



PowerQuery

A Beginners Course

01

Be able to familiarize with Query Editor layout

02

Be able to learn the fundamentals for PowerQuery

03

Be able to learn the fundamentals of M-Language

04

Be able to learn the fundamentals of Data Modelling

Course Objectives



Your Facilitator

- Franco Angelo Cipriano
- Founder of Cocotech Solutions
- Bachelor and Masters Degree in IT
- Have been using MS Tools since 2008
- Have been training MS Excel since 2016





Power Query



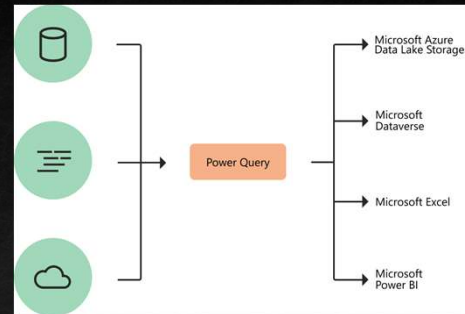
Introduction

Get to know PowerQuery

PAGE 4

Power Query

- This is a data transformation and data preparation engine.
- Allows you to perform ETL operation (Extract > Transform > Load)
- Has a GUI for getting data from sources and perform the transformation. Query Editor
- Essential tool in data modelling
- Included in 2016 and later versions. 2013 and earlier versions needs Add-ins to be installed.



How Power Query helps in ETL

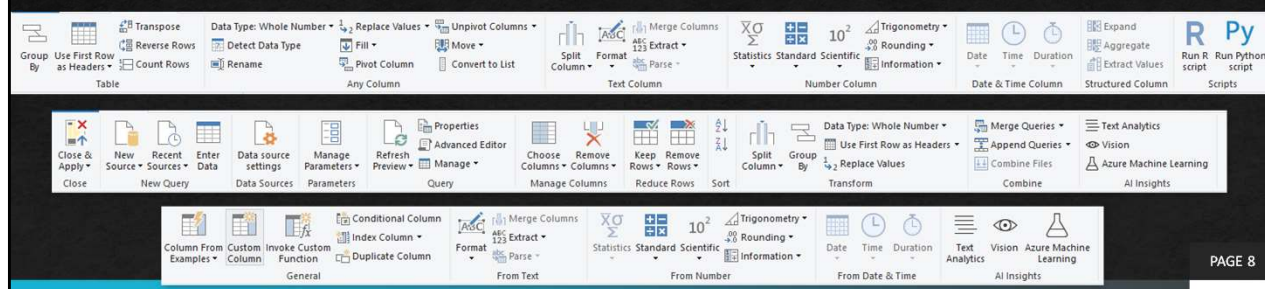
Existing challenge	How does Power Query help?
Finding and connecting to data is too difficult	Power Query enables connectivity to a wide range of data sources, including data of all sizes and shapes.
Experiences for data connectivity are too fragmented	Consistency of experience, and parity of query capabilities over all data sources.
Data often needs to be reshaped before consumption	Highly interactive and intuitive experience for rapidly and iteratively building queries over any data source, of any size.

How Power Query helps in ETL

Existing challenge	How does Power Query help?
Any shaping is one-off and not repeatable	Steps are recorded as you create your query in Power Query.
Volume (data sizes), velocity (rate of change), and variety (breadth of data sources and data shapes)	Power Query provides connectivity to hundreds of data sources and over 350 different types of data transformations for each of these sources, you can work with data from any source and in any shape.

Power Query Editor

- Is the GUI (Graphical User Interface) used to transform your query.
- This makes transformation a lot easier. No coding needed.
- Power Query converts steps to M Language automatically
- Included pre-built transformation engine functions

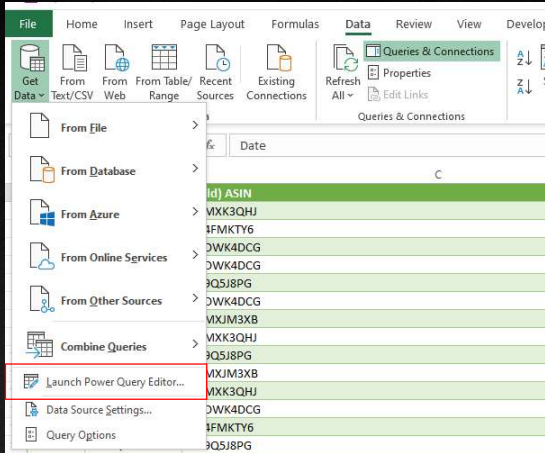


Power Query M Language

- The Power Query engine uses a scripting language behind the scenes for all Power Query transformations: the Power Query M formula language, also known as M.
- Anything that happens in the query is ultimately written in M.
- This allows you to customize the query and build a more dynamic query.

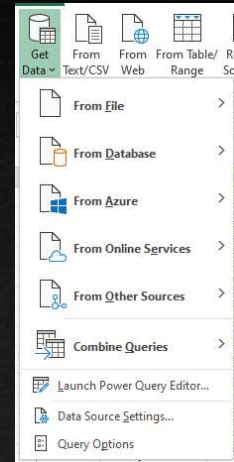
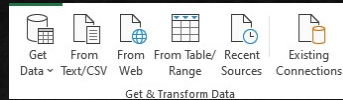
```
let
    Source = Folder.Files("C:\Cocotech\Training\Practice Exercise\VBA\activity"),
    #"Filtered Rows" = Table.SelectRows(Source, each ([Extension] = ".csv")),
    #"Replaced Value" = Table.ReplaceValue(#"Filtered Rows", ".csv", "", Replacer.ReplaceText, {"Name"}),
    #"Filtered Hidden Files1" = Table.SelectRows(#"Replaced Value", each [Attributes][Hidden]? <> true),
    #"Invoke Custom Function1" = Table.AddColumn(#"Filtered Hidden Files1", "Transform File", each #"Transform File"([Content])),
    #"Removed Other Columns1" = Table.SelectColumns(#"Invoke Custom Function1", {"Name", "Transform File"}),
    #"Expanded Table Column1" = Table.ExpandTableColumn(#"Removed Other Columns1", "Transform File", Table.ColumnNames(#"Transform File"("#Sa
    #"Removed Columns" = Table.RemoveColumns(#"Expanded Table Column1", {"Column1"}),
    #"Filtered Rows2" = Table.SelectRows(#"Removed Columns", each ([Column2] <> "")),
    #"Promoted Headers" = Table.PromoteHeaders(#"Filtered Rows2", [PromoteAllScalars=true]),
    #"Renamed Columns" = Table.RenameColumns(#"Promoted Headers", {"Feb-2021", "Date"}),
    #"Filtered Rows1" = Table.SelectRows(#"Renamed Columns", each ([Title] <> "SKU" and [Title] <> "Title"))
in
    #"Filtered Rows1"
```

Opening the Query Editor



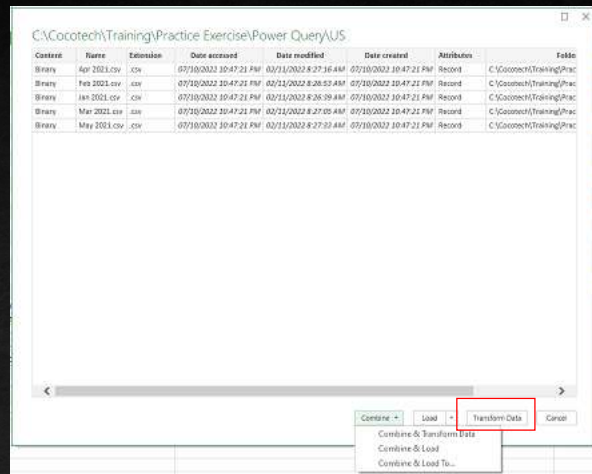
Get Data

- You can get data from multiple sources
- You can transform this data and combine them using Power Query



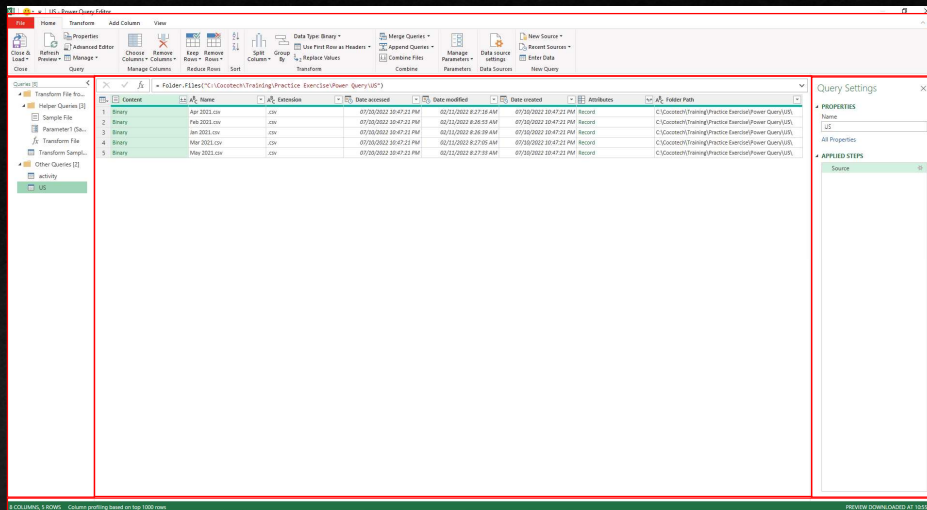
Transforming Data

- We start by selecting the data source
- Click the Transform Data
- Or select Combine & Transform Data
- Selecting Combine & Load will automatically load your data in an MS Excel worksheet



<https://services.odata.org/V4/Northwind/Northwind.svc/>

Power Query Editor



1.Ribbon: the ribbon navigation experience, which provides multiple tabs to add transforms, select options for your query, and access different ribbon buttons to complete various tasks.

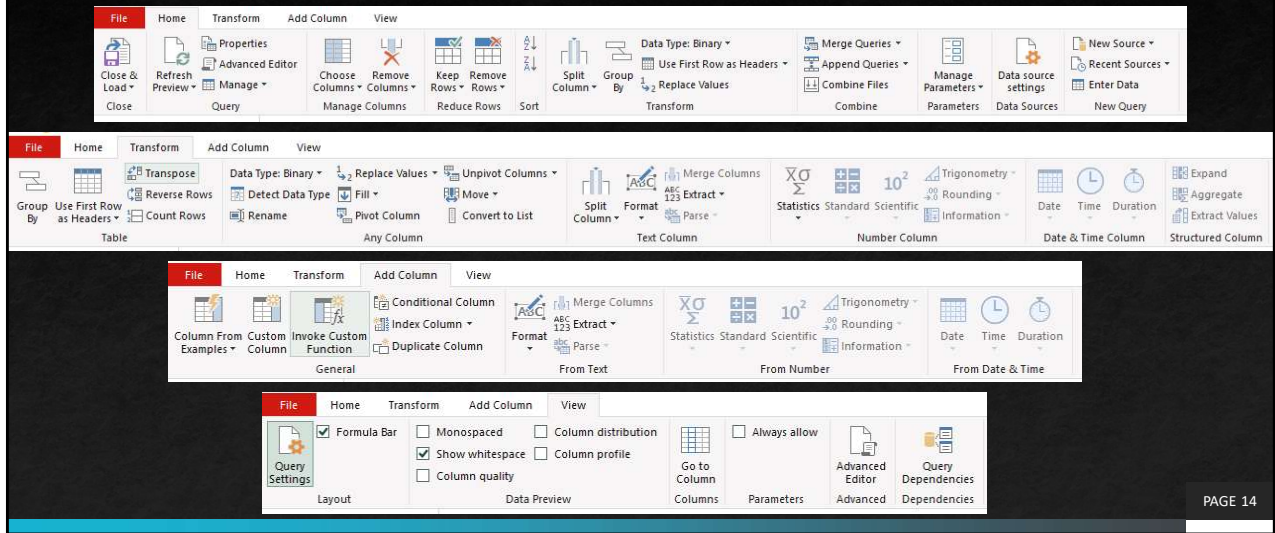
2.Queries pane: a view of all your available queries.

3.Current view: your main working view, that by default, displays a preview of the data for your query. You can also enable the [diagram view](#) along with the data preview view. You can also switch between the [schema view](#) and the data preview view while maintaining the diagram view.

4.Query settings: a view of the currently selected query with relevant information, such as query name, query steps, and various indicators.

5.Status bar: a bar displaying relevant important information about your query, such as execution time, total columns and rows, and processing status. This bar also contains buttons to change your current view.

Ribbon



- Home
- Transform
- Add Column
- View

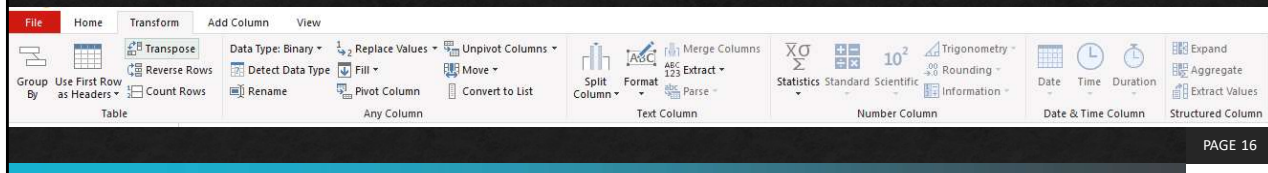
Home Ribbon

- This is where you can:
 - Manage Query
 - Remove rows and columns
 - Transform columns
 - Combine Queries
 - Manage Parameters
 - Manage data source



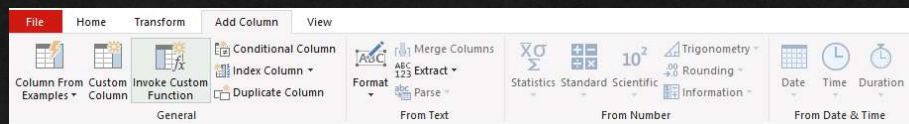
Transform Ribbon

- This ribbon provides you more commands for transformation.
- This allows you to transform the whole table



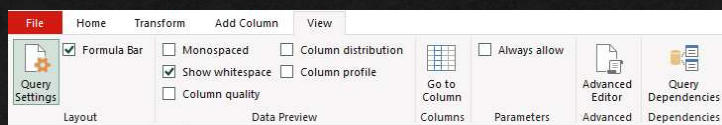
Add Column Ribbon

- This allows you to add different types of columns



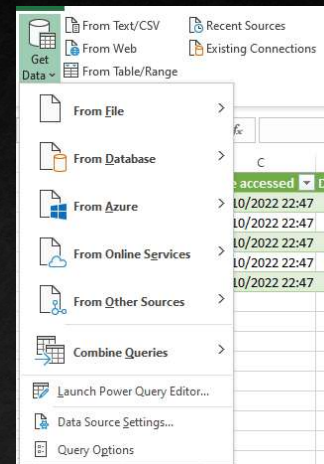
View Ribbon

- Allows you to manage the view of your query editor



Creating a query

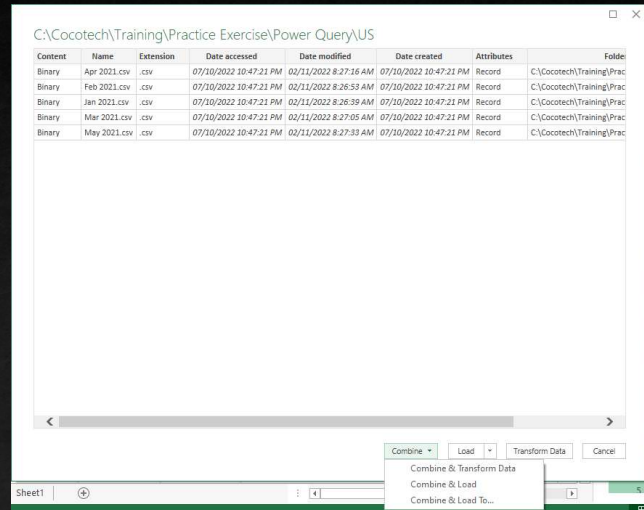
- You start creating query by selecting your data source
- You can only select one source at a time, then combine them later



<http://finance.yahoo.com/q/hp?s=MSFT>

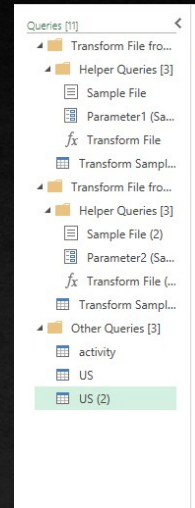
Creating a query

- You will see a preview of the data available from your source.
- You have an option to combine them right away or transform the data
- Selecting Transform Data will take you to the Query Editor



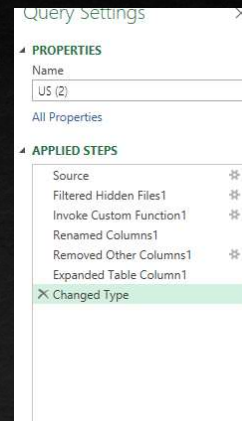
Queries

- You will see available queries in the Queries Panel
- You can delete, duplicate, move query



Applied Steps

- This is the recording of steps performed to transform the query
- You can go back to the step and change it



Demo and explain applied steps

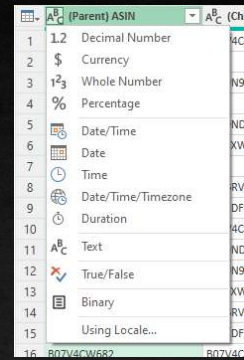
Let's Try

Transforming Columns

- Power Query allows you to transform the columns according to your data model
- Transforming allows you to remove, add, or edit columns, while retaining the original state of the columns from the source file.

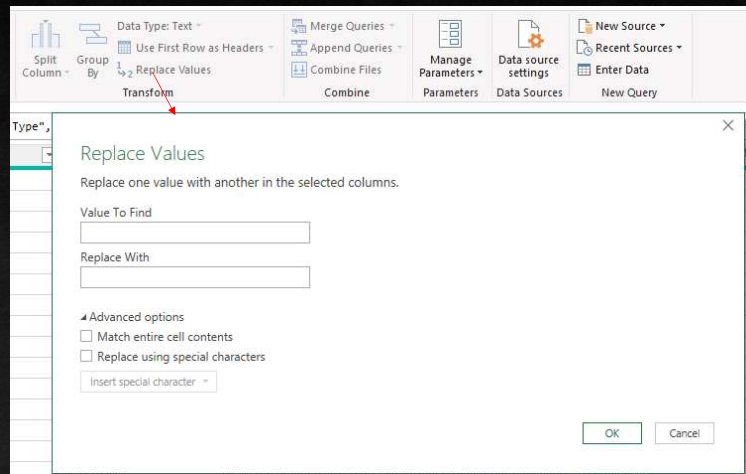
Data Types

- The column can be assigned with a data type.
- It is important to select the right data type for the column
- Power Query can detect data type of the value in the column



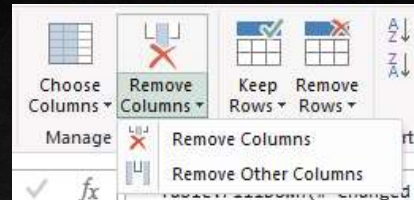
Replacing Values

- This allows you to find and replace values in a each row



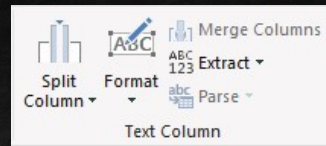
Removing Columns

- Allows you to keep or remove selected columns



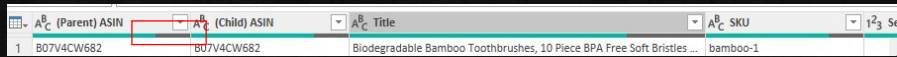
Text Columns

- Split Columns
- Format Columns
- Extract characters
- Parsing



NULL Rows

- Power Query gives you a hint that your rows have NULL or blank values
- The indicator bar shows that there is a blank or NULL

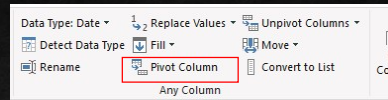


The screenshot shows a table with the following columns: (Parent) ASIN, (Child) ASIN, Title, and SKU. The first row contains the values B07V4CW682, B07V4CW682, Biodegradable Bamboo Toothbrushes, 10 Piece BPA Free Soft Bristles ..., and bamboo-1. A red box highlights the (Child) ASIN cell, which has a small indicator bar on its right side, signifying a NULL or blank value.

	(Parent) ASIN	(Child) ASIN	Title	SKU
1	B07V4CW682	B07V4CW682	Biodegradable Bamboo Toothbrushes, 10 Piece BPA Free Soft Bristles ...	bamboo-1

Pivot Columns

- Allows you to create a table that contains an aggregate value for each unique value in a column.

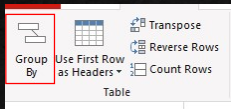


The diagram shows a transformation of a data table. On the left, a table with 9 rows and 2 columns is shown. The first column is labeled 'Attributes' and contains values A1, A2, A3, A1, A2, A3, A1, A2, A3. The second column is labeled 'Values' and contains values V1, V2, V3, V4, V5, V6, V7, V8, V9. An arrow points to the right, where a pivot table is shown. The pivot table has 3 columns labeled 'Attributes' (A1, A2, A3) and 3 rows labeled 'Values' (V1, V4, V7). The cells in the pivot table contain the following values: (A1, V1) is V1, (A1, V4) is V4, (A1, V7) is V7, (A2, V1) is V2, (A2, V4) is V5, (A2, V7) is V8, (A3, V1) is V3, (A3, V4) is V6, and (A3, V7) is V9.

Demo Pivot table

Group By

- group values in various rows into a single value by grouping the rows according to the values in one or more columns. You can choose from two types of grouping operations:
 - Column groupings.
 - Row groupings.

A screenshot of the 'Group By' dialog box. The title is 'Group By' with a close button (X) in the top right. Below the title is the instruction 'Specify the column to group by and the desired output.' There are two radio buttons: 'Basic' (selected) and 'Advanced'. Below this is a dropdown menu showing '(Parent) ASIN'. Underneath are three fields: 'New column name' with the text 'Count', 'Operation' with a dropdown menu showing 'Count Rows', and 'Column' with an empty dropdown menu. At the bottom right are 'OK' and 'Cancel' buttons.

Advanced Group By

- Allows you to group multiple Columns and perform multiple aggregations.

Group By

Specify the columns to group by and one or more outputs.

Basic Advanced

Date

(Child) ASIN

Add grouping

New column name	Operation	Column
Count	Count Rows	
Total Sales	Sum	Total Orders

Add aggregation

OK Cancel

Let's Try

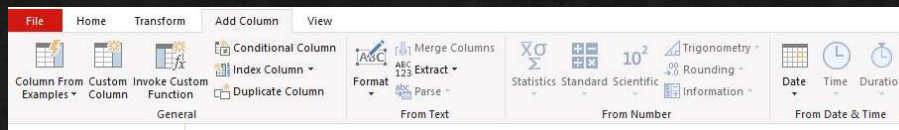


PAGE 33

- Create a duplicate of CA query
- Group the totals sales by Child ASIN

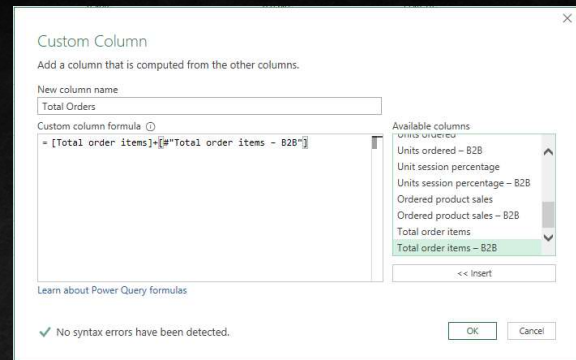
Adding Columns

- Allows you to add new columns without changing the raw data
- You can add custom columns using M Language
- You can add column using an IF statement
- You can add columns based on values



Add Custom Column

- This will create a new column based on the formula you specified.



Conditional Columns

- Allows you to add new columns based on a conditional statement

Add Conditional Column

Add a conditional column that is computed from the other columns or values.

New column name
Acq?

Column Name	Operator	Value	Output
Total Orders	is greater than	Average	Acq
Else If			

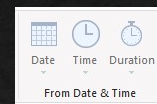
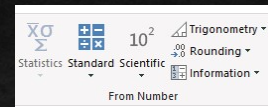
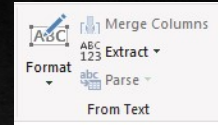
Add Clause

Else
Acq

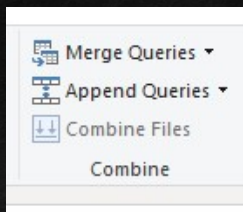
OK Cancel

Other Add Columns

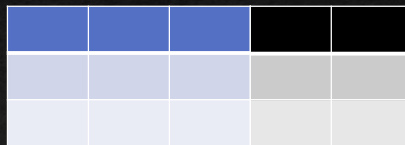
- From Text
- From Number
- From Date & Time



Combining Queries



Merge Queries



Blue	Blue	Blue	Black	Black
Light Blue	Light Blue	Light Blue	Light Grey	Light Grey
Light Blue	Light Blue	Light Blue	Light Grey	Light Grey

Append Queries



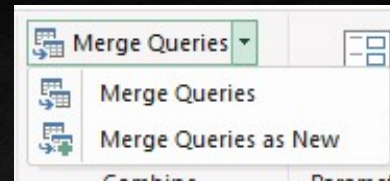
Blue	Blue	Blue	Black	Black
Light Blue	Light Blue	Light Blue	Black	Black
Light Blue	Light Blue	Light Blue	Black	Black
Black	Black	Black	Black	Black
Black	Black	Black	Black	Black
Black	Black	Black	Black	Black

There are two primary ways of combining queries: *merging* and *appending*.

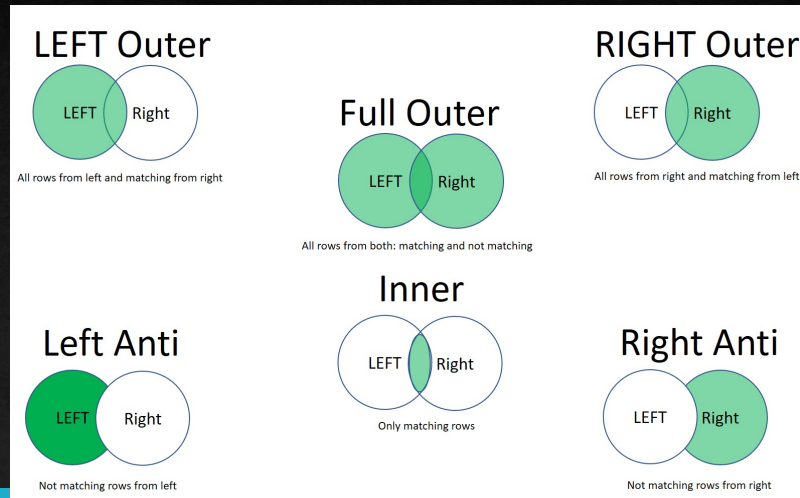
- When you have one or more columns that you'd like to add to another query, you *merge* the queries.
- When you have additional rows of data that you'd like to add to an existing query, you *append* the query.

Merge Queries

- Joins the records from one query to the records in another by matching on a unique identifier.
- The kind of join you apply is important because it determines which records are returned from each data set.
- You can merge it to an existing query
- You can merge it as a new query



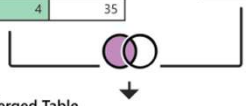
JOIN Kinds



Unlike SQL, you can only JOIN 2 tables at a time.

LEFT Outer Join

Left Table			Right Table	
Date	CountryID	Units	ID	Country
1/1/2020	1	40	1	USA
1/2/2020	1	25	2	Canada
1/3/2020	3	30	3	Panama
1/4/2020	4	35		

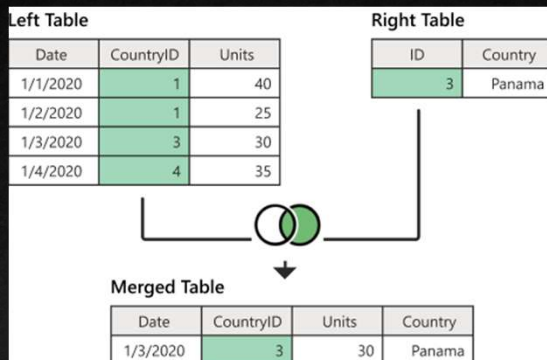


Merged Table			
Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	<i>null</i>

left outer join keeps all the rows from the left table and brings in any matching rows from the right table

<demo>

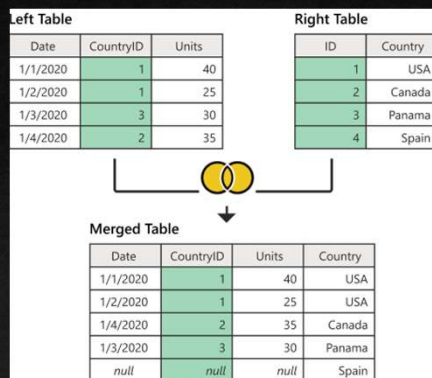
RIGHT Outer Join



right outer join, which keeps all the rows from the right table and brings in any matching rows from the left table

<demo>

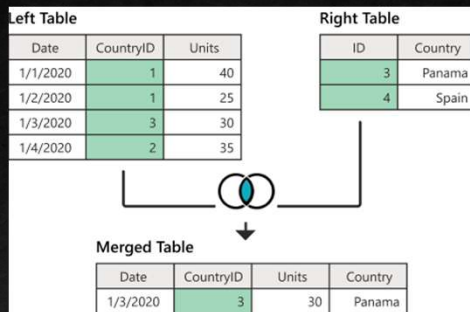
FULL Outer Join



full outer join, which brings in all the rows from both the left and right tables

<demo>

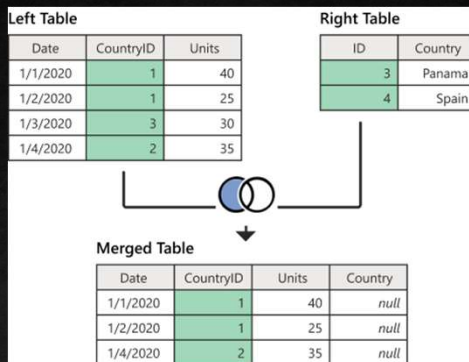
INNER Join



inner join, which brings in only matching rows from both the left and right tables.

<demo>

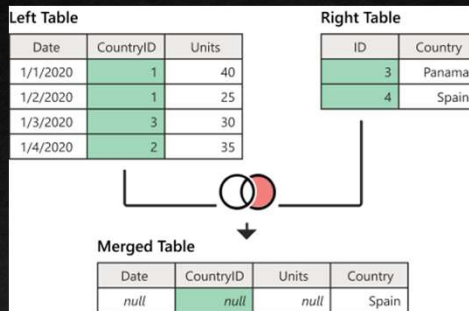
LEFT anti join



left anti join, which brings in only rows from the left table that don't have any matching rows from the right table

<demo>

RIGHT anti join



right anti join, which brings in only rows from the right table that don't have any matching rows from the left table

<demo>

Fuzzy Matching

	Questions
1	Answer
2	Apple
3	Aple
4	Apol
5	Apples
6	Water melon
7	watermelon
8	watermeln
9	Banana
10	Bananan
11	Bnana

	Fruits
1	Apple
2	Banana
3	Watermelon

Fuzzy merge is a smart data preparation feature you can use to apply fuzzy matching algorithms when comparing columns, to try to find matches across the tables that are being merged

<demo>

Append Queries

A	B	C
1	1	1
2	2	2
3	3	3

A	B	D
4	4	4
5	5	5

A	B	C	D
1	1	1	null
2	2	2	null
3	3	3	null
4	4	null	4
5	5	null	5

The append operation creates a single table by adding the contents of one or more tables to another, and aggregates the column headers from the tables to create the schema for the new table.

NOTE

When tables that don't have the same column headers are appended, all column headers from all tables are appended to the resulting table. If one of the appended tables doesn't have a column header from other tables, the resulting table shows *null* values in the respective column, as shown in the previous image in columns C and D.

<demo>

Let's Try

```
c=function(b){this.element=a(D)};c.VERSION=
),d=b.data("target");if(d||(d=b.attr("href"),d=d&&d.repla
ent("hide.bs.tab",{relatedTarget:b[0]}),g=a.Event("show.bs
({}){var h=a(d);this.activate(b.closest("li"),c),this.act
"shown.bs.tab",relatedTarget:e[0]}})}},c.prototype.act
.removeClass("active").end().find('[data-toggle="tab"]').
!0),h?(b[0].offsetWidth,b.addClass("in")):b.removeClass(
a-toggle="tab"]').attr("aria-expanded",!0),e&&e()}var g=
("> .fade").length);g.length&&h?g.one("bsTransitionEnd"
ab;a.fn.tab=b,a.fn.tab.Constructor=c,a.fn.tab.noConflict
ument).on("click.bs.tab.data-api",[data-toggle="tab"]
nction b(b){return this.each(function(){var d=a(this),e
b({}))}var c=function(b,d){this.options=a.extend({},c.D
s.checkPosition,this)).on("click.bs.affix.data-api",a.f
nedOffset=null,this.checkPosition());c.VERSION="3.3.7"
m(a,b,c,d){var e=this.$target.scrollTop(),f=this.$elem
s.affixed)return null!=c?!(e+this.unpin<=f.top)\&&"bot
op":null!=d&&i+j>=a-d&&"bot"
```

Introduction to M Language

M Language

- M is a functional programming language
 - computation through evaluation of mathematical functions
 - Programming involves writing expressions instead of statements
 - M does not support changing state or mutable data
 - Every query is a single expression that returns a single value
 - Every query has a return type
- Get Started with M
 - Language is case sensitive
 - It's all about writing expressions
 - Query expressions can reference other queries by name

Let Statement

- Queries usually created using let statement
 - Allows a single expressions to contain inner expressions
 - Each line in **let** block represents a separate expression
 - Each line in **let** block has variable which is named step
 - Each line in **let** block requires comma at end except for last line
 - Expression inside **in** block is returned as **let** statement value

APPLIED STEPS

Source	✖
Filtered Rows	✖
Split Column by Delimiter	✖
Renamed Columns2	
Filtered Hidden Files1	✖
Invoke Custom Function1	✖
Renamed Columns1	
Removed Other Columns1	✖
Expanded Transform File	✖
Renamed Columns3	
Filtered Rows1	✖
Changed Type	
Replaced Value	✖
Replaced Value1	✖
Changed Type1	
Renamed Columns	
Filtered Rows2	
Grouped Rows	✖

Comments and Variable names

- M supports using C style comments
 - Multiline comments created using `/**/`
 - Single line comments created using `//`
- Variable names with spaces must be enclosed in `#"`
 - Variable names with spaces created automatically by query designer

```
/**  
  This is my most excellent query  
*/  
let  
  var1 = 42, // the secret of life
```

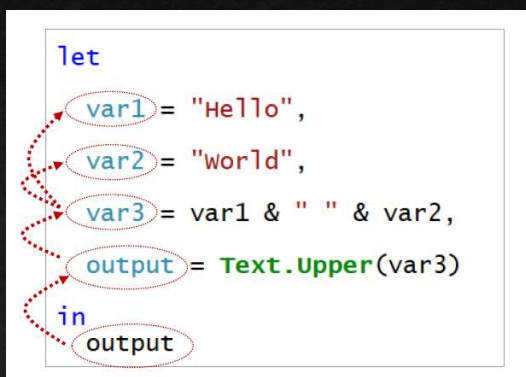
```
let  
  var1 = "Spaces in ",  
  #"var 2" = "variable names ",  
  #"Bob's your unkle" = "are evil",  
  #"Kitchen sink" = var1 & #"var 2" & #"Bob's your unkle"  
in  
  #"Kitchen sink"
```

APPLIED STEPS

```
var1  
var 2  
Bob's your unkle  
X Kitchen sink
```

Flow of Statement Evaluation

- Evaluation starts with expression inside in block
 - Expression evaluation triggers other expression evaluation



Will This M Code Work?

- Yes, the Mashup Engine has no problem with this
 - The order of expressions in let block doesn't matter
 - However, the Power Query designer might get confused

```
let
    var4 = Text.Upper(var3),
    var3 = var1 & " " & var2,
    var2 = "world",
    var1 = "Hello"
in
    var4
```

QUERY SETTINGS

PROPERTIES

Name

Hello World Reversed

[All Properties](#)

APPLIED STEPS

Hello World Reversed



Data Types

Values

- This is the data produced by evaluating an expression.

Kind	Literal
<i>Null</i>	<code>null</code>
<i>Logical</i>	<code>true</code> <code>false</code>
<i>Number</i>	<code>0</code> <code>1</code> <code>-1</code> <code>1.5</code> <code>2.3e-5</code>
<i>Time</i>	<code>#time(09,15,00)</code>
<i>Date</i>	<code>#date(2013,02,26)</code>
<i>DateTime</i>	<code>#datetime(2013,02,26, 09,15,00)</code>
<i>DateTimeZone</i>	<code>#datetimezone(2013,02,26, 09,15,00, 09,00)</code>
<i>Duration</i>	<code>#duration(0,1,30,0)</code>

<i>Text</i>	<code>"hello"</code>
<i>Binary</i>	<code>#binary("AQID")</code>
<i>List</i>	<code>{1, 2, 3}</code>
<i>Record</i>	<code>[A = 1, B = 2]</code>
<i>Table</i>	<code>#table({ "X", "Y" }, {{0,1}, {1,0}})</code>
<i>Function</i>	<code>(x) => x + 1</code>
<i>Type</i>	<code>type { number }</code> <code>type table [A = any, B = text]</code>

Types

- **Primitive types**, which classify primitive values (*binary, date, datetime, datetimezone, duration, list, logical, null, number, record, text, time, type*) and also include a number of abstract types (function, table, any, and none)
- **Record types**, which classify record values based on field names and value types
- **List types**, which classify lists using a single item base type

Types

- **Function types**, which classify function values based on the types of their parameters and return values
- **Table types**, which classify table values based on column names, column types, and keys
- **Nullable types**, which classifies the value null in addition to all the values classified by a base type
- **Type types**, which classify values that are types

M Type System

- Built in types
 - any, none, null, logical, number, text, binary, time, date, datetime, datetimezone, duration
- Complex types
 - list, record, table, function
- User defined types
 - You can create custom types for records and tables

```
CustomerRecordType = type [FirstName = text, LastName = text],
```

Examples of programming with M Datatypes

```
let
  // primitives
  var1 = 123, // number
  var2 = true, // boolean
  var3 = "hello", // text
  var4 = null, // null

  // creating lists
  list1 = {1, 2, 3}, // list of three numbers

  // accessing list elements
  var5 = list1[1],

  // create records
  record1 = [ FirstName="Soupy", LastName="Sales", ID=3 ],

  // accessing records
  var6 = record1[FirstName],

  // table
  table1 = #table( {"A", "B"}, { {1, 2}, {3, 4} } ),

  // creating function
  function1 = (x) => x * 2,

  // calling function
  output = function1(var1)
in
  output
```

let

```
var =123,
var2= true,
var3="text value",
var4=null,
```

```
function1 = (x) =>x*2,
output = function1(var)
```

in

```
output
```

Initializing Dates and Times

```
// time
var1 = #time(09,15,00),

// date
var2 = #date(2013,02,26),

// date and time
var3 = #datetime(2013,02,26, 09,15,00),

// date and time in specific timezone
var4 = #datetimezone(2013,02,26, 09,15,00, 09,00),

// time durement
var5 = #duration(0,1,30,0),
```

Operators

- Types of operator arranged by precedence
 - Primary – i, (), x[i],x{y}, x(..),{x,y,...},[i=x...]
 - Unary - +x, -x, not x
 - Metadata – x meta y
 - Multiplicative - *,/
 - Additive - +,-
 - Relational - <, >, <=, >=
 - Equality - =, <>
 - Type assertion – x as y
 - Type conformance – x is y
 - Logical AND – x and y
 - Logical OR – x or y

Lists

- List is a single dimension array
 - Literal list can be created using `{ }` operators
 - List elements accessed using `{ }` operator and zero-based index

```
let
  RatPack = { "Frank", "Dean", "Sammy" } ,
  FirstRat = RatPack{0} ,
  SecondRat = RatPack{1} ,
  ThirdRat = RatPack{2} ,
  output = FirstRat & ", " & SecondRat & " and " & ThirdRat
in
output
```

- User `{ }?` to avoid error when index range is out-of-bounds

```
Rat4 = RatPack{4}, // error - index range out of bounds
Rat5 = RatPack{5}? , // no error - Rat5 equals null
```


Text.Select

- Text.Select can be used to clean up text value
 - You create a list of characters to include

```
// take a text value with unwanted characters
input = "!!My text has some @bad things !&A",

// get upper and lower case letters
set1 = {"A".."Z"},
set2 = {"a".."z"},

// get digits 0-9 and convert to text
set3 = List.Transform({0..9}, each Number.ToText(_)),

// add any other allowed characters
set4 = {" ", "-", "_", "."},

// combine all allowed characters in single list
allowedChars = set1 & set2 & set3 & set4,

// call Text.Select to strip out unwanted characters
output = Text.Select(input, allowedChars)
```

Records

- Record contains fields for single instance of entity

```
// create records by using [] and defining fields
Person1 = [FirstName="Chris", LastName="webb"],
Person2 = [FirstName="Reza", LastName="Rad"],
Person3 = [FirstName="Matt", LastName="Masson"],

// access field inside a record using [] operator
FirstName1 = Person1[FirstName],
LastName2 = Person2[LastName],
```

- You must often create records to call M library functions

```
// create a record to define HTTP request headers
RequestHeaders = [ Accept="application/json",
                  #"odata-maxversion"="4.0" ],

// create a second record which contains the first record
OptionsRecord = [ Headers=RequestHeaders ],

// pass the second record as parameter to web.contents
Response = web.contents(ur[, OptionsRecord),
```

Combination Operator (&)

- Used to combine strings, arrays and records

```
// text concatenation: "ABC"  
var1 = "A" & "BC",  
  
// list concatenation: {1, 2, 3}  
var2 = {1} & {2, 3},  
  
// record merge: [ a = 1, b = 2 ]  
var3 = [ a = 1 ] & [ b = 2 ],
```

Table.FromRecords

- Table.FromRecords can be used to create table
 - Table columns are not strongly typed

```
Query1
let
    CustomersTable = Table.FromRecords({
        [{"First Name" = "Bruce", LastName="Wayne"}],
        [{"First Name" = "Clark", LastName="Kent"}],
        [{"First Name" = "Barry", LastName="Allen"}],
        [{"First Name" = "Dianna", LastName="Prince"}]
    })
in
    CustomersTable
```

Creating a table

```
let
    CustomersTable = Table.FromRecords({
        [{"First Name" = "Bruce", LastName="Wayne"}],
        [{"First Name" = "Clark", LastName="Kent"}],
        [{"First Name" = "Barry", LastName="Allen"}],
        [{"First Name" = "Dianna", LastName="Prince"}]
    })
in
    CustomersTable
```

IF Statement

- *if-expression* selects from two expressions based on the value of a logical input value and evaluates only the selected expression.

```
if if-condition then true-expression else false-expression
```

```
if 2 > 1 then 2 else 1 // 2  
if 1 = 1 then "yes" else "no" // "yes"
```

NESTED IF

- Allows you to create multiple condition
- Conditional Column will generate the IF or Nested IF Statement

```
#"Added Conditional Column" = Table.AddColumn("#Changed Type2", "Custom", each  
if [Country] = "CA" then "CAD"  
else if [Country] = "UK" then "GBP"  
else if [Country] = "US" then "USD"  
else null),
```

if [Country] = "CA" then "CAD" else if [Country] = "UK" then "GBP" else if [Country] = "US" then "USD" else null

Catching Errors

- Error handling in M done using **try .. Otherwise**

```
try Date.FromText([Raw Date]) otherwise null
```

- Error handling can avoid evaluation errors

```
AddedDateColumn1 = Table.AddColumn(Source, "Date1", each Date.FromText([Raw Date])),  
AddedDateColumn2 = Table.AddColumn(AddedDateColumn1, "Date2", each ( try Date.FromText([Raw Date]) otherwise null ) )
```

- Expression causing errors replace with value such as **null**

	Raw Date	Date1	Date2
1	Feb 30, 2019	Error	null
2	March 4, 2019	3/4/2019	3/4/2019
3	Cinco de mayo, 2019	Error	null
4	6/4/2019	6/4/2019	6/4/2019
5	07-04-2019	7/4/2019	7/4/2019

Creating User-defined Types

- M allows you to create user defined types
 - Here is a user defined type for a record and a table

```
customerRecordType = type [FirstName = text, LastName = text],
customerTableType = type table CustomerRecordType,
```

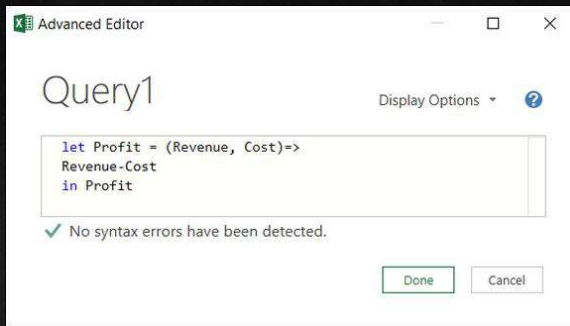
- User defined table used to create table with strongly typed columns

```
let
  CustomerRecordType = type [FirstName = text, LastName = text],
  CustomerTableType = type table CustomerRecordType,
  CustomersTable =
    #table(CustomerTableType, {
      { "Matt", "Masson" },
      { "Chris", "Webb" },
      { "Reza", "Rad" },
      { "Chuck", "Sterilicious" }
    })
in
  CustomersTable
```

	FirstName	LastName
1	Matt	Masson
2	Chris	Webb
3	Reza	Rad
4	Chuck	Sterilicious

Custom Functions

- This allows you to create functions that you can reuse in different queries.
- It can have a parameter, or with no parameter



The screenshot shows a window titled "Advanced Editor" with a tab labeled "Query1". The window contains a text area with the following code:

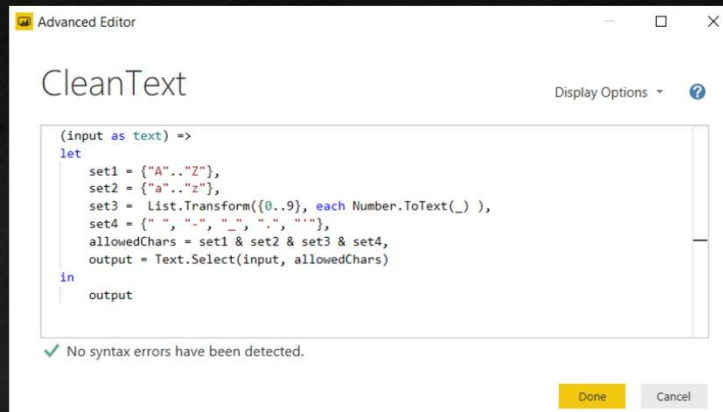
```
let Profit = (Revenue, Cost)=>
Revenue-Cost
in Profit
```

Below the text area, there is a green checkmark and the text "No syntax errors have been detected." At the bottom of the window, there are two buttons: "Done" and "Cancel".

Demo

Creating a Function Query

- You can write a function by wrapping the whole **let** statement

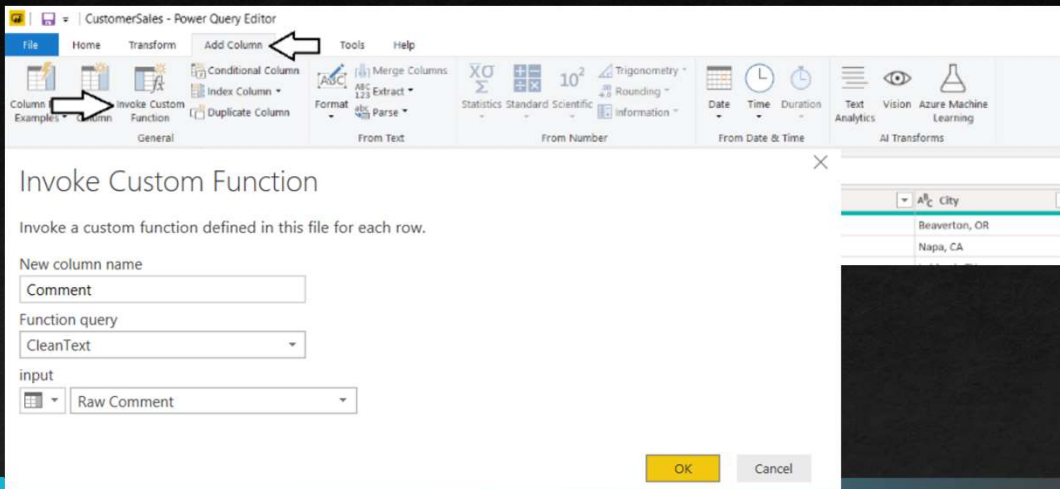


```
(input as text) =>
let
    set1 = {"A".."Z"},
    set2 = {"a".."z"},
    set3 = List.Transform({0..9}, each Number.ToText(_)),
    set4 = {" ", "-", "_", ".", ""},
    allowedChars = set1 & set2 & set3 & set4,
    output = Text.Select(input, allowedChars)
in
    output
```

✓ No syntax errors have been detected.

Done Cancel

Calling a Function Query



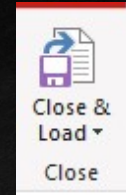
Parameters

- A parameter query is a kind of query that relies on one or more parameters to run
- A parameter query is one where you provide the parameters

Let us Try

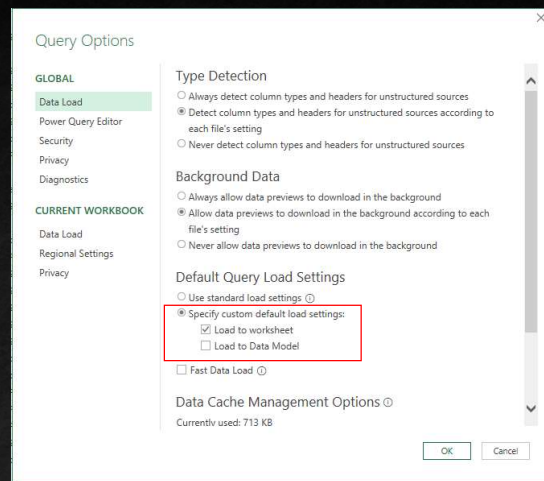
Loading to worksheet

- To load your queries you can click Close & Load button
- Unlike PowerBI, MS Excel Power Query will load all your query to your worksheet.



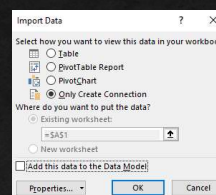
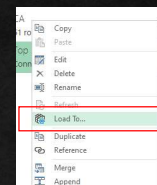
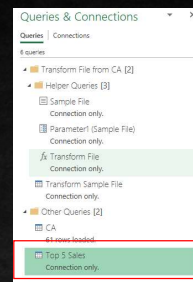
Selecting which to load

- Disabling Load to Worksheet by default
- File > Options & Settings > Query Options > Data Load tab
- Under Default Query Load Settings. Load to Worksheet is checked by default.
- Uncheck to refrain loading queries to Worksheet



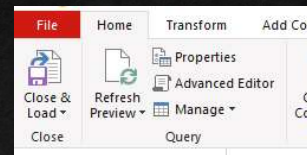
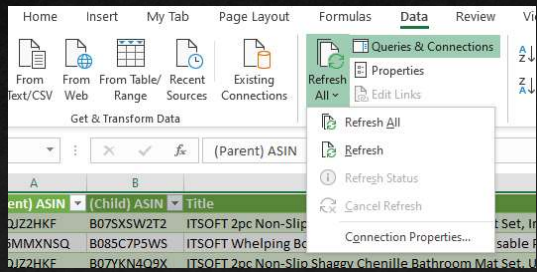
Selecting which to load

- Queries not loaded will show “Connection only” status.
- Right click the query you want to load, and select **Load To...**
- Select where and how you want to load your data



Refresh Data Source

- You can refresh your data from the worksheet or from the query editor
- This will perform the steps you have created, no need to redo the steps.
-



Let's Try

THANK YOU!



PAGE 83