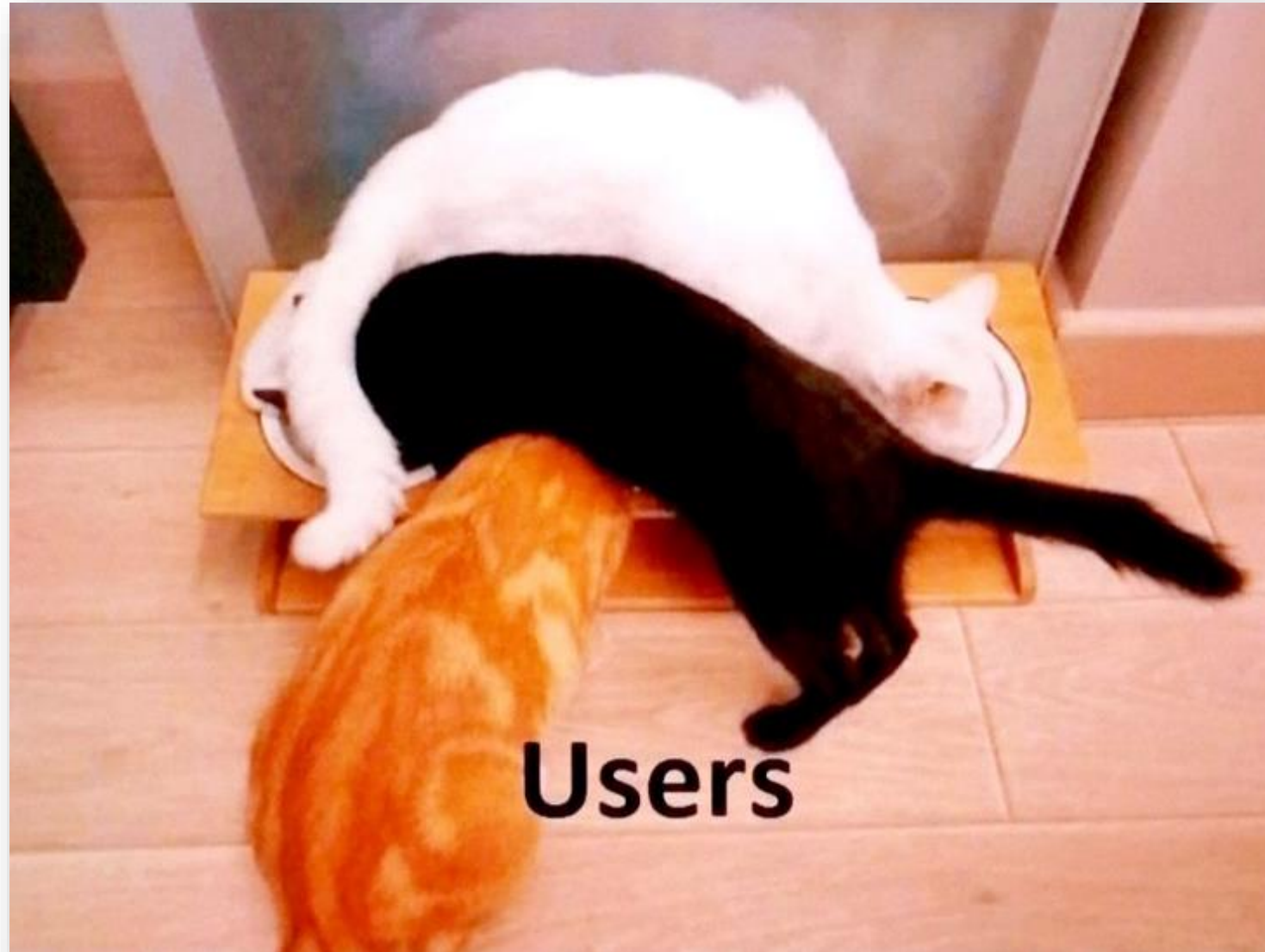


- Power Query
  - kann vielfältige, auch unterschiedliche Datenquellen „anzapfen“
  - kann große Mengen an Daten und Dateien aggregieren  
=> bequeme und leistungsstarke Zahlen- UND Text-Aggregation)
  - Datenausgabe kann zielgruppenorientiert praktisch beliebig angepasst werden
  - eignet sich besonders gut für sich wiederholende Aufgaben
- => Kandidat für Geschäftsprozessunterstützung und -optimierung



**Developer: Makes a simple, intuitive UI**









- „normaler“ Excel-Nutzer
  - keine bis wenig Kenntnisse von Power Query
  - beherrscht kein M
  - arbeitet nur in der Excel-Oberfläche
- „fortgeschrittener“ Excel-Nutzer
  - wäre (ggf. nach Einweisung) in der Lage, Parameter o. ä. in Power Query zu bearbeiten
  - traut sich ggf. das Lesen (und Verstehen) von M-Code zu (einfaches Debugging)
- Power Query-Nutzer
  - kann bzw. würde wenigstens über PQ-UI sinnvolle Abfragen erzeugen



- ✓ Das Tool ist für den Nutzer da, nicht der Nutzer für das Tool
- ✓ Auge mit Farbe und Linien leiten, auf Kontraste achten
  - ✓ Kontraste auf Barrierefreiheit prüfen, z. B. <https://contrastchecker.com/>
- ✓ „schlank“, ohne Schnickschnack
- ✓ Bei langen Tabellen Nutzerhilfen integrieren
  - ✓ Gruppierung
  - ✓ Filter
- ✓ Ladeverhalten PQ-Aktualisierung prüfen
  - ✓ Tabellentools: Entwurf > Externe Tabellendaten > Eigenschaften > Spaltenbreite anpassen deaktivieren



## Zu empfehlen

- Intelligente Tabellen/Listobjects (eine pro Tabellenblatt)
- Sprechende Namen
  - Tabellenblatt
  - Tabellen
  - Datei
- Anleitung und Hinweise vorne in komplexen Dokumenten
  - Verständliche Sprache (einfach, gegliedert/logisch, kurz, ansprechend, adressatenorientiert)
- Integrierte Formatvorlagen z. B. für Überschriften nutzen
  - erforderlichenfalls anpassen
  - Serifenlose Schriften
- Fokus auf Zelle A1 auf dem ersten Tabellenblatt
- Datenende markieren
- Barrierefreiheitsprüfung



## Besser meiden

- Komplexe Spaltenköpfe
- Verbundene und geteilte Zellen
- Text in Textboxen
- Übermäßiger Gebrauch von *Kursivschreibung* und Unterstreichungen
- Informationen, die ausschließlich über Farbe, Position o. ä. wahrnehmbar sind
- Relevante Informationen in Kopf- und Fußzeile



## Mögliche Fehlerursachen

- Abfragegestaltung deckt nicht alle Datenzustände ab
  - Index-Zugriff bei zu kleiner Enumeration
- Datenquelle wurde geändert
  - Speicherort
  - Spaltenanzahl, -reihenfolge, -bezeichnung
  - Datenformat wird geändert
- Power Query-Abfrage wurde geändert
  - Manipulation
  - Versehentliche Änderung
- Datenschutzeinstellungen
  - Einstellungen wurden verändert
  - aktualisierender Nutzer ändert sich (bei restriktiver Einstellung)
  - Formula.Firewall greift
- Arbeitsspeicher reicht nicht aus, um Abfrage zu verarbeiten



## Ⓟ kryptische Fehlermeldungen

### ✓ Fehler meiden durch Prävention / robuste Abfragen

#### ✓ Abfragen dynamisch gestalten

#### ✓ Hartcodiertes ersetzen

- Spaltentypumwandlung > List.Transform/Table.ColumnNames/Text.Contains... nutzen
- Spaltenauswahl > List.Select(..., (x)=>...)

#### ✓ ggf. Parameter-Steuerung für Nutzer einbauen

#### ✓ ggf. regionenspezifische Umwandlungen definieren

### ✓ Error-Handler einbauen

#### ✓ Abfrageabbruch (Standard) ggf. ersetzen durch besondere Rückgabewerte (oder Ignorieren)

- if then else
- try otherwise
- MissingField.Ignore (nur im Ausnahmefall!)



- ✓ Arbeitsschritte auf Erforderlichkeit prüfen
  - ✓ Abhängigkeiten auf das notwendige Maß reduzieren
  - ✓ Überflüssige Schritte entfernen
    - Lazy Evaluation übergeht ggf. nicht alle nicht programmierten Schritte!
- ✓ Menge der zu bewegenden Daten hinterfragen
  - ✓ Query Folding (SQL Server) nutzen
    - möglichst frühzeitig horizontal und vertikal filtern
    - Check: Befehl „Systemeigene Abfrage anzeigen“ ist verfügbar
    - Typische PQ-Transformationen erst mit gefilterten Daten
- ✓ Rechenintensive Operationen analysieren und ggf. ersetzen
  - Exkurs: Filtern anhand Liste ①
  - Exkurs: Transponieren ②
  - Exkurs: Nachschlagen von Werten ③
  - Exkurs: Verarbeitung strukturierter Werte ④



## ✓ weiter: Performance optimieren

### ✓ Anzahl Datei-Quellen reduzieren

- verschiedenartige Daten aus einer Datei kommen schneller als Daten aus zwei Dateien

### ✓ Quell-Typ ändern

- Konnektoren sind unterschiedlich performant
- CSV schneller als XLSX schneller als ACCDB/ACCDE

### ✓ Aktualisierungssteuerung überprüfen

- ggf. müssen nicht alle Abfragen bei „Alle Aktualisieren“ aktualisiert werden
- Verbindungseigenschaften > Aktualisierungssteuerung

### ✓ Excel-Daten von „außen“ auslesen statt öffnen

- für Power Query irrelevante Formatierungen, Formeln etc. können Excel-Datei verlangsamen
- Excel.Workbook statt Excel.CurrentWorkbook





- ✓ Nutzereingaben soweit wie möglich steuern (Vorgabewerte/Auswahl)
- ✓ Für Freitext „Putz-Algorithmus“ implementieren
  - Zeichenersetzungen
  - Splitten/Trimmen/Zusammensetzen (auch für Spaltennamen!!!!)
  - Personennamen
    - ggf. Tausch *Nachname, Vorname* implementieren (wenn Komma, dann..., sonst...)



- ✓ Sprechende Variablennamen verwenden
  - ✓ Leerzeichen, führende Zahlen und Bindestriche vermeiden (“#“...“”),
  - ✓ besser camel case, Unterstrich
  - ✓ notfalls kommentieren
- ✓ Logische Gruppen bilden
  - ✓ Logische Einrückungen (indent)
  - ✓ Nested let statements verwenden
    - Variablen, die nur für einen einzigen Schritt benötigt werden
    - Variablen, die in einem Schritt mehrfach benötigt werden
- ✓ UI-Code mit M-Code optimieren, z. B.
  - ✓ überflüssigen Code löschen (z. B. nicht benötigte Spaltenumbenennung beim Ausklappen von Tabellenspalten – bei nur wenigen Umbenennungen siehe ① )
  - ✓ Mehrere Spalten in einem Schritt erstellen, dynamisch expandieren ⑤
  - ✓ harte Codierungen durch weiche ersetzen
  - ✓ wiederholende komplexe Arbeitsschritte durch Funktionen ersetzen



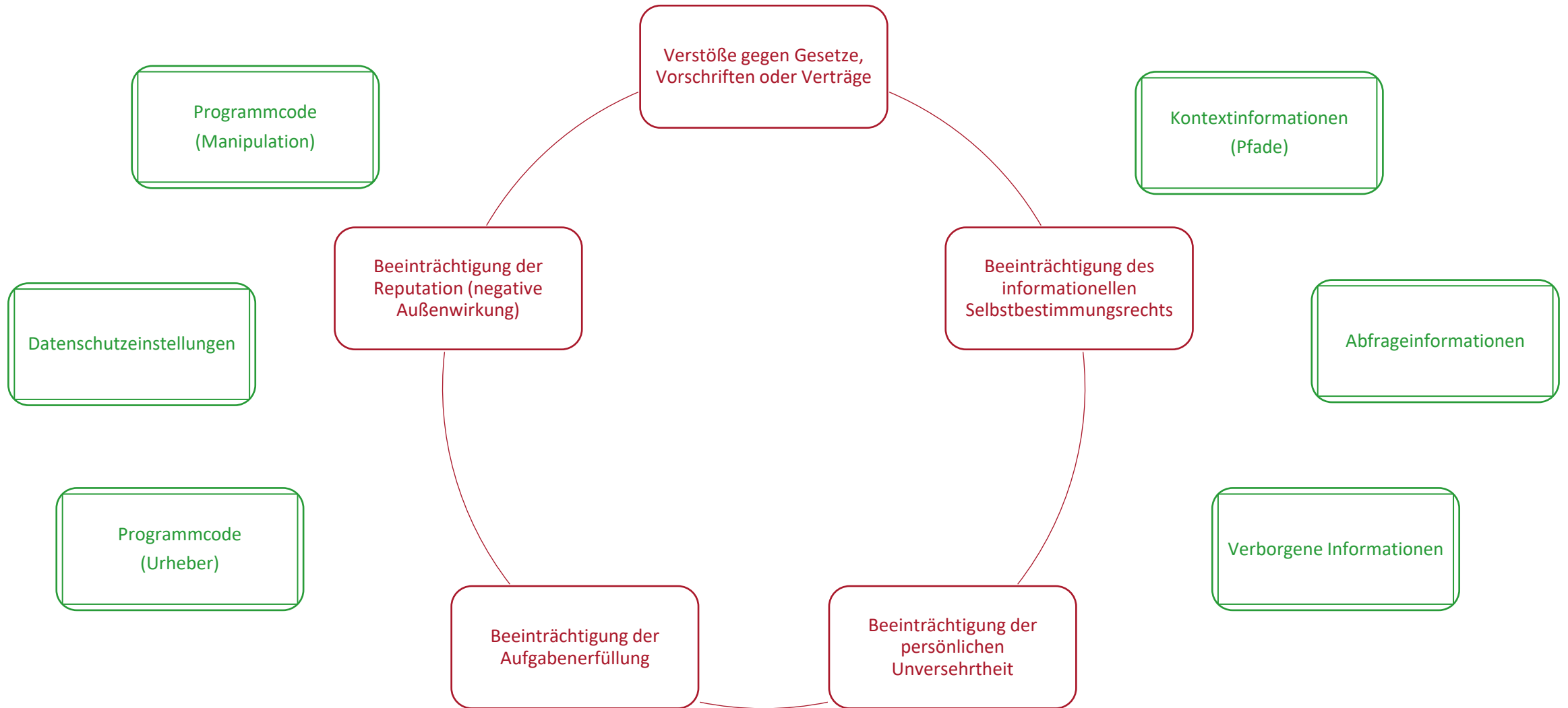
- ✓ Bereiche/Objekte/Werttypen nach Muster benennen (Namens-Manager Excel-UI, Power Query)
  - Einzelwert: „pv“ & \_
  - Strukturierter Wert (Liste, Tabelle): „sv“ & \_
  - Temporäre Spalte: „TEMP“
  - Intelligente Tabellen: „tbl“ & \_
  - Eigene PQ-Funktion: „fx“ & \_
- ✓ Gruppen von Abfragen mit gemeinsamem Präfix
  - erlaubt z. B. dynamisches Kombinieren mittels Funktion #sections ⑥
- ✓ Variablen für typische Schritte projektübergreifend gleich benennen
  - Spaltenauswahl
  - Spalte\_...
  - Expand\_Spaltenname
  - Join\_Tabellenname



- ✓ UI-generierter Code muss überprüft werden
  - ✓ Filterbedingungen
  - ✓ Verwendete Funktionen
  - ✓ Automatisch übergebene Argumente nachbessern
    - z. B. Excel.Workbook – Argument useHeaders setzen auf true, um Schritt #\"Höher gestufte Header\" zu sparen
- ✓ M-Code muss zur Office-Version passen
  - ✓ Geänderten Funktionsumfang beachten
    - Code: neue Funktionen und Operatoren in älteren Versionen ggf. nicht verfügbar
  - ✓ Syntaxunterschiede beachten
    - Funktionen: tw. Änderungen in den geforderten Argumenten
- ✓ Performance im Editor ungleich Performance in Excel
  - ✓ Bei Arbeit mit sehr großen Datenmengen „Entwicklermodus“ einbauen
    - (wahrscheinlich) wegen Vorschau-puffer für Schritte
    - Reduzierung Anzahl Datensätze über Parameter (Entwicklermodus: true/false) und if-Funktion
  - ✓ Abfrageperformance mit Datenmenge für Realszenario testen, ggf. optimieren
    - Zugriffsgeschwindigkeit auf Realdaten variiert ggf. in Abhängigkeit von Systemumgebung
    - Die Verarbeitungsgeschwindigkeit sinkt bei ungünstiger Abfragegestaltung bei Anstieg der Datenmenge stark ab



- ✓ für das Optimieren von Abfragen zunächst Kopie erstellen, in der Kopie arbeiten, bis zufriedenstellend
- ✓ Schreibhilfen M-Code / Cheaten mit UI
  - ✓ Spaltentypänderung
  - ✓ Spaltentransformation:  
Transformieren>Textspalte>Extrahieren (>Schritt umbenennen ;-))
  - ✓ Gruppierung: Spalte „Daten“>Alle Zeilen als Referenz erstellen  
(abschließend „kondensieren“ / auflösen)
  - ✓ Excel 2016: zum Schreiben und Prüfen von Code Notepad++ und ggf. selbst erstellte Sprache (UDL)
- ✓ Funktion #shared verwenden
  - ✓ vor Veröffentlichung löschen
  - **Record.ToTable(#shared)**



- 

Weiter



×

 Aktuelle Arbeitsmappe 



Abbrechen





- Sicherheitsfrage
  - taucht auf, sobald Nutzer nicht mit dem letzten Nutzer übereinstimmt, der Abfragen aktualisiert oder M-Code bearbeitet hat
  - bewirkt, dass Abfragen zunächst nicht ausgeführt werden
- Datenquelleneinstellungen
  - Sicherheitsstufen / Datenschutzebenen können jederzeit durch aktuellen Nutzer bearbeitet werden
    - PQE > Datei > Optionen und Einstellungen > Datenquelleneinstellungen
  - bieten eine schnelle Übersicht über die in den Abfragen verwendeten Datenquellen
    - lokale Dateien
    - Webseiten
    - SQL-Server
  - gezielter Austausch verwendeter Datenquellen über Dialog



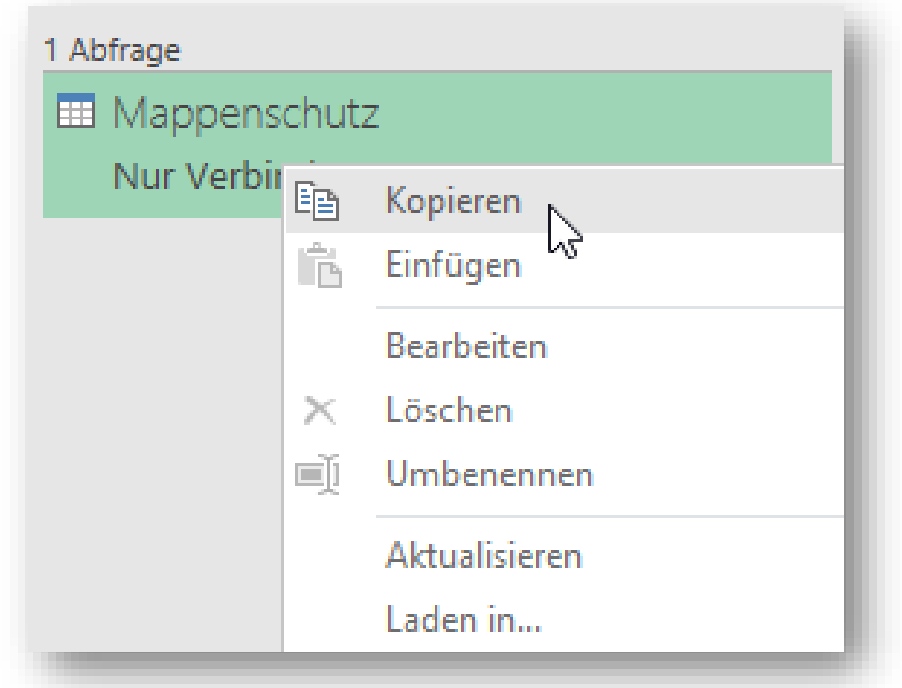
- Sicherheitsstufen / Datenschutzstufen
  - können für jede angegebene Datenquelle einzeln konfiguriert werden
    - Öffentlich
    - Organisation
    - Privat
  - werden durch die PQ-Engine ausgewertet
    - soll verhindern, dass PQ Daten an Quellen sendet, die einem höheren Schutzgrad unterliegen
    - Datenschutzabfrage kommt z. B. bei Join SQL-Server-Datenquelle mit Arbeitsmappendaten
    - lösen insbesondere bei Staged Queries Formula.Firewall aus (Beispiel: Pfad-Übergabe per Excel-UI)
  - Schutzmechanismus der Stufen unklar
    - Exkurs: Sicherheitsstufen ⑤
  - Mehrwert: erkennen, wenn andere Person Änderungen an der Abfrage oder einen Datenabruf vorgenommen hat



- ④ • Bewertung der Sicherheitsstufen setzt voraus:
  - Kenntnisse zur Bedienung des Power Query Editors
  - Kenntnisse zu Schutzbedarf der Daten
- Restriktive Einstellungen führen in der Regel zur Blockade von Abfragen (Formula.Firewall)
  - Können nicht immer durch einheitliche Klassifizierung aller Datenquellen aufgelöst werden
- Persönliche Best Practice (für meine aktuellen Einsatzgebiete): „Ignorieren“



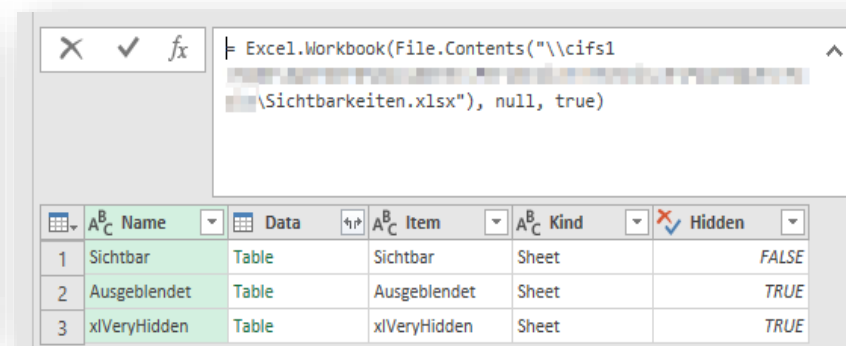
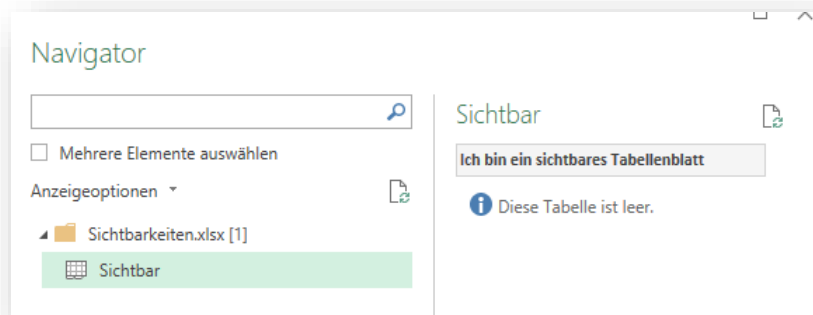
- Aktivierung
  - Reiter Überprüfen
    - > Gruppe Änderungen bzw. Gruppe Schützen
    - > Arbeitsmappe schützen
    - > In der Arbeitsmappe schützen: Struktur
- Auswirkung
  - Abfragen (Namen) sichtbar, M-Code nicht direkt einseh- oder bearbeitbar, aber: kopierbar!
  - Abfragen können nicht aktualisiert werden
  - Tabellenblätter können bearbeitet werden (wenn nicht zusätzlich Blattschutz aktiv)
  - => Schutz des M-Codes einer bestehenden Datei vor Manipulation





\*außer Formatierungen

- Bereiche, die mit Blattschutz gegen Kopieren gesichert wurden
- Versteckte Namen (Namens-Manager), die per VBA auf .Visible = False gesetzt wurden
  - Auch wenn Namen nicht im Namens-Manager erscheinen, können diese per Power Query ausgelesen werden!
- Ausgeblendete Tabellenblätter (auch xlSheetVeryHidden!)
  - Power Query Navigator zeigt zunächst nur sichtbare Daten an, liest aber alle aus!





Pfade können vertrauliche Informationen enthalten

- Excel-Funktion =ZELLE("dateiname")
  - hilft, relative Pfade in Power Query zu nutzen
  - aktualisiert sich erst beim Öffnen der Datei
  - Power Query - aber auch andere Tools - können geschlossene Dateien auslesen!
- Arbeitsmappenverbindungen
  - enthalten Pfadangaben
  - bleiben erhalten, auch wenn Arbeitsblatt in eine neue Datei kopiert wird



- ✓ prüfen, welche Daten durch wen erhoben werden müssen
- ✓ vorzugsweise Datenanlieferung von Rohdaten
  - ✓ möglichst auf PQ-Code bei „Anwendungsteilen“, die durch nicht näher definierte Personengruppe genutzt werden sollen, verzichten
  - ✓ hilfsweise Nutzer sensibilisieren wegen Datenschutzeinstellungen, Überprüfen Datenquellen
  - ✓ ggf. enthaltene PQ-Abfrage nicht zwingend einzusetzen
- ✓ nicht auf PQ-Berechnungen zugesandter Dateien vertrauen
  - Manipuliert? Nicht aktualisiert?
- ✓ zugesandte Dateien „von außen“ auslesen
  - Excel.Workbook statt Excel.CurrentWorkbook
- ✓ Arbeitsmappenverbindungen vor Versand an Externe entfernen
- ✓ auf Excel-Funktion ZELLE() verzichten
  - bei Dateien, die zwischen verschiedenen Bereichen/Parteien ausgetauscht werden





- PQ starkes Tool, geeignet für GPO
  - Analyse und Aufbereitung großer Datenmengen
  - #nomorecopyandpaste
- Einschränkungen in Prozessdesign berücksichtigen
  - vorzugsweise Datenanlieferung von Rohdaten
  - Vorsicht beim Abruf von Daten von externen Datenquellen (Parameter)
- Nutzer sensibilisieren (auch wenn nicht in PQE arbeitend)
  - Konfiguration der Datenschutzeinstellungen



## BI Gorilla

- <https://www.youtube.com/@BIGorilla>



## Chandeep Chhabra

- <https://www.youtube.com/@GoodlyChandeep>
- <https://goodly.co.in/blog/>
- Goodly Kurs:  
[Master Power Query {M}](#)  
#UnbezahlteWerbung



## Machen!

- Nichts schult so sehr wie echte Praxis-Herausforderungen!

# Danke!

---



## Danke für eure Aufmerksamkeit!





# Exkurse

Eintauchen in Spezialthemen

- Performance
- Sicherheit

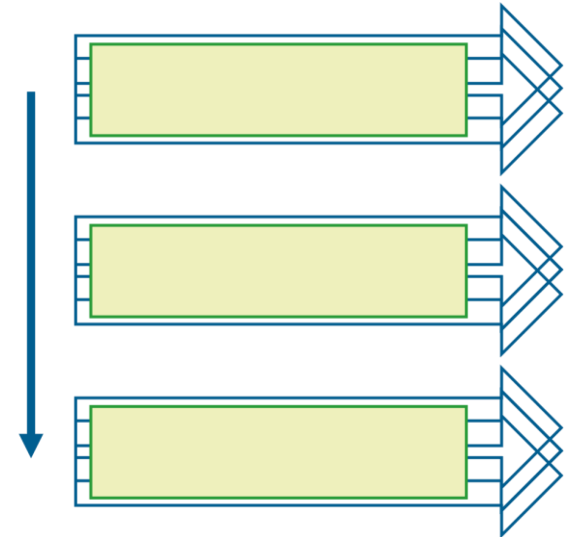




Aufgabe: Entferne aus einer Tabelle alle Zeilen, die in einer bestimmten Spalte einem Wert entsprechen, der in einer „Negativ-Liste“ enthalten ist

Lösungsansatz 1:

- **Table.SelectRows**(EineTabelle, **each not List.MatchesAny**(NegativListe, (x)=> x=[BestimmteSpalte])) )
- sehr langsam auch bei kurzer Negativliste





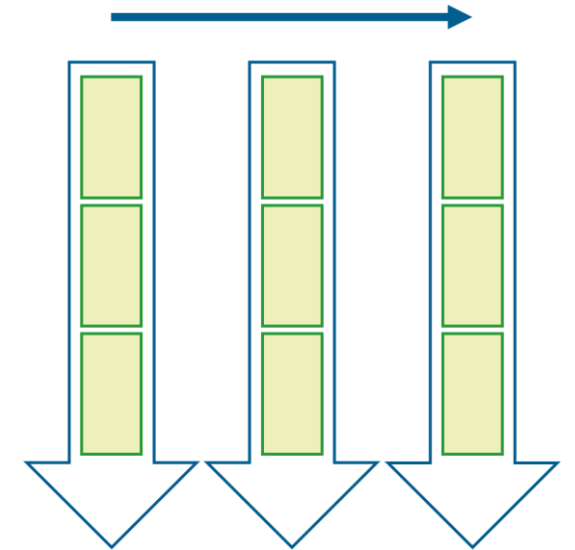
Aufgabe: Entferne aus einer Tabelle alle Zeilen, die in einer bestimmten Spalte einem Wert entsprechen, der in einer „Negativ-Liste“ enthalten ist

Lösungsansatz 2:

- **List.Accumulate**(NegativListe, EineTabelle, (s,c)=> **Table.SelectRows**(s, each [BestimmteSpalte] <> c))
- Extrem schnell!

Weitere Methode

- **Table.NestedJoin**(EineTabelle, {"BestimmteSpalte"}, NegativListeAlsTabelle, {"Ausfiltern"}, "TEMP", **JoinKind.LeftAnti**)
- Klassische Datenbank-Methode, „Negativ-Liste“ muss als Tabelle zugeführt werden



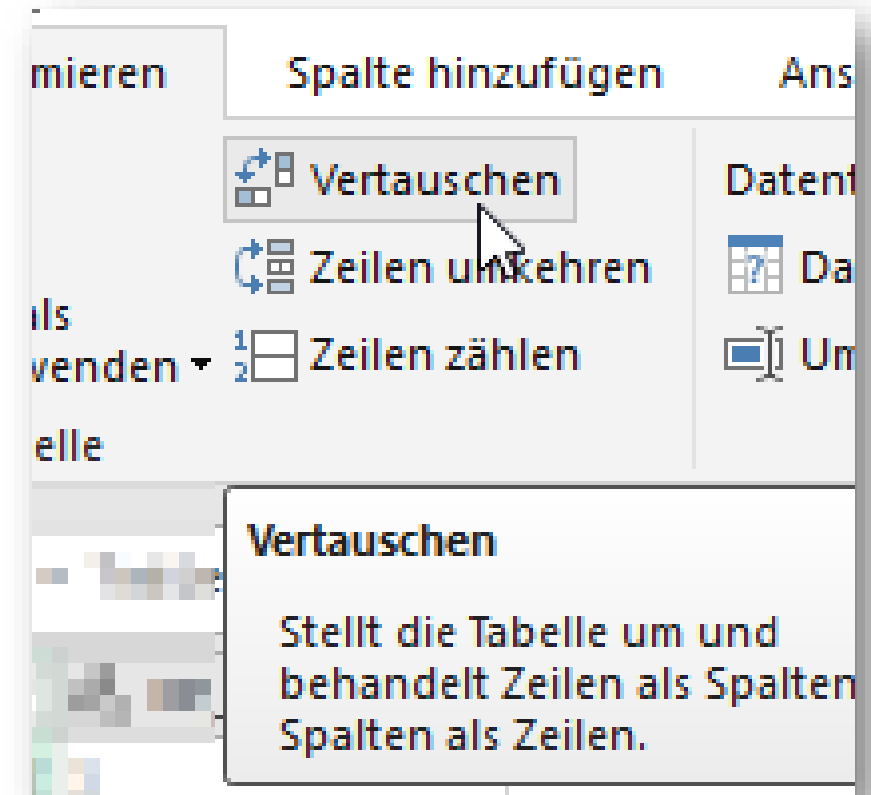




### Aufgabe: Spalten und Zeilen einer Tabelle vertauschen (transponieren)

#### Lösungsansatz 1:

- UI-Befehl Transformieren > Tabelle > Vertauschen
- verwendet die rechenintensive Funktion Table.Transpose > langsam bei größeren Datenmengen





Aufgabe: Spalten und Zeilen einer Tabelle vertauschen (transponieren)

Lösungsansatz 2:

**Table.FromColumns(Table.ToRows(Quelle))**

- Sehr schnell!
- Achtung! Nicht:
  - **Table.FromRows(**  
    **Table.ToColumns(Quelle))**
- Logisch? ;-) Ausprobieren!





Aufgabe: Daten ähnlich SVERWEIS nachschlagen

Daten aus der Tabelle 1 (tblDim) sollen als neue Spalte in die Tabelle 2 (tblFact) eingefügt werden.

tblDim		Tabelle tblFact	
TagKurz	TagLang	Tag	Wert
Mo	Montag	Mo	3
Di	Dienstag	Di	4
Mi	Mittwoch	WE	6
WE	Wochenende	Mi	8
		Di	2
		Di	3
		Mi	6

1

2





### Aufgabe: Daten ähnlich SVERWEIS nachschlagen

#### Lösungsansatz 1:

```
Join_tblDim =  
    Table.NestedJoin(Quelle, {"Tag"},  
        tblDim, {"TagKurz"}, "TEMP",  
        JoinKind.LeftOuter),  
Expand_tblDim =  
    Table.ExpandTableColumn(Join_tblDim,  
        "TEMP", {"TagLang"})
```

- Zugriff auf Nachschlagewert über Tabellenjoin
- 2 Schritte
- Reihenfolge der Datensätze kann sich ändern
- sehr schnell, über UI umsetzbar (Start > Abfragen zusammenführen)

**Tabelle tblFact**

Tag	Wert
Mo	3
Di	4
WE	6
Mi	8
Di	2
Di	3
Mi	6

2

**tblFactDim1**

Tag	Wert	TagLang
Mo	3	Montag
Di	4	Dienstag
Di	2	Dienstag
WE	6	Wochenende
Mi	8	Mittwoch
Di	3	Dienstag
Mi	6	Mittwoch

3





### Aufgabe: Daten ähnlich SVERWEIS nachschlagen

#### Lösungsansatz 2:

```
Spalte_TagLang =  
    Table.AddColumn(Quelle, "TagLang",  
        each tblDim[TagLang]  
        {List.PositionOf(tblDim[TagKurz],  
            [Tag])})
```

- Zugriff auf Nachschlagewert über Positionsindex, Ausgabe in neuer Spalte
- Nur ein Schritt
- Reihenfolge der Datensätze wird beibehalten
- langsamer als Join, nur über M umsetzbar
- Performance-Nachteil ähnlich dem „Selbstbau“ von Tabellen mit der Funktion #table

Tabelle tblFact

Tag	Wert
Mo	3
Di	4
WE	6
Mi	8
Di	2
Di	3
Mi	6

2

tblFactDim2

Tag	Wert	TagLang
Mo	3	Montag
Di	4	Dienstag
WE	6	Wochenende
Mi	8	Mittwoch
Di	2	Dienstag
Di	3	Dienstag
Mi	6	Mittwoch

3





### Abruf von Daten

Der Abruf von Daten ist kein reiner Lesevorgang! Um Daten abzurufen, müssen auch Daten gesendet werden. Insbesondere Parameter können sensible Daten enthalten.

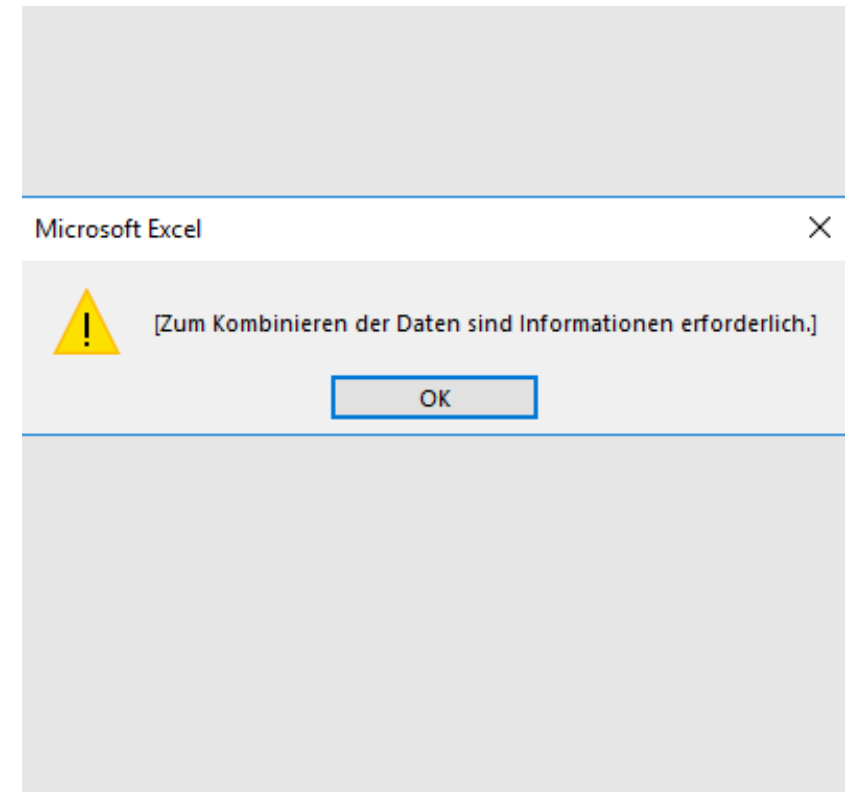




### Aufgabe: Verknüpfen von Arbeitsmappendaten mit einem SQL-Server (Table.NestedJoin)

#### Einstellung

- Server – Keine; Mappe – Privat  
> Sicherheitsabfrage
- Server – Privat; Mappe – Keine  
> Sicherheitsabfrage
- Server – Öffentlich; Mappe – Privat  
> Daten werden geladen
- Server – Privat; Mappe – Öffentlich  
> Daten werden geladen
- Server – Privat; Mappe – Organisation  
> Daten werden geladen
- Server – Organisation; Mappe – Privat  
> Daten werden geladen

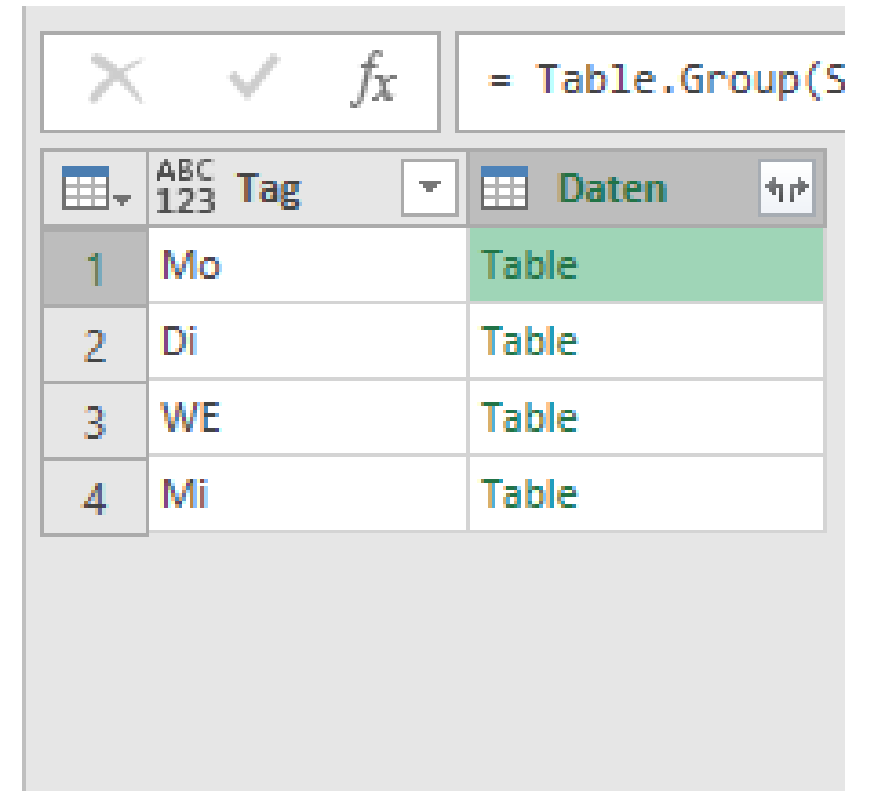




Aufgabe: Strukturierte Zellwerte sollen weiterverarbeitet werden

### Zugriff auf strukturierten Zellwert:

- Auslesen Wert für neue Spalte, Aggregation, Manipulation etc.
- Gruppierte Werte: sehr schnell
- Gejointe Werte: optisch kein Unterschied, aber sehr langsam
- ✓ Joins erweitern, mit erweiterten Daten weiterarbeiten



	ABC 123	Tag	Daten
1		Mo	Table
2		Di	Table
3		WE	Table
4		Mi	Table







### Aufgabe: Global gruppieren, Spalte „Daten“ als Referenz verwenden

Schaltfläche *Gruppieren nach*

> *Neuer Spaltenname*: Daten,

> *Vorgang*: Alle Zeilen

- nur Hilfsspalte – Datenvorschau hilft bei der Erstellung der eigentlich beabsichtigten Aggregationen
- Eintrag {„Daten, each \_“}, kann nach Erstellung der Gruppierungsspalten gelöscht werden

The screenshot shows the Power Query Editor interface. At the top, the formula bar contains the M code: `= Table.Group(Quelle, {"Monat"}, {{"Daten", each _}, {"Zahlenreihe", each Text.Combine(List.Transform(List.Sort(_[HO], Order.Ascending), Text.From), "-")})`. Below the formula bar, the 'Gruppieren nach' (Group By) dialog box is open, showing the 'Monat' column selected for grouping. The 'Zahlenreihe' column is selected for summarization. The 'Zahlenreihe' column is highlighted in green. Below the dialog box, the data preview is shown, displaying the grouped data with columns 'Monat', 'HO', 'BO', and 'NO'.

Monat	HO	BO	NO
Jan	61	42	72
Jan	60	59	66
Jan	54	22	63





### Aufgabe: Global gruppieren, Groß- und Kleinschreibung ignorieren

- Integrierte comparer function nutzen

### Lösungsansatz 1:

Gruppieren =  
**Table.Group**(MeineTabelle,  
{„Lied“},  
{{"Daten", each \_}},  
**GroupKind.Global**,  
**Comparer.OrdinalIgnoreCase**)

Schließen & laden > voranschaual aktualisieren > Verwalten > Kopieren > Verwalten > Sp



### Aufgabe: Global gruppieren, Groß- und Kleinschreibung ignorieren

- Integrierte comparer function nutzen
- [Custom Comparer Function for Table.Group in Power Query M \(youtube.com\)](https://www.youtube.com/watch?v=...)

### Lösungsansatz 2:

```
Gruppieren =  
    Table.Group(MeineTabelle,  
        {„Lied“},  
        {{„Daten“, each _}},  
        GroupKind.Global,  
        (c, n) =>  
            Value.Compare(Text.Lower(c[Lied]),  
                          Text.Lower(n[Lied]))  
    )
```

A	B	C
Lied		Lied
alle		alle
Meine		Meine
Entchen		Entchen
ALLe		
MEInE		
ENTchen		
allE		
mElne		
entCHen		

ABC	Lied	Daten
1	alle	Table
2	Meine	Table
3	Entchen	Table



### Aufgabe: Global nach Quartal gruppieren

- Integrierte comparer function nutzen
- [Custom Comparer Function for Table.Group in Power Query M \(youtube.com\)](https://www.youtube.com/watch?v=...)

```
Gruppieren =  
    Table.Group(MeineTabelle,  
        {„Lied“},  
        {{„Daten“, each _}},  
        GroupKind.Global,  
        (c, n) =>
```

```
    Value.Compare(Date.StartOfQuarter(c[Datum]),  
        Date.StartOfQuarter(n[Datum])  
    )  
)
```



## ⑥ Exkurs | Gruppieren (V)



Aufgabe: Zusammenhängende Datensätze anhand Merkmal für Gruppenstart gruppieren > GroupKind.Local mit comparer function

- Neue Gruppe beginnt mit nächstem Datensatz, wenn Rückgabewert der Funktion  $\neq 0$
- Number.From verwenden, um ggf. Wahrheitswert in Zahl (0 oder 1) umzuwandeln

```
Gruppieren =  
    Table.Group(Quelle,  
        {"MeineTiere"},  
        {"Daten", each _},  
        GroupKind.Local,  
        (c, n) =>  
            Number.From(  
                Text.StartsWith(n[MeineTiere],  
                    "Gruppe:")
```

	MeineTiere
Gruppe: Hühner	Helga
	Henna
	Claudia
Gruppe: Bienen	Willi
	Maja
Gruppe: Spinnen	Thekla
	Oma Langbein
	Vogel
	Quatsch erzählen

	MeineTiere
Gruppe: Hühner	3
Gruppe: Bienen	2
Gruppe: Spinnen	3



## Aufgabe: Lokal Gruppieren mit comparer function, Gruppenstart über zwei Zeilen

```
Gruppieren = Table.Group(Quelle,
    {"MeinZeitplan"}, {"Daten", each _},
    GroupKind.Local,
    (c, n) => Number.From(n[MeinZeitplan] is number)),
Alternierend =
    Table.FromColumns(
        {Table.AlternateRows(Gruppieren, 1, 1, 1)[MeinZeitplan]}
        &
        Table.ToColumns(
            Table.AlternateRows(Gruppieren, 0, 1, 1)))
SpaltenTransformieren = Table.TransformColumns(Alternierend,
    {
        {"Column1", each Text.From(Time.From(_))},
        {"Column2", each Text.From(Time.From(_))},
        {"Column3", each Text.Combine(
            Table.Skip(_, 1)[MeinZeitplan],
            Character.FromNumber(10))}
    })
```

MeinZeitplan	Column1	Column2	Column3
08:30	08:30	09:15	Vortrag BlaBlupp Thors Hammer Raum 1234
09:15	09:15	09:45	Workshop Basteln Gänseblümchenketten
Vortrag BlaBlupp Thors Hammer	10:00	10:15	Inspiration
Raum 1234	12:00	12:30	Power Nap mit Schokolade
09:15	13:00	14:00	Verabschiedung Onkel Theo sagt Tschüß
09:45			
Workshop Basteln Gänseblümchenketten			
10:00			
10:15			
Inspiration			
12:00			
12:30			
Power Nap mit Schokolade			
13:00			
14:00			
Verabschiedung Onkel Theo sagt Tschüß			



# Goldstückchen

Kleine Musterlösungen für häufige Probleme



## ① #Goldstück 1 | Spaltenumbenennung (I)



Spalten dynamisch umbenennen (hier: Präfix setzen)

PräfixSetzen =

```
let Spalten = Table.ColumnNames(Quelle)
in Table.RenameColumns(Quelle,
    List.Zip({Spalten,
        List.Transform(Spalten, (x)=> "MeinPräfix" & x)}))
```

```
PräfixSetzen = Table.RenameColumns(Quelle,
    List.Transform(Table.ColumnNames(Quelle),
        each {_, "MeinPräfix" & _}))
```





## ① #Goldstück 1 | Spaltenumbenennung (II)



Einzelne Spalten beim Ausklappen umbenennen (hier: Spaltennamen tauschen)

Table.ExpandTableColumn mit Umbenennung einzelner Spalten übersichtlicher und robust gestalten

Spalten =

```
{"Anton", "Berti", "Conni", "Det", "Edi", "Fritzchen"},
```

Renames =

```
{ {"Anton", "Gertrude"},  
  {"Conni", "Cornelius"},  
  {"Fritzchen", "Gunnar"} },
```

Umbenennen = **List.ReplaceMatchingItems**(Spalten, Renames),

SpaltenAusklappen = **Table.ExpandTableColumn**(MeineTabelle,  
 "JoinTabelle",  
 Spalten,  
 Umbenennen)



## ② #Goldstück 2 | Spaltensortierung



Originale Spaltenreihenfolge wiederherstellen, dabei einzelne Spalten an Anfang bzw. Ende einer Tabelle verschieben

durch Transformationen, Bearbeitungen etc. kann sich Reihenfolge von Spalten verändert haben  
einer der wenigen sinnvollen Einsätze für `MissingField.Ignore`

`ColOrder =`

`let`

`ColStart = {"ErsteSpalte"},`

`ColEnd = {"LetzteSpalte"},`

`ColMid = List.RemoveMatchingItems(  
    Table.ColumnNames(OriginaleTabelle),  
    ColStart & ColEnd)`

`in`

`Table.ReorderColumns(FastFinaleTabelle,  
    ColStart & ColMid & ColEnd, MissingField.Ignore)`

## ③ #Goldstück 3 | Spaltentyp-Festlegung (I)



### Spaltentypen dynamisch festlegen

Alle Spalten als Text-Spalten formatieren

AlleSpaltenText =

```
Table.TransformColumnTypes(Quelle,  
  List.Transform(Table.ColumnNames(Quelle),  
    each {_, type text}))
```

## ③ #Goldstück 3 | Spaltentyp-Festlegung (II)



### Spaltentypen dynamisch festlegen

Spaltentypen anhand Spaltenbezeichnung festlegen

AnhandNamen =

```
Table.TransformColumnTypes(Quelle,  
    List.Transform(Table.ColumnNames(Quelle),  
        (x)=> if Text.Contains(x, „ID“)  
            then {x, Int64.Type}  
            else {x, type text}))
```

## ③ #Goldstück 3 | Spaltentyp-Festlegung (III)



### Spaltentypen dynamisch festlegen

Spaltentypen anhand Daten in erster Zeile festlegen

```
AnhandDatenInErsterZeile =  
    let  
        TypenErsteZeile =  
            List.Transform(Table.ToRows(Quelle){0},  
                (x)=> if not (try Value.Type(x))[HasError]  
                    then  
                        if x = null then type any else Value.Type(x)  
                    else type any,  
    in  
        Table.TransformColumnTypes(Quelle,  
            List.Zip({Table.ColumnNames(Quelle), TypenErsteZeile}))
```



### Spalten auswählen

Per Funktion oder per Projektion

`AuswahlPerFunktion = Table.Selectcolumns(Quelle, {„Col1“, „Col2“})`

`AuswahlPerProjektion = Quelle[[Col1], [Col2]]`

`Quelle[[Col1]]` > Rückgabewert = Tabelle mit einer Spalte

`Quelle[Col1]` > Rückgabewert = Liste

## 5 #Goldstück 5 | Mehrere Spalten auf einmal anlegen



Mehrere Spalten einschließlich Spaltentyp auf einmal anlegen

```
NeueSpaltenAlsRecord = Table.AddColumn(Queue, "TEMP",  
    each [  
        NeueSpalteA = BELIEBIGE BERECHNUNG/BELIEBIGER WERT,  
        NeueSpalteB = BELIEBIGE BERECHNUNG/BELIEBIGER WERT,  
        NeueSpalteC = BELIEBIGE BERECHNUNG/BELIEBIGER WERT,  
        NeueSpalteD = BELIEBIGE BERECHNUNG/BELIEBIGER WERT  
    ],  
    type [NeueSpalteA = text, NeueSpalteB = Int64.Type,  
        NeueSpaltenD = number, KomischeSpalteDieEsGarNichtGibt = any]),  
Expand_TEMP = Table.ExpandRecordColumn(NeueSpaltenAlsRecord,  
    "TEMP", Record.FieldNames(NeueSpaltenAlsRecord[TEMP]{0})
```

## ⑥ #Goldstück 6 | Abfragen dynamisch auswählen



Abfragen anhand Namen dynamisch auswählen  
(hier: Tabellen mit bestimmtem Präfix kombinieren)

Tabellenauswahl =

**let**

AlleAbfragen = #sections[Section1],

Liste\_Abfragen =

**List.Select**(Record.FieldNames(AlleAbfragen),

(x)=> **Text.StartsWith**(x, "BestimmterPräfix")),

MeineTabellen = **Record.ToList**(Record.SelectFields(

AlleAbfragen, Liste\_Abfragen))

**in**

MeineTabellen,

TabellenKombinieren = **Table.Combine**(Tabellenauswahl)



## ⑦ #Goldstück 7 | Zahlen/Zeichen extrahieren (I)



### Zahlen extrahieren

Quelle = "dfjhsjFG3287483ncgk,m,dmfsd", //Beispieltext

ÜberFehlerauswertung =

```
Text.Combine(List.Select(  
    Text.ToList(Quelle),  
    (x)=> (try Number.From(x) is number)[HasError] = false) )
```

ÜberAsciiCode =

```
Text.Combine(List.Select(  
    Text.ToList(Quelle),  
    (x)=> List.MatchesAny({48..57},  
        each _ = Character.ToNumber(x))))
```

## 7 #Goldstück 7 | Zahlen/Zeichen extrahieren (II)



### Zeichen extrahieren

Quelle = "dfjhsjFG3287483ncgk,m,dmfsd", //Beispieltext

ÜberZeichen =

```
Text.Combine(List.Select(
    Text.ToList(Quelle),
    (x)=> List.MatchesAny({"F", "G", "k", "p", "3", "7"},
        each _ = x)))
```

ÜberStartzeichenUndLänge =

let

```
PosStart = Text.PositionOf(Quelle, "mfs")
```

in

```
if PosStart = -1 then null
```

```
else Text.Middle(Quelle, PosStart, 10)
```

## 8 #Goldstück 8 | Namensliste „putzen“ (I)



### Namensliste putzen

Ersetzung1 = {"é", "e"}, {"ä", "ae"}, {"v.", "von"}, {" ", " "}, {"ü", „ue“} //Buchstabenwechsel

Ersetzung2 = List.Transform({10, 13}, (x)=> {Character.FromNumber(x), " "}) //Umbrüche gg. Leerzeichen

Ersetzung3 = List.Transform({0..31, 33..43, 46..64, 91..96, 123..127, 128..197}, (x)=> {Character.FromNumber(x), ""}) //ungültige Zeichen löschen

```
NichtDruckbareZchEntf =  
    Text.Clean(EingegebenerName),  
TeileZwKommaGetrimmt =  
    Text.Combine(  
        List.Transform(  
            Text.Split(  
                NichtDruckbareZchEntf, ","),  
            Text.Trim),  
        ",",  
    ),
```

```
MehrfachLeerzeichenEntfernen =  
Text.Combine(  
    List.RemoveMatchingItems(  
        Text.Split(TeileZwKommaGetrimmt,  
            " " ),  
        {" "}),  
    " " ),  
Ersetzungen = List.Combine(  
    {Ersetzung1, Ersetzung2, Ersetzung3}),  
MeinTextNeu = List.Accumulate(  
    Ersetzungen,  
    MehrfachLeerzeichenEntfernen,  
    (s, c)=> Text.Replace(s, c{0}, c{1}) )
```

## ⑧ #Goldstück 8 | Namensliste „putzen“ (II)

---



### Namensliste putzen

**Text.Clean** entfernt nichtdruckbare Zeichen.

Für das Ersetzen von (vielen) Zeichen wird eine geschachtelte Liste (Liste von Listen) benötigt, die Paare (alt/neu) von Zeichenketten enthält. Mithilfe von **List.Accumulate** wird diese Liste einzeln durchlaufen.

Die Funktion **Character.FromNumber** wandelt Ascii-Code in Zeichen um.

## ⑧ #Goldstück 8 | Namensliste „putzen“ (III)



### Namensliste putzen

Vor- und Nachname tauschen

```
//fxVNzuNV
```

```
(NameVN as text) =>
```

```
let
```

```
    NameGesplittet = Text.Split(NameVN, " "),
```

```
    Nachname = NameGesplittet[List.Count(NameGesplittet)-1],
```

```
    Vornamen = Text.Combine(  
        List.RemoveMatchingItems(NameGesplittet, {Nachname}), " "),
```

```
    NameNV = Text.Combine({Nachname, Vornamen}, ", ")
```

```
in
```

```
    NameNV
```



### Namensliste putzen

fxVNzuNV wandelt einen Namen im Format >Vorname Nachname< in das Format >Nachname, Vorname< um.

Enthält eine Namensliste mal Einträge im Format „Vorname(n) Nachname“ und „Nachname, Vorname“

kann so vor dem Abgleich mit anderen Datenquellen die Darstellung vereinheitlicht werden (Anwendung der Funktion von Vorhandensein Komma abhängig machen!

z. B. `if not Text.Contains(NameVN, ",") then ... else ...`

## 9 #Goldstück 9 | Erster Wert, der nicht null ist (I)



Aus einer Liste von Werten den ersten „gültigen“ Wert ermitteln

Excel 365:

Wert1 ?? Wert2 ?? Wert3

Excel 2016:

fxErsteNichtNull =

(Input as list) as any =>

**List.Accumulate**(Input, null, (s,c)=> if s = null then c else s)

Aufruf:

fxErsteNichtNull({Wert1, Wert2, Wert3})

## ⑨ #Goldstück 9 | Erster Wert, der nicht null ist (II)

---



Aus einer Liste von Werten den ersten „gültigen“ Wert ermitteln

Erst in neueren Power Query Versionen ist der Coalesce-Operator „??“ verfügbar. In früheren Versionen muss man sich behelfen mit if then else oder List.Accumulate.

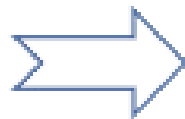


## 10 #Goldstück 10 | Pivotieren mit Aggregation von Textwerten (I)



Beim Pivotieren von Daten Textwerte zusammenfassen

Tag	Klasse	Fach
Montag	1a	Deutsch
Montag	1a	Mathe
Dienstag	1a	Zeichen
Mittwoch	1b	Sport
Montag	1b	Grammatik
Dienstag	1b	Leichtathletik
Dienstag	1a	Volleyball
Freitag	1c	Turnen am Reck



Tag	1a	1b	1c
Dienstag	Zeichen/Volleyball		Leichtathletik
Freitag			Turnen am Reck
Mittwoch		Sport	
Montag	Deutsch/Mathe	Grammatik	

## 10 #Goldstück 10 | Pivotieren mit Aggregation von Textwerten (II)



### Beim Pivotieren von Daten Textwerte zusammenfassen

Textaggregation ist nur über M-Code erreichbar

**Table.Pivot(Quelle, List.Distinct(Quelle[Klasse]), "Klasse", "Fach",  
each Text.Combine(\_, "/"))**

	ABC 123 Tag	ABC 123 1a	ABC 123 1b	ABC 123 1c
1	Dienstag	Error	Leichtathletik	null
2	Freitag	null	null	Turnen am Reck
3	Mittwoch	null	Sport	null
4	Montag	Error	Grammatik	null

	ABC 123 Tag	ABC 123 1a	ABC 123 1b	ABC 123 1c
1	Dienstag	Zeichen/Volleyball	Leichtathletik	
2	Freitag			Turnen am Reck
3	Mittwoch		Sport	
4	Montag	Deutsch/Mathe	Grammatik	



## Mehrere Spalten einer Tabelle zu einer Spalte zusammenführen, dabei die Spalten-Überschriften erhalten

Funktion > Spaltenauswahl durch Übergabe des nullbasierten Spaltenindexes

Aufruf: fxSpaltenKondensieren(Quelle, {1..3}, "Kondensiert")

Tag	Aktion	Eine Zahl	Noch was
Montag	Nichts tun	34	Katze
Dienstag	Akkordeon spielen	7727	Hund
Freitag	Teppich knüpfen	478	Schaf



Tag	Kondensiert
Montag	Aktion = Nichts tun Eine Zahl = 34 Noch was = Katze
Dienstag	Aktion = Akkordeon spielen Eine Zahl = 7727 Noch was = Hund
Freitag	Aktion = Teppich knüpfen Eine Zahl = 478 Noch was = Schaf

## 11 #Goldstück 11 | Spalten zusammenführen, Header erhalten (II)

Mehrere Spalten einer Tabelle zu einer Spalte zusammenführen, dabei die Spalten-Überschriften erhalten

```
//fxSpaltenKondensieren
```

```
(Tabelle as table, Spaltenindizes as list,  
optional NeuerSpaltenname as text, optional AlteSpaltenLöschen as logical) =>  
let  
    Spalten = List.Buffer(Table.ColumnNames(Tabelle)),  
    GewählteSpalten = List.Transform(List.Distinct(Spaltenindizes), each Spalten{_}),  
    ColName = if NeuerSpaltenname = null then "Kondensat" else NeuerSpaltenname,  
    Spalte_Kondensat = Table.AddColumn(Tabelle, ColName, each let  
        Werte = Record.ToList(Record.SelectFields(_, GewählteSpalten)),  
        WerteAlsText = List.Transform(Werte, each Text.From(_)),  
        HorizKombi = List.Transform(List.Zip({GewählteSpalten, WerteAlsText}), each Text.Combine(_, " = ")),  
        in Text.Combine(HorizKombi, Character.FromNumber(10)), type text),  
    SpaltenLöschen = Table.RemoveColumns(Spalte_Kondensat, GewählteSpalten)  
in  
    if AlteSpaltenLöschen = false then Spalte_Kondensat else SpaltenLöschen
```