M

Daten abfragen und verarbeiten mit Excel und Power BI

10 Abfragecode bearbeiten mit M

M ist eine leistungsfähige Programmiersprache, genauer gesagt eine Formelsprache, die Microsoft eigens für die Abfragetechnologie Power Query entwickelt hat. Der offizielle Name ist *Power Query Formula Language*, obwohl selbst Microsoft auch den inoffiziellen Terminus *M* zu bevorzugen scheint.

Viele Nutzer, die bereits erfolgreich Power-Query-Abfragen erstellt haben, schrecken davor zurück, sich mit M auseinanderzusetzen. Vielleicht hat man schon mit Mühe diverse Excel-Formeln oder sogar die eine oder andere Programmiersprache gelernt und hat keine Lust, sich noch eine zusätzliche komplexe Sprachlogik anzueignen.

Nun, die gute Nachricht ist: Wenn Sie bereits erfolgreich Abfragen im Abfrage-Editor erstellt haben, haben Sie bereits in M programmiert. Jeder Transformationsschritt, den Sie erzeugen, erzeugt eigentlich M-Code. Dieses Prinzip erleichtert das Erlernen von M ungemein. Sie können Abfrageschritte über die Benutzeroberfläche erstellen und anschließend den Code betrachten. Sie werden sehen, dass er so relativ einfach zu verstehen ist. Der Anteil an Code, den Sie selbst schreiben, ist üblicherweise ziemlich gering. Den größten Teil Ihrer Abfragen können Sie weiterhin über die Benutzeroberfläche erstellen und nur dort Ihre M-Kenntnisse einsetzen, wo die vorgefertigten Werkzeuge an ihre Grenzen stoßen.

10.1 Die Bearbeitungsleiste

In der Bearbeitungsleiste sehen Sie jederzeit, welcher Code für den aktuell ausgewählten Transformationsschritt hinterlegt ist. Aktivieren Sie im Abfrage-Editor auf dem Register **Ansicht** die Option **Bearbeitungsleiste**, damit sie immer angezeigt wird.

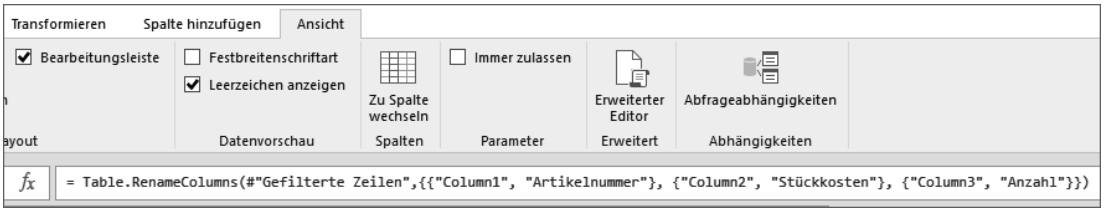


Bild 10.1 Die Bearbeitungsleiste zeigt den M-Code des aktiven Schritts an.

Mithilfe der Bearbeitungsleiste können die erstellten Transformationen überprüft und auch angepasst werden. Dazu ändern Sie einfach den Text und bestätigen mit der **Enter**-Taste. Sie können auch die Schaltflächen auf der linken Seite der Bearbeitungsleiste zum Bestätigen bzw. Abbrechen nutzen. Beachten Sie, dass Formeln in der Bearbeitungsleiste immer mit einem „=“ beginnen.

Ein Blick auf die Formel in der Bearbeitungsleiste lohnt sich fast immer. Sie müssen den M-Code nicht immer komplett verstehen, um etwaige Fehler oder Verbesserungsmöglichkeiten zu erkennen.

10.2 Das Editor-Fenster

In der Liste *Angewendete Schritte* können Sie einzelne Schritte auswählen und den zugehörigen Code in der Bearbeitungsleiste betrachten. Sie können aber auch den gesamten M-Code einer Abfrage in voller Pracht begutachten und verändern. Öffnen Sie hierfür den Editor über die Schaltfläche **Erweiterter Editor**. Sie finden sie sowohl im Register **Start** als auch unter **Ansicht**.

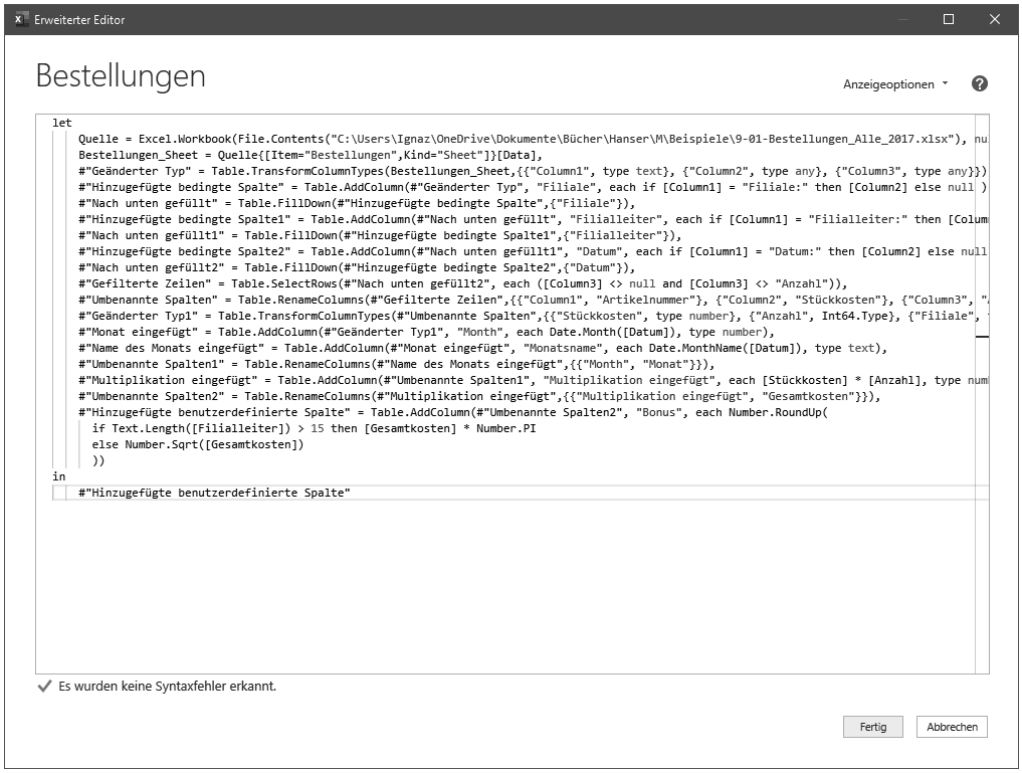


Bild 10.2 Der erweiterte Editor zeigt den gesamten Abfrage-Code.

Im Editor-Fenster können Sie den M-Code wie in einem Texteditor anpassen. Falls der Code Syntaxfehler enthält, erscheint eine entsprechende Meldung unterhalb des Textfelds. Sie haben dann die Möglichkeit, auf *Fehler anzeigen* zu klicken, woraufhin die fehlerhafte Stelle hervorgehoben wird.

Ähnlich wie in der Bearbeitungsleiste erhalten Sie auch hier Textvorschläge während der Eingabe. Außerdem werden bestimmte Wörter farblich hervorgehoben: M-Schlüsselwörter sind blau, Zahlen und Datentypen sind grün und Texte in Anführungszeichen sind orange. Sehr nützlich ist das Hervorheben von zusammengehörenden Klammern oder gleichen Wörtern. Wenn Sie zum Beispiel den Cursor auf eine öffnende Klammer setzen, wird die entsprechende schließende Klammer hervorgehoben. Wenn der Cursor in einem Spaltentitel steht, wird dieser Spaltentitel im ganzen Code hervorgehoben, sooft er vorkommt.

10.3 Erstellen einer leeren Abfrage

Bevor wir uns den Grundlagen der M-Sprache zuwenden, bekommen Sie die Gelegenheit, eine Abfrage nur anhand des Code-Textes zu erstellen. Anstatt beim Erstellen der Abfrage schon anzugeben, zu welcher Quelle die Verbindung hergestellt werden soll, erstellen Sie einfach eine leere Abfrage. Auf diese Weise können Sie zum Beispiel Abfrage-Code zwischen verschiedenen Dokumenten übertragen. Beispiel-Code aus einem Buch (wie diesem) oder aus dem Web kann so ebenfalls ohne große Umstände in eine neue Abfrage eingefügt werden.

EXCEL:

- Öffnen Sie eine neue Arbeitsmappe und klicken Sie im Register **Daten** auf **Daten abrufen – Aus anderen Quellen – Leere Abfrage**.

POWER BI:

- Klicken Sie auf **Daten abrufen** und wählen Sie in der Rubrik **Sonstige** den Eintrag **Leere Abfrage**.
- Klicken Sie auf **Verbinden**, um zum Abfrage-Editor zu gelangen.

Da keine Verbindung zu irgendeiner Datenquelle hergestellt wurde, zeigt der Abfrage-Editor logischerweise keine Tabelle als Vorschau. Wie immer gibt es aber schon den ersten Transformationsschritt namens *Quelle*. In der Bearbeitungsleiste sehen Sie aber, dass es keinen Code dazu gibt.

- Öffnen Sie den erweiterten Editor.

Im Editor-Fenster sehen Sie die Minimalform einer Abfrage: vier Codezeilen, bestehend aus den Ausdrücken `let` und `in` sowie dem einen vorhandenen Transformationsschritt *Quelle* (vgl. Bild 10.3).

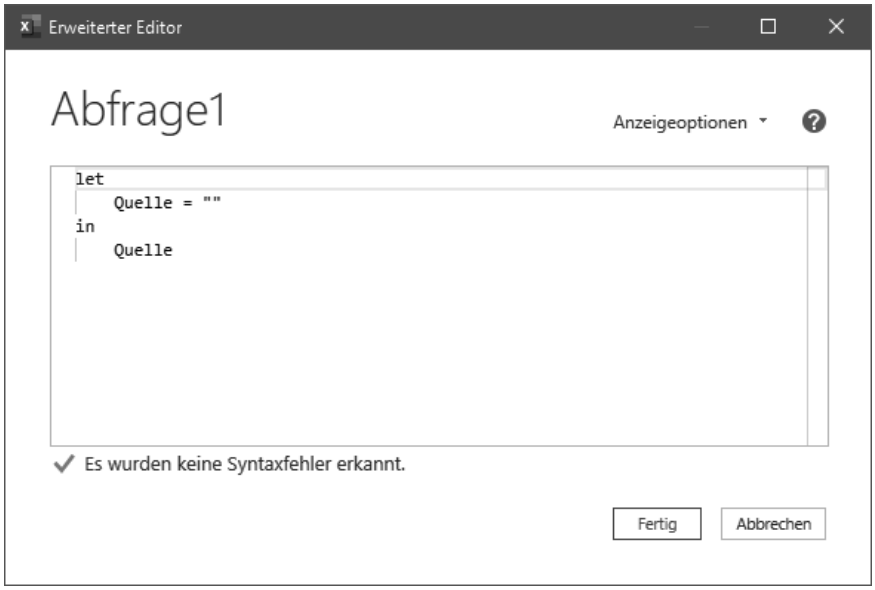


Bild 10.3 Der Code einer leeren Abfrage.

10.4 Die Grundstruktur des Abfrage-Codes

Das Grundprinzip von Power Query bzw. der Abfragesprache M basiert auf Ausdrücken und Rückgabewerten. Ein Ausdruck kann eine Berechnung sein, eine M-Funktion oder eine Kombination aus beidem. Rückgabewerte können einzelne Zahlen, Texte oder auch ganze Tabellen sein. Im Grunde ist eine Abfrage auch ein Ausdruck, der einen Wert zurückgibt. In den meisten Fällen ist der Rückgabewert eine Tabelle.

Da bei einer Abfrage oft mehrere Schritte nötig sind, um zum gewünschten Rückgabewert zu gelangen, gibt es die Anweisung `let ... in`. Dadurch wird ein einzelner Ausdruck aufgespaltet in Unterausdrücke bzw. -schritte. Betrachten Sie den Beispielcode in Listing 10.1.

Listing 10.1 Eine einfache let-Anweisung.

```
let
   Schritt1 = "Guten",
   Schritt2 = Schritt1 & " Tag",
   Schritt3 = Schritt2 & " Ignaz!"
in
   Schritt3
```

Zwischen `let` und `in` können Sie beliebig viele Zwischenschritte einfügen. Wichtig ist, dass jeder Schritt einen eindeutigen Namen hat, gefolgt von einem „=“ und dem Ausdruck, der ausgewertet werden soll. Wie Sie in *Schritt2* und *Schritt3* sehen, kann der Ausdruck wiede-

rum den Namen eines anderen Schritts enthalten. Üblicherweise ist das ein vorangegangener Schritt, was aber nicht zwingend notwendig ist.

Alle Schritte mit Ausnahme des letzten werden mit einem Komma abgeschlossen. Nach dem letzten Schritt folgt die Anweisung `in`. Daraufhin wird der Rückgabewert der `let...in`-Anweisung festgelegt, was üblicherweise (aber nicht zwingend) der Name des letzten Schritts ist.

- Geben Sie den Code aus Listing 10.1 im Editor-Fenster ein. Wenn Sie nicht auch Ignaz heißen, geben Sie stattdessen Ihren eigenen Namen ein.

Beachten Sie immer folgende Punkte beim Schreiben von M-Code:

- Groß- und Kleinschreibung wird unterschieden. „schritt1" wäre ein anderer Schritt als „Schritt1".
- Textwerte müssen immer in Anführungszeichen stehen.
- Alle Transformationsschritte werden mit einem Komma abgeschlossen – nur nicht der letzte.

- Unterhalb des Eingabefelds wird angezeigt, ob Ihr Code noch Fehler enthält. Stellen Sie sicher, dass keine Syntaxfehler enthalten sind.
- Bestätigen Sie den neuen Code mit der Schaltfläche **Fertig**.

Als Rückgabewert Ihrer Abfrage sollten Sie einen kurzen Begrüßungstext wie in Bild 10.4 sehen.

Bild 10.4
Das Abfrageergebnis muss nicht immer eine Tabelle sein.

10.5 Schritte und Schrittnamen verstehen

Die beste Möglichkeit für das Kennenlernen von M ist, eine Abfrage über die Benutzeroberfläche zu erstellen und dann den so erzeugten M-Code zu studieren. Im folgenden Beispiel sehen Sie auch, warum es immer nützlich ist, den M-Code im Auge zu behalten.

- Erstellen Sie eine Abfrage auf die Excel-Datei *10-01-Zahlungsziele.xlsx*.
- Wählen Sie *Tabelle1* aus und klicken Sie auf **Bearbeiten**, um zum Abfrage-Editor zu gelangen.

Werfen wir zunächst einen Blick auf die Schritte, die Power Query bereits automatisch erstellt hat.

1. *Quelle:* Die Verbindung zur Quelldatei wurde hergestellt.
2. *Navigation:* Das Arbeitsblatt *Tabelle1* wurde ausgewählt.
3. *Höher gestufte Header:* Die Spaltenüberschriften wurden als Überschriften erkannt.
4. *Geänderter Typ:* Die Datentypen wurden erkannt und eingestellt.

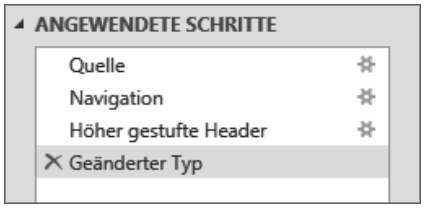


Bild 10.5
Vier Transformationsschritte sind bereits vorhanden.

- Nehmen wir an, für Ihre Auswertung benötigen Sie nur die Spalten *Kunde* und *Netto-Betrag*. Markieren Sie also diese beiden Spalten mit gedrückter **Strg**-Taste, machen Sie einen Rechtsklick auf einen der Spaltentitel und wählen Sie im Kontextmenü **Andere Spalten entfernen**.
- Sortieren Sie die Spalte *Netto-Betrag* absteigend, sodass die größten Beträge zuerst erscheinen.

Betrachten Sie nun den M-Code der eben erstellten Abfrage.

Listing 10.2 Der M-Code der Abfrage.

```
let
   Quelle = Excel.Workbook
      (File.Contents("C:\Beispiele\10-01-Zahlungsziele.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",{{"Auftragsnummer", type text}, {"Kunde",
      type text}, {"Rechnungs-Datum", type date}, {"Zahlungs-Ziel", Int64.Type},
      {"Zahlung", type date}, {"Netto-Betrag", Int64.Type}, {"Mwst", type number},
      {"Gesamt-Betrag", type number}, {"Bemerkung A. Friedrich 18.04.2018", type
      text}}),
   #"Andere entfernte Spalten" = Table.SelectColumns(#"Geänderter Typ",
      {"Kunde", "Netto-Betrag"}),
   #"Sortierte Zeilen" = Table.Sort(#"Andere entfernte Spalten",
      {{"Netto-Betrag", Order.Descending}})
in
   #"Sortierte Zeilen"
```

Normalerweise nimmt jeder Transformationsschritt eine Zeile ein. Die Zeilenumbrüche wurden hier nur für das Buchlayout eingefügt. Der Code würde aber auch in dieser Form funktionieren.

Sie müssen den Abfrage-Code nicht komplett verstehen, Sie wissen ja bereits, was die einzelnen Schritte bewirken. Sie erkennen jedoch auch hier das Schema, das in Abschnitt 10.4 skizziert wurde: Zwischen `let` und `in` sind die einzelnen Transformationsschritte aufgelistet. Nach jedem Schrittnamen folgt eine Zuweisung mit dem Gleichheitszeichen. Eine Besonderheit fällt jedoch bei den Schrittbezeichnungen auf:

Wenn ein Transformationsschritt mit mehr als einem Wort benannt wurde, wird der Schrittname im M-Code in Anführungszeichen gestellt und mit dem Raute-Symbol markiert. Die ersten beiden Schritte werden daher ohne besondere Kennzeichnung geschrieben (`Quelle`), wohingegen die längeren Schrittnamen etwas aufwendiger markiert sind (`#"Höher gestufte Header"`).

Beachten Sie, dass jeder Schrittname auch in der Zuweisung des darauffolgenden Schritts vorkommt. Um den Code etwas übersichtlicher zu gestalten, kann es sinnvoll sein, Leerzeichen in den Bezeichnungen zu vermeiden. Sie können die Bezeichnungen jederzeit im M-Code ändern oder sie alternativ in der Liste **Angewendete Schritte** umbenennen.

- Ersetzen Sie alle Leerzeichen in den Schrittbezeichnungen durch einen Unterstrich („_“). Klicken Sie dazu in der Liste *Angewendete Schritte* mit der rechten Maustaste auf die Bezeichnung und wählen Sie **Umbenennen**. Auf diese Weise wird der gesamte Code nach dem jeweiligen Schrittnamen durchsucht und der Name bei jedem Vorkommen ersetzt.

10.6 Fehler im M-Code vermeiden

Nachdem nun die ganzen Rauten und Anführungszeichen weggefallen sind, ist der Code etwas angenehmer zu lesen. Betrachten Sie jetzt den automatisch eingefügten Schritt *Geänderter_Typ*. Mithilfe der M-Funktion *Table.TransformColumnTypes* werden dort die Datentypen aller Spalten angepasst. Im darauffolgenden Schritt werden jedoch alle Spalten bis auf zwei entfernt. Die Auflistung aller Spalten bei der Typänderung ist also völlig unnötig.

Aber nicht nur das: In der Beispieltabelle findet sich die Spalte namens *Bemerkung A. Friedrich 18.04.2018*. Hier hat scheinbar ein Kollege eine Kommentarspalte eingefügt und in gutem Willen noch im Spaltentitel seinen Namen und das Datum angegeben. Es handelt sich also offensichtlich um eine Spalte, bei der Sie nicht davon ausgehen können, dass sie bei einer künftigen Aktualisierung der Abfrage noch genauso vorhanden ist.

- Schließen Sie die Abfrage ab und öffnen Sie die Beispieldatei in Excel.
- Ändern Sie den Spaltentitel der letzten Spalte in „Bemerkung Neu“.
- Speichern Sie die Datei.

Wenn Sie Ihre Abfrage nun aktualisieren, werden Sie feststellen, dass sie nicht mehr funktioniert – und das, obwohl die fragliche Spalte für die Auswertung gar nicht relevant ist.

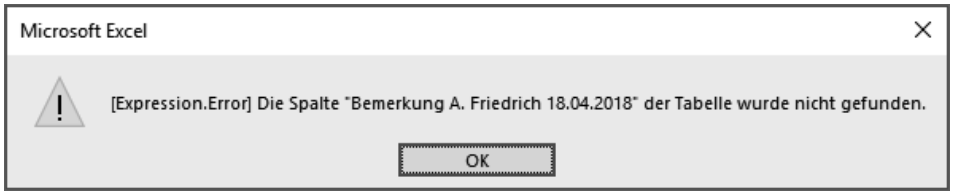


Bild 10.6 Der geänderte Spaltentitel erzeugt einen Fehler.

Dieses Beispiel zeigt, wie hilfreich es ist, den erstellten M-Code zu überprüfen, auch wenn die Abfrage über die Benutzeroberfläche des Abfrage-Editors erstellt wird. Um den Fehler zu vermeiden, achten Sie darauf, dass die problematische Spalte nicht im Code vorkommt.

- Entfernen Sie den Transformationsschritt *Geänderter_Typ* aus der Liste der angewendeten Schritte.
- Markieren Sie den letzten Schritt in der Liste und erstellen Sie einen neuen Transformationsschritt, indem Sie die Datentypen der verbleibenden Spalten anpassen.

Bei der Bestimmung der Datentypen ist es eine gute Angewohnheit, diesen Schritt so spät wie möglich durchzuführen – also wenn nötig, sobald Sie eine typspezifische Funktion anwenden oder ganz am Ende der Abfrage, wenn alle Spalten in ihrer finalen Form vorliegen.

Den automatisch eingefügten Schritt *Geänderter Typ* können Sie in vielen Fällen sofort löschen.

Es gibt sogar auch eine Einstellung dafür. Wenn Sie die automatische Typerkennung generell deaktivieren wollen, ändern Sie die Option unter **Datei – Optionen und Einstellungen – Abfrageoptionen – Daten laden**.

10.7 Schritte zusammenfassen

In den meisten Fällen enthalten Transformationsschritte eine Referenz zu einem anderen Transformationsschritt. Betrachten Sie zum Beispiel den Code-Ausschnitt in Listing 10.3. Sie stellen fest, dass der Schrittname *Höher_gestufte_Header* im darauffolgenden Schritt vorkommt.

Listing 10.3 Zwei Transformationsschritte aus dem Abfrage-Code.

```
Höher_gestufte_Header = Table.PromoteHeaders(Tabelle1_Sheet,
  [PromoteAllScalars=true]),
Andere_entfernte_Spalten = Table.SelectColumns(Höher_gestufte_Header,
  {"Kunde", "Netto-Betrag"}),
```

Vereinfacht kann der obenstehende Code folgendermaßen gelesen werden:

- *Höher_gestufte_Header:* Nehme den Rückgabewert des Transformationsschritts *Tabelle1_Sheet* und übernehme die erste Zeile als Überschriften.
- *Andere_entfernte_Spalten:* Nehme die Tabelle, die im Schritt *Höher_gestufte_Header* erzeugt wurde, und reduziere sie auf die Spalten *Kunde* und *Netto-Betrag*.

Sie erinnern sich, dass der Transformationsschritt *Andere_entfernte_Spalten* erstellt wurde, indem über das Kontextmenü im Abfrage-Editor alle Spalten außer *Kunde* und *Netto-Betrag* entfernt wurden. Im M-Code geschieht dies über die M-Funktion *Table.SelectColumns*. Als erstes Argument erhält diese Funktion den Namen des vorhergehenden Schritts *Höher_gestufte_Header*.

Wenn Sie den M-Code verkürzen wollen, können Sie die beiden Schritte aber auch zu einem einzigen Schritt zusammenfassen. Hierfür wird anstatt des Schrittnamens „Höher_gestufte_Header“ einfach der Code-Teil eingetragen, der durch diesen repräsentiert wird.

Bei den vielen Kommas kann man leicht den Überblick verlieren, besonders, wenn wie hier (aus Layoutgründen) noch Zeilenumbrüche eingefügt wurden. Merken Sie sich folgende Faustregel: Der Code eines Transformationsschritts beginnt nach dem Gleichheitszeichen und endet vor dem abschließenden Komma. Kommas, die innerhalb von Klammern stehen, gehören jedoch noch zum Transformationsschritt. ■

Listing 10.4 Die zwei Schritte wurden zu einem zusammengefasst.

```
Andere_entfernte_Spalten = Table.SelectColumns(Table.PromoteHeaders(
  Tabelle1_Sheet, [PromoteAllScalars=true]), {"Kunde", "Netto-Betrag"}),
```

Durch das Zusammenfassen von Schritten können Sie Ihren Code verkürzen und dadurch übersichtlicher gestalten. Für die Programmausführung macht es keinen Unterschied, ob die Befehle in einem Schritt abgearbeitet oder auf mehrere Schritte aufgeteilt werden. Theoretisch könnten Sie sogar die ganze Abfrage in einem Schritt schreiben. Das würde allerdings die Lesbarkeit des Codes ziemlich erschweren.

Letztendlich ist es Geschmackssache, welche Stellen Sie lieber zusammenfassen und wo Sie lieber Schritt für Schritt vorgehen.

10.8 Kommentare

Jeder Programmierer kennt das Phänomen, dass man seinen eigenen Code betrachtet und nicht mehr weiß, was man da genau gemacht hat. Noch verwirrender ist der Versuch, Code von jemand anderem zu verstehen.

Kommentare sind daher ein unerlässliches Werkzeug in fast jeder Programmiersprache. Durch das Einfügen von kurzen Beschreibungen, Zwischenüberschriften und Erläuterungen machen Sie Ihren Code für sich selbst und für andere verständlich.

In M gibt es zwei Möglichkeiten, Kommentare einzufügen: Kommentare, die sich über mehrere Zeilen erstrecken, beginnen mit /* und enden mit */. Einzeilige Kommentare beginnen mit //. Listing 10.5 zeigt den Code aus dem letzten Beispiel mit eingefügten Kommentaren.

Listing 10.5 M-Code mit Kommentaren.

```
/* Die Abfrage lädt die Zahlungen aller Kunden aus der Datei "Zahlungsziele".
Die Liste ist nach Betrag sortiert.
(Dieser Text ist ein Kommentar über mehrere Zeilen, der für die Funktion der Abfrage
keine Bedeutung hat.) */

let
   //Verbindung zur Datenquelle (<-- Einzeiliger Kommentar)
   Quelle = Excel.Workbook
      (File.Contents("C:\Beispiele\10-01-Zahlungsziele.xlsx"), null, true),
   Tabelle1_Sheet = Quelle{[Item="Tabelle1",Kind="Sheet"]}[Data],
   Höher_gestufte_Header = Table.PromoteHeaders(Tabelle1_Sheet,
      [PromoteAllScalars=true]),
   //Auswahl der Spalten "Kunde" und "Netto-Betrag"
   Andere_entfernte_Spalten = Table.SelectColumns(Höher_gestufte_Header,
      {"Kunde", "Netto-Betrag"}),
   //Sortierung
   Sortierte_Zeilen = Table.Sort(Andere_entfernte_Spalten,
      {{"Netto-Betrag", Order.Descending}}), //Größter Betrag zuerst
   Geänderter_Typ = Table.TransformColumnTypes(Sortierte_Zeilen,{{"Kunde",
      type text}, {"Netto-Betrag", type number}})
in
   Geänderter_Typ
```

Die Kommentare im M-Code sind leider nur im erweiterten Editor vollständig sichtbar. Wenn Sie Code in der Bearbeitungsleiste bearbeiten, können Sie dort zwar auch Kommentare einfügen. Wenn Sie aber einen anderen Transformationsschritt auswählen und zurückkehren, werden die Kommentare am Anfang oder am Ende des Befehls nicht angezeigt. Kommentare, die innerhalb eines Befehls stehen, werden kurioserweise auch in der Bearbeitungsleiste gezeigt.

Listing 10.6 Nicht alle Kommentare erscheinen in der Bearbeitungsleiste.

```
 Sortierte_Zeilen = Table.Sort(Andere_entfernte_Spalten, /*Dieser Kommentar wird in
der Bearbeitungsleiste sichtbar*/ {{"Netto-Betrag", Order.Descending}}), /*dieser
nicht*/
```

Wenn Sie Kommentare über oder vor Transformationsschritten einfügen, werden diese auch angezeigt, wenn Sie mit der Maus auf einen Schrittnamen unter *Angewendete Schritte* zeigen. Ein kleines i-Symbol zeigt an, wo sich ein Kommentar verbirgt.

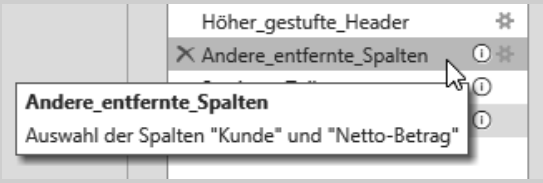


Bild 10.7
Ein Info-Feld zeigt Schrittname und Kommentar.

11 Werte und Datentypen

Bei Werten in Power Query unterscheidet man verschiedene Datentypen, wobei jeder Datentyp bestimmte Eigenheiten bei der Erzeugung oder Berechnung von Werten mit sich bringt. Einige Datentypen kennen Sie bereits vom Zuweisen eines Spaltentyps (vgl. Kapitel 3.7).

Dieses Kapitel gibt einen detaillierten Überblick über alle existierenden Datentypen, von einfachen Datentypen, die einzelne Werte enthalten, über übergeordnete Datentypen, die mehrere Einzelwerte zusammenfassen, bis hin zu speziellen Datentypen, zu denen auch die Funktionen zählen.

11.1 Einfache Datentypen

11.1.1 Null

Ein Null-Wert bezeichnet genau genommen die Abwesenheit eines Werts und sollte nicht mit der Zahl *0* verwechselt werden. Wenn Sie eine Tabelle mit leeren Zellen einlesen, haben diese beispielsweise den Wert Null. Null wird immer mit dem Schlüsselwort *null* gekennzeichnet. Sie können dieses Wort auch benutzen, wenn Sie leere Zellen suchen oder ersetzen wollen.

11.1.2 Logical (true/false)

Daten vom Datentyp *Logical* können genau zwei mögliche Werte annehmen: *true* oder *false*. Üblicherweise ist ein derartiger Wahrheitswert das Ergebnis einer Prüfung. Wenn Sie zum Beispiel zwei Werte mithilfe eines Vergleichsoperators (>, <, =) miteinander vergleichen, erhalten Sie als Rückgabewert entweder *true* oder *false*.

Geben Sie zum Testen folgende Prüfungen in die Bearbeitungsleiste ein und bestätigen Sie mit **Enter**:

- `= 5 > 4`
- `= 5 < 4`

- `= "A" = "B"`
- `= 3 * 4 = 15`
- `= 3 * 4 = 12`

Wenn die Gleichung stimmt, wird *true* ausgegeben, wenn nicht, ist der Rückgabewert *false*.

Der häufigste Verwendungszweck dieses Datentyps ist dort, wo Sie ihn gar nicht zu Gesicht bekommen: Im *if*-Befehl verwenden Sie Bedingungen, die geprüft werden und entweder wahr oder falsch sind. Anstatt *true* oder *false* erhalten Sie hier jedoch das Ergebnis, das mit *then* oder *else* festgelegt wird. ■

Anstelle der einfachen Rechnungen mit Zahlen und Buchstaben in den obigen Beispielen können Sie natürlich auch Schrittnamen oder M-Funktionen für Ihre Vergleiche verwenden.

Operatoren

Wie in jeder Programmiersprache gibt es auch in M bestimmte Operatoren, um Elemente zu verändern, zu berechnen oder zu vergleichen. Es hängt vom jeweiligen Datentyp ab, welche Operatoren gültig sind. Für Logical-Werte gibt es drei Operatoren: *and*, *or* und *not*.

Geben Sie folgendes Beispiel in die Bearbeitungsleiste ein:

```
= (5 > 1) and (1 = 7)
```

In diesem Fall haben Sie zwei Prüfungen, die einen Logical-Wert zurückgeben: *true* und *false*. Durch den Operator *and* wird geprüft, ob *beide* Werte wahr sind. Das Ergebnis ist somit *false*.

Mit dem Vergleichsoperator *or* prüfen Sie dagegen, ob *mindestens einer* der beiden Logical-Werte wahr ist. Versuchen Sie folgenden Code:

```
= (5 > 1) or (1 = 7)
```

In diesem Fall ist das Ergebnis *true*, denn die linke Prüfung ergibt *true*.

Der letzte Logical-Operator ist *not*. Damit kehren Sie das Ergebnis um:

```
= not (1 = 7)
```

Die Gleichung 1=7 ist zwar falsch, aber durch das Schlüsselwort *not* wird der Wert ins Gegenteil umgewandelt. Der Rückgabewert ist somit *true*.

Die Operatoren *and*, *or* und *not* funktionieren nur mit Logical-Werten. Die Klammern in diesen Beispielen bewirken, dass der Klammerinhalt zuerst ausgewertet wird und *TRUE* bzw. *FALSE* erzeugt, bevor die außenstehenden Operatoren angewendet werden. ■

11.1.3 Number (Zahl)

Anders, als man es angesichts der Spalten-Datentypen erwarten würde, gibt es in M nur einen Haupt-Datentyp für Zahlen. Egal, ob es sich um ganze Zahlen oder Dezimalzahlen handelt, sie haben immer den Typ *number*.

Folgende Punkte sind wichtig, wenn Sie Zahlen in M-Code schreiben:

- Im Code wird als Dezimaltrennzeichen immer der Punkt verwendet, unabhängig von der jeweiligen Regionaleinstellung. Statt `1,5` müssen Sie also `1.5` schreiben. Die Anzeige im Vorschaubereich des Abfrage-Editors wird jedoch an Ihre Regionaleinstellung angepasst.
- Längere Zahlen können mit dem Exponenten-E abgekürzt werden:

  `1e5` entspricht *100000*

  `34e5` entspricht *3400000*

  `1e-5` entspricht *0,00001*

  `34e-5` entspricht *0,00034*
- Die Präzision, in der Zahlenwerte gespeichert werden, entspricht dem Datentyp *Double* nach der IEEE-Norm. Dies bedeutet eine Genauigkeit auf ca. 15 Stellen.

Es gibt auch Untertypen zu Number, deren Genauigkeit eingeschränkt ist (Int64, Currency, Single, …).

- Auch wenn es nur selten gebraucht wird, gibt es Unendlich als Zahlenwert. Er wird bei bestimmten Rechenoperationen erzeugt, zum Beispiel bei der Rechnung `1/0`. Unendlich wird im M-Code geschrieben als `#infinity` und im Abfrage-Editor angezeigt als *infinity* bzw. ∞. Analog gibt es auch `-#infinity` als das negative Unendlich.
- Zahlenwerte können auch in Hexadezimal-Schreibweise eingegeben werden. Sie können z. B. statt `255` auch `0xff` schreiben.

Operatoren

Folgende Operatoren sind für Zahlenwerte gültig:

Tabelle 11.1 Operatoren für Zahlen.

Operator	Bedeutung
`x + y`	Summe
`x - y`	Differenz
`x * y`	Produkt
`x / y`	Quotient
`-x`	Negation
`x > y`	Größer (Vergleichsoperator)
`x >= y`	Größer oder gleich (Vergleichsoperator)

Operator	Bedeutung
`x < y`	Kleiner (Vergleichsoperator)
`x <= y`	Kleiner oder gleich (Vergleichsoperator)
`x = y`	Gleich (Vergleichsoperator)
`x <> y`	Ungleich (Vergleichsoperator)

11.1.4 Time (Zeit)

Im Abfrage-Editor werden Zeitwerte im selben Format angezeigt, wie Sie es von Digitaluhren kennen, d. h. mit Stunden, Minuten und Sekunden (z. B. *14:30:05*). Intern werden Werte dieses Datentyps jedoch als einfache Dezimalzahlen gespeichert und verarbeitet. Dies ist nützlich zu wissen, besonders wenn es um die Umwandlung von Datentypen geht.

Ein Time-Wert gibt eine Tageszeit zurück und ist definiert als die Anzahl von *Ticks* seit Mitternacht. Ein „Tick" entspricht 100 Nanosekunden. Sie können also auch mit sehr kleinen Zeitintervallen arbeiten. Wenn Sie einen Time-Wert in eine Dezimalzahl umwandeln, erhalten Sie eine Zahl zwischen 0 (=0 Uhr) und 1 (=Mitternacht).

Um Zeitwerte im M-Code zu schreiben, benutzen Sie am besten die Anweisung *#time*:

```
#time(Stunde, Minute, Sekunde)
```

Der Befehl `#time(14, 30, 5)` erzeugt also die Uhrzeit *14:30:05* im Time-Datentyp.

Operatoren

Die möglichen Operatoren bei Time-Werten werden im Abschnitt zu Duration erklärt.

11.1.5 Date (Datum)

Ähnlich wie der Time-Datentyp, werden auch Date-Werte im Abfrage-Editor anders angezeigt, als sie intern gespeichert werden. Im Vorschau-Bereich des Editors erscheinen Datumswerte so, wie es die jeweiligen Regionaleinstellungen vorgeben, also zum Beispiel *21. 04. 2018*. Intern wird eine ganze Zahl gespeichert, die die Anzahl an Tagen angibt, welche seit dem festgelegten Epochenbeginn vergangen sind. Die *1* entspricht dem *31. 12. 1899* und die höchstmögliche Zahl beträgt *2958465*, was dem *31. 12. 9999* entspricht.

Power Query unterstützt auch Datumswerte vor dem 31. 12. 1899. Der zugehörige Zahlenwert ist dann einfach negativ bzw. 0 (für den 30. 12. 1899).

Derartige Werte können aber nur in Power BI angezeigt werden. In Excel werden sie außerhalb des Abfrage-Editors nicht unterstützt. ■

Datumswerte werden im M-Code über die Anweisung *#date* erzeugt, wobei anders als bei uns üblich zuerst das Jahr, dann der Monat und zuletzt der Tag angegeben werden:

```
#date(Jahr, Monat, Tag)
```

Die Anweisung `#date(2018, 4, 21)` erzeugt somit den 21. April 2018.

Operatoren

Die möglichen Operatoren bei Date-Werten werden im Abschnitt zu Duration erklärt.

11.1.6 Datetime (Datum/Uhrzeit)

Der Datentyp Datetime ist die Verbindung aus den Datentypen Date und Time. Im Abfrage-Editor wird er mit Tag, Monat, Jahr, Stunde, Minute und Sekunde in der jeweiligen Regionaleinstellung angezeigt, also zum Beispiel *01.04.2012 08:30:00*.

Die Zahl, die intern gespeichert wird, folgt der gleichen Logik wie bei Date und Time, wobei die Zahl vor dem Komma das Datum und die Dezimalstellen die Uhrzeit angeben.

Zeitpunkte dieses Datentyps werden mit der Anweisung *#datetime* nach folgendem Schema erzeugt:

```
#datetime(Jahr, Monat, Tag, Stunde, Minute, Sekunde)
```

Operatoren

Die möglichen Operatoren bei Datetime-Werten werden im Abschnitt zu Duration erklärt.

11.1.7 Datetimezone (Datum/Uhrzeit/Zeitzone)

Der Datentyp Datetimezone wird eher selten verwendet, weil er im Grunde nur eine andere Darstellungsweise des Datentyps Datetime ist. Zusätzlich zu Datum und Uhrzeit können Sie hierbei die jeweilige Zeitzone anzeigen lassen. Die Werte werden jedoch intern umgerechnet in Datetime-Werte und entsprechend auch ohne die Zeitzonen-Angabe ausgegeben.

Um einen solchen Wert zu erzeugen, benutzen Sie die *#datetimezone*-Anweisung nach folgendem Schema:

```
#datetimezone(Jahr, Monat, Tag, Stunde, Minute, Sekunde, Zeitverschiebung-Stunden,
Zeitverschiebung-Minuten)
```

Operatoren

Die möglichen Operatoren bei Datetimezone-Werten werden im Abschnitt zu Duration erklärt.

11.1.8 Duration (Dauer)

Um den Abstand zwischen zwei Zeitpunkten anzugeben, wird in Power Query der Datentyp Duration verwendet. Er ist ähnlich aufgebaut wie der Datentyp Datetime. Der wesentliche Unterschied ist, dass ein Duration-Wert auch negativ sein kann.

Duration-Werte werden mit der Anweisung *#duration* erzeugt:

```
#duration(Tage, Stunden, Minuten, Sekunden)
```

Es steht Ihnen dabei frei, ob Sie Ihren Zeitraum in Tagen, Stunden, Minuten oder Sekunden angeben. Bei Date- oder Time-Werten würde eine Angabe von mehr als 12 Monaten bzw. mehr als 24 Stunden einen Fehler erzeugen. Der Duration-Datentyp ist flexibler. Die folgenden Anweisungen erzeugen alle einen Zeitraum von zwei Tagen:

- `#duration(2, 0, 0, 0)`
- `#duration(0, 48, 0, 0)`
- `#duration(1, 24, 0, 0)`
- `#duration(0, 0, 2880, 0)`

Auch wenn bei der Konstruktionsanweisung als größtmögliche Zeiteinheit Tage verwendet werden, müssen Sie sich übrigens auch bei sehr großen Duration-Werten keine Sorgen machen. Der Maximalwert für den Datentyp Duration liegt bei über 29 000 Jahren. Bei kleinen Zeitintervallen reicht die Präzision wie bei Time und Datetime bis zu 100 Nanosekunden. Um Sekundenbruchteile anzugeben, benutzen Sie einfach den Punkt als Dezimaltrennzeichen. Folgende Anweisung erzeugt einen Zeitraum von einer halben Sekunde: `#duration(0, 0, 0, 0.5)`

Operatoren

Die Tatsache, dass die Datentypen Date, Time, Datetime, Datetimezone und Duration alle programmintern als Dezimalzahlen gespeichert werden, ermöglicht eine Vielzahl von Berechnungen mit Zeit- und Datumswerten, die sonst nicht ohne Weiteres möglich wären. So können Sie beispielsweise problemlos ein Datum mit einer Uhrzeit verknüpfen oder die Differenz aus zwei Datetime-Werten ermitteln.

Dabei ist jedoch zu beachten, dass Berechnungen mit Zeit- oder Datumswerten unter Umständen Werte mit unterschiedlichen Datentypen ergeben. In der folgenden Tabelle erhalten Sie einen Überblick über die möglichen Rechenoperationen und die sich ergebenden Datentypen.

Tabelle 11.2 Rechenoperationen mit Zeit- und Datumswerten.

Datentyp 1	Operator	Datentyp 2	Ergebnis	Beschreibung
Time	–	Time	Duration	Abstand zwischen zwei Tageszeiten
Date	–	Date	Duration	Abstand zwischen zwei Datumswerten
Datetime	–	Datetime	Duration	Abstand zwischen zwei Zeitpunkten
Time	+ / –	Duration	Time	Um Intervall versetzte Tageszeit
Date	+ / –	Duration	Date	Um Intervall versetztes Datum

Fortsetzung nächste Seite

Tabelle 11.2 Rechenoperationen mit Zeit- und Datumswerten. *(Fortsetzung)*

Datentyp 1	Operator	Datentyp 2	Ergebnis	Beschreibung
Datetime	+ / –	Duration	Datetime	Um Intervall versetzter Zeitpunkt
Date	&	Time	Datetime	Zusammengesetzter Datetime-Wert
Duration	+	Duration	Duration	Summe von Zeitintervallen
Duration	–	Duration	Duration	Differenz aus Zeitintervallen
Duration	*	Number	Duration	Vielfaches von Zeitintervallen
Duration	/	Number	Duration	Teil von Zeitintervallen

Der Datentyp Datetimezone verhält sich genauso wie der Datentyp Datetime und ist daher in der Tabelle nicht aufgeführt.

Neben den oben aufgeführten Rechenoperatoren gelten für Zeit- und Datumstypen dieselben Vergleichsoperatoren wie für Zahlen. Sie können jeweils aber immer nur Werte des gleichen Typs miteinander vergleichen. Die folgende Prüfung ist daher gültig:

```
#date(2000, 1, 1) < #date(2005, 1, 1)
```

Diese Prüfung ist dagegen nicht möglich und führt zu einem Fehler:

```
#date(2000, 1, 1) < #datetime(2005, 1, 1, 18, 30, 00)
```

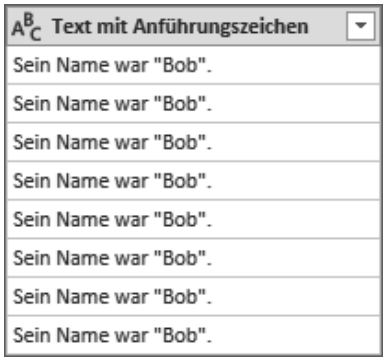
11.1.9 Text

Ein Textwert ist eine Sequenz aus Unicode-Zeichen. Der Unicode-Standard umfasst so gut wie alle Buchstaben, Zahlen und Sonderzeichen, die Sie aus der Textverarbeitung kennen (inklusive Emojis). Um die Länge der Zeichenkette müssen Sie sich auch keine Sorgen machen, sie darf bis zu 268 Mio. Zeichen betragen.

Im Normalfall definieren Sie einen Textwert, indem Sie den entsprechenden Text in (doppelte) Anführungszeichen schreiben. Was aber, wenn Ihre Zeichenfolge Anführungszeichen enthalten soll? In diesem Fall scheiben Sie einfach ein Anführungszeichen vor das Anführungszeichen. Damit signalisieren Sie, dass der Textwert noch nicht zu Ende ist, sondern Anführungszeichen enthält.

```
= "Sein Name war ""Bob""."
```

Wenn Sie diesen Code in der Bearbeitungsleiste oder in der Formel für eine benutzerdefinierte Spalte eingeben, sehen Sie, dass der Name „Bob“ in Anführungszeichen erscheint.



Text mit Anführungszeichen
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".
Sein Name war "Bob".

Bild 11.1
Textwerte mit Anführungszeichen.

Die Escape-Sequenz in M

Es gibt noch eine Reihe weiterer Zeichen, die nicht ohne Weiteres im Programmcode dargestellt werden können. Diese müssen daher kodiert werden. Als Zeichencode wird die Unicode-Nummer (im Hex-Format) verwendet, die für jedes Zeichen festgelegt ist. Sie finden einige Beispiele in Tabelle 11.3. Für die am häufigsten verwendeten Zeichen gibt es alternativ auch Abkürzungen in M.

Um anzuzeigen, dass nun ein kodiertes Zeichen verwendet wird, schreiben Sie den Zeichencode innerhalb von runden Klammern und setzen das Raute-Zeichen davor. Das Zeichen für Tab wird beispielsweise folgendermaßen dargestellt: `#(tab)`

Das Unendlich-Zeichen erstellen Sie auf diese Weise:

```
#(221E)
```

Natürlich dürfen auch bei Sonderzeichen wie bei allen Texten nicht die Anführungszeichen vergessen werden. ■

Tabelle 11.3 Häufig verwendete Sonderzeichen

Unicode-Nummer	Abkürzung in M	Bedeutung
0009	tab	Tab
000A	lf	Zeilenumbruch: Line Feed (Zeilenvorschub)
000D	cr	Zeilenumbruch: Carriage Return (Wagenrücklauf)
00A0		Geschütztes Leerzeichen
202F		Enges geschütztes Leerzeichen
00B6		Absatz-Zeichen
221E		Unendlich-Symbol
00A3		Pfund-Symbol
20AC		Euro-Symbol

Um die Unicode-Nummern für andere Zeichen zu finden, können Sie das Windows-Tool "Zeichentabelle" benutzen. In der linken unteren Ecke sehen Sie den jeweiligen Code. Es gibt auch diverse Quellen im Web, z. B. *https://unicode-table.com*.

Sie benötigen nur die vierstellige Zahlen-/Buchstabenreihe des Codes, nicht das vorangestellte „U+".

Zeilenumbrüche

Leider gibt es keinen einheitlichen Standard dafür, welches Zeichen einen Zeilenumbruch markiert. Es gibt zwei relevante Zeichen: Wagenrücklauf (`"#(cr)"`) und Zeilenvorschub (`"#(lf)"`). Dies ist ein Relikt aus Zeiten, in denen der Schreibmaschinen- oder Druckkopf gesondert an den Anfang der Zeile geschickt werden musste.

Es hängt vom Betriebssystem und vom Textverarbeitungsprogramm ab, welches Zeichen verwendet bzw. erwartet wird. In der Windows-Umgebung ist es oft eine Kombination aus beiden: `"#(cr)#(lf)"`

Falls Sie übrigens aus irgendeinem Grund die Zeichenfolge `#(` als Teil eines Textwerts schreiben wollen, müssen Sie dies auch gesondert markieren. Ansonsten würde es so interpretiert, als wollten Sie ein Spezialzeichen einfügen. Damit `#(` als Text erscheint, müssen Sie folgende Zeichenfolge schreiben: `#(#)(`

Operatoren

Mit Zeichenketten kann nicht gerechnet werden, daher funktionieren die üblichen Operatoren (`+`, `-` usw.) bei Textwerten nicht. Sie können aber zwei Textwerte mithilfe des Zeichens `&` verketten. Der folgende Code verbindet zwei Textbausteine und erzeugt den Satz „Das Wetter ist schön.":

```
= "Das Wetter ist " & "schön."
```

Um zwei Texte miteinander zu vergleichen, benutzen Sie wie auch bei Zahlen die Operatoren `=` (gleich) oder `<>` (ungleich).

Die anderen Vergleichsoperatoren (`>`, `<`) können verwendet werden, um Texte und Zeichen in Bezug auf ihre Position im Alphabet zu vergleichen. Beachten Sie hierbei, dass alle Großbuchstaben vor den Kleinbuchstaben eingeordnet werden (d. h. `"Z" < "a"`). Umlaute und Sonderzeichen kommen aber hinter allen anderen Buchstaben (d. h. `"z" < "ü"`). Die Grundlage für den Vergleich bilden die Unicode-Nummern der einzelnen Buchstaben.

Um zu sehen, in welcher Reihenfolge die Zeichen angeordnet werden, geben Sie diesen Code in die Bearbeitungsleiste ein:

```
= List.Transform({1..9999}, each Character.FromNumber(_))
```

11.1.10 Binary (Binär)

Ein binärer Wert besteht aus einer Folge von Bytes. Typischerweise dient der Datentyp Binary dazu, eingelesene Dateien, zum Beispiel bei der Ordnerabfrage, zu speichern.

Wenn Sie sich mit der Struktur von Binärdaten auskennen, können Sie Binary-Werte natürlich auch selbst erzeugen:

```
= #binary({0x00, 0x01, 0x02, 0x03})
```

Operatoren

Für Binärdaten stehen die üblichen Vergleichs- und Rechenoperatoren sowie eine Reihe von Funktionen zur Verfügung.

11.2 Übergeordnete Datentypen

Im Gegensatz zu den bisher vorgestellten Datentypen, die immer nur einzelne Werte enthalten, gibt es auch Datentypen, die mehrere Einzelwerte in einem Wert zusammenfassen können. Werte vom Typ List, Record oder Table können beliebig viele Werte enthalten, die selbst einen beliebigen Datentyp haben und sogar wieder andere Werte in sich vereinen können.

11.2.1 List (Liste)

Eine Liste in Power Query ist eine beliebig lange Reihe von Werten. Die Werte können dabei jeden beliebigen Typ haben, sogar Listen von Listen sind möglich.

In M werden Listen immer in geschweiften Klammern angegeben. Die einzelnen Listenelemente werden durch Kommas getrennt. Sollte es sich dabei um Texte handeln, werden sie wie immer in Anführungszeichen geschrieben.

Im Folgenden sind einige Beispiele für Listen aufgeführt:

- Eine Liste aus Zahlen:

```
={2, 3, 5, 7, 11, 13, 17, 19}
```

- Eine Liste aus Texten:

```
={"Asterix", "Obelix", "Idefix"}
```

- Eine gemischte Liste:

```
={0, "Giraffe", -5, 8.1, "0815", "xyz", 42}
```

- Eine Liste, die drei Listen enthält:

```
={ {"Ketchup", "Senf"}, {"Salz", "Pfeffer"}, {"Essig", "Öl"} }
```