Data Transformation with Power Query M in Power BI

Learn Power BI at www.DataCamp.com

> Definitions

Power Query is a tool for extract-transform-load (ETL). That is, it lets you import and prepare your data for use in Microsoft data platforms including Power BI, Excel, Azure Data Lake Storage and Dataverse.

Power Query Editor is the graphical user interface to Power Query.

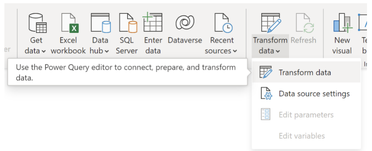
Power Query M ("M" for short) is the functional programming language used in Power Query.

DAX is the other programming language available for Power BI. DAX is used for data analysis rather than ETL. Learn more about it in the DataCamp DAX cheat sheet.

An **expression** is a single formula that returns a value.
A **query** is a sequence of expressions used to define more complex data transformations. Queries are defined using let-in code blocks.

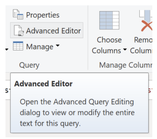
> Accessing M in Power BI

M code can be seen in Power Query Editor. In the ribbon, click on 'Transform data' to open the Power Query Editor.



M code is shown in the Formula Bar.

M code can also be seen in the Advanced Editor window. Click 'Advanced Editor' to open the Advanced Editor window.



> Creating values

```
999 // Define a number

true // Define a logical value

"DataCamp" // Define a text value

null // Define a null (missing value)

#date(2023, 12, 31) // Define a date with #date()

#datetime(2022, 9, 8, 15, 10, 0) // Define a datetime
with #datetime()
```

> Variables

```
// Variables are assigned by writing a
query with let-in
let
  // Intermediate calculations
  TempF = 50
  TempC = 5 / 9 * (TempF - 32)
in
  // Resulting variable
  TempC

// By convention, variable names are
UpperCamelCase
HeightM

// Quote variable names and prefix with
# for non-standard names
#"Height in Meters"
```

> Operators

Arithmetic operators

```
102 + 37 // Add two numbers with +
102 - 37 // Subtract a number with -
4 * 6    // Multiply two numbers with *
22 / 7   // Divide a number by another with /
```

Numeric comparison operators

```
3 = 3  // Test for equality with =
3 <> 3 // Test for inequality with <>
3 > 1  // Test greater than with >
3 >= 3 // Test greater than or equal to with >=
3 < 4  // Test less than with <
3 <= 4 // Test less than or equal to with <=
```

Logical Operators

```
not (2 = 2)          // Logical NOT with not
(1 <> 1) and (1 < 1) // Logical AND with and
(1 >= 1) or (1 < 1)  // Logical OR with or
```

Text Operators

```
"fish" & " & " & "chips" // Combine text with &
```

> Numbers

Arithmetic

```
Number.Power(3, 4) // Raise to the power with
Power()
Number.IntegerDivide(22, 7) // Integer divide
a number with IntegerDivide()
Number.Mod(22, 7) // Get the remainder after
division with Mod()
Value.Equals(1.999999, 2, Precision.Double) //
Check number close to equal with Equals()
```

Math functions

```
Number.Ln(10) // Calculate natural logarithm
with Ln()
Number.Exp(3) // Calculate exponential with
Exp()
Number.Round(12.3456, 2) // Round to n decimal
places with Round()
Number.Abs(-3) // Calculate absolute values
with Abs()
Number.Sqrt(49) // Calculate the square root
with Sqrt()
Number.IsNaN(Number.NaN) // Returns true if
not a number
```

> Text Values

Creating text

```
// Text values are double-quoted, and can
span multiple lines
"M is a programming
language for ETL"

// Embed quotes in strings by doubling them
"""M is magnificent"", mentioned Mike."

// Include control characters with #()
"Split text with a tab character, #(tab), or start
a new line with carriage-return line feed,
#(cr,lf)"

// Embed # in strings with #(#)
"Hex codes for colors start with #(#)"
```

Creating text

```
// Text values are double-quoted, and can
span multiple lines
"M is a programming
language for ETL"

// Embed quotes in strings by doubling them
"""M is magnificent"", mentioned Mike."

// Include control characters with #()
"Split text with a tab character, #(tab), or
start a new line with carriage-return line feed,
#(cr,lf)"

// Embed # in strings with #(#)
"Hex codes for colors start with #(#)"
```

Indexing

```
// Get the number of characters in text with Length()
Text.Length("How long will dinner be? About 25cm.")

// Get a substring with Middle()
Text.Middle("Zip code: 10018", 10, 5)
```

Splitting and combining text

```
// Combine text, optionally separated with Combine()
Text.Combine({"fish", "chips"}, " & ")

// Split text on a delimiter with Split()
Text.Split("fish & chips", " & ")
```

Mutating text

```
// Convert text to upper case with Upper()
Text.Upper("iN cAsE oF eMeRgEnCy") // Returns
"IN CASE OF EMERGENCY"

// Convert text to LOWer case with Lower()
Text.Lower("iN cAsE oF eMeRgEnCy") // Returns
"in case of emergency"

// Convert text to title case with Proper()
Text.Proper("iN cAsE oF eMeRgEnCy") // Returns
"In Case Of Emergency"

// Replace characters in text with Replace()
Text.Replace("Have a nice trip", " n", "n ")
// Returns "Have an ice trip"
```

Type Conversion

```
// Convert value to number with Number.From()
Number.From(true) // Returns 1

// Dates and datetimes given as time in days
since 1899-12-30
Number.From(#datetime(1969, 7, 21, 2, 56, 0)) //
Returns 25405.12

// Convert text to number with Number.FromText()
Number.FromText("4.5E3") // Returns 4500

// Convert number to text with Number.ToText()
- Formats: "D": integer digits, "E":
exponential, "F": fixed, G": general, "N":
number, "P": percent
Number.ToText(4500, "E") // Returns "4.5E3"

// Convert value to logical with
Logical.From()
Logical.From(2)
```

Functions

```
// Define a function with (args) => calculations
let
  Hypotenuse = (x, y) => Number.Sqrt(Number.Power(x, 2) + Number.Power(y, 2))
in
  Hypotenuse
// each is syntactic sugar for a function with 1 arg named _
// Use it to iterate over lists and tables
each Number.Power(_, 2) // Same as (_) => Number.Power(_, 2)
```

> Lists

Creation

```
// Define a list with {}
//You can include different data types including null
{999, true, "DataCamp", null}

// Define a sequence of numbers with m..n
{-1..3, 100} // Equivalent to {-1, 0, 1, 2, 3, 100}

// Lists can be nested
{"outer", {"inner"}}

// Concatenate lists with &
{1, 4} & {4, 9} // Returns {1, 4, 4, 9}
```

Example lists

```
let
  Fruits = {"apple", null, "cherry"}
in
  Fruits

let
  Menage = {1, -1, 0, 1, 2, 13, 80, 579}
in
  Menage
```

Counting

```
// Access list elements with {}, zero-indexed
Fruits{0} // 1st element; returns "apple"

// Accessing elements outside the range
throws an error
Fruits{3} // Throws an Expression.Error

// Append ? to return null if the index is
out of range
Fruits{3}? // Returns null

// Get the first few elements with FirstN()
List.FirstN(Fruits, 2) // Returns {"apple", null}

// Get the last few elements with LastN()
List.LastN(Fruits, 2) // Returns {null, "cherry"}

// Get unique elements with Distinct()
List.Distinct(Menage) // Returns {1, -1, 0, 2,
13, 80, 579}
```

Selection

```
// Access list elements with {}, zero-
indexed
Fruits{0} // 1st element; returns "apple"

// Accessing elements outside the range
throws an error
Fruits{3} // Throws an Expression.Error

// Append ? to return null if the index is
out of range
Fruits{3}? // Returns null

// Get the first few elements with FirstN()
List.FirstN(Fruits, 2) // Returns {"apple",
null}

// Get the last few elements with LastN()
List.LastN(Fruits, 2) // Returns {null,
"cherry"}

// Get unique elements with Distinct()
List.Distinct(Menage) // Returns {1, -1, 0, 2, 13,
80, 579}

// Get elements that match a criteria with Select()
List.Select(Menage, each _ > 1) // Returns {2, 13,
80, 579}

// Return true if all elements match a criteria with
MatchesAll()
List.MatchesAll(Menage, each _ > 1) // Returns false

// Return true if any elements match a criteria with
MatchesAny()
List.MatchesAny(Menage, each _ > 1) // Returns true

// Get value from list of length 1, or return
default, with SingleOrDefault()
List.SingleOrDefault(Menage, -999) // Returns -999
```

Manipulation

```
// Sort items in ascending order with Sort()
List.Sort(Menage) // Returns {-1, 0, 1, 1, 2,
13, 80, 579}

// Sort items in descending order
List.Sort(Menage, Order.Descending) // Returns
{579, 80, 13, 2, 1, 1, 0, -1}

// Reverse the order of items in a list with
Reverse()
List.Reverse(Menage) // Returns {579, 80, 13,
2, 1, 0, -1, 1}

// Get non-null values with RemoveNulls()
List.RemoveNulls(Fruits) // Returns {"apple",
"cherry"}

// Remove items by position with RemoveRange()
List.RemoveRange(Menage, 2, 3) // Returns {1,
-1, 13, 80, 579}

// Repeat elements with Repeat()
List.Repeat({"one", "two"}, 2) // Returns
{"one", "two", "one", "two"}

// Split list into lists of specified size with
Split()
List.Split(Menage, 2) // Returns {{1, -1}, {0,
1}, {2, 13}, {80, 579}}

// Flatten lists by removing 1 level of nesting
with Combine()
List.Combine({{"alpha"}, {"bravo", {"charlie",
"delta"}}}) // Returns {"alpha", "bravo",
{"charlie", "delta"}}
```

Equality & membership

```
// Lists are equal if they contain the same elements in the same order
{1, 2} = {1, 2} // true     {1, 2} = {2, 1} // false     {1, 2} = {1, 2, 3} // false
```

Calculations

```
// Get the minimum value in a list with Min()
List.Min({0, 7, -3, 2, 1})

// Get the minimum value in a list with Max()
List.Max({0, 7, -3, 2, 1})

// Get quantile values from a list with
Percentile()
List.Percentile(
  {0, 7, -3, 2, 1}, {0.25, 0.5, 0.75},
  [PercentileMode=PercentileMode.SqlDisc]
)

// Get the sum of values in a list with Sum()
List.Sum({0, 7, -3, 2, 1})

// Get the product of values in a list with
Product()
List.Product({0, 7, -3, 2, 1})

// Get the mean of values in a list with Average()
List.Average({0, 7, -3, 2, 1})

// Get the standard deviation of values in a list
with StandardDeviation()
List.StandardDeviation({0, 7, -3, 2, 1})
```

Generation

```
// Generate random numbers between 0 and 1
with Random()
List.Random(3)

// Generate a sequence of dates with Dates()
List.Dates(#date(2023, 1, 1), 3, #duration(7,
0, 0, 0))

// Generate a sequence of numbers with Numbers()
List.Numbers(1, 5, 2)

// Mimic a for loop with Generate()
List.Generate(
  () => 2,
  each _ < 20,
  each Number.Power(_, 2))
```

Set operations

```
// Get values in all inputs with Intersect()
List.Intersect({{1, 3, 6, 10, 15}, {21, 15, 9, 3}, {0, 3, 6}}) // Returns {3}

// Get values in any inputs with Union()
List.Union({{1, 3, 6, 10, 15}, {21, 15, 9, 3}, {0, 3, 6}}) // Returns {0, 1, 3, 6, 9, 10, 15,
21}

// Get value in one set but not the other with Difference()
List.Difference({1, 3, 6, 10, 15}, {21, 15, 9, 3}) // Returns {1, 6, 10}
```