HANSER

Leseprobe

zu

M
Daten abfragen und verarbeiten mit Excel und Power BI

von Ignaz A. Schels

ISBN (Buch): 978-3-446-45588-7
ISBN (E-Book): 978-3-446-45685-3

Weitere Informationen und Bestellungen unter
https://www.hanser-fachbuch.de/
sowie im Buchhandel

Inhalt

Vorwort ..... **XI**

1 Grundlagen der Abfrage-Logik **1**

2 Abrufen: Verbindungen zu externen Quellen herstellen **3**
2.1 Einfache Abfrage auf Excel-Datei 5
2.2 Aktualisieren der Abfrage 8
2.3 Abfragen automatisch aktualisieren 10

3 Transformieren: Aufbereiten der abgerufenen Daten **13**
3.1 Der Abfrage-Editor 13
3.2 Transformationsschritte löschen 15
3.3 Zeilen filtern 16
3.4 Überschriften (Header) einfügen 17
3.5 Spalten entfernen 17
3.6 Werte ersetzen 18
3.7 Datentypen bestimmen 18
3.8 Transformationsschritte prüfen und benennen 22
3.9 Abschließen der Transformation 23
3.10 Erneutes Bearbeiten der Abfrage 23

4 Erweiterte Transformationen **25**
4.1 Nach unten bzw. oben ausfüllen 27
4.2 Zeilen und Spalten vertauschen (Transponieren) 28
4.3 Spalten verbinden 29
4.4 Entpivotieren 30
4.5 Spalten teilen und Abschluss der Transformation 31

5 Anpassen der Rohdaten **33**
5.1 Texttransformationen 33
5.2 Datumstransformationen 36

5.3 Zahlentransformationen ... 37
5.4 Gruppieren ... 38

6 Abfragen auf Webseiten ... 41
6.1 Webabfrage auf eine Tabelle ... 41
6.2 Webabfrage ohne Tabellen ... 44

7 Abfragen auf relationale Datenbanken ... 49
7.1 Beispiel mit Oracle-Datenbank ... 49
7.2 Query Folding ... 55

8 Mehrere Quellen kombinieren ... 59
8.1 Abfragen anfügen ... 59
8.2 Abfragen zusammenführen ... 62
8.3 Alle Dateien in einem Ordner einlesen ... 67

9 Spalten hinzufügen – erste Schritte mit M ... 73
9.1 Bedingte Spalten ... 73
9.2 Datums- und Rechenfunktionen ... 77
9.3 Benutzerdefinierte Spalten ... 77
9.3.1 Spaltenformeln für Berechnungen eingeben ... 78
9.3.2 Bedingungslogik mit if ... 79
9.3.3 M-Funktionen ... 80

10 Abfragecode bearbeiten mit M ... 83
10.1 Die Bearbeitungsleiste ... 83
10.2 Das Editor-Fenster ... 84
10.3 Erstellen einer leeren Abfrage ... 85
10.4 Die Grundstruktur des Abfrage-Codes ... 86
10.5 Schritte und Schrittnamen verstehen ... 87
10.6 Fehler im M-Code vermeiden ... 89
10.7 Schritte zusammenfassen ... 90
10.8 Kommentare ... 91

11 Eingabehilfen mit Visual Studio ... 93
11.1 Die Vorteile von Visual Studio ... 94
11.2 Installation ... 95

12 Werte und Datentypen ... 99
12.1 Einfache Datentypen ... 99
12.1.1 Null ... 99
12.1.2 Logical (true/false) ... 99
12.1.3 Number (Zahl) ... 101
12.1.4 Time (Zeit) ... 102

12.1.5 Date (Datum) ... 102
12.1.6 Datetime (Datum/Uhrzeit) ... 103
12.1.7 Datetimezone (Datum/Uhrzeit/Zeitzone) ... 103
12.1.8 Duration (Dauer) ... 104
12.1.9 Text ... 105
12.1.10 Binary (Binär) ... 108
12.2 Übergeordnete Datentypen ... 108
12.2.1 List (Liste) ... 108
12.2.2 Record (Datensatz) ... 110
12.2.3 Table (Tabelle) ... 111
12.3 Spezielle Datentypen ... 114
12.3.1 Type (Datentyp) ... 114
12.3.2 Function (Funktion) ... 116

13 Abfragen flexibel gestalten ... 119
13.1 Nicht-lineare Abfragen ... 119
13.2 Unterschiedliche Tabellenzeilen vergleichen ... 125
13.3 Abfragen mit Parametern ... 128
13.3.1 Power-Query-Parameter ... 128
13.3.2 Parameter aus anderen Quellen ... 132

14 Fehlerbehandlung ... 137
14.1 Fehler vorbeugen ... 137
14.2 Error-Werte entfernen ... 138
14.3 Fehler abfangen mit ***try*** ... 140
14.4 Fehlermeldungen erzeugen ... 141

15 M-Funktionen ... 145
15.1 Das Syntax-Schema ... 145
15.2 Abruf der Funktionsliste ... 146
15.3 Listenfunktionen ... 150
15.3.1 Listenerstellung ... 150
15.3.2 Informationen über Listenelemente ... 153
15.3.3 Berechnungen ... 157
15.3.4 Ordnung und Reihenfolge ... 158
15.3.5 Auswahl ... 162
15.3.6 Transformationen ... 167
15.3.7 Vergleiche mehrerer Listen ... 174
15.4 Datensatzfunktionen ... 175
15.4.1 Erstellung und Umwandlung ... 175
15.4.2 Informationen über Datensätze ... 177
15.4.3 Auswahl ... 177
15.4.4 Transformationen ... 179
15.5 Tabellenfunktionen ... 182
15.5.1 Informationen über Tabellen ... 183

15.5.2 Umwandlungen ... 186
15.5.3 Spalten ... 187
15.5.4 Zeilen ... 194
15.5.5 Ordnung und Sortierung ... 207
15.5.6 Erstellung ... 210
15.5.7 Transformationen ... 213
15.5.8 Arbeiten mit mehreren Tabellen ... 221
15.5.9 Sonstige ... 224
15.6 Textfunktionen ... 224
15.6.1 Erstellung und Konvertierung ... 224
15.6.2 Informationen über Texte ... 228
15.6.3 Auszüge von Textteilen ... 230
15.6.4 Modifikationen ... 231
15.6.5 Transformationen ... 233
15.7 Zahlenfunktionen ... 238
15.7.1 Informationen über Zahlen ... 238
15.7.2 Rechenoperationen ... 239
15.7.3 Rundung ... 242
15.7.4 Zufallszahlen ... 244
15.7.5 Trigonometrie ... 245
15.7.6 Konvertierung und Formatierung ... 247
15.8 Logical-Funktionen ... 253
15.9 Datumsfunktionen ... 254
15.9.1 Erzeugung und Umwandlung ... 254
15.9.2 Informationen über Datumswerte ... 256
15.9.3 Berechnungen ... 262
15.9.4 Vergleiche mit 'Jetzt' ... 263
15.10 DateTime-Funktionen ... 267
15.10.1 Erzeugung und Umwandlung ... 267
15.10.2 Vergleiche mit 'Jetzt' ... 270
15.11 DateTimeZone-Funktionen ... 272
15.12 Duration-Funktionen ... 277
15.13 Zeitfunktionen ... 280
15.14 Hilfsfunktionen ... 282
15.14.1 Comparer-Funktionen ... 282
15.14.2 Replacer-Funktionen ... 284
15.14.3 Combiner-Funktionen ... 285
15.14.4 Splitter-Funktionen ... 288

16 Eigene Funktionen erstellen ... 293
16.1 Funktionen innerhalb einer Abfrage ... 293
16.2 Das Schlüsselwort *each* ... 294
16.3 Funktionen als eigene Abfrage ... 295
16.4 Funktionen, die aus mehreren Schritten bestehen ... 298

16.5 Rekursive Funktionen ... 302
16.6 Funktionsbeschreibungen in der Metadata ... 304
16.7 Funktionen teilen ... 307
16.7.1 Aktivierung und Vorbereitung ... 307
16.7.2 Funktionserweiterungen ... 308

17 M und VBA ... 311
17.1 Abfragen per VBA-Befehl aktualisieren ... 311
17.1.1 Schaltflächen zum Aktualisieren nutzen ... 311
17.1.2 Aktualisieren bei Ereignissen ... 314
17.2 Abfrage-Code per VBA anpassen ... 315
17.3 Abfrageergebnisse mit VBA auslesen ... 318

18 Datenschutz und Firewall ... 321
18.1 Datenschutz bei verschiedenen Datenquellen ... 321
18.2 Der Formula.Firewall-Fehler ... 322

19 Tipps für mehr Geschwindigkeit ... 325
19.1 Schnelleres Entwerfen der Abfrage ... 325
19.2 Schnelleres Aktualisieren ... 325
19.3 Messen der Geschwindigkeit ... 327

Index ... 331

Vorwort

Es mag ungewöhnlich erscheinen, ein Fachbuch zu schreiben, das nur einen Teilbereich von zwei so vielseitigen Programmen wie Excel und Power BI Desktop behandelt. Warum nicht stattdessen ein Buch, das entweder Excel oder Power BI in allen Facetten erklärt?

Wenn Sie dieses Buch gekauft haben, ahnen Sie zumindest, welches Potenzial in den Features schlummert, die unter dem Namen *Power Query* zusammengefasst werden. Excel-Nutzer verbringen unzählige Stunden damit, Daten aus den verschiedensten Quellen zusammenzukopieren und dabei händisch zu modellieren. Etliche Power-BI-Nutzer wissen zwar, wie Daten abgerufen werden, stoßen aber schnell an ihre Grenzen, wenn die Datenbasis nicht richtig aufbereitet ist. Wenn Sie nicht zu diesen Benutzern gehören wollen, haben Sie das richtige Buch gefunden.

Das Thema des vorliegenden Buchs

Was als Add-In für Excel begann, ist nun fester Teil von einer wachsenden Menge an Microsoft-Produkten, allen voran Excel und Power BI Desktop. Die Funktion der Abfragen und die Benutzeroberfläche des Abfrage-Editors sind dabei bis auf wenige Ausnahmen programmübergreifend gleich. Der Hauptunterschied ist die Hintergrundfarbe (hellgrau bei Excel und dunkelgrau bei Power BI). Deshalb ist es durchaus möglich, Power Query sowohl für Excel als auch Power BI Desktop zu erklären.

Aus genau diesem Grund findet aber auch eine starke Beschränkung der abgedeckten Themen statt: Das Buch behandelt nur das Abrufen von externen Daten und die Bearbeitung (Transformation) im Abfrage-Editor, basierend auf der Abfragesprache M. Sie werden lernen, wie Sie Daten in Ihr Programm laden und dabei genau in die Form bringen, in der Sie sie benötigen. Die so aufbereiteten Tabellen können dann in Excel und Power BI nach Belieben interpretiert und visualisiert werden – doch das ist nicht mehr Thema des Buchs.

Unterschiede zwischen den Versionen

Die Grundlagen der Abfrage-Logik und der Formelsprache M haben sich seit den ersten Versionen von Power Query kaum verändert. Die Benutzeroberfläche und die Entwicklungsumgebung des Abfrage-Editors werden jedoch laufend optimiert.

Die regelmäßige Verbesserung eines Programms ist natürlich zu begrüßen, doch erschwert sie das Lernen und Lehren der Grundlagen. Es ist nicht auszuschließen, dass das eine oder andere Beispiel auf Ihrem Computer nicht genauso funktioniert wie hier beschrieben – sei

es, dass Sie eine ältere Version benutzen oder sei es, dass nach der Buchveröffentlichung weitere Neuerungen eingeführt wurden.

Seien Sie aber bitte nicht frustriert, wenn eine Beschreibung oder eine Abbildung nicht exakt zutrifft. In den meisten Fällen ist das entsprechende Werkzeug nur anders benannt oder befindet sich an einer anderen Stelle. Spätestens wenn es um das Schreiben von M-Code geht, sind Versionsunterschiede ohnehin kaum mehr von Belang.

Kontakt zum Autor

Auch bei einem so starren Medium wie einem Buch darf der Kontakt zwischen Leser und Autor nicht zu kurz kommen. Scheuen Sie sich nicht, mir zu schreiben, wenn Sie Fragen zu einem der behandelten Themen haben. Ein Software-Spezialist lebt von solchen Herausforderungen. Feedback jeder Art, egal ob Lob oder Kritik, ist natürlich auch immer willkommen. Schicken Sie einfach eine Mail an *ignaz.alexander@schels.de*.

13 Abfragen flexibel gestalten

Mit dem Verständnis der M-Grundlagen eröffnen sich neue Möglichkeiten, um Abfragen flexibler und sicherer zu gestalten. Die meisten Abfragen beruhen auf festen Annahmen: Die Datei bzw. Datenbank muss an einem vordefinierten Ort liegen. Ihr Inhalt muss exakt dasselbe Layout haben wie zu dem Zeitpunkt, als die Abfrage erstellt wurde. Die Abfrage arbeitet Schritt für Schritt, wobei sich jeder Schritt auf den vorherigen bezieht.

Obwohl diese Aussagen auf mindestens 90 % aller Abfragen zutreffen, sind sie keineswegs zwingend notwendig. Mit geschickten Anpassungen des M-Codes können Sie starre Abfragen auf die jeweiligen Bedürfnisse anpassen.

13.1 Nicht-lineare Abfragen

Die meisten Programmiersprachen beinhalten das Konzept von Variablen: Ein Zeichen oder Wort kann als Platzhalter für einen Wert dienen und bei Bedarf als Referenz auf diesen Wert benutzt werden. Das, was in diesem Buch als Schrittnamen bezeichnet wird, sind im Prinzip Variablen: Ihnen wird ein Wert zugewiesen, damit sie später an Stelle dieses Werts verwendet werden können.

Werfen Sie ein Blick auf den Code in Listing 13.1:

Listing 13.1 Eine einfache Abfrage auf eine Excel-Datei.

```
let
    Quelle = Excel.Workbook(File.Contents(
             "C:\Beispiele\13-01-Pegel-Isar.xlsx"), null, true),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
             [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes(#"Höher gestufte Header",
             {{"Datum", type date}, {"Uhrzeit", type time},
             {"Wasserstand cm über NN", Int64.Type}})
in
    #"Geänderter Typ"
```

Die Abfrage besteht aus vier Schritten: `Quelle`, `Tabelle1_Sheet`, `Höher gestufte Header` und `Geänderter Typ`. Mit Ausnahme des ersten Schritts finden sich in allen Wertzuweisun-

gen Verweise auf den vorhergehenden Schritt. Diese Reihenfolge erscheint zwar logisch, ist aber für das Ergebnis völlig irrelevant. Die Abfrage würde genauso funktionieren, wenn die Schritte in einer anderen Reihenfolge geschrieben wären.

Listing 13.2 Die Transformationsschritte werden vertauscht.

```
let
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
             [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes(#"Höher gestufte Header",
             {{"Datum", type date}, {"Uhrzeit", type time},
             {"Wasserstand cm über NN", Int64.Type}}),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    Quelle = Excel.Workbook(File.Contents(
             "C:\Beispiele\13-01-Pegel-Isar.xlsx"), null, true)
in
    #"Geänderter Typ"
```

Der Befehl `Quelle`, der im ursprünglichen Code am Anfang stand, steht nun am Ende der Abfrage. Der erste Transformationsschritt `Höher gestufte Header` kann natürlich nicht ausgeführt werden, bevor die Verbindung zur Quelldatei hergestellt wurde. Dennoch liefert die Abfrage in Listing 13.2 dasselbe Ergebnis wie die Original-Abfrage aus Listing 13.1. Power Query ist also im Stande, die sinnvollste Reihenfolge der Abarbeitung selbst zu ermitteln.

Diese Erkenntnis kann von großem Nutzen sein, denn sie ermöglicht es, Abfragen zu schreiben, die nicht linear ablaufen, sondern Abzweigungen oder sogar Schleifen enthalten.

Betrachten wir folgendes Beispiel, das einen einfachen, aber häufigen Fall zeigt.

```
Export 004 vom 12.03.2018
Zeitraum: 01.02.2018 - 28.02.2018
Abteilung: P2
Anmerkungen: (keine)

Bestell-Nr      Artikel Anzahl  Datum
10023   b-62    3       02.02.2018
10023   u-33    5       02.02.2018
10023   f-94    87      02.02.2018
10024   s-55    21      04.02.2018
10024   d-86    3       04.02.2018
10024   u-57    51      04.02.2018
10024   u-25    6       04.02.2018
10024   d-24    3       04.02.2018
10024   d-36    16      04.02.2018
10025   s-55    5       05.02.2018
10026   f-52    24      10.02.2018
10026   f-93    4       10.02.2018
10027   b-23    8       17.02.2018
10027   d-12    3       17.02.2018
10027   u-25    2       17.02.2018
10027   f-54    15      17.02.2018
10027   d-72    6       17.02.2018
10027   s-91    12      17.02.2018
10027   f-94    3       17.02.2018
10027   d-33    4       17.02.2018
10027   u-25    5       17.02.2018
Bestellungen: 5
```

Bild 13.1 Die Struktur der Beispieldatei.

Die Beispieldatei *13-02-Export_Feb.txt* enthält eine Tabelle, deren Spalten durch Tab-Zeichen getrennt sind. Allerdings gibt es vor und nach der Tabelle Zeilen, die nicht zur Tabelle gehören. Das Problem ist, dass sich die Anzahl der Zeilen vor der Tabelle sowie die Anzahl der Zeilen in der Tabelle ändern können. Um die Tabelle vom Rest isolieren zu können, kann also keine fixe Zeilenzahl angenommen werden. Hinzu kommt, dass die Abteilungsnummer in den Kopfzeilen in die jeweiligen Tabellenzeilen übernommen werden sollte. Sie muss also irgendwie gespeichert werden, bevor die Kopfzeilen entfernt werden.

- Erstellen Sie eine Abfrage auf die Textdatei *13-02-Export_Feb.txt* und bearbeiten Sie sie im Abfrage-Editor. Stellen Sie dabei sicher, dass als Trennzeichen *Tab* eingestellt ist.
- Löschen Sie den automatisch eingefügten Transformationsschritt *Geänderter Typ*, er ist hier überflüssig.
- Zunächst soll die Abteilungsnummer in der vierten Zeile extrahiert werden. Filtern Sie hierfür die erste Spalte nach Werten beginnend mit "Abteilung:".

Ein normaler Textfilter wäre hier problematisch. Wenn Sie die Filterfunktion benutzen und in das Textfeld das Wort "Abteilung" eingeben, sieht es zwar so aus, als würde nur nach diesem Begriff gefiltert.

Ein Blick auf die Bearbeitungsleiste zeigt jedoch, dass der Befehl den kompletten Wert „Abteilung: P2" enthält, auch wenn Sie nur nach „Abteilung" gesucht haben.

```
= Table.SelectRows(Quelle, each ([Column1] = "Abteilung: P2"))
```

Bild 13.2 Der Befehl in der Bearbeitungsleiste.

Damit auch bei anderen Abteilungen der richtige Wert extrahiert wird, müssen Sie daher den Textfilter *Beginnt mit…* verwenden.

- Da wir nur die Abteilungsnummer, hier „P2" benötigen, muss der Rest des Werts entfernt werden. Klicken Sie dafür mit der rechten Maustaste auf die erste Spalte, wählen Sie **Werte ersetzen** und ersetzen Sie die Zeichenfolge „Abteilung: " (mit Doppelpunkt und Leerzeichen) durch nichts.
- Der momentan angezeigte Wert ist eine Tabelle. Es fehlt aber nur noch ein Schritt, um den gesuchten Textwert zu isolieren: Klicken Sie mit der rechten Maustaste auf den Zellwert *P2* und wählen Sie im Kontextmenü den Eintrag **Drilldown ausführen**.

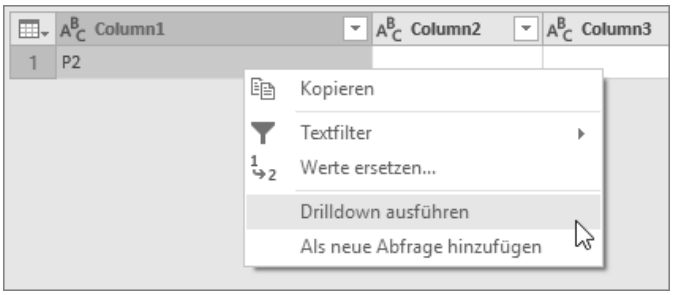

Bild 13.3 Ein Drilldown wählt einen einzelnen Wert aus einer Tabelle.

In der Bearbeitungsleiste sehen Sie, wie der zugehörige M-Befehl aussieht:

```
= #"Ersetzter Wert"{0}[Column1]
```

Aus der Tabelle, die der vorherige Schritt zurückgegeben hat, wurde der Wert in der ersten Zeile (d. h. der Eintrag *0* in der Liste) und der ersten Spalte (mit dem Namen *Column1*) ausgewählt. Der aktuelle Schritt gibt also den Textwert *P2* zurück.

- Benennen Sie den aktuellen Transformationsschritt um in *Abteilung*.
- Wie kommen Sie nun wieder an die Ausgangstabelle? Ganz einfach: Erstellen Sie einen neuen Transformationsschritt über das *fx*-Symbol in der Bearbeitungsleiste und geben Sie als Code für diesen Schritt Folgendes ein:

  ```
  = Quelle
  ```

- Bestätigen Sie mit **Enter**. Da *Quelle* der Name des ersten Schritts war, erscheint nun der Rückgabewert dieses Schritts wieder.

Auf ähnliche Weise sollen nun die Zeilennummer für den Beginn und das Ende der Tabelle extrahiert werden. Der besseren Übersicht wegen, reduzieren wir die Tabelle auf eine Spalte. Sie wissen ja nun, wie Sie die Original-Tabelle bei Bedarf wieder zurückholen.

- Entfernen Sie alle Spalten außer der ersten über den Befehl **Andere Spalten entfernen**.
- Klicken Sie im Register **Spalte hinzufügen** auf **Indexspalte**.

	Column1	Index
1	Export 004 vom 12.03.2018	0
2	Zeitraum: 01.02.2018 - 28.02.2018	1
3	Abteilung: P2	2
4	Anmerkungen: (keine)	3
5		4
6	Bestell-Nr	5
7	10023	6
8	10023	7
9	10023	8
10	10024	9
11	10024	10
12	10024	11

Bild 13.4 Die Indexspalte liefert Zeilennummern, beginnend mit 0.

- Um an den Beginn der gesuchten Tabelle zu kommen, filtern Sie die erste Spalte nach Werten, die mit dem Text „Bestell-Nr“ beginnen.
- Führen Sie einen Drilldown auf die Zeilennummer in der Indexspalte aus.
- Benennen Sie den letzten Abfrageschritt um in *Anfang*.
- Um die Abfrage etwas zu verkürzen, können Sie die letzten beiden Schritte zu einem zusammenfassen. Öffnen Sie dazu den Erweiterten Editor.

- Der vorletzte Abfrageschritt sollte *Gefilterte Zeilen1* heißen. Markieren Sie den zugehörigen Ausdruck rechts vom Gleichheitszeichen bis vor das Komma am Zeilenende. Kopieren Sie den markierten Text in den Transformationsschritt *Anfang* und ersetzen Sie dort den Namen des alten Schritts `#"Gefilterte Zeilen1"`. Der Befehl sollte nun so aussehen:

```
Anfang = Table.SelectRows(#"Hinzugefügter Index", each
    Text.StartsWith([Column1], "Bestell-Nr")){0}[Index]
```

- Anschließend können Sie den Schritt *Gefilterte Zeilen1* löschen.

Für das Ende der Tabelle soll die gleiche Logik angewandt werden. Nun, da Sie alles, was dafür nötig ist, in einem Schritt haben, können Sie den Befehl direkt im Editor schreiben.

- Fügen Sie unter dem Schritt *Anfang* einen neuen Schritt *Ende* ein. Der Code dafür ist fast identisch, nur statt `"Bestell-Nr"` lautet der Filter-Text `Bestellungen`. Der Schritt Ende hat also diesen Code:

```
Ende = Table.SelectRows(#"Hinzugefügter Index", each
    Text.StartsWith([Column1], "Bestellungen")){0}[Index]
```

- Geben Sie ein Komma am Ende des Schritts Anfang ein.
- Ersetzen Sie den Schrittnamen in der letzten Zeile nach dem `in` durch `Ende`.
- Schließen Sie den Erweiterten Editor.
- Fügen Sie einen neuen Schritt ein und geben Sie als Code dafür `= Quelle` ein. Somit erscheint wieder die Ursprungstabelle.

Die Werte für Abteilung, Anfang und Ende können nun in weiteren Transformationen verwendet werden.

- Klicken Sie im Register **Start** auf **Zeilen beibehalten – Erste Zeilen beibehalten**.
- Geben Sie als Anzahl die Zahl *10* ein – sie ist nur ein Platzhalter und wird gleich ersetzt.
- In der Bearbeitungsleiste sehen Sie den Code, den Sie damit generiert haben:

```
= Table.FirstN(Benutzerdefiniert2, 10)
```

- Anstelle der Zahl 10 soll die vorhin ermittelte Zeilenanzahl stehen. Ersetzen Sie die *10* durch *Ende*.
- Der Schritt *Benutzerdefiniert2* verweist eigentlich nur auf den ersten Schritt *Quelle*. Er ist daher überflüssig. Ersetzen Sie *Benutzerdefiniert2* im letzten Befehl durch *Quelle*. Der Befehl sollte nun also so aussehen:

```
= Table.FirstN(Quelle, Ende)
```

- Bestätigen Sie mit **Enter** und löschen Sie den Schritt *Benutzerdefiniert2*.
- Klicken Sie im Register **Start** auf **Zeilen entfernen – Erste Zeilen entfernen**. Geben Sie eine beliebige Zahl ein, zum Beispiel *3*.
- Die eingegebene Zahl wird nun in der Bearbeitungsleiste ersetzt durch *Anfang*.
- Klicken Sie im **Start**-Register auf **Erste Zeile als Überschriften verwenden**. Die Tabelle ist nun isoliert.

- Zum Abschluss wird noch die Abteilungsnummer angefügt. Klicken Sie im Register **Spalte hinzufügen** auf **Benutzerdefinierte Spalte**.
- Als Formel für die neue Spalte geben Sie einfach den Namen des Transformationsschritts an, in dem der Wert gespeichert wurde. Geben Sie also sowohl als Formel als auch als Spaltentitel *Abteilung* ein.

	Bestell-Nr	Artikel	Anzahl	Datum	Abteilung
1	10023	b-62	3	02.02.2018	P2
2	10023	u-33	5	02.02.2018	P2
3	10023	f-94	87	02.02.2018	P2
4	10024	s-55	21	04.02.2018	P2
5	10024	d-86	3	04.02.2018	P2
6	10024	u-57	51	04.02.2018	P2
7	10024	u-25	6	04.02.2018	P2
8	10024	d-24	3	04.02.2018	P2
9	10024	d-36	16	04.02.2018	P2
10	10025	s-55	5	05.02.2018	P2
11	10026	f-52	24	10.02.2018	P2
12	10026	f-93	4	10.02.2018	P2
13	10027	b-23	8	17.02.2018	P2
14	10027	d-12	3	17.02.2018	P2
15	10027	u-25	2	17.02.2018	P2
16	10027	f-54	15	17.02.2018	P2
17	10027	d-72	6	17.02.2018	P2
18	10027	s-91	12	17.02.2018	P2
19	10027	f-94	3	17.02.2018	P2
20	10027	d-33	4	17.02.2018	P2
21	10027	u-25	5	17.02.2018	P2

Bild 13.5 Die Tabelle ist nun abgegrenzt.

Damit ist die Transformation abgeschlossen. Sie haben die Tabelle aus dem Dateiinhalt isoliert, ohne wichtige Informationen zu verlieren und ohne feste Zeilenangaben zu verwenden. Die Abfrage ist daher flexibel, was die Position und Höhe der Tabelle angeht. Sie können es ausprobieren, indem Sie im Transformationsschritt *Quelle* den Pfad für die Beispieldatei *13-03-Export_Mrz.txt* eingeben. Auch hier werden die richtigen Zeilen eingelesen.

13.2 Unterschiedliche Tabellenzeilen vergleichen

Es wurde bereits gezeigt, wie neue Spalten an Tabellen angefügt werden, in denen Berechnungen mit Werten aus anderen Spalten angestellt werden. Doch was, wenn sich die Berechnung auf unterschiedliche Zeilen innerhalb der Tabelle beziehen soll?

Zeiterfassung			
Personal-Nr.:		Mitarbeiter:	
273		Schneider, Friedhelm	
Datum	Uhrzeit	Aktion	Pausen
03.07.2017	05:52 Uhr	kommt	
03.07.2017	09:00 Uhr	geht	0 Min.
06.07.2017	05:57 Uhr	kommt	
06.07.2017	14:01 Uhr	geht	30 Min.
07.07.2017	05:58 Uhr	kommt	
07.07.2017	14:05 Uhr	geht	30 Min.
10.07.2017	14:02 Uhr	kommt	
10.07.2017	22:01 Uhr	geht	30 Min.
11.07.2017	13:59 Uhr	kommt	
11.07.2017	22:03 Uhr	geht	30 Min.

Bild 13.6 Ausschnitt aus einer Zeiterfassungsliste.

Bild 13.6 zeigt einen Ausschnitt aus einer Beispieldatei, die Schichtzeiten mehrerer Mitarbeiter enthält. Das Ziel ist, eine zusammenhängende Tabelle mit den Arbeitsstunden der Mitarbeiter anhand ihrer jeweiligen Personalnummer zu erzeugen. Dabei stellen sich zwei Herausforderungen:

1. Die Personalnummer der jeweiligen Mitarbeiter steht über den einzelnen Tabellen. Das Wort „Personal-Nr“ steht aber in einer anderen Zeile, als die Nummer selbst. Die Nummer kann also nur anhand der vorhergehenden Zeile gefunden werden.
2. Die Zeiten für Schichtbeginn und Schichtende stehen in unterschiedlichen Zeilen. Um die Stundenzahl zu berechnen, muss also die Differenz aus untereinanderliegenden Werten ermittelt werden.

Wie schon im letzten Beispiel besteht der „Trick“ auch hier in der Verwendung einer Indexspalte. Sie ermöglicht es, eine bestimmte Zeilennummer aus der Tabelle auszuwählen.

- Erstellen Sie eine neue Abfrage auf die Excel-Datei *13-04-Zeiterfassung.xlsx*.
- Fügen Sie eine Indexspalte hinzu, indem Sie im Register **Spalte hinzufügen** auf **Indexspalte** klicken. Achten Sie darauf, dass der Index mit 0 beginnt.
- Ändern Sie den Namen des gerade erstellten Transformationsschritts in *neuIndex*. Wenn der Schrittname kein Leerzeichen enthält, lässt er sich leichter in Formeln verwenden (man spart sich das `#"..."`).

- Fügen Sie als Nächstes eine **Benutzerdefinierte Spalte** hinzu. Geben Sie als Spaltentitel *Personal-Nr.* ein und als Spaltenformel folgenden Code:

```
if [Column1] = "Personal-Nr.:" then
   neuIndex[Column1]{[Index]+1}
else
   null
```

	Column1	Column2	Column3	Column4	Index	Personal-Nr.
1	Zeiterfassung	*null*	*null*	*null*	*0*	*null*
2	*null*	*null*	*null*	*null*	*1*	*null*
3	Personal-Nr.:	*null*	Mitarbeiter:	*null*	*2*	*273*
4	*273*	*null*	Schneider, Friedhelm	*null*	*3*	*null*
5	*null*	*null*	*null*	*null*	*4*	*null*
6	Datum	Uhrzeit	Aktion	Pausen	*5*	*null*
7	03.07.2017	05:52 Uhr	kommt	*null*	*6*	*null*
8	03.07.2017	09:00 Uhr	geht	0 Min.	*7*	*null*
9	06.07.2017	05:57 Uhr	kommt	*null*	*8*	*null*
10	06.07.2017	14:01 Uhr	geht	30 Min.	*9*	*null*

Bild 13.7 Die Personalnummer wird über den Index eingefügt.

Der Transformationsschritt *neuIndex* wird hier verwendet, um einen Wert aus der entsprechenden Tabelle zu extrahieren. In den eckigen Klammern wird die gesuchte Spalte angegeben, in den geschweiften Klammern die Zeilennummer. Anstelle der Zeilennummer wird hier `[Index]+1` angegeben. Da `[Index]` keinen Schrittnamen vorangestellt hat, bezieht es sich auf die Spalte im aktuellen Schritt und da wir uns innerhalb der Formel für eine benutzerdefinierte Spalte befinden, bezieht es sich auf den Wert der jeweiligen Zeile. Die Zeilennummer wird um eins erhöht, um die darunterliegende Zeile auszulesen.

- Wenden Sie die Funktion **Ausfüllen – Nach unten** auf die neu hinzugefügte Spalte an.
- Die Kopf- und Leerzeilen können nun entfernt werden. Filtern Sie die zweite Spalte so, dass alle *Null*-Werte entfernt werden.
- Klicken Sie im **Start**-Register auf **Erste Zeile als Überschriften verwenden.**
- Durch den letzten Schritt gingen die Titel der letzten beiden Spalten verloren. Ändern Sie die Überschriften wieder auf *Index* und *Personal-Nr.*.
- Es müssen noch einige Zwischenüberschriften entfernt werden. Filtern Sie die Spalte *Pausen*, indem Sie dort das Wort *Pausen* entfernen.
- Entfernen Sie den Text „ Min.“ (mit Leerzeichen und Punkt) aus der Spalte *Pausen* mithilfe von **Werte ersetzen.**
- Ändern Sie die Datentypen der Spalten: *Datum* zu *Datum*, *Uhrzeit* zu *Zeit*, *Pausen* zu *ganze Zahl* und *Personal-Nr.* zu *Text*.

	Datum	Uhrzeit	Aktion	Pausen	Index	Personal-Nr.
1	03.07.2017	05:52:00	kommt	null	6	273
2	03.07.2017	09:00:00	geht	0	7	273
3	06.07.2017	05:57:00	kommt	null	8	273
4	06.07.2017	14:01:00	geht	30	9	273
5	07.07.2017	05:58:00	kommt	null	10	273
6	07.07.2017	14:05:00	geht	30	11	273
7	10.07.2017	14:02:00	kommt	null	12	273
8	10.07.2017	22:01:00	geht	30	13	273
9	11.07.2017	13:59:00	kommt	null	14	273
10	11.07.2017	22:03:00	geht	30	15	273

Bild 13.8 Die Tabelle ist nun abgegrenzt.

Nun, da die Tabelle ordentliche Überschriften und keine Leerräume mehr hat, kann die letzte Herausforderung in Angriff genommen werden: Anstelle der Uhrzeiten für Kommen und Gehen, soll die Anzahl der Stunden für jede Arbeitseinheit errechnet werden. Es muss also die Differenz aus den Zeiten *Kommt* und *Geht* abzüglich der Pausen-Minuten errechnet werden.

- Nachdem durch die Filteraktionen mehrere Zeilen entfernt wurden, ist die Indexspalte nicht mehr fortlaufend und dadurch unbrauchbar. Entfernen Sie sie und erstellen Sie eine neue Indexspalte.
- Ändern Sie den Namen des letzten Transformationsschritts zu *neuIndex2*.
- Fügen Sie eine neue **Benutzerdefinierte Spalte** hinzu. Geben Sie als Namen der neuen Spalte *Dauer mit Pause* ein. Die Spaltenformel lautet folgendermaßen:

```
if [Aktion] = "geht" then
   [Uhrzeit] - neuIndex2[Uhrzeit]{[Index]-1}
else
   null
```

- Ändern Sie den Spaltentyp in *Dauer*.
- Filtern Sie alle *Null*-Werte heraus.
- Entfernen Sie die Spalten *Uhrzeit*, *Aktion* und *Index*.
- Fügen Sie eine neue **Benutzerdefinierte Spalte** hinzu. Geben Sie den Namen *Dauer ohne Pause* und folgende Spaltenformel ein:

```
[Dauer mit Pause] - #duration(0, 0, [Pausen], 0)
```

Das Ergebnis kann sich sehen lassen: Anstelle der Uhrzeiten wird nun die Dauer jeder Schicht im Format Duration angezeigt.

	Datum	Pausen	Personal-Nr.	Dauer mit Pause	Dauer ohne Pause
1	03.07.2017	0	273	0.03:08:00	0.03:08:00
2	06.07.2017	30	273	0.08:04:00	0.07:34:00
3	07.07.2017	30	273	0.08:07:00	0.07:37:00
4	10.07.2017	30	273	0.07:59:00	0.07:29:00
5	11.07.2017	30	273	0.08:04:00	0.07:34:00
6	12.07.2017	30	273	0.07:56:00	0.07:26:00
7	13.07.2017	30	273	0.08:11:00	0.07:41:00
8	14.07.2017	30	273	0.07:59:00	0.07:29:00
9	17.07.2017	30	273	0.07:23:00	0.06:53:00
10	18.07.2017	30	273	0.08:04:00	0.07:34:00
11	19.07.2017	30	273	0.08:01:00	0.07:31:00
12	24.07.2017	30	273	0.08:00:00	0.07:30:00

Bild 13.9 Die finale Tabelle.

Wenn Sie anstelle des Formats Duration lieber die Stunden als Zahl anzeigen möchten, multiplizieren Sie die Dauer mit 24.

13.3 Abfragen mit Parametern

Abfragen mit Power Query sind, nachdem sie einmal erstellt wurden, kinderleicht zu bedienen: Im Prinzip muss man nur wissen, wie sie aktualisiert werden. Diese Einfachheit kann aber auch ein Nachteil sein, wenn man den Ablauf der Abfrage „von außen“ beeinflussen möchte, d. h. ohne den Abfrage-Code zu ändern.

Stellen Sie sich vor, Sie haben eine grandiose Abfrage auf eine Datei erstellt und möchten sie mit all Ihren Kollegen teilen. Nun möchten diese Kollegen aber unterschiedliche Quelldateien verwenden. Es gibt nun drei Möglichkeiten, um ihnen zu helfen: Sie zeigen jedem Kollegen, wie der entsprechende Transformationsschritt im Abfrage-Editor angepasst wird. Oder Sie bitten sie, die Quelldateien genauso zu benennen und an genau den Ordnerpfad zu kopieren, wie es in der Abfrage angegeben ist. Oder – und das ist vermutlich der Weg, mit dem Sie am wenigsten Verwirrung stiften – Sie fügen einen Platzhalter in den Abfragecode ein, dessen Wert relativ einfach angepasst werden kann.

13.3.1 Power-Query-Parameter

- Erstellen Sie als Erstes eine normale Abfrage auf die Datei *13-05-Pegel-Donau.xlsx*.
- Falls die Datentypen nicht richtig erkannt werden, stellen Sie sie entsprechend ein.

Der Abfragecode sollte nun in etwa so aussehen:

Listing 13.3 Der Abfragecode ohne Parameter.

```
let
    Quelle = Excel.Workbook(
         File.Contents("C:\Beispiele\13-05-Pegel-Donau.xlsx"), null, true),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
         [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes(
         #"Höher gestufte Header",{{"Datum", type date},
         {"Uhrzeit", type time}, {"Wasserstand cm über NN", Int64.Type}})
in
    #"Geänderter Typ"
```

Im ersten Transformationsschritt *Quelle* ist der Pfad der Datei enthalten, auf die Sie die Abfrage gemacht haben. Das ist der Teil, der angepasst werden muss, damit verschiedene Dateien an beliebigen Orten eingelesen werden können.

- Im **Start**-Register gibt es das Menü **Parameter verwalten**. Wählen Sie den Menüeintrag **Neuer Parameter**.
- Es erscheint ein Dialogfenster, in dem Sie die Eigenschaften Ihres Parameters eingeben können. Stellen Sie Folgendes ein:
  - *Name:* Dateipfad
  - *Beschreibung:* Pfad und Dateiname der Quelldatei
  - *Erforderlich:* Aktivieren (bewirkt, dass Nutzer eine Warnung erhalten, wenn das entsprechende Feld leer ist)
  - *Typ:* Text
  - *Vorgeschlagene Werte:* beliebiger Wert
  - *Aktueller Wert:* Geben Sie hier Ihren Dateipfad und Dateinamen ein.

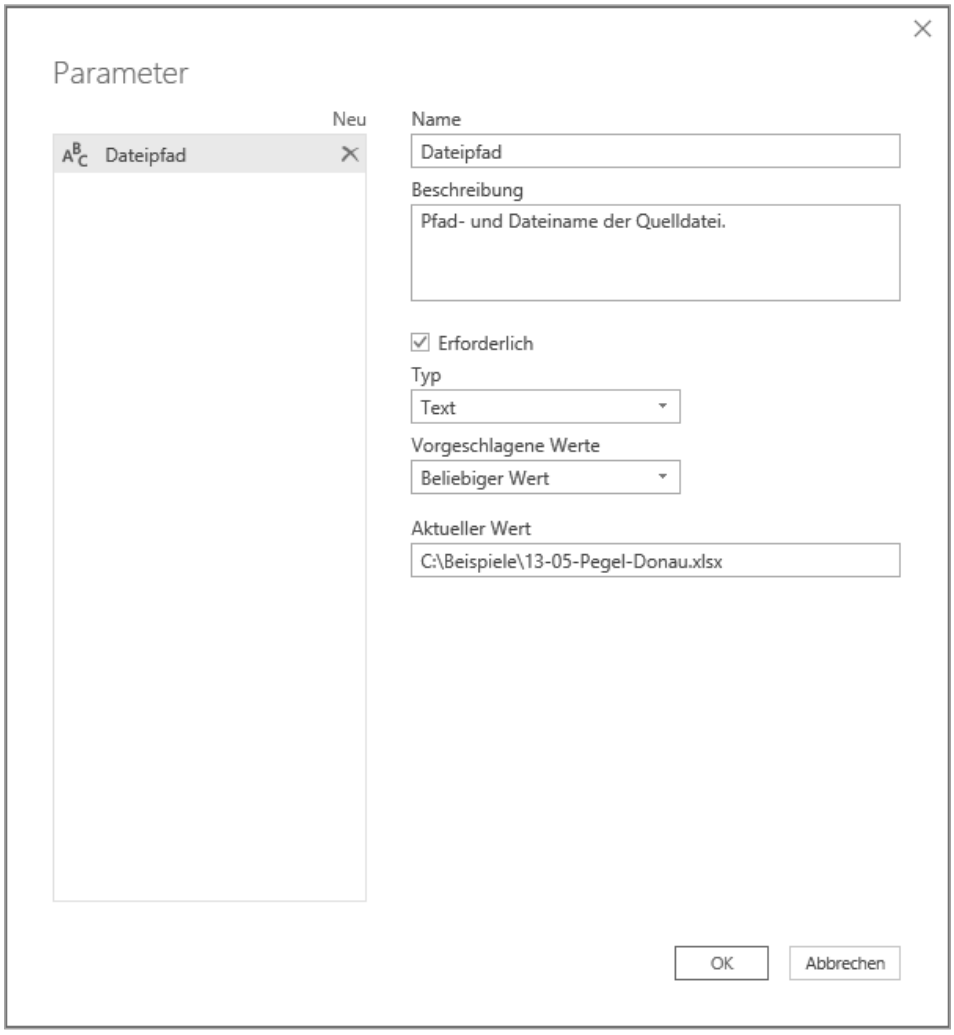

Bild 13.10 Das Dialogfenster zur Erstellung des Parameters.

- Bestätigen Sie mit **OK**.

Daraufhin wird der Parameter in der Liste der Abfragen angezeigt, wo er auch angepasst werden kann.

- Wählen Sie in der Abfragenliste wieder die ursprüngliche Abfrage und öffnen Sie den Erweiterten Editor.
- Ersetzen Sie im ersten Abfrageschritt den Dateipfad durch den Parameternamen. Der neue Abfragecode ist in Listing 13.4 zu sehen:

Listing 13.4 Der Abfragecode mit Parameter.

```
let
    Quelle = Excel.Workbook(
         File.Contents(Dateipfad), null, true),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
         [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes(
         #"Höher gestufte Header",{{"Datum", type date},
         {"Uhrzeit", type time}, {"Wasserstand cm über NN", Int64.Type}})
in
    #"Geänderter Typ"
```

- Geben Sie der Abfrage den Namen *AbfrageWasserstand* und schließen Sie sie ab.

Der Pfad der Quelldatei ist nun nicht fest im Abfragecode verankert, sondern nur als Parameterwert hinterlegt. Er kann ganz einfach geändert werden, ohne dass man sich mit Power Query auskennen muss.

EXCEL:

- Öffnen Sie das Menü für Abfragen und Verbindungen durch die entsprechende Schaltfläche im Register **Daten**.
- Durch Doppelklick auf den Eintrag für den Parameter *Dateipfad* gelangen Sie in den Abfrage-Editor. Hier können Sie den Parameterwert direkt ändern.

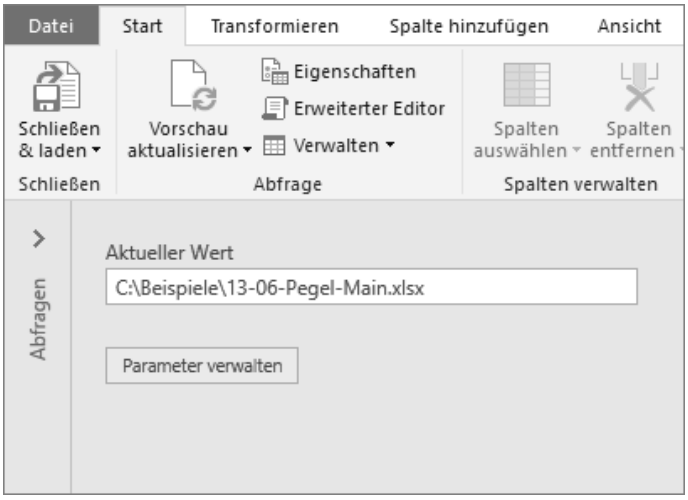

Bild 13.11 Ändern des Parameterwerts in Excel.

- Geben Sie nun den Dateipfad für die Beispieldatei *13-06-Pegel-Main.xlsx* ein.
- Bestätigen Sie mit **Schliessen & Laden**.
- Aktualisieren Sie die Abfrage. Es wird nun eine neue Datei eingelesen.

POWER BI:

In Power BI ist die Anpassung noch einfacher als in Excel. Sie müssen nicht einmal den Abfrage-Editor öffnen, um den Wert eines Parameters zu ändern.

- Klicken Sie im Hauptfenster (nicht im Abfrage-Editor) im Register **Start** auf die Schaltfläche **Abfragen bearbeiten** und wählen Sie den Menüeintrag **Parameter bearbeiten**.

Bild 13.12 In Power BI Desktop haben Sie im Hauptfenster Zugriff auf Parameter.

- Geben Sie den Dateipfad für die Beispieldatei *13-06-Pegel-Main.xlsx* ein.
- Aktualisieren Sie die Abfrage. Es wird nun eine neue Datei eingelesen.

13.3.2 Parameter aus anderen Quellen

Alternativ zum Erstellen eines Parameters können Sie auch Werte aus anderen Quellen einlesen, um damit Ihre Abfrage zu steuern. Eine einfache Excel-Tabelle kann beispielsweise dafür verwendet werden, den Dateipfad einer Quelldatei oder die Filteroptionen einer Tabelle einzustellen. Diese Technik ist für Excel-Nutzer besonders vorteilhaft, denn Sie können Ihre selbstdefinierten Parameter in der gleichen Datei speichern wie die dazugehörige Abfrage. Für Nutzer von Power BI ist dies zwar nicht möglich, die zugrunde liegende Technik funktioniert aber trotzdem.

Die Beispieldatei *13-07-Parameterabfrage.xlsx* enthält eine einfache Tabelle, in der Parameter und die zugehörigen Werte eingetragen sind (vgl. Bild 13.13). Diese Tabelle kann benutzt werden, um eine Abfrage auf eine Datei zu verändern.

	A	B	C
1			
2		Abfrage-Parameter	Wert
3		Dateipfad	C:\Beispiele\13-08-Einwohnerzahlen_1970.xlsx
4		Filter: Stadt	Köln
5			

Bild 13.13 Die Parameter liegen in einer Excel-Tabelle.

- Öffnen Sie die Datei *13-07-Parameterabfrage.xlsx*
- Tragen Sie in der rechten Spalte den Pfad ein, an dem Ihre Beispieldateien liegen.

EXCEL:

- Markieren Sie eine Zelle in dieser Tabelle und erstellen Sie eine neue Abfrage auf die Tabelle. Klicken Sie hierfür im Register *Daten* auf **Daten abrufen – Aus anderen Quellen – Aus Tabelle / Bereich**.
- Löschen Sie den automatisch eingefügten Transformationsschritt *Geänderter Typ*.
- Klicken Sie mit der rechten Maustaste auf den ersten Wert in der rechten Spalte und wählen Sie den Befehl **Drilldown ausführen**.
- Öffnen Sie den Erweiterten Editor.
- Fassen Sie die beiden Transformationsschritte zu einem Schritt zusammen. Listing 13.5 zeigt die so entstehende Anweisung.

Listing 13.5 Der Befehl zum Auslesen eines Werts aus der Tabelle.

```
Quelle = Excel.CurrentWorkbook(){[Name="tbl_Parameter"]}[Content]{0}[Wert]
```

- Nun wissen Sie, welcher Befehl nötig ist, um einen Wert aus der Parametertabelle zu extrahieren. Sie werden diesen Befehl später brauchen. Die Abfrage selbst ist nicht mehr notwendig. Beenden Sie den Abfrage-Editor, ohne zu speichern.
- Erstellen Sie nun eine Abfrage auf die Datei *13-12-Einwohnerzahlen_2010.xlsx* und bearbeiten Sie sie im Abfrage-Editor.
- Filtern Sie die Spalte *Stadt* nach einem einzelnen Wert. Welche Stadt Sie wählen, ist egal, weil sie später durch den Parameterwert ersetzt wird.
- Öffnen Sie den Erweiterten Editor. Hier werden nun die Teile des Codes, die flexibel sein sollen (Dateipfad und Stadt) an den entsprechenden Stellen eingesetzt.

  Der erste Transformationsschritt sieht zunächst so aus (wobei natürlich der Pfad auf Ihrem System unterschiedlich sein wird):

```
Quelle = Excel.Workbook(File.Contents(
    "C:\Beispiele\13-12-Einwohnerzahlen_2010.xlsx"), null, true),
```

- Anstelle des eingetragenen Dateipfads in Anführungszeichen fügen Sie den Befehl ein, der den Dateipfad aus der Parametertabelle liest (siehe oben). Der neue Befehl sieht somit folgendermaßen aus:

```
Quelle = Excel.Workbook(File.Contents(
 Excel.CurrentWorkbook(){[Name="tbl_Parameter"]}[Content]{0}[Wert]
 ), null, true),
```

- Setzen Sie ebenso für die herausgefilterte Stadt den entsprechenden Code-Baustein ein. Dabei müssen Sie nur beachten, dass nun die zweite Zeile aus der Tabelle ausgelesen wird. Die Zahl in Klammern muss somit nicht *0*, sondern *1* sein.

Der Abfragecode sollte am Ende so aussehen wie in Listing 13.6.

Listing 13.6 Der Abfragecode mit den eingesetzten Parameterabfragen.

```
let
    Quelle = Excel.Workbook(File.Contents(
        Excel.CurrentWorkbook(){[Name="tbl_Parameter"]}[Content]{0}[Wert]
        ), null, true),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
        [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes
        (#"Höher gestufte Header",{{"Stadt", type text},
        {"Einwohnerzahlen", Int64.Type}}),
    #"Gefilterte Zeilen" = Table.SelectRows(#"Geänderter Typ", each ([Stadt] =
        Excel.CurrentWorkbook(){[Name="tbl_Parameter"]}[Content]{1}[Wert]
        ))
in
    #"Gefilterte Zeilen"
```

- Schließen Sie die Abfrage nun ab. Wählen Sie dabei am besten **Schliessen & Laden in...** und legen Sie fest, dass die Abfragetabelle direkt unter der Parametertabelle angezeigt wird. So haben Sie alles im Blick.

Parameter:

Abfrage-Parameter	Wert
Dateipfad	C:\Beispiele\13-12-Einwohnerzahlen_2010.xlsx
Filter: Stadt	Bielefeld

Ausgabe:

Stadt	Einwohnerzahlen
Bielefeld	323270

Bild 13.14 Flexible Datei- und Filterwahl in einem Excel-Blatt.

- Testen Sie Ihre Abfrageparameter, indem Sie verschiedene Städte als Parameter eingeben und die Abfrage aktualisieren. Sofern die gesuchte Stadt in der Liste ist, sollte die entsprechende Einwohnerzahl erscheinen.
- Geben Sie unterschiedliche Dateipfade an, um die Einwohnerzahlen aus verschiedenen Jahren zu erhalten. Wählen Sie eine dieser Dateinamen (zusammen mit dem jeweiligen Pfad):
  - 13-08-Einwohnerzahlen_1970.xlsx
  - 13-09-Einwohnerzahlen_1980.xlsx
  - 13-10-Einwohnerzahlen_1990.xlsx
  - 13-11-Einwohnerzahlen_2000.xlsx
  - 13-12-Einwohnerzahlen_2010.xlsx

Mit ein wenig Excel-Bastelei können Sie Ihre Parametertabelle noch benutzerfreundlicher machen. Sie können zum Beispiel mithilfe der **Datenüberprüfung** auf dem Register **Daten** ein Dropdown einfügen, mit dem man die Stadt auswählen kann. Werfen Sie hierfür einen Blick auf die Datei *13-13-Parameterabfrage_Lösung.xlsx*.

POWER BI:

In Power BI können Sie ebenfalls Abfragen erstellen, deren Parameter in einer Excel-Tabelle liegen. Im Unterschied zum obigen Beispiel wäre die Parametereingabe dann jedoch in einer anderen Datei als die Abfrage, was eher selten zweckmäßig ist.

Nichtsdestotrotz ist es natürlich möglich. Das Vorgehen ist das gleiche wie oben beschrieben, mit dem Unterschied, dass die Funktion *Excel.CurrentWorkbook* in Power BI kein Ergebnis liefert. Stattdessen müssen Sie die Excel-Datei über die Funktion *Excel.Workbook* einlesen. Der Code-Baustein, der den ersten Parameter aus der Tabelle zurückgibt, lautet folgendermaßen:

```
Excel.Workbook(File.Contents
    ("C:\Beispiele\13-07-Parameterabfrage.xlsx"), null, true)
    {[Item="tbl_Parameter",Kind="Table"]}[Data]{0}[Wert]
```

Dabei müssen Sie natürlich die Pfadangabe in der zweiten Zeile durch den richtigen Pfad auf Ihrem System ersetzen.

Der komplette Code für Power BI wird in Listing 13.7 gezeigt.

Listing 13.7 Der Abfragecode mit Parametern aus einer externen Excel-Datei.

```
let
    Quelle = Excel.Workbook(File.Contents(
        Excel.Workbook(File.Contents
        ("C:\Beispiele\13-07-Parameterabfrage.xlsx"), null, true)
        {[Item="tbl_Parameter",Kind="Table"]}[Data]{0}[Wert]
        ), null, true),
    Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
    #"Höher gestufte Header" = Table.PromoteHeaders(Tabelle1_Sheet,
        [PromoteAllScalars=true]),
    #"Geänderter Typ" = Table.TransformColumnTypes
        (#"Höher gestufte Header",{{"Stadt", type text},
        {"Einwohnerzahlen", Int64.Type}}),
    #"Gefilterte Zeilen" = Table.SelectRows(#"Geänderter Typ", each ([Stadt] =
        Excel.Workbook(File.Contents
        ("C:\Beispiele\13-07-Parameterabfrage.xlsx"), null, true)
        {[Item="tbl_Parameter",Kind="Table"]}[Data]{1}[Wert]
        ))
in
    #"Gefilterte Zeilen"
```

Index

A

Ablaufverfolgung 328
Access 49
Add-In 3
Aktualisieren 8
– automatisch (Excel) 10
Anfügen 59
Anführungszeichen 105
Ausfüllen (nach oben/unten) 27
Azure 49

B

Bearbeitungsleiste 83
Bedingte Spalte 73
Beispieldateien 2
Benutzerdefinierte Spalte 77
Binary 108

C

Combiner-Funktionen 285
Comparer-Funktionen 282
Connector 307
CSV 14

D

Data Connector 307
Date 102
Date-Funktionen 254
Datenbank 49
Datenmodell 7
Datensatz 110
Datensatzfunktionen 175
Datenschutzstufen 321
Datentyp 99, 114
– Spalte 18
– Übersicht 20
Datetime 103
DateTime-Funktionen 267
Datetimezone 103
DateTimeZone-Funktionen 272
Datum 102
Datumstransformationen 36
Dauer 104
DB2 49
DirectQuery 9
Duration 104
Duration-Funktionen 277

E

Each 294
Entpivotieren 30, 193
Error 138, 141
Ersetzen 18
Erweiterter Editor 84
Escape-Sequenz 106
Euro 106

F

False 99
Fehler 89, 137
Filter 16
Formula.Firewall-Fehler 322

Funktionen
– Datentyp 116
– Erstellen 293
– Übersicht 145
Funktionsbeschreibung 116, 304
Funktionsbibliothek 148

G

Geänderter Typ 90
Glätten 34
Großschreiben 34
Gruppieren 38

H

Header 17
Hexadezimalzahl 101

I

If 79
Index 125

K

Kommentar 91

L

Laden in (Excel) 6, 60
Leere Abfrage 85
let...in 86, 299
Liste 108
Listenfunktionen 150
Logical 99
Logical-Funktionen 253

M

M
– Groß- & Kleinschreibung 81
– Grundstruktur 86
– Komma 87, 91
– Leerzeichen 78
– Operatoren 79
– Schrittnamen 87
– Variablen 87
– Zeilenumbrüche 80
Makro 311
Metadata 304
MSDN 146

N

Navigator-Fenster 6
Null 99
Number 101

O

Oracle 49
Ordnerabfrage 67

P

Parameter 128
Pivotieren 192
Pivot-Tabelle 26
– Entpivotieren 30

Q

Query Folding 55, 321, 327

R

Record 110
Record-Funktionen 175
Rekursive Funktionen 302
Replacer-Funktionen 284
Runden 21, 242

S

Schlüssel 53, 223
Schnelles Laden (Excel) 326
shared 148
Sicherheitsstufen 321
Sortieren 207
Spalte
– Löschen 17

– Teilen 31
– Typ 18
– Verschieben 43
– Zusammenführen/Verbinden 29
Splitter-Funktionen 288
SQL Server 49
Suffix 35
Switch-Funktion 299
Syntax 145

T

Tab 106
Tabelle 111
Tabellenfunktionen 182
Text 105
Textfunktionen 224
Texttransformationen 33
Time 102
Timefunktionen 280
Trace-Datei 328
Transformationsschritte
– Anzeigen 15
– Löschen 15
– Umbenennen 23
– Zusammenfassen 90
Transponieren 28
True 99
try...otherwise 140
Type 114

U

Überschrift 17
Unendlich 101, 106
Unicode 107
Ursprüngliche Spaltennamen als Präfix verwenden 66

V

VBA 311
Verbindung erstellen 7, 60
Vertauschen 28
Visual Studio 93

W

Web-Abfrage 41

Z

Zahl 101
Zahlenfunktionen 238
Zahlentransformationen 37
Zeilenumbruch 107
Zeit 102
Zeitfunktionen 280
Zusammenführen 59, 62