



Table of Contents

What does Power Query do and how does it work? (from Only App Matter Book)..... 2

Excel and Power BI Power Query Ribbon Tabs..... 3

Power Query Editor Window in Excel 4

Power Query Editor Window in Power BI Desktop..... 4

Excel Power Query Load To 5

Power BI Desktop Close & Apply 5

The story of how M Code works in Power Query: 6

 Microsoft Power Query M Code Specifications Guide 6

 Define Power Query..... 6

 Values possible in Power Query..... 6

 Power Query Data Types 7

 Data Types vs. Values..... 7

 Where you can view M Code 8

 Expressions..... 8

 Identifiers 8

 Keywords..... 9

 Operators and Punctuators..... 9

 Standard Library..... 9

 M Code Tips: #shared, Tab, Linefeed, Return, Lists of Numbers & Letters and Joining Lists, Records & Tables 9

 Let expression 10

 M Code Lookup 12

 Primary Keys..... 14

 Custom Functions 15

 each and underscore..... 18

What does Power Query do and how does it work? (from Only App Matter Book)

Power Query

Power Query was first introduced in Excel in 2013, exactly 20 years after the PivotTable was invented. And as it turns out, Power Query is the greatest invention in Excel since the PivotTable. When you have to deal with data, Power Query is a dream tool that can do everything. In Excel 365, most data analysis projects start with Power Query, which is a perfect tool for all the data preparation you need to do before you use a PivotTable to create summary reports with conditional calculations.

Power Query can connect to data sources such as text files, databases, websites, other Excel files, and XML files. It can also bring data into the Power Query Editor window, which is an interface you can use to invoke commands to import, clean, transform, and manipulate data. Finally, Power Query can load data to locations such as the worksheet, the PivotTable cache, or the Data Model in the Excel app; and to the Data Model in the Power BI Desktop app. Power Query can also perform many other tasks that involve data, such as financial and statistical calculations that are not related to reporting, visualizations, and dashboards.

When you use the Power Query user interface to do things like import data, add a data type, or combine multiple tables, behind the scenes Power Query records every step for you so that you can go back and see previous steps or even go back and edit previous steps in the query. Power Query records these steps using a case-sensitive, function-based formula programming language called *M code*, where M stands for *data mashup*. There are more than 700 functions in the M code language, but none of them are identical to Excel worksheet functions. Luckily, most of the time you can use the Power Query user interface to create queries by clicking on buttons and commands, and Power Query will write the M code for you. Sometimes, however, the best option is to write your own M code. In this chapter, you will learn how to create queries both ways.

As you begin to learn how to use Power Query, it is helpful to understand that Power Query queries are different from Excel worksheet formulas in two main ways:

- With worksheet formulas, you must type out and write your formulas manually, but when you create a query, most of the time you click on buttons and invoke commands, and the M code formulas are written for you.
- In a worksheet, formulas mostly deliver individual values in individual cells or arrays of values, but in Power Query, M code can deliver *values* such as tables with fields, records, lists, functions, binary files, and individual values such as text, numbers, dates, times, nulls, and Boolean values.

You can work with more types of objects (or *values*, as they are officially called in M code) in Power Query than in an Excel worksheet. This makes sense because in data analysis, the objects that you often work with are not simply numbers, text, and dates but instead are tables with fields, records, lists, and binary files.

Luckily, Microsoft put the Power Query tool in both the Excel app and in the Power BI Desktop app. Figure 18.64 shows that in the Excel app, Power Query is in the Data tab in the Excel Ribbon. Figure 18.65 shows that in the Power BI Desktop app, Power Query is in the Home tab in the Power Query Ribbon. Although the Power Query user interface is somewhat different in the two tools, almost everything else is the same. Whatever you learn about Power Query in the Excel app can help you in the Power BI Desktop app and vice versa. In this example, you will use Power Query in the Excel app, and then in Example 3, you will use Power Query in the Power BI Desktop app.

Excel and Power BI Power Query Ribbon Tabs

Excel (Figure 18.64):

The screenshot shows the Excel ribbon with the **Data** tab selected. The ribbon is divided into two main groups: **Get & Transform Data** and **Queries & Connections**. The **Get & Transform Data** group includes options like 'From Text/CSV', 'From Web', and 'From Table/Range'. The **Queries & Connections** group includes 'Refresh All', 'Queries & Connections', 'Properties', and 'Edit Links'. A large text box on the right says '<= Power Query'. Below the ribbon, two callout boxes provide instructions: one for starting a new query and another for refreshing existing queries.

Start new query to:
import, clean, transform,
and load data

Refresh or access:
existing queries and other
connections to data sources
in the Excel workbook

Power BI (Figure 18.65):

The screenshot shows the Power BI ribbon with the **Home** tab selected. The ribbon is divided into two main groups: **Data** and **Queries**. The **Data** group includes options like 'Get data', 'Excel', 'Power BI datasets', 'SQL Server', 'Enter data', and 'Recent sources'. The **Queries** group includes 'Transform data' and 'Refresh data'. A large text box on the left says 'Power Query =>'. Below the ribbon, two callout boxes provide instructions: one for starting a new query and another for refreshing existing queries.

Start new query to:
import, clean,
transform, and load
data

Refresh or access:
existing queries

Power Query Editor Window in Excel

Power Query Editor window inside Excel app

Power Query Ribbon: 4 tabs with buttons and commands

Data source settings

Advanced Editor for M Code

Data Type Icons

M Code Formula Bar

Default query name

Query steps automatically created

Preview of data

The screenshot shows the Power Query Editor window in Excel. The ribbon has four tabs: File, Home, Transform, and View. The M Code Formula Bar contains the formula: `= Table.TransformColumnTypes("#Promoted Headers",{{"ProductID", Int64.Type}, {"Units Sold", Int64.Type}})`. The data preview shows a table with columns ProductID and Units Sold. The Query Settings pane on the right shows the query name "Ch18UnitsTableExample02" and applied steps: Source, Promoted Headers, and Changed Type.

ProductID	Units Sold
1043	6
1069	3
1069	11
1043	8
2005	6
2005	13
1043	5
1043	5

Power Query Editor Window in Power BI Desktop

Power Query Editor window inside Power BI Desktop

Data source settings

Power Query Ribbon: 6 tabs with buttons and commands

Advanced Editor for M Code

M Code Formula Bar

Default query name

Query steps automatically created

Preview of data

List of queries in Power BI Desktop file

The screenshot shows the Power Query Editor window in Power BI Desktop. The ribbon has six tabs: File, Home, Transform, Add Column, View, Tools, and Help. The M Code Formula Bar contains the formula: `= Table.TransformColumnTypes(fUnits_Table,{{"ProductID", Int64.Type}, {"Units Sold", Int64.Type}})`. The data preview shows a table with columns ProductID and Units Sold. The Query Settings pane on the right shows the query name "fUnits" and applied steps: Source, Navigation, and Changed Type.

ProductID	Units Sold
1043	6
1069	3
1069	11
1043	8
2005	6
2005	13
1043	5
1043	5

Excel Power Query Load To

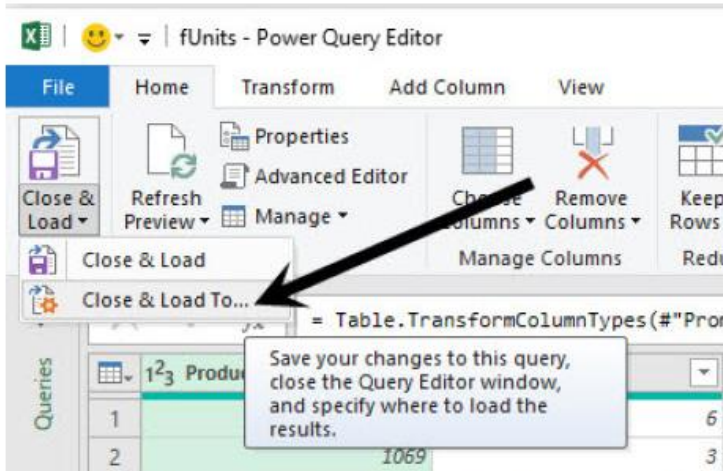


Figure 18.76 Use the Close & Load To option to choose where the data is loaded.

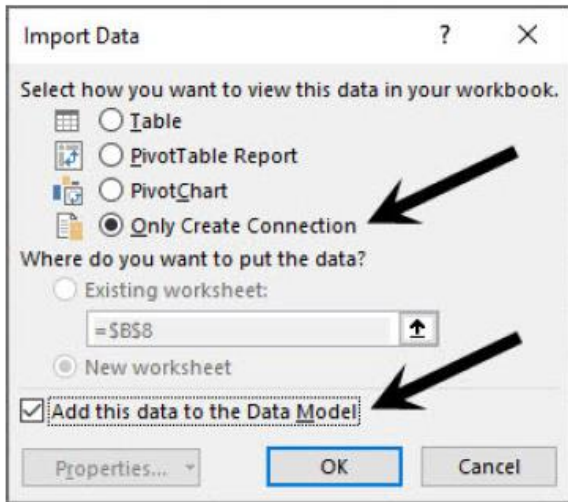


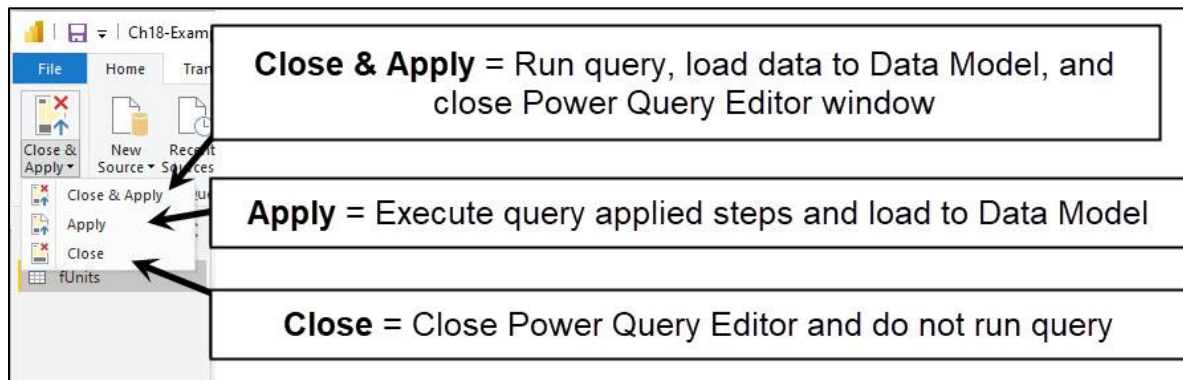
Figure 18.77 When you load to the Data Model, you must select both of these options.

2. In the Import Data dialog box that appears, to load the table to the Power Pivot Data Model, select the Only Create Connection option button and check the Add This Data to the Data Model checkbox, as shown in Figure 18.77.
3. Click OK in the Import Data dialog box. The fact table is loaded into the Data Model's columnar database, and you are taken back to the Excel app window.

Take a closer look at the Import Data dialog box in Figure 18.77. The top four options allow you to select how to view the data in the Excel workbook file:

- **Table:** The query is loaded as an Excel Table in the worksheet
- **PivotTable Report:** The query is loaded into the PivotTable cache.
- **PivotChart:** The query is loaded into the PivotTable cache.
- **Only Create Connection:** The query is not loaded to the worksheet or the PivotTable cache. This option allows Power Query to run the query steps on the source data in the Power Query Editor but then pass the data along to other queries—or, in this case, pass the data into the Data Model's columnar database.

Power BI Desktop Close & Apply



The story of how M Code works in Power Query:

Microsoft Power Query M Code Specifications Guide

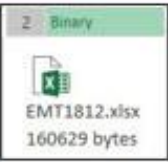
<https://learn.microsoft.com/en-us/powerquery-m/power-query-m-language-specification>

Define Power Query








- Power Query is a tool to import, clean, transform and load data, or as Microsoft calls it “Data Mashup”.
- The amazing tool Power Query is in Excel and Power BI Desktop (Power BI Online has a similar tool called Data Flows).
- Power Query uses a case sensitive, function-based M Code language to create **Values**.

Values possible in Power Query

Value	Description	Literal (Type a hard code value into M Code)						
1 Null	Absence of data.	Example: null						
2 Logical	Boolean: true or false.	Examples: true false						
3 Text	A string of Unicode characters.	Example: "Quad"						
4 Number	Used for numeric and arithmetic operations.	Examples: -43 , 0 , 43.46 , 9.9e-5						
5 Time	Time for 24 hour day as decimal.	#time(hour, minute, seconds) Example: #time(11,10,57) = 11:10:57						
6 Date	Date as a serial number (Common Era Gregorian calendar). Day 1 = 01/01/0001. Last day = 12/31/9999.	#date(year, month, day) Example: #date(1598, 01, 15) = 1/15/1598						
7 DateTime	Datetime value contains both date and time.	#datetime(year, month, day, hour, minutes, seconds) Example: #datetime(2021,10,01,11,10,57) = 10/01/21 11:10:57 AM						
8 DateTime Zone	Represents a UTC (Universal Coordinated Time) date/time with a time-zone offset (last two arguments: hours & minutes).	#datetimezone(year, month, day, hour, minute, second, offset-hours, offset-minutes) Example: #datetimezone(2021,10,01,11,10,57,09,00) = 10/01/21 11:10:57 AM + 9 hours						
9 Duration	Serial Number Length of Date and Time (can be positive or negative).	#duration(days, hours, minutes, seconds) Example: #duration(01,01,10,0) = 1 day and 1 hour 10 minutes						
10 Table	Table with field names and records.	Example: #table({"Boom Product", "Sales"}, {"Quad",43}, {"Aspen",35}) = <table border="1" data-bbox="1224 1381 1520 1503"> <thead> <tr> <th>Boom Product</th> <th>Sales</th> </tr> </thead> <tbody> <tr> <td>Quad</td> <td>43</td> </tr> <tr> <td>Aspen</td> <td>35</td> </tr> </tbody> </table>	Boom Product	Sales	Quad	43	Aspen	35
Boom Product	Sales							
Quad	43							
Aspen	35							
11 Record	An ordered sequence of fields, like a row from a table.	Example: [Boom Product = "Quad", Sales = 43] = <table border="1" data-bbox="1224 1560 1446 1650"> <thead> <tr> <th>Boom Product</th> <th>Quad</th> </tr> </thead> <tbody> <tr> <td>Sales</td> <td>43</td> </tr> </tbody> </table>	Boom Product	Quad	Sales	43		
Boom Product	Quad							
Sales	43							
12 List	A sequence of M Code values house in curly brackets. The list can have varied value types in list. When you lookup a column, a list of values from the column is returned.	Example: {"Quad", "Aspen"} = <table border="1" data-bbox="1224 1692 1377 1801"> <thead> <tr> <th>List</th> </tr> </thead> <tbody> <tr> <td>1 Quad</td> </tr> <tr> <td>2 Aspen</td> </tr> </tbody> </table>	List	1 Quad	2 Aspen			
List								
1 Quad								
2 Aspen								

Value	Description	Literal (Type a hard code value into M Code)
		Example: Excel file = 
13 Binary	Represents a sequence of bytes.	
14 Function	An M Code Custom Function is a user-created and defines the variables and the mapping for those variables to deliver a value.	Example: Function calculates the effective rate = (APR, Periods) => Number.Power(1+APR/Periods, Periods)-1
15 Type	The Data Types we use on values in Power Query.	Example: "type number" for defining the Decimal Data Type

Power Query Data Types

Data Type	Icon	Short Description	M Code
1 Decimal number	1.2	Number up to 15 decimals	type number
2 Currency (Fixed decimal number)	\$	Number up to 4 decimals	Currency.Type
3 Whole number	1 ² 3	Number with no digit to right of decimal	Int64.Type
4 Percentage	%	Number up to 15 decimals with % Number Format	Percentage.Type
5 Date/Time		Serial number date and time together	type datetime
6 Date		Serial number date	type date
7 Time		Serial number time	type time
8 Date/Time/Timezone		Represents a UTC date/time with a time-zone offset	type datetimetimezone
9 Duration		Serial Number Length of Date and Time	type duration
10 Text	A ^B C	Text	type text
11 True/False		Boolean	type logical
12 Binary		File like Excel file or Text file	type binary
13 Any	ABC 123	Sets numbers such as dates and decimals according to regional settings	type any
14 nullable		You can add the keyword nullable to data types like number so that the column can have the number or a null.	type nullable number
15 anynonnull		Any non-null value (all values excluding null).	type anynonnull
16 none		No values are classified	type none

Data Types vs. Values

1. **Data Types** on Fields aid consistent data in field and help create accurate calculations from the data.
2. Power Query **Values** are the possible outputs that M Code can deliver.

Where you can view M Code

1. Applied Steps = List of query step names, called identifiers (As shown below).
2. Formula Bar = Shows M Code formula for selected step in the Applied Step list (As shown below).
3. Advanced Editor = Shows all the M Code for a given query, which consists of the let expression with all its query steps that work in succession to deliver the final value of the query (As shown below).

Picture illustrates the three places to view and edit M Code:

The screenshot shows the Microsoft Excel Power Query interface. The ribbon includes File, Home, Transform, Add Column, and View. The 'Advanced Editor' button is highlighted with a red box and labeled '3)'. The 'Applied Steps' pane on the right is labeled '1)'. The 'Formula Bar' at the top is labeled '2)'. The 'Advanced Editor' window is open, showing the M code for the 'DayTypeSalesReport' query. A table of data is visible in the background.

ABC 123	TypeDay	ABC 123	TotalSales
1	Workday		15956.8
2	Weekend		4669.8
3	Holiday		1919.97
4	Donation		6311.83

```
let
Source = Excel.CurrentWorkbook()[[Name="DayTypeSales"]][Content],
DayTypeSales = Table.Group(Source, {"TypeDay"}, {"TotalSales", each List.Sum([Sales])}),
AddDataTypes = Table.TransformColumnTypes(DayTypeSales, {"TypeDay", type text}, {"TotalSales", type number})
in
AddDataTypes
```

Notice:
1) Query Step Name (Identifier) "DayTypeSales" is also listed in Applied Steps List
2) Formula is also listed in the Formula Bar

Expressions

- An **expression** is any M Code that results in a value.
- Some examples of expressions:
 - A function, like:
 - = Table.Group(Source, {"TypeDay"}, {"TotalSales", each List.Sum([Sales])}).
 - A list, like: = {1,2,3}.
 - A number and math operator, like: = 43 + 7.
 - A query step within a let expression, like:
 - DayTypeSales = Table.Group(Source, {"TypeDay"}, {"TotalSales", each List.Sum([Sales])}),
 - A full let expression, as shown in the above picture.

Identifiers

- An **identifier** is used to refer to an expression (value).
 - If the identifier has a space, you must distinguish it from text by recording it with a leading # sign and place identifier in quotes, such as: #"Identifier Name".
 - If there is no space, you record the identifier as: IdentifierName.

- **Generalized Identifier** are identifiers that allow spaces without the # sign and quotes.
 - Generalized Identifiers are allowed in either:
 1. The name of a Field in a Record Literal like: [Boom Product = "Quad", Sales = 43] or
 2. The name of a Field in a Field Access Operator like: [Boom Product].

Keywords

- Reserved words that have an assigned meaning and not be used as identifiers for expressions.
- Here are some of the Keywords:

and as each else error false if in is let meta not otherwise or section shared then true try type
 #binary #date #datetime #datetimezone #duration #infinity #nan #sections #shared #table #time

Operators and Punctuators

- There are several kinds of operators and punctuators.
- Operators are used in expressions to describe operations involving one or more operands. For example, the expression a + b uses the + operator to add the two operands a and b.
- Punctuators are for grouping and separating. For Example { } for grouping items in a list: {1, 2, 3}.
- Examples:

, ; = < <= > >= <> + - * / & () [] { } @ ! ? => ..

Online article about operators; <https://learn.microsoft.com/en-us/powerquery-m/operators>

Standard Library

- Standard Library is a list of built-in Functions and constants that deliver values.
- The Power Query Excel Standard Library has 837 items and the Power BI Desktop Power Query Standard Library has 1032 items (many more data connectors)
- Four examples of Standard Library:
 - Excel.CurrentWorkbook() function.
 - Number.Round() function.
 - Number.E or Number.PI (delivers the number e or pi).

M Code Tips: #shared, Tab, Linefeed, Return, Lists of Numbers & Letters and Joining Lists, Records & Tables

- If you type **#shared** into power Query Editor Formula Bar, you get a full list of all elements in Standard Library. If you convert to a Table, you can search the list.
- Carriage-return, linefeed, or tab character in a text literal, the #(cr), #(lf), and #(tab).
- If you type the list: {1.. 43}, you get a list of numbers 1 to 43.
- If you type {"a".."z"} you get a lower case alphabet.
- You can join two lists with Join Operator, like: {1,3}&{7,43} = {1,3,7,43}
- You can join two records with Join Operator, like: [A=3]&[B=43] = [A=3, B=43]
- You can append tables using the Join Operator, like: Table01&Table02, as shown here:

Table01:	Table02:	Join Operator To Append Tables:																											
<table border="1"> <tr><td>P</td><td>S</td></tr> <tr><td>A</td><td>38</td></tr> <tr><td>F</td><td>37</td></tr> </table>	P	S	A	38	F	37	<table border="1"> <tr><td>P</td><td>S</td></tr> <tr><td>Q</td><td>43</td></tr> <tr><td>A</td><td>19</td></tr> </table>	P	S	Q	43	A	19	<table border="1"> <tr> <th colspan="3">= Table01 & Table02</th> </tr> <tr> <td>1</td> <td>Q</td> <td>43</td> </tr> <tr> <td>2</td> <td>A</td> <td>19</td> </tr> <tr> <td>3</td> <td>A</td> <td>38</td> </tr> <tr> <td>4</td> <td>F</td> <td>37</td> </tr> </table>	= Table01 & Table02			1	Q	43	2	A	19	3	A	38	4	F	37
P	S																												
A	38																												
F	37																												
P	S																												
Q	43																												
A	19																												
= Table01 & Table02																													
1	Q	43																											
2	A	19																											
3	A	38																											
4	F	37																											

Let expression

- let expression allows you to define variables (steps) and use them throughout the let expression to define a final value. let expressions can be used in any M Code, but is most often used to define a new query.
- The rules for a let expression are:
 1. Start the let expression with lower case let.
 2. Each variable, or step, starts with an identifier, then an equal sign, then an expression.
 3. Variables are usually used in subsequent steps, can be used anywhere throughout the let expression, but cannot be used outside the let expression.
 4. Each variable is followed by a comma.
 5. The last query step does not end with a comma.
 6. End the let expression with lower case in followed by the identifier for the final value to be delivered by the new query. The result of the query is almost always the identifier of the last query step.
- Visual summary of let rules:

let expression structure:

```
let
  VariableName1 = M Code,
  #" Variable Name 2" = M Code,
  VariableName3 = M Code
in
  Output
```

*Output is usually the last variable name, but can be any variable name, query name or expression

- A **new query** created with the Power Query tool uses a let expression:
 1. In a query, a let expression is designed to combine all query steps to deliver a final value.
 2. The query name is the identifier for the let expression. This identifier can be used throughout the workbook.
 3. Example of a new query that uses let to deliver a Sales By Day Type Report:



```
Advanced Editor

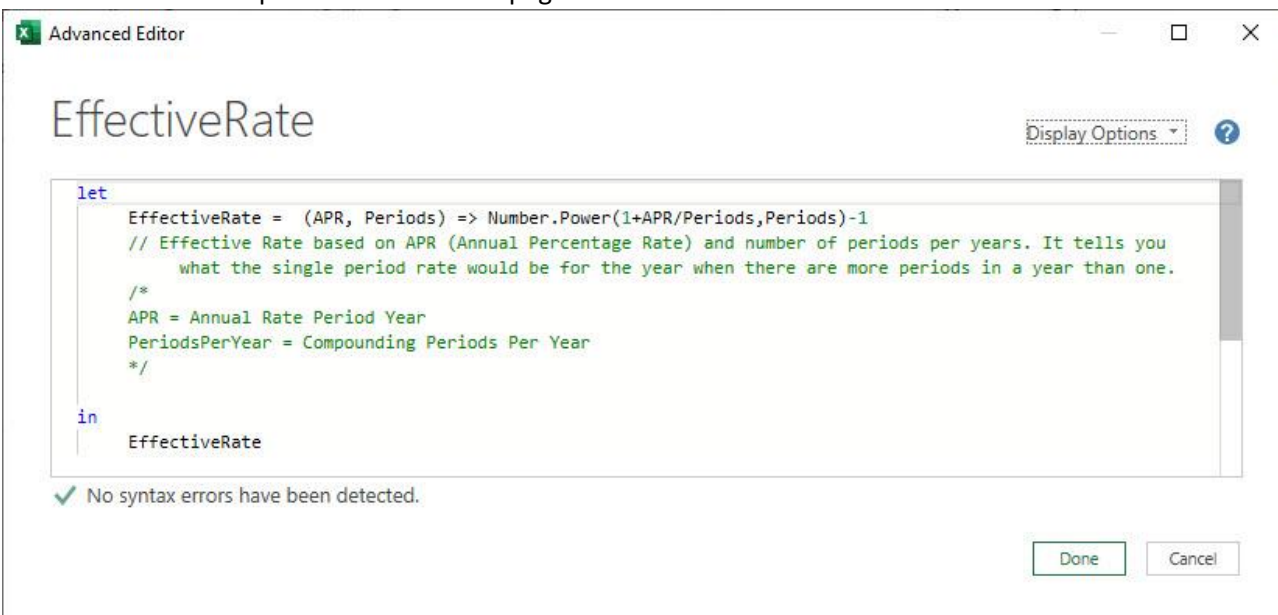
DayTypeSalesReport

let
    Source = Excel.CurrentWorkbook(){[Name="DayTypeSales"]}[Content],
    DayTypeSales = Table.Group(Source, {"TypeDay"}, {{"TotalSales", each List.Sum([Sales])}},
    AddDataTypes = Table.TransformColumnTypes(DayTypeSales,{{"TypeDay", type text}, {"TotalSales", type number}})
in
    AddDataTypes
```

- You can use the **let expression anywhere**, here is an example of a let expression that is being used inside the Table.AddColumn function:



- You can add **notes** to M Code with the following syntax:
 - Start the note with two forward slashes, like `//`, followed by written note (before a hard return).
 - If you have multiple lines, start at the first line with forward slash and an asterisk, `/*`, followed by as many lines (hard returns) as you want, and then end with asterisk and forward slash, `*/`.
 - Example is shown on next page:



M Code Lookup

- Note: M Code is "base zero", which means: Row 1 = 0, Row 2 = 1, Row 3 = 2, and so on...
- M Code Lookup is similar to lookup use the Excel Worksheet function INDEX because INDEX can lookup value from a table based on a row index number and column index number, Like:

INDEX(Array, Index Number, Column Number)

- Two Types of M Code Lookup:

1. Row Index Lookup

- M Code Syntax:

Table { Row Index Number } [Field Name]

- This method uses a hard coded row number to determine the row position of the lookup. The row position does not change when the sort or content of the column changes.
- This method uses a hard coded row index number in curly brackets (called **positional index operator**) and a field name in square brackets (called a **field access operator**) to get a value from a table at the intersection of the row number and designated column.
- Examples:
 - **Source{0}[Content]** to get the first item from the Content field in the Source table.
 - **Source{0}** to get the first record from the Source table.
 - **Source[Content]** to get the Content field from the Source table as a list.
 - **Source[[Content]]** to get the Content field from the Source table as a field.
 - **{43,86}{0} = 43** (get the first item from the list).
 - **{43,86}{2} = error** (error because there is not a third item).
 - **{43,86}{2}? = null** (? is Optional Operator which avoids error and delivers a null when a match is not made)

2. Key Match Lookup:

- M Code Syntax:

Table{[Field Name = LookupValue]}[Field Name]

- This method uses an exact match lookup (called **key match lookup**) to dynamically determine the row position based on a lookup value in a specified column. The row position changes when the sort or content of the column changes. You use this type of lookup when you have a field that contains a unique list, or the field is the primary key in the table.
- This method uses a key match logical test inside square brackets (called **lookup operator**) inside the positional index operator to dynamically determine the row position of the lookup. The field name in the field access operator determines the column position. Then the intersecting value is retrieved from the table.
- Examples:
 - **Source{[Product="Sunshine"]}[Content]** to get the item in the Content field that corresponds to the row position of the Sunshine item in the Product field.
 - **Source{[Product="Sunshine"]}** to get the record in the Source table where the item in the Product field is equal to Sunshine. There are no duplicates in the Product field and so it works.
 - **Source{[Product="Quad"]}** yields the error: "The key matched more than one row in the table" because there was more than one "Quad" in the Product field.
 - **Source{[Product="Aspen"]}** yields the error: "The key didn't match any rows in the table" because there were no Aspen items in Product field.

- Pictorial Summaries of M Code Lookup on next page:

1) **Row Index Lookup** formula has this structure:

=Table { Row Index Number } [Field Name]

{ } = Row Positional
Index Operator

[] = Field Access
Operator

2) **Key Match Lookup** formula has this structure:

=Table { [Field Name = Lookup Value] } [Field Name]

{ } = Row Positional
Index Operator

[] = Field Access
Operator

[] = Lookup Operator when used inside of { }

1) **Lookup Record in table (Lookup Row):**

=Table { Row Index Number }

=Table { [Field Name = Lookup Value] }

2) **Lookup List from a Field in a table (Lookup Column):**

=Table [Field Name]

3) **Lookup Field from table (Lookup Column):**

=Table [[Field Name]]

4) **Lookup Fields from table:**

=Table [[Field Name], [Field Name2]]

Note:

Relational Algebra & Database Theory:

Projection (π , pi) = pick a column (attribute) from a table (relation)

Selection (σ , sigma) = select a row (tuple) from a table (relation)

Two types of Power Query Exact Match Two-Way Lookup:

1) **Row Index Lookup** = not dynamic because it always gets the first row

= Excel.CurrentWorkbook() { 0 } [Content]

[] = Field Access Operator

{ } = Row Positional Index Operator

Excel.CurrentWorkbook() = Table with Content and Name fields

2) **Key Match Lookup** = dynamic because it always gets the row with the table name "dSalesDiscount"

= Excel.CurrentWorkbook() { [Name="dSalesDiscount"] } [Content] → [] = Field Access Operator

[] = Lookup Operator when used inside of { }

{ } = Row Positional Index Operator

Excel.CurrentWorkbook() = Table with Content and Name fields

Primary Keys

- Primary Keys for Power Query Tables are nearly invisible in Power Query. And yet they play a significant role in doing Lookup in Power Query and for some calculations such as Group By Aggregate.
- There is no way to see the Primary Keys in the User Interface.
- You can determine if a Table has a Primary key by using the **Table.Keys** function, as shown in below example:
- Power Query will define a Primary Key in a Table in these situations:
 - If you use the Remove Duplicates feature, i.e., the Table.Distinct function.
 - If you are connected to a database like an SQL database and a Primary Key has been defined in a table that you import.
 - If you use **Table.AddKey** function to add a primary key to a table.
 - When you use Excel.CurrentWorkbook() function to import Excel Tables from the current workbook, the table of objects that the function creates, adds a primary key to the object name field names "Name".
- The main area in Power Query where a primary key comes into play is with the Drill Down feature.
 - If you Drill Down on a Primary Key, Key Match Lookup is preformed, like:

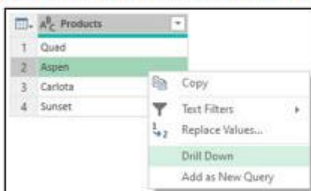
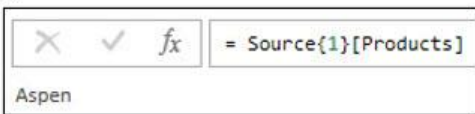
= Excel.CurrentWorkbook(){[Name="fTransactions"]}[Content]"

- If you Drill Down on a non-Primary Key, Row Index Lookup is preformed, like:


= Excel.CurrentWorkbook(){0}[Content]"

- Primary Key Example:


1) If you Drill Down on Aspen in Products Field when there is no Primary Key, Row Index Lookup is performed:

1)  2) 

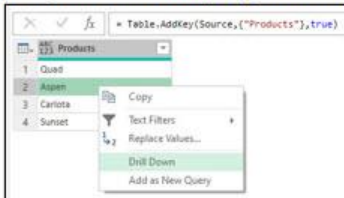

2) Check For Primary Key. If there is no Primary Key an empty list is returned:



3) You can add a Primary Key to a table with Table.AddKeys function:



4) Now if you Drill Down, Key Match Lookup is performed:

1)  2) 

Custom Functions

- An M Code Custom Function is a user-created function that defines the variables and the mapping for those variables to deliver a value.
- Custom functions can be created in any bit of M Code, but are most often created in three places:
 1. In a new query as a re-usable function (in a let expression that delivers a custom function value that can be used throughout the workbook).
 2. In function arguments that require functions
 3. As a step in a let expression
- Syntactical rules for defining a Custom Function are:
 - Type variables (function inputs) separated by commas in parentheses.
 - Type the go to operator: => (Equal, Greater Than).
 - Type a formula that uses zero or more of the variables.
 - Two Examples:
 - To add the inputs x and y:
(x, y) => x + y
 - Calculate the effective rate based on the inputs APR and Periods:
(APR, Periods) => Number.Power(1+APR/Periods,Periods)-1
- Example of a Custom Function in a Custom Column that helps to perform approximate match lookup:

Custom Function in second argument of Table.SelectRows function

```

each List.Last(
    Table.SelectRows(
        dDiscount,
        (IT) => IT[Sales] <= [Sales]
    )
)
  
```

Date	1.2 Sales	1.2 Discount-CustomFunction
1/20/23	3913.46	0.075

- Example of a let expression in a Custom Column that helps to perform approximate match lookup:

```

= Table.AddColumn(GetDiscountByCustomFunction, "Discount-LetExpression", each
    let Sales = [Sales]
    in
    List.Last(Table.SelectRows(dDiscount, each [Sales] <= Sales)[Discount]), type number)
  
```

1.2 Sales	1.2 Discount-CustomFunction	1.2 Discount-LetExpression
1/20/23	3913.46	0.075
1/20/23	774.61	0.025
1/20/23	2088.31	0.05
1/20/23	2515.81	0.075
1/20/23	473.81	0

- Example of a Re-usable Custom Function created in the Advanced Editor that calculates the effective rate:

```

let
    EffectiveRate = (APR, Periods) => Number.Power(1+APR/Periods,Periods)-1
in
    EffectiveRate
  
```

- You can define data types for your Custom Function like this:

```

let
    EffectiveRate = (APR as number, Periods as number) as number => Number.Power(1+APR/Periods,Periods)
in
    EffectiveRate
  
```

Defined number data type for APR variable
Defined number data type for Periods variable
Defined number data type for the output of the function

- You can add notes to your Custom Function:
 1. Use two forward slashes, like // , to add a single line with notes (one hard return)
 2. When you have multiple lines (multiple hard returns) use /* to start and then */ to close.
 3. Examples below:

```

let
    EffectiveRate = (APR, Periods) => Number.Power(1+APR/Periods,Periods)-1
    // Effective Rate based on APR (Annual Percentage Rate) and number of periods per years. It tells you
    // what the single period rate would be for the year when there are more periods in a year than one.
    /*
    APR = Annual Rate Period Year
    PeriodsPerYear = Compounding Periods Per Year
    */
in
    EffectiveRate
  
```

✓ No syntax errors have been detected.

Done Cancel

- Four Examples of an Effective Rate Custom Function:

Advanced Editor
Display Options ?

FVTable

```

let
    Source = Excel.CurrentWorkbook(){[Name="FVTable"]}[Content],
    AddDataTypes = Table.TransformColumnTypes(Source,{{"APR", type number}, {"Years", Int64.Type}, {"PeriodsPerYear", Int64.Type}, {"Invest", Int64.Type}}),
    1
    ERReach = Table.AddColumn(AddDataTypes, "ERReach", each Number.Power(1+[APR]/[PeriodsPerYear],[PeriodsPerYear])-1, type number),
    2
    ERUniversalFx = Table.AddColumn(ERReach, "ERUniversalFx", each EffectiveRate([APR], [PeriodsPerYear]), type number),
    // Effective Rate Math Formula: (1+i/n)^n
    ERQueryFx = (i as number,n as number) as number => Number.Power(1+i/n,n)-1, ← Defined Custom Function as query step
    3
    ERInvokeQueryFx = Table.AddColumn(ERUniversalFx,"ERInvokeQueryFx", each ERQueryFx([APR],[PeriodsPerYear]), type number),
    4
    ERInvokeQueryFxByCC = Table.AddColumn(ERInvokeQueryFx, "ERQueryFxByCC", each ERQueryFx([APR],[PeriodsPerYear]), type number)
in
    ERInvokeQueryFxByCC
  
```

Uses each keyword → 1

Invokes Universal Query Function → 2

Invokes function from above query step → 3

Invokes function from above query step in the Custom Column dialog box → 4

each and underscore

- each = syntactical shorthand for defining an unnamed function taking a single untyped variable named underscore _ .
- You can use each shorthand anywhere a function can be declared.
- each is often used to pass a function to an argument in another function, like Table.AddColumn.
- each can be thought of as "allowing you to make a calculation in each row of a table or list."
- The Underscore (_) can be thought of as extracting everything from the row the function is working in.
- Examples of equivalent Custom Functions with different syntax:
 1. = Table.SelectRows(Source, each [TypeDay] = "Workday")
 2. = Table.SelectRows(Source, (_) => _[TypeDay] = "Workday")
 3. = Table.SelectRows(Source, (x) => x[TypeDay] = "Workday")
- Examples of equivalent Custom Functions with different syntax to extract a record from a table in a Custom Column:
 1. = Table.AddColumn(CustomFunction, "Custom", each _)
 2. = Table.AddColumn(CustomFunction, "Custom", (_) => _)
 3. = Table.AddColumn(CustomFunction, "Custom", (Record) => Record)

Here is picture of advanced editor with above examples of the different syntax available for Custom Functions:



```
Advanced Editor

EachAndUnderScore

let
    Source = DayTypeSales,

    // Next Three Steps Filter Table, each using different syntax:
    FunctionWithEach = Table.SelectRows(Source, each [TypeDay] = "Workday"),

    FunctionWithUnderScore = Table.SelectRows(Source, (_) => _[TypeDay] = "Workday"),

    CustomFunction = Table.SelectRows(Source, (x) => x[TypeDay] = "Workday"),

    // Next Three Steps extract a record in each row of a Custom Column, each using different syntax:
    RecordWithEachAndUnderScore = Table.AddColumn(CustomFunction, "Custom", each _),

    RecordWithWhatEachAndUnderScoreReplce = Table.AddColumn(CustomFunction, "Custom", (_) => _),

    RecordWithCustomFunction = Table.AddColumn(CustomFunction, "Custom", (Record) => Record)
in
    RecordWithCustomFunction
```