M 365 Excel Class Video 13: Power Query to Import & Transform Data, Excel and Power BI Desktop

p

Import Dynamic Folder Path from Current Excel Workbook File:

1. This formula will show the folder path for the file that the formula lives in:

=TEXTBEFORE(CELL("filename",A1),"[")

Or

=LEFT(CELL("filename",A1),SEARCH("[",CELL("filename",A1))-1)

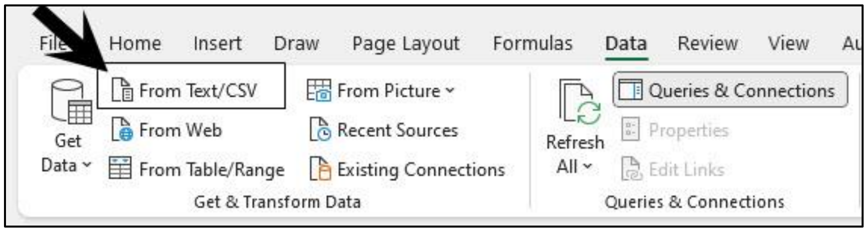
2. If you name the cell with the above formula "FolderPath", the resultant folder path can be used to import all files from the folder regardless of where you move the folder using this Power Query formula in a blank query:

= Folder.Files(Excel.CurrentWorkbook(){[Name="FolderPath"]}[Content]{0}[Column1])

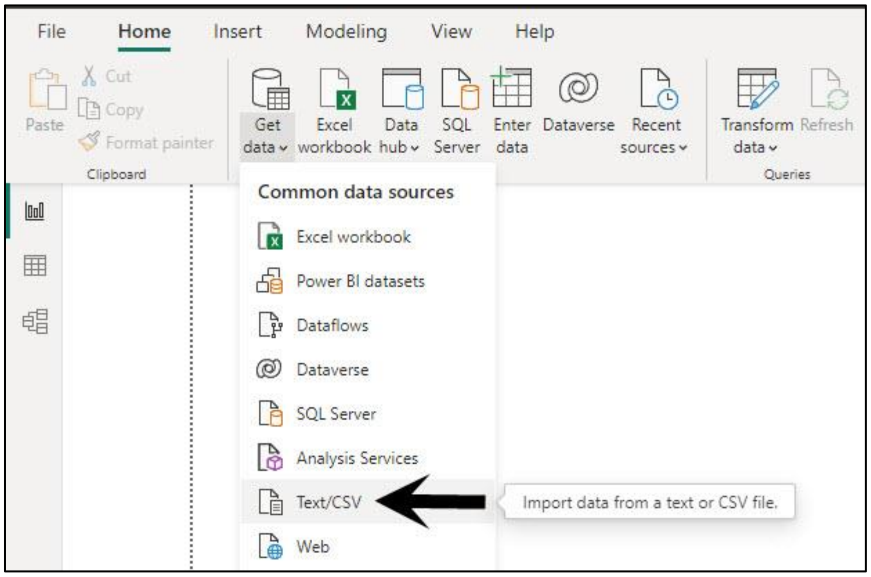
3. In Excel, together these two formulas will always deliver a table of Excel Objects (Excel Tables, Defined Names and Arrays that have previously been imported into the Power Query Editor) regardless of where the folder is moved to.
4. These formulas are not available in Power BI Desktop.

Import Single Object Files: TXT and CSV

1. Text files with the extension ".txt" (Text) are Tab delimited files that are designed to move data from one system to another.
2. Csv files with the extension ".csv" (Comma Separated Values) are Comma delimited files that are designed to move data from one system to another.
3. Here are the two connector paths in Power Query:
   - Excel:

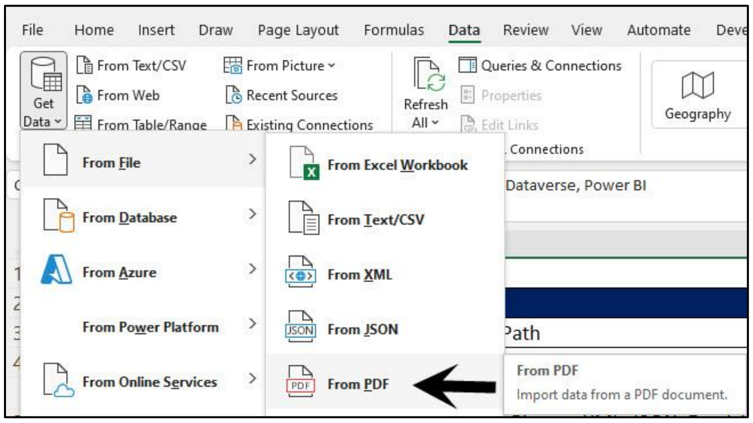


   - Power BI Desktop:

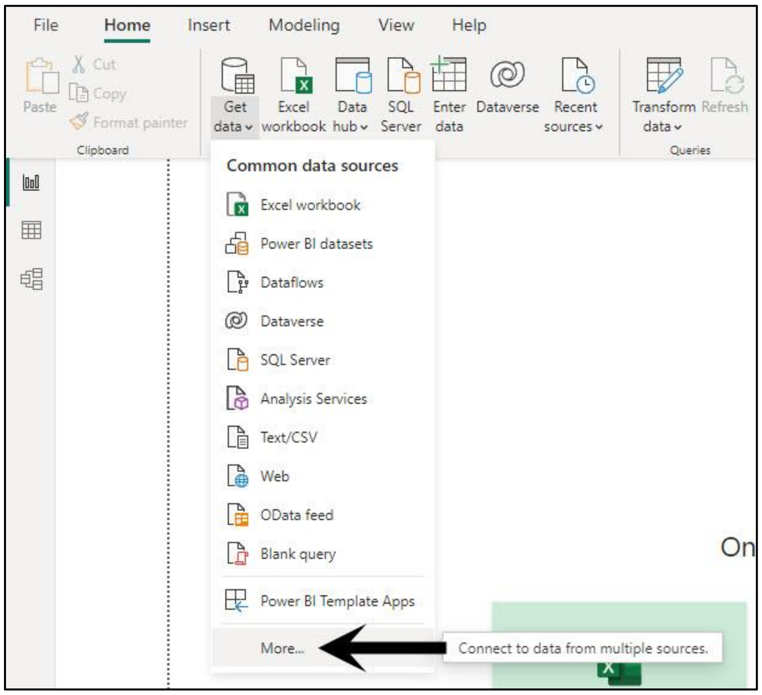


Import Single Object Files: PDF

1. Pdf files are NOT designed to move data from one system to another. However, Power Query can interpret data from a pdf file with somewhat good results:
2. Here are the two connector paths in Power Query:
   - Excel:

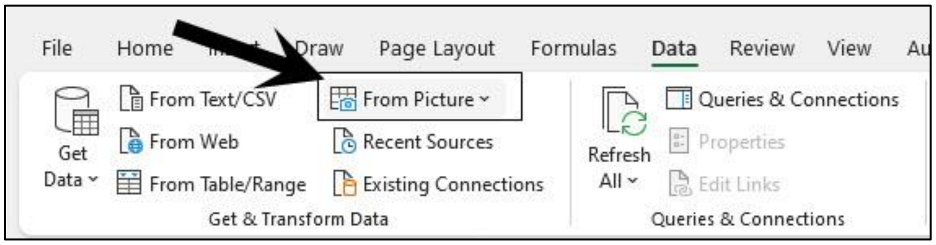


   - Power BI:



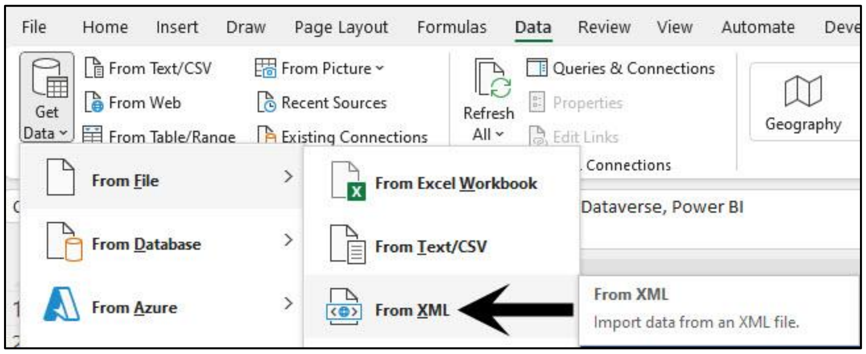
Import Single Object Files: Picture

1. Picture files are NOT designed to move data from one system to another. However, Excel can interpret data from a pdf file with varying results:
2. Here is connector the Get & Transform group in the Data tab in the Excel Ribbon (this feature is NOT Power Query):
   - Excel:

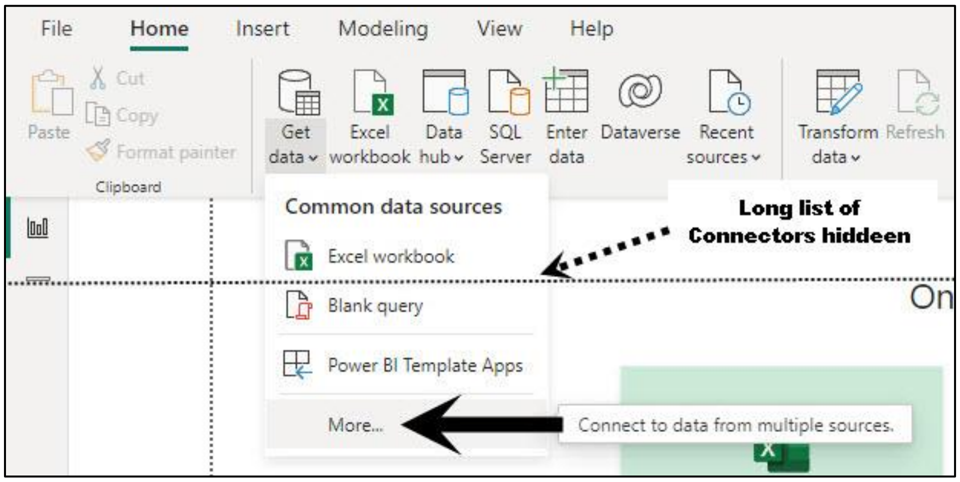


Import Single Object Files: XML

1. XML files with the extension ".xml" (eXtensible Markup Language) and are designed to move data from one system to another.
2. Here are the two connector paths in Power Query:
   - Excel:

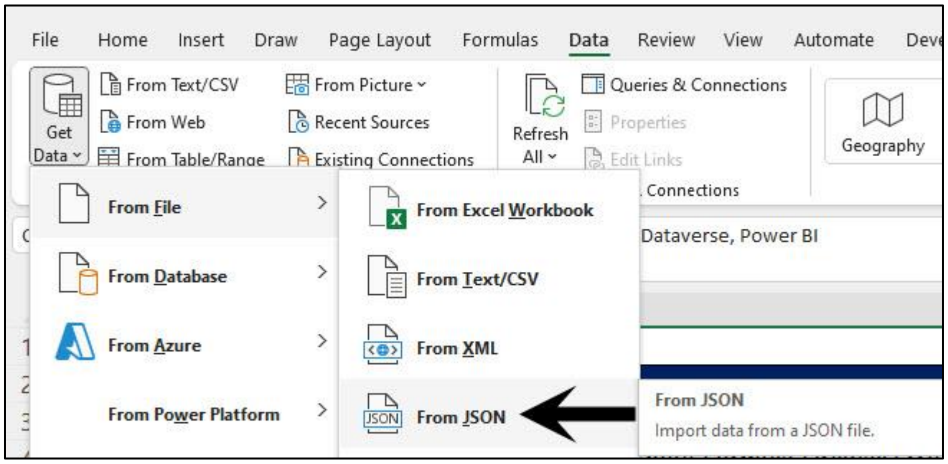


   - Power BI:

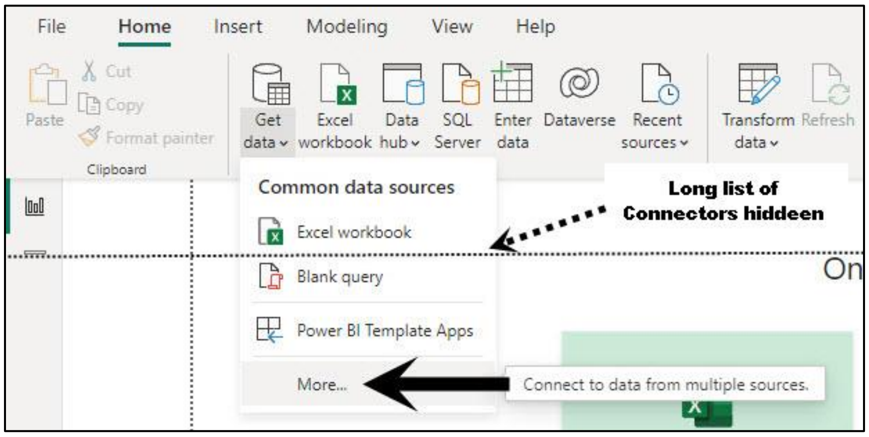


Import Single Object Files: JSON

1. Json files with the extension ".json" (JavaScript Object Notation) and are designed to move data from one system to another.
2. Here are the two connector paths in Power Query:
   - Excel:

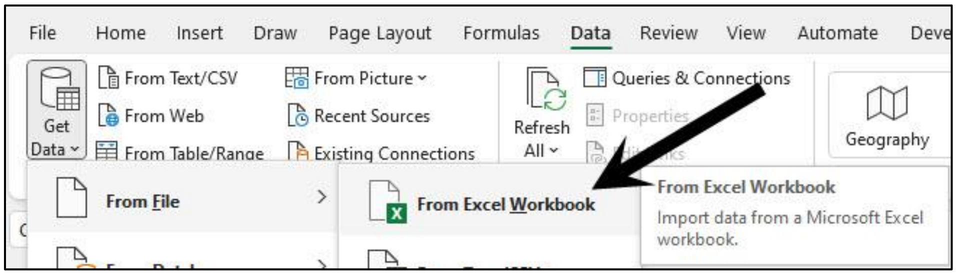


- Power BI Desktop:

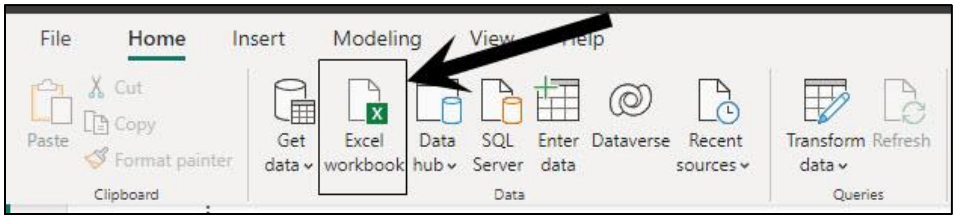


Import Single Object Files: Excel

1. Excel files can have many objects such as Excel Tables, Worksheets, Defined Names or Dynamic Arrays. When you import a single Excel file, a Navigator windows allows you to select one or more objects to import, each as a different query.
2. Here are the two connector paths in Power Query:
   - Excel:

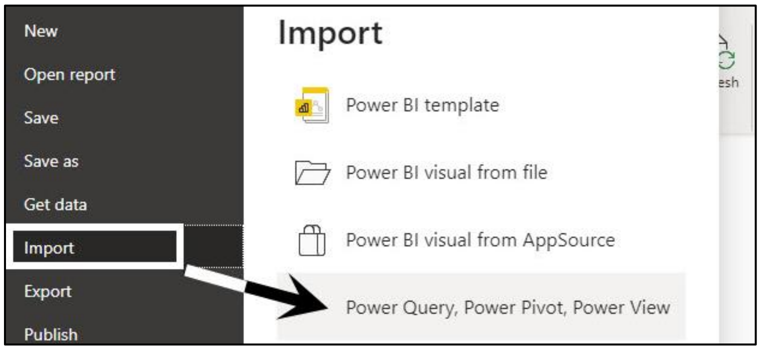


   - Power BI Desktop:



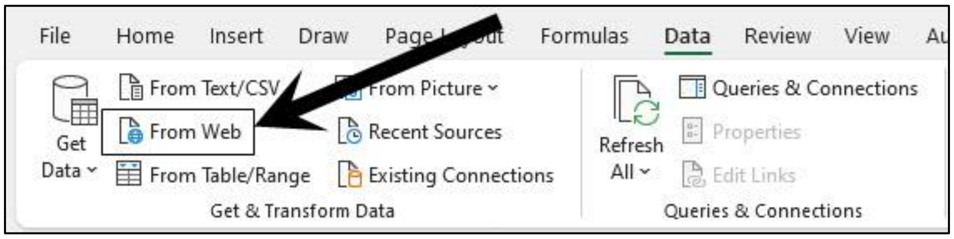
Import Excel Query or Data Model into Power BI Desktop (Demonstrated in MECS video #4)

1. When you want to import Queries or Data Model objects (Tables, Relationships and DAX Formulas) from an Excel file into Power BI Desktop you DO NOT use the Excel Import Button as shown in previous example. Instead, you use this connector from the File tab in the Power BI Desktop Ribbon:

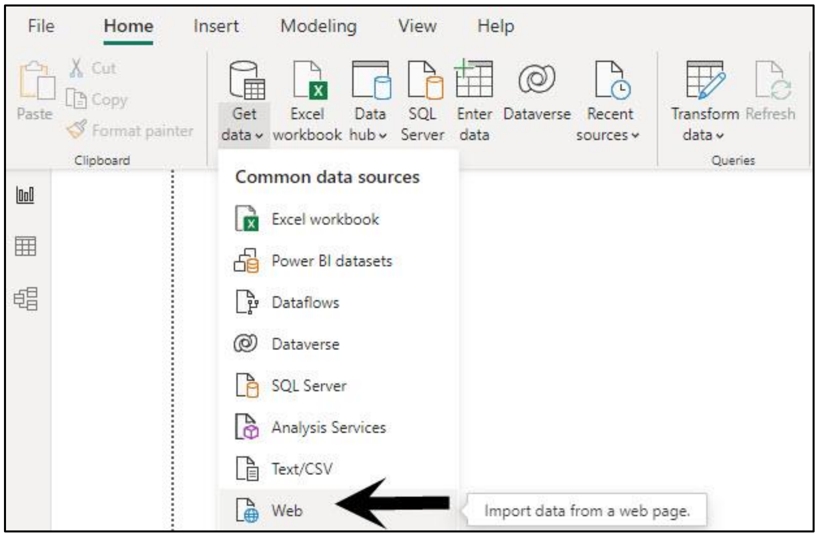

New
Open report
Save
Save as
Get data
Import
Export
Publish
Import
Power BI template
Power BI visual from file
Power BI visual from AppSource
Power Query, Power Pivot, Power View

Import from Web

1. Power Query has a import connector that allows you to import data from a web site if the web site has been built and allows data to be exported (not all web sites allow data export).
2. Here are the two connector paths in Power Query:
    - Excel:

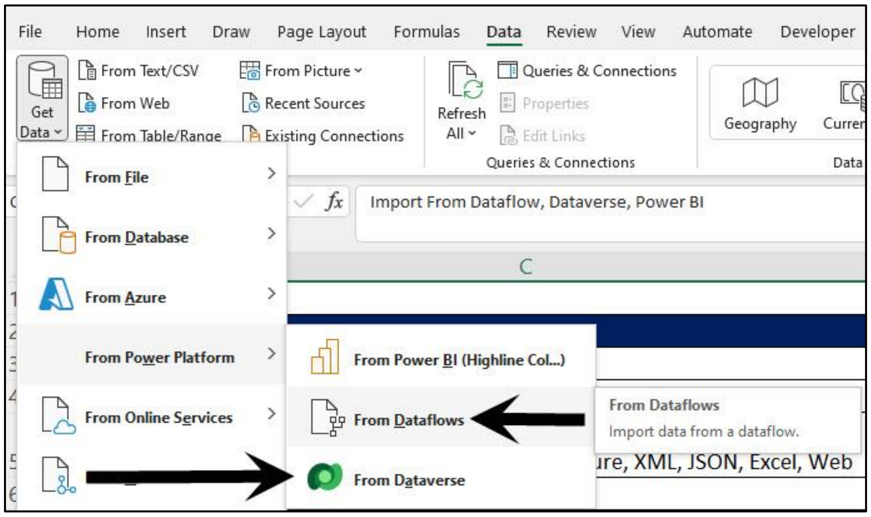


    - Power BI Desktop:

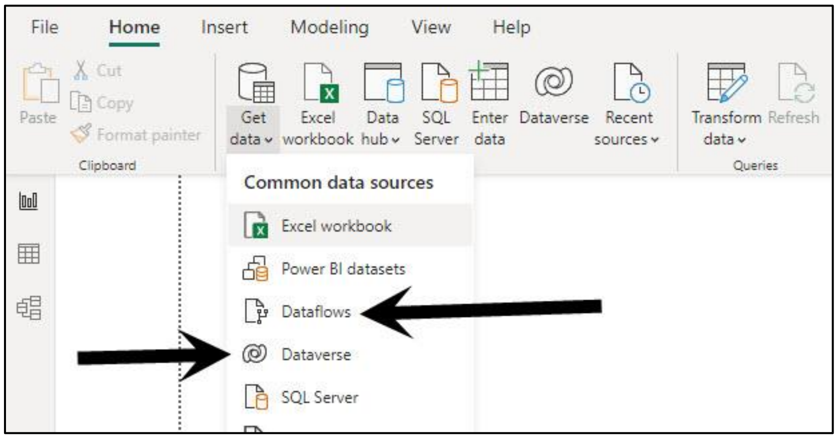


Import From: Dataflow or Dataverse

1. Dataflow and Dataverse are Microsoft Power Platforms that allow you to connect to data stored in the platform and download it into Power Query:
2. Here are the two connector paths in Power Query:
    - Excel:

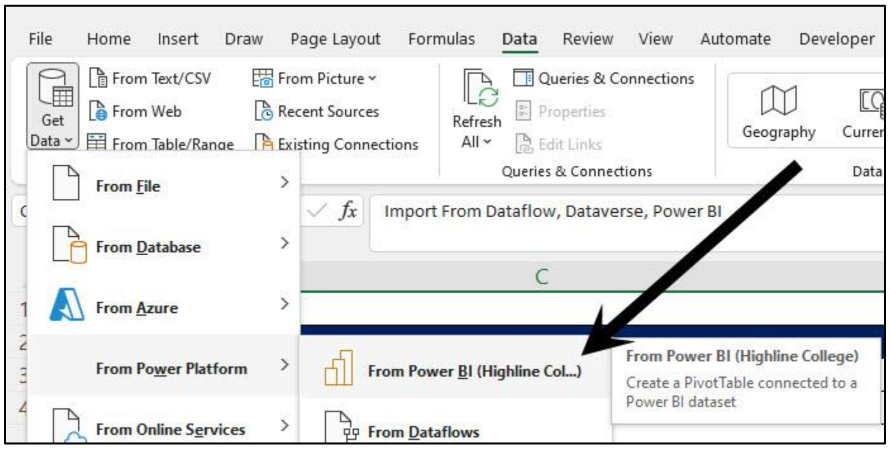


- Power BI Desktop:

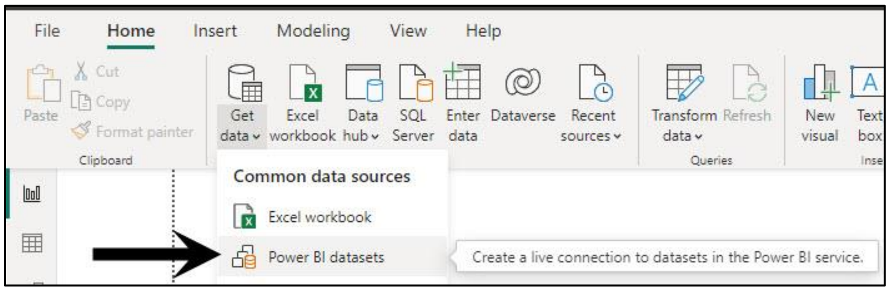


Import From: Power BI

1. The Power BI data source does not directly allow you to download the data, but instead, you connect to a data source that is stored and managed in one location in the Microsoft Power Platform (powerbi.com as discussed in MECS video #4) and build reports and visuals through a connection to that online data source. In Excel you can build PivotTable and PivotCharts from the data source. In power BI Desktop, you can build report pages with visuals from the data source.
2. Here are the two connector paths in Power Query:
   - Excel:

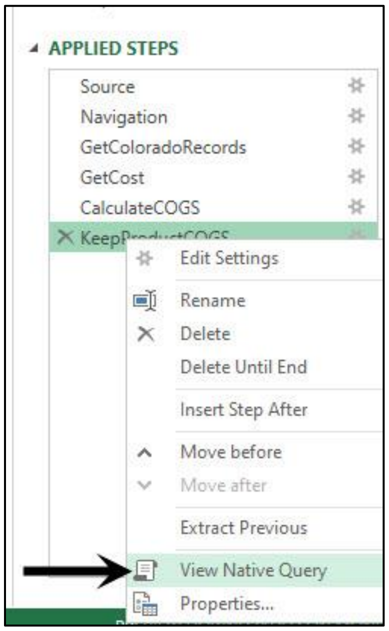


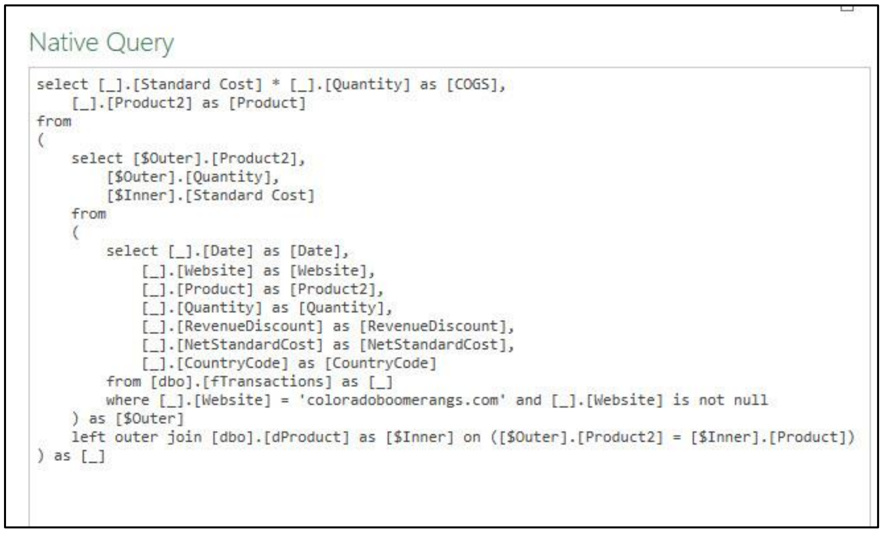
   - Power BI Desktop:



Import From SQL Database

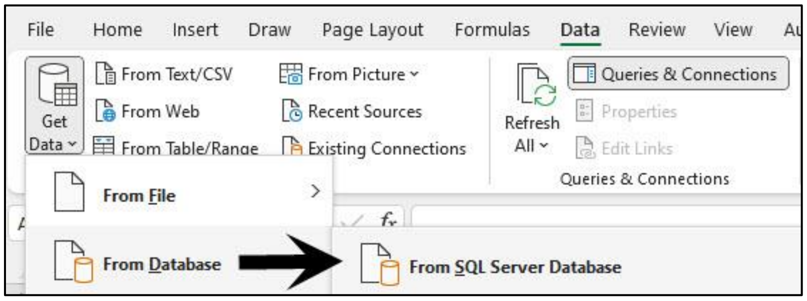
1. Most data in the world is stored in an SQL Database. SQL means "Structured Query Language" This computer language can be used to build and query databases.
2. When the SQL computer language is used to query a database, the methods used to extract and transform the data from that database are usually very efficient.
3. Usually, the equivalent SQL query is much more efficient than an equivalent M Code query. Because of this fact, when using Power Query to connect to and query an SQL database, a process called "**Query Folding**" takes place which sends the M Code generated query back to the source SQL database to be executed in the more efficient SQL environment.
4. To determine if a query step is sent back to the SQL database, right-click the query step and if you see "View Native Query" in the dropdown and it is not grayed-out, you know the step is being sent back (see picture below). If you click "View Native Query" from the dropdown, you can see the SQL code being sent back (see picture below).



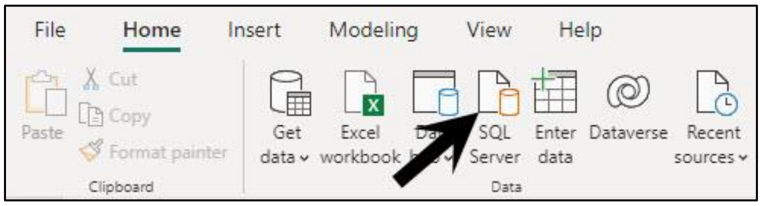

Native Query

```
select [_].[Standard Cost] * [_].[Quantity] as [COGS],
    [_].[Product2] as [Product]
from
(
    select [$Outer].[Product2],
        [$Outer].[Quantity],
        [$Inner].[Standard Cost]
    from
    (
        select [_].[Date] as [Date],
            [_].[Website] as [Website],
            [_].[Product] as [Product2],
            [_].[Quantity] as [Quantity],
            [_].[RevenueDiscount] as [RevenueDiscount],
            [_].[NetStandardCost] as [NetStandardCost],
            [_].[CountryCode] as [CountryCode]
        from [dbo].[fTransactions] as [_]
        where [_].[Website] = 'coloradoboomerangs.com' and [_].[Website] is not null
    ) as [$Outer]
    left outer join [dbo].[dProduct] as [$Inner] on ([$Outer].[Product2] = [$Inner].[Product])
) as [_]
```

5. The rule in Power Query is to put as many steps as possible that can be sent back to the SQL database at the top of the Applied Steps list, because once you have a step that can't be sent back to the SQL database, then all remaining steps in the Applied Steps list will not be sent back to the SQL database. For example, if you apply Data Types, this step cannot be sent back for query folding, so this step should be kept until the very end of the query. For a list of steps that can and cannot be used in query folding, check out this Microsoft article with full details about query folding: https://learn.microsoft.com/en-us/power-query/power-query-folding
6. Here are the two connector paths in Power Query:
   - Excel:

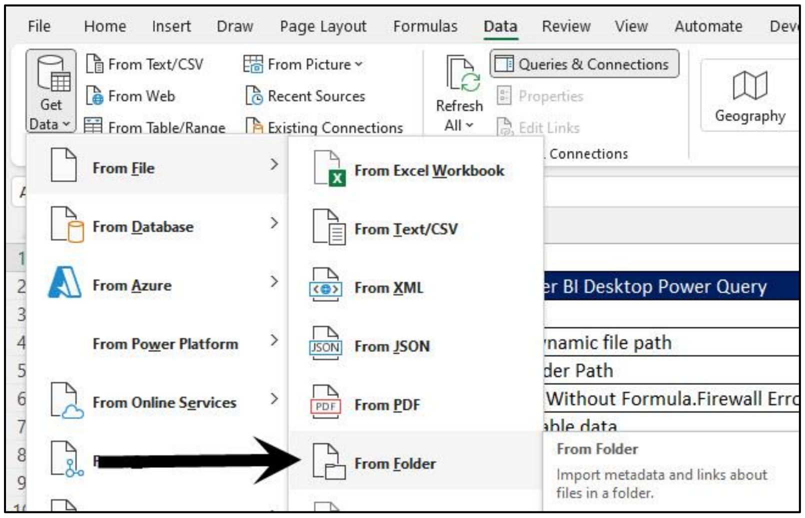


- Power BI Desktop:



Import From Folder (All Files Have One Object): Text, CSV or Excel file

1. When you use this feature you point Power Query to a folder and it can list all files and folders with attributes for the files and folders.
2. Often this feature is used to collect tables of data from files and then append the tables into a single table. The most important tip when appending tables from a folder, is that you must make sure that each table being appended has the same structure, such as same number of fields, same data types for each field and same field names for each field.
3. Here are the two connector paths in Power Query:
   - Excel:



   - Power BI Desktop:
     - Go to Home tab, Data group, Get Data dropdown, More, Folder.

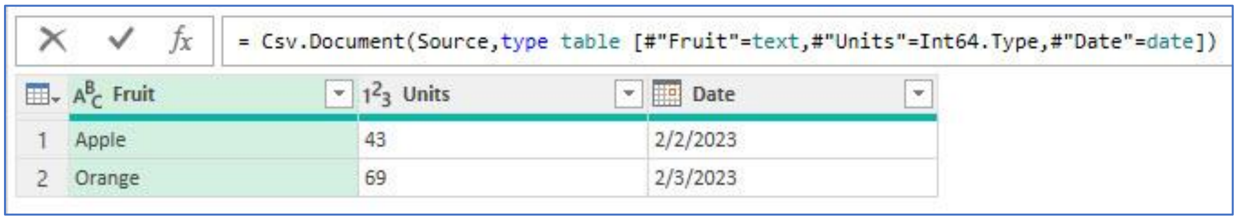
4. The M Code functions Folder.Contents and Folder.File are described in the next section.
5. When importing Tab or Comma Delimited files you can use the Csv.Documnet M Code function as described 2 sections ahead.
6. When importing Excel files, you can use the Excel.Workbook M Code function as described 3 sections ahead.

Folder.Contents and Folder.File M Code functrions

- **Folder.Contents** = This M Code functions imports all content directly inside folder, both files and folders. This function does not retrieve any files from within sub-folders.
- **Folder.File** = This M Code functions imports all files directly inside a folder and from within all subfolders.

Csv.Document function

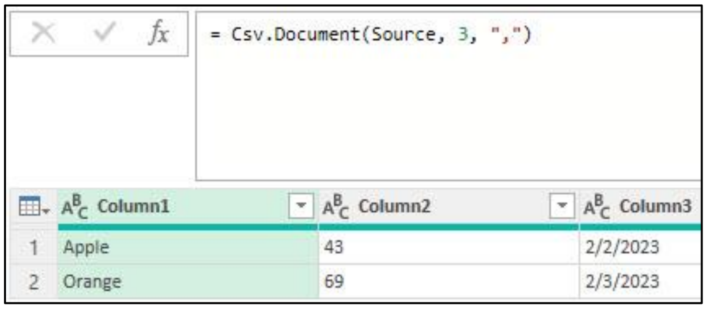
- This function can extract a table of data from a CSV file.
- Arguments:
- **Csv.Document(File Source, Columns, Delimiter, ExtraValues)**
    - **File Source** = file such as ".csv" or ".txt".
    - **Columns** = can be null, the number of columns, a list of column names, a table type, or an options record. **Default** = all columns. Type table option allows you to add Data Types, like:



```
= Csv.Document(Source,type table [#"Fruit"=text,#"Units"=Int64.Type,#"Date"=date])
```

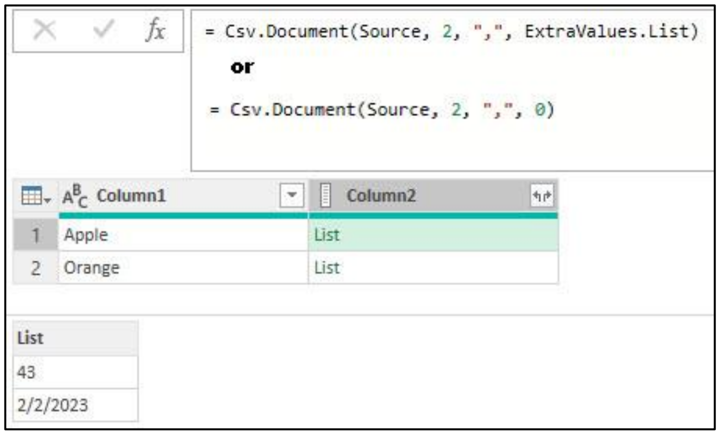
	Fruit	Units	Date
1	Apple	43	2/2/2023
2	Orange	69	2/3/2023

    -
    - **Delimiter** = can be a single character, a list of characters (multiple delimiters), a list of fixed widths, or the value "", which indicates rows should be split by consecutive whitespace characters. **Default** = "," (comma).
    - **ExtraValues** =
        i. ExtraValues.List = 0 = If the splitter function returns more columns than the table expects, they should be collected into a list.
        ii. ExtraValues.Error = 1 = If the splitter function returns more columns than the table expects, an error should be raised. This is the **default**.
        iii. ExtraValues.Ignore = 2 = If the splitter function returns more columns than the table expects, they should be ignored.
        iv. Pictures:



```
= Csv.Document(Source, 3, ",")
```

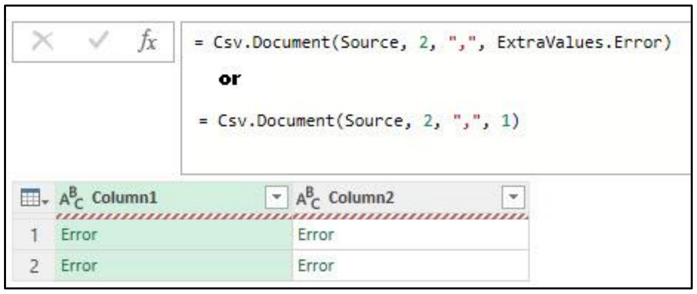
	Column1	Column2	Column3
1	Apple	43	2/2/2023
2	Orange	69	2/3/2023



```
= Csv.Document(Source, 2, ",", ExtraValues.List)
or
= Csv.Document(Source, 2, ",", 0)
```

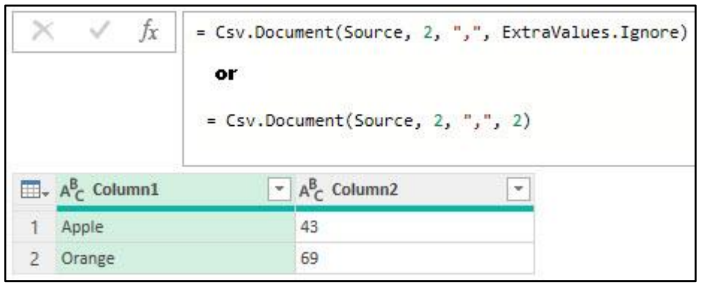
	Column1	Column2
1	Apple	List
2	Orange	List

List
43
2/2/2023



```
= Csv.Document(Source, 2, ",", ExtraValues.Error)
or
= Csv.Document(Source, 2, ",", 1)
```

	Column1	Column2
1	Error	Error
2	Error	Error



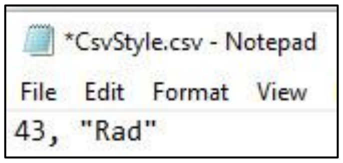
```
= Csv.Document(Source, 2, ",", ExtraValues.Ignore)
or
= Csv.Document(Source, 2, ",", 2)
```

	Column1	Column2
1	Apple	43
2	Orange	69

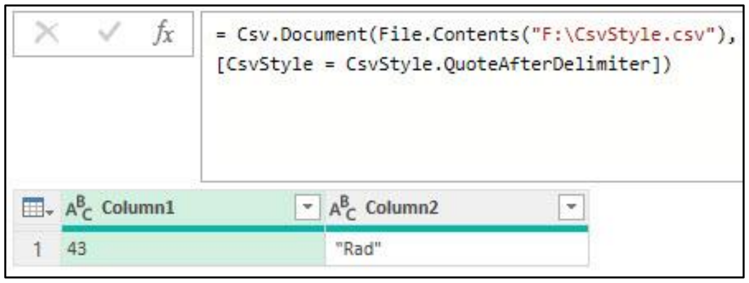
- For the Csv.Document function, if a record is specified for columns (and delimiter, extraValues, and encoding are null), the following record fields may be provided: [Delimiter, Columns, Encoding, CsvStyle, QuoteStyle]
    i. Example of a record in second argument:

Csv.Document([Content],
[Delimiter="#(tab)", Columns=2, Encoding=1252, CsvStyle=CsvStyle.QuoteAfterDelimiter, QuoteStyle=QuoteStyle.None])

    ii. Settings inside Record:
        1. **Delimiter** = Set delimiter in record like: Delimiter="#(tab)" for tab or Delimiter="," for comma.
        2. **Columns** = Set columns in record like: Columns=3 for three fields in resulting table. If you omit this setting, the Csv.Documnet function will automatically take the number of columns as determined by the delimiter. If you use the user interface (CSV/Text button), the number of columns will be hard coded in, such as Columns=3 for three columns. If the number of columns changes later, this setting will not automatically update (in this case it may be better to not include the setting in the record.
        3. **Encoding** = type of file encoding =
            - TextEncoding.Utf16 = 1200 = UTF16 little endian binary form.
            - TextEncoding.Unicode = 1200 = UTF16 little endian binary form.
            - TextEncoding.BigEndianUnicode = 1201 = UTF16 big endian binary form.
            - TextEncoding.Windows = 1252 = Windows binary form.
            - TextEncoding.Ascii = 20127 = ASCII binary form.
            - TextEncoding.Utf8 = 65001 = UTF8 binary form. This is the **default**.
        4. **CsvStyle** = Specifies how quotes are handled =
            - **CsvStyle.QuoteAfterDelimiter** = 0 = Quotes in a field are only significant immediately following the delimiter. This is the **default**.

Source File: 

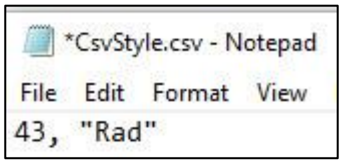
```
*CsvStyle.csv - Notepad
File  Edit  Format  View
43, "Rad"
```

Becomes: 

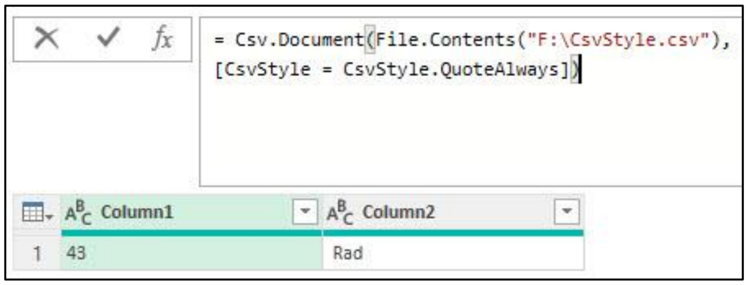
```
= Csv.Document(File.Contents("F:\CsvStyle.csv"),
[CsvStyle = CsvStyle.QuoteAfterDelimiter])
```

	Column1	Column2
1	43	"Rad"

- **CsvStyle.QuoteAlways** = 1 = Quotes in a field are always significant, regardless of where they appear.

Source File: 

```
*CsvStyle.csv - Notepad
File  Edit  Format  View
43, "Rad"
```

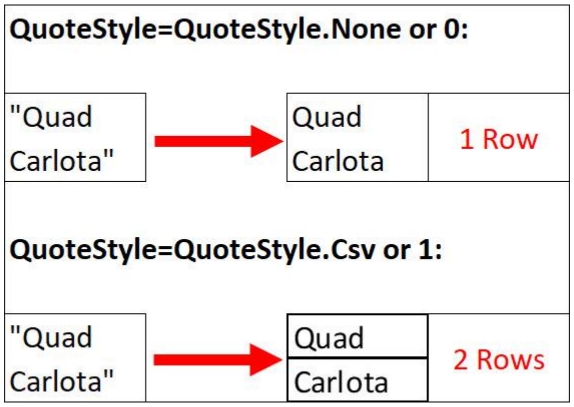
Becomes: 

```
= Csv.Document(File.Contents("F:\CsvStyle.csv"),
[CsvStyle = CsvStyle.QuoteAlways])
```

	Column1	Column2
1	43	Rad

iii. Settings inside Record (continued):
    1. **QuoteStyle** = Specifies how quoted line breaks are handled =
        - QuoteStyle.None = 0 = Ignore quoted line breaks. This is the default.
        - QuoteStyle.Csv = 1 = Apply all line breaks.

        - Examples:

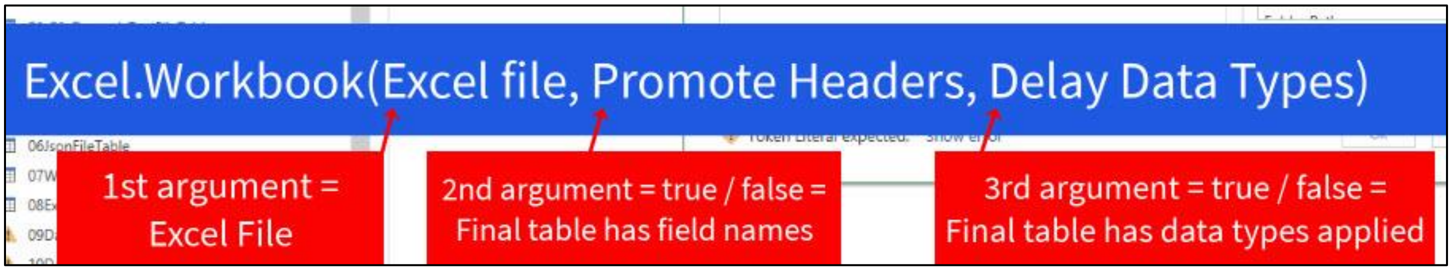

QuoteStyle=QuoteStyle.None or 0:
"Quad Carlota"
Quad Carlota
1 Row
QuoteStyle=QuoteStyle.Csv or 1:
"Quad Carlota"
Quad
Carlota
2 Rows

Import Excel Files with Multiple Objects From Folder

1. When you use the From Folder feature on Excel files with multiple objects, you must use the Excel.Workbook function to extract all the objects from the Excel file and then filter to get just the objects that you want.
2. The following section gives you information about the Excel.Workbook function.

Excel.Workbook function

- **Excel.Workbook function** will extract all the objects from a specified external Excel workbook file and deliver a table to the Power Query Editor that lists all the objects and object attributes so that you can chose which objects to import.
- Although the official Microsoft web site does not define what sort of objects this function will extract, and instead just states that the function extracts "content", it is important to know what type of objects the function will extract because some of the objects can cause unexpected results. Some of the possible Excel objects that the function extracts are listed here:
    - Excel worksheets with all content on the sheet.
    - Excel Tables.
    - Defined Names that are manually created.
    - Defined Name that is automatically created when you define a print range.
    - Defined Name that is automatically created when you use the Auto Filter feature.
    - The Defined Names, Criteria and Extract, that are created when you use the Advanced Filter feature.
- The three arguments in the function are:
    - Excel.Workbook(Excel File, Promote Headers, Delay Data Types)
    - **Excel File** = Excel File as binary
    - **Promote Headers** = true or false to indicate if headers (fields) in final table should be promoted from text in the first row to headers (fields). Default = false.
    - **Delay Data Types** = true or false to indicate if Data Types should be applied. Default = false. Note: using false and letting the Excel.Workbook() set the Data Types has two important points to think about: **1)** It may not always get the right Data Type (for example, it may interpret a whole number as a Decimal), and **2)** the query may run slower than setting the argument to true and then added an extra query step to change Data Types.


Excel.Workbook(Excel file, Promote Headers, Delay Data Types)
1st argument = Excel File
2nd argument = true / false = Final table has field names
3rd argument = true / false = Final table has data types applied

Import All Excel Tables From Inside Current Workbook

- To import all the Excel Tables from within the current workbook, you can use the Excel.CurrentWorkbook M Code function. Information about this function is presented in the next section.

Excel.CurrentWorkbook()

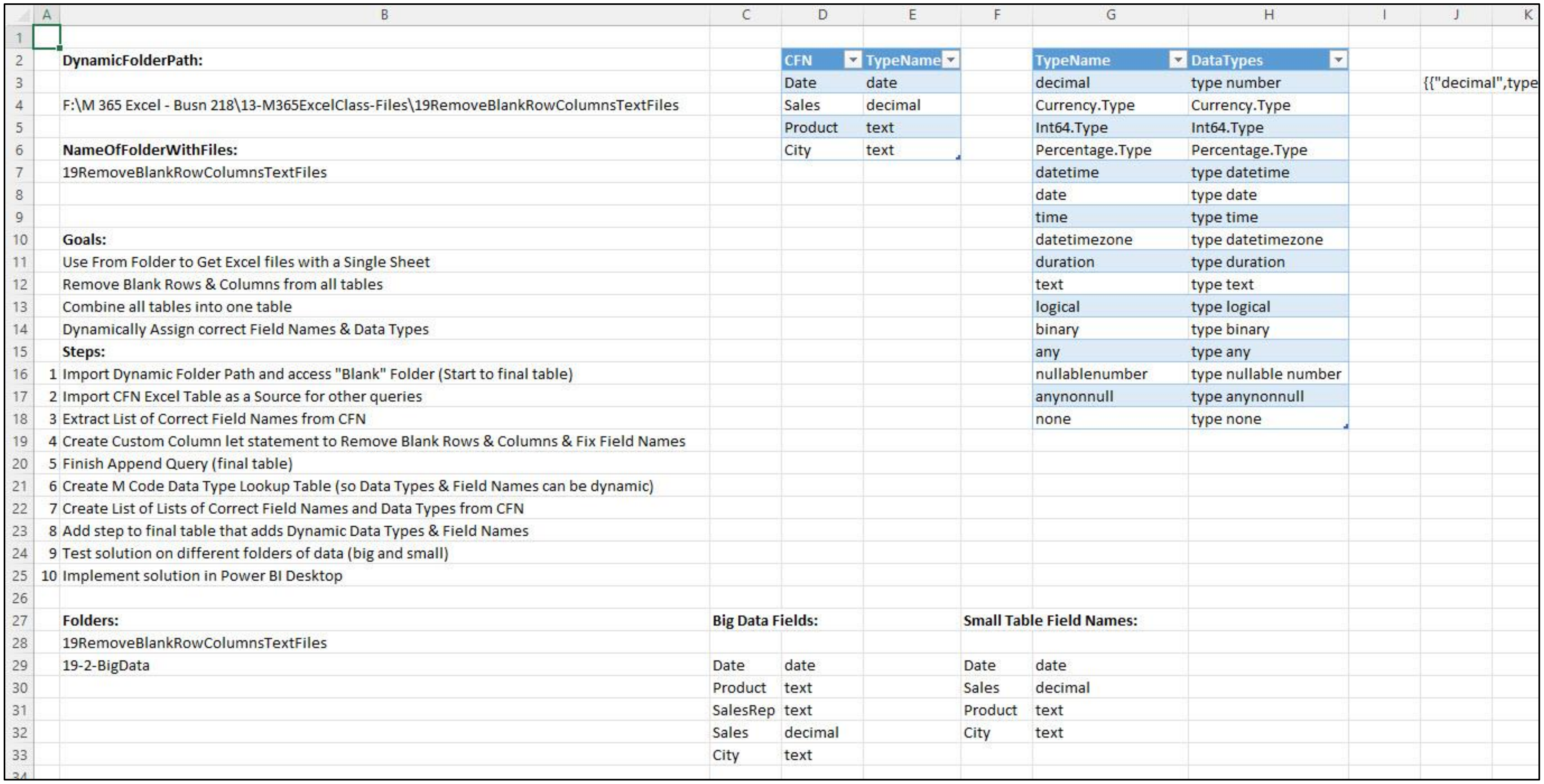
- Excel.CurrentWorkbook() is an argumnetless function that rreturns the contents of the current Excel workbook.
    - It returns Excel Tables, Named Ranges (including those automatic names created when you use a print range, filter or advanced filter), and dynamic arrays.
    - Unlike Excel.Workbook, it does not return sheets.
    - When you use the Excel.CurrentWorkbook() function If you load a query result to the worksheet, you must filter out the query name to avoid recursion, which is the query result being imported to itself (because the query result in an Excel Table).
    - You do not have the recursion problem if you load the data to a PivotTable Cache or the Data Model.

Import All Excel Worksheets From Inside Current Workbook

Example not in this video, but here is video link):
https://www.youtube.com/watch?v=KfuYxBDBkAo&lc=Ugwl4Ney9Q_xVcmla6l4AaABAg

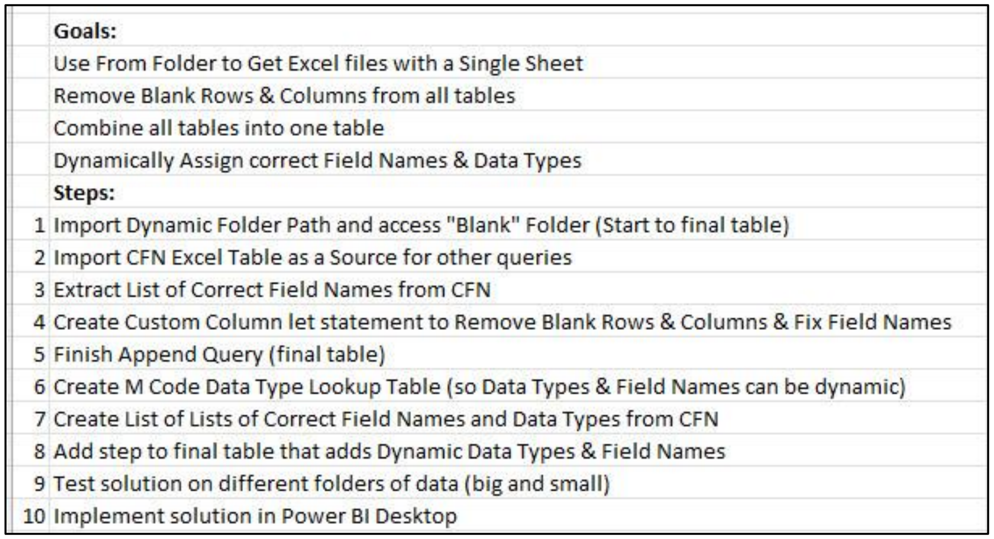
Query 19 (Solution for Blank Rows & Columns and Dynamic Field Names an Data Types)

- This is the Control Sheet in the Excel file named “13-M365-BlankRowsColsFinished.xlsx”:



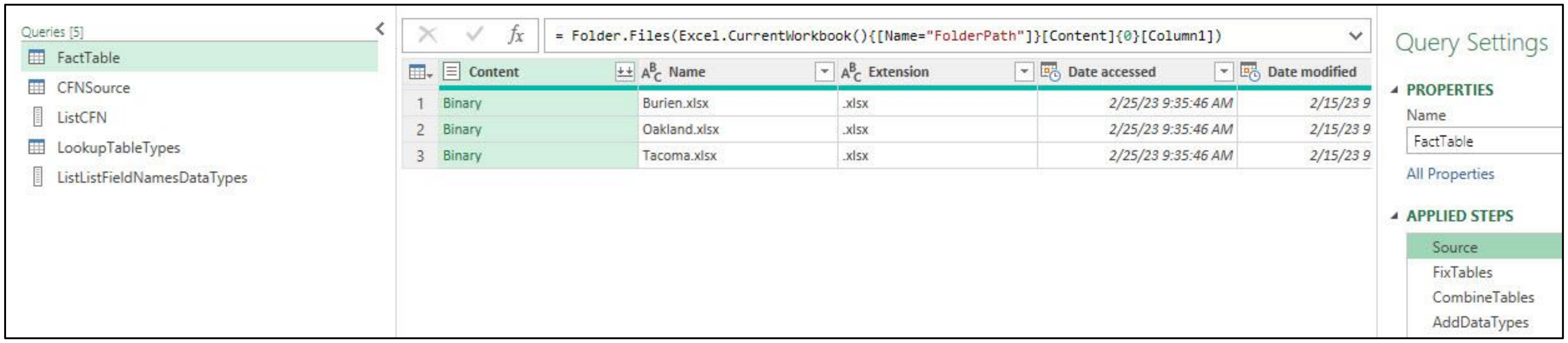
	A	B	C	D	E	F	G	H	I	J	K
1											
2		**DynamicFolderPath:**		CFN	TypeName		TypeName	DataTypes			
3				Date	date		decimal	type number		{{"decimal",type	
4		F:\M 365 Excel - Busn 218\13-M365ExcelClass-Files\19RemoveBlankRowColumnsTextFiles		Sales	decimal		Currency.Type	Currency.Type			
5				Product	text		Int64.Type	Int64.Type			
6		**NameOfFolderWithFiles:**		City	text		Percentage.Type	Percentage.Type			
7		19RemoveBlankRowColumnsTextFiles					datetime	type datetime			
8							date	type date			
9							time	type time			
10		**Goals:**					datetimezone	type datetimezone			
11		Use From Folder to Get Excel files with a Single Sheet					duration	type duration			
12		Remove Blank Rows & Columns from all tables					text	type text			
13		Combine all tables into one table					logical	type logical			
14		Dynamically Assign correct Field Names & Data Types					binary	type binary			
15		**Steps:**					any	type any			
16	1	Import Dynamic Folder Path and access "Blank" Folder (Start to final table)					nullablenumber	type nullable number			
17	2	Import CFN Excel Table as a Source for other queries					anynonnull	type anynonnull			
18	3	Extract List of Correct Field Names from CFN					none	type none			
19	4	Create Custom Column let statement to Remove Blank Rows & Columns & Fix Field Names									
20	5	Finish Append Query (final table)									
21	6	Create M Code Data Type Lookup Table (so Data Types & Field Names can be dynamic)									
22	7	Create List of Lists of Correct Field Names and Data Types from CFN									
23	8	Add step to final table that adds Dynamic Data Types & Field Names									
24	9	Test solution on different folders of data (big and small)									
25	10	Implement solution in Power BI Desktop									
26											
27		**Folders:**	**Big Data Fields:**			**Small Table Field Names:**					
28		19RemoveBlankRowColumnsTextFiles									
29		19-2-BigData	Date	date		Date	date				
30			Product	text		Sales	decimal				
31			SalesRep	text		Product	text				
32			Sales	decimal		City	text				
33			City	text							

- Goals and Steps of Project:



	Goals:
	Use From Folder to Get Excel files with a Single Sheet
	Remove Blank Rows & Columns from all tables
	Combine all tables into one table
	Dynamically Assign correct Field Names & Data Types
	Steps:
1	Import Dynamic Folder Path and access "Blank" Folder (Start to final table)
2	Import CFN Excel Table as a Source for other queries
3	Extract List of Correct Field Names from CFN
4	Create Custom Column let statement to Remove Blank Rows & Columns & Fix Field Names
5	Finish Append Query (final table)
6	Create M Code Data Type Lookup Table (so Data Types & Field Names can be dynamic)
7	Create List of Lists of Correct Field Names and Data Types from CFN
8	Add step to final table that adds Dynamic Data Types & Field Names
9	Test solution on different folders of data (big and small)
10	Implement solution in Power BI Desktop

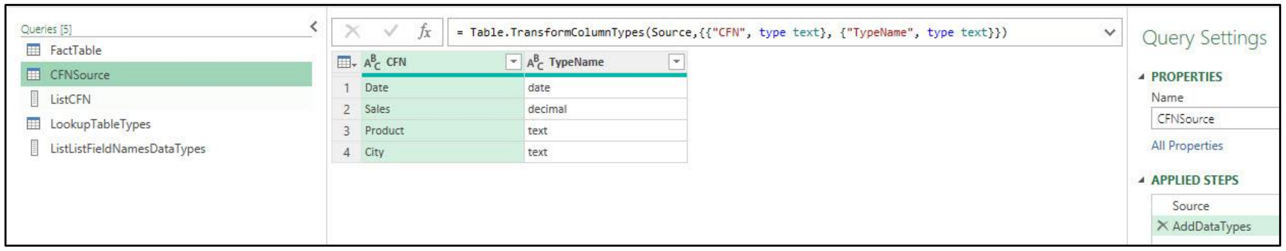
- Step 1 in Project: Import Dynamic Folder Path and access "Blank" Folder (Start to final table)



```
= Folder.Files(Excel.CurrentWorkbook(){[Name="FolderPath"]}[Content]{0}[Column1])
```

	Content	Name	Extension	Date accessed	Date modified
1	Binary	Burien.xlsx	.xlsx	2/25/23 9:35:46 AM	2/15/23 9
2	Binary	Oakland.xlsx	.xlsx	2/25/23 9:35:46 AM	2/15/23 9
3	Binary	Tacoma.xlsx	.xlsx	2/25/23 9:35:46 AM	2/15/23 9

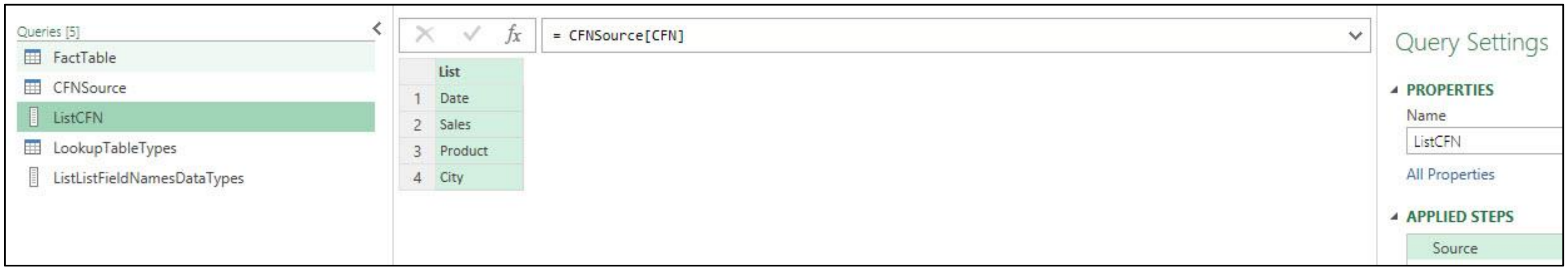
- Step 2 in Project: Import CFN Excel Table as a Source for other queries



```
= Table.TransformColumnTypes(Source,{{"CFN", type text}, {"TypeName", type text}})
```

	CFN	TypeName
1	Date	date
2	Sales	decimal
3	Product	text
4	City	text

- Step 3 in Project: Extract List of Correct Field Names from CFN

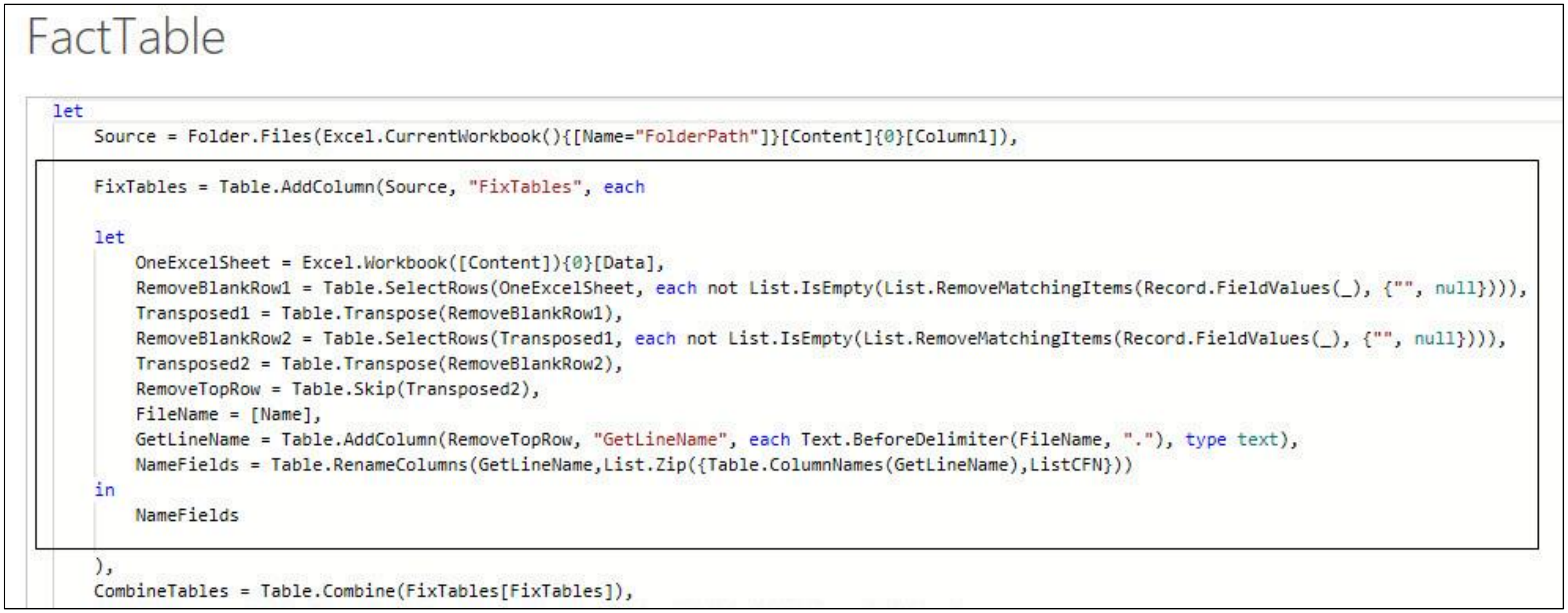


```
= CFNSource[CFN]
```

	List
1	Date
2	Sales
3	Product
4	City

Name: ListCFN

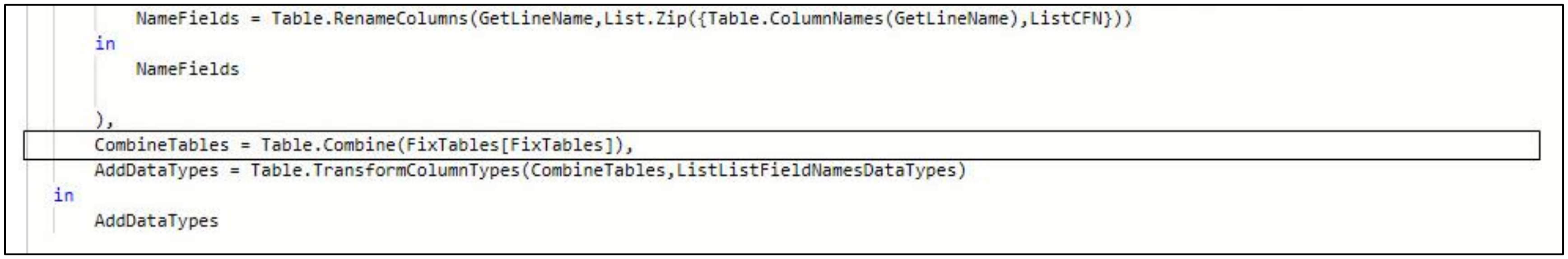
- Step 4 in Project: Create Custom Column let statement to Remove Blank Rows & Columns & Fix Field Names



FactTable

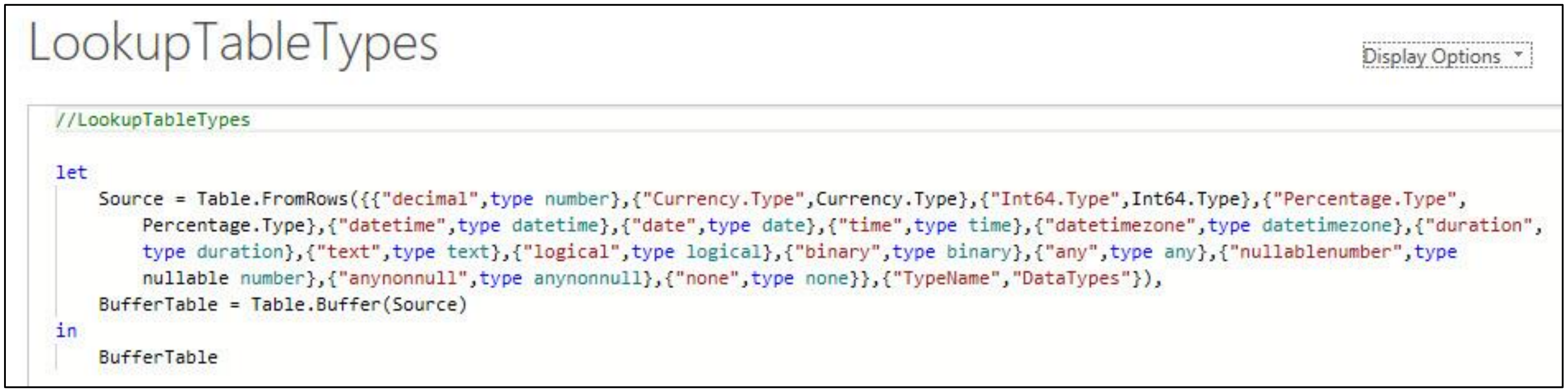
```
let
    Source = Folder.Files(Excel.CurrentWorkbook(){[Name="FolderPath"]}[Content]{0}[Column1]),

    FixTables = Table.AddColumn(Source, "FixTables", each

    let
        OneExcelSheet = Excel.Workbook([Content]){0}[Data],
        RemoveBlankRow1 = Table.SelectRows(OneExcelSheet, each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))),
        Transposed1 = Table.Transpose(RemoveBlankRow1),
        RemoveBlankRow2 = Table.SelectRows(Transposed1, each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))),
        Transposed2 = Table.Transpose(RemoveBlankRow2),
        RemoveTopRow = Table.Skip(Transposed2),
        FileName = [Name],
        GetLineName = Table.AddColumn(RemoveTopRow, "GetLineName", each Text.BeforeDelimiter(FileName, "."), type text),
        NameFields = Table.RenameColumns(GetLineName,List.Zip({Table.ColumnNames(GetLineName),ListCFN}))
    in
        NameFields

    ),
    CombineTables = Table.Combine(FixTables[FixTables]),
```

- Step 5 in Project: Finish Append Query (final table)



```
            NameFields = Table.RenameColumns(GetLineName,List.Zip({Table.ColumnNames(GetLineName),ListCFN}))
        in
            NameFields

        ),
        CombineTables = Table.Combine(FixTables[FixTables]),
        AddDataTypes = Table.TransformColumnTypes(CombineTables,ListListFieldNamesDataTypes)
    in
        AddDataTypes
```

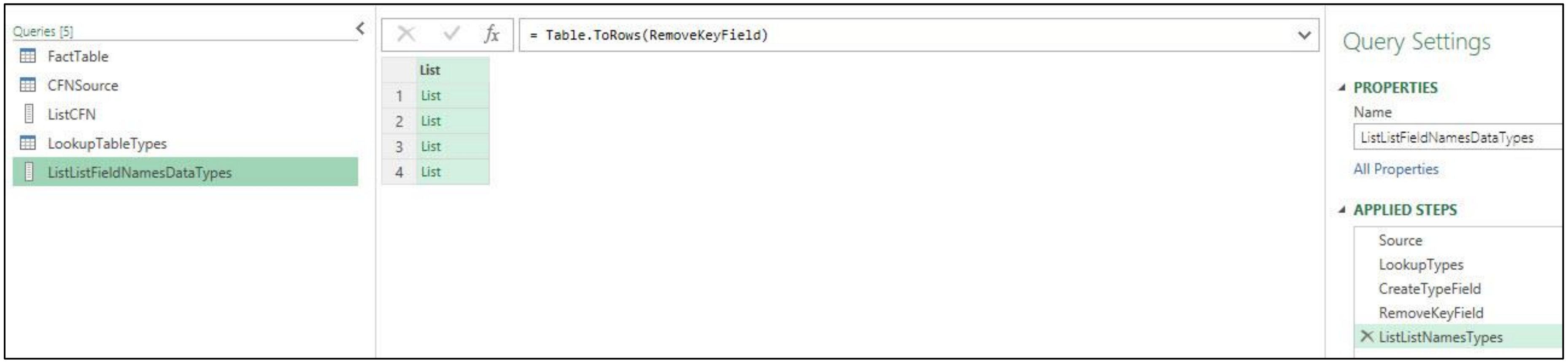
- Step 6 in Project: Create M Code Data Type Lookup Table (so Data Types & Field Names can be dynamic)



LookupTableTypes

```
//LookupTableTypes

let
    Source = Table.FromRows({{"decimal",type number},{"Currency.Type",Currency.Type},{"Int64.Type",Int64.Type},{"Percentage.Type",
        Percentage.Type},{"datetime",type datetime},{"date",type date},{"time",type time},{"datetimezone",type datetimezone},{"duration",
        type duration},{"text",type text},{"logical",type logical},{"binary",type binary},{"any",type any},{"nullablenumber",type
        nullable number},{"anynonnull",type anynonnull},{"none",type none}},{"TypeName","DataTypes"}),
    BufferTable = Table.Buffer(Source)
in
    BufferTable
```

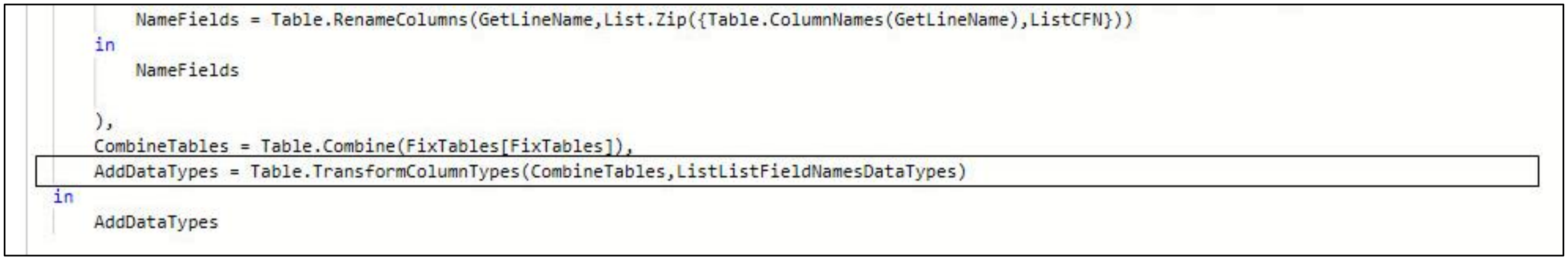
- Step 7 in Project: Create List of Lists of Correct Field Names and Data Types from CFN



```
= Table.ToRows(RemoveKeyField)
```

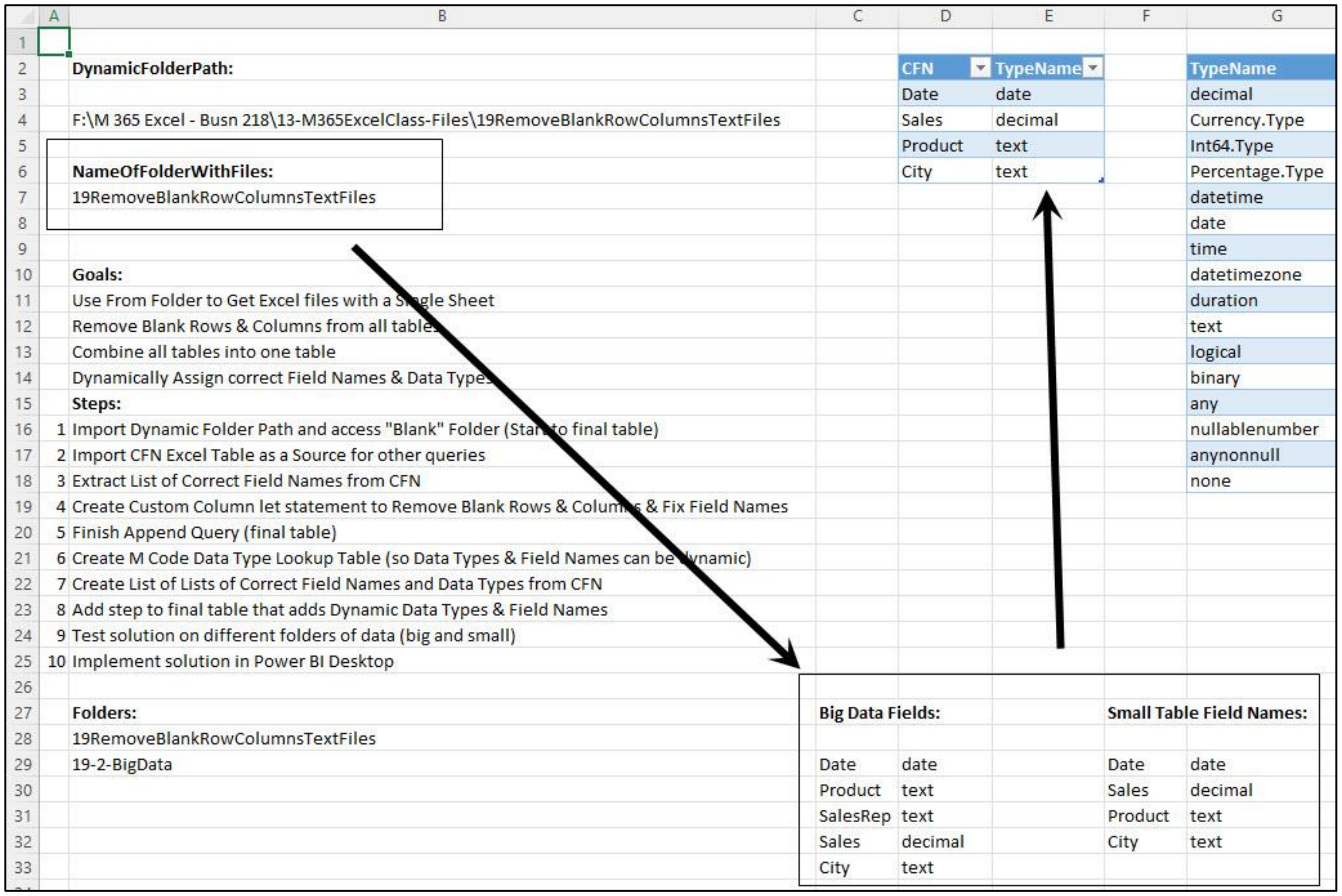
	List
1	List
2	List
3	List
4	List

- Step 8 in Project: Add step to final table that adds Dynamic Data Types & Field Names



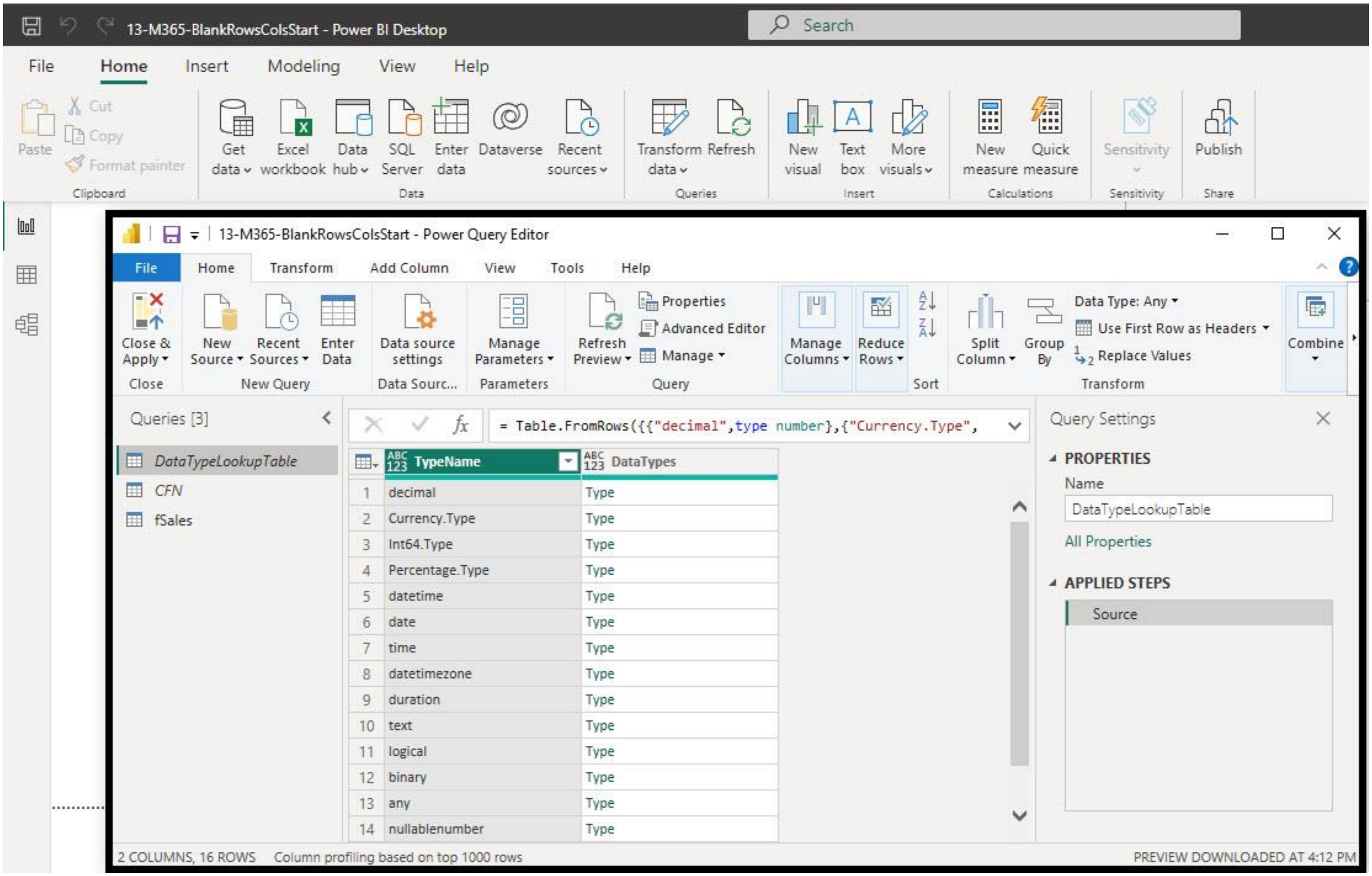
```
            NameFields = Table.RenameColumns(GetLineName,List.Zip({Table.ColumnNames(GetLineName),ListCFN}))
        in
            NameFields

        ),
        CombineTables = Table.Combine(FixTables[FixTables]),
        AddDataTypes = Table.TransformColumnTypes(CombineTables,ListListFieldNamesDataTypes)
    in
        AddDataTypes
```

- Step 9 in Project: Test solution on different folders of data (big and small)



	A	B	C	D	E	F	G
1							
2		**DynamicFolderPath:**		CFN	TypeName		TypeName
3				Date	date		decimal
4		F:\M 365 Excel - Busn 218\13-M365ExcelClass-Files\19RemoveBlankRowColumnsTextFiles		Sales	decimal		Currency.Type
5				Product	text		Int64.Type
6		**NameOfFolderWithFiles:**		City	text		Percentage.Type
7		19RemoveBlankRowColumnsTextFiles					datetime
8							date
9							time
10		**Goals:**					datetimezone
11		Use From Folder to Get Excel files with a Single Sheet					duration
12		Remove Blank Rows & Columns from all tables					text
13		Combine all tables into one table					logical
14		Dynamically Assign correct Field Names & Data Types					binary
15		**Steps:**					any
16	1	Import Dynamic Folder Path and access "Blank" Folder (Start to final table)					nullablenumber
17	2	Import CFN Excel Table as a Source for other queries					anynonnull
18	3	Extract List of Correct Field Names from CFN					none
19	4	Create Custom Column let statement to Remove Blank Rows & Columns & Fix Field Names					
20	5	Finish Append Query (final table)					
21	6	Create M Code Data Type Lookup Table (so Data Types & Field Names can be dynamic)					
22	7	Create List of Lists of Correct Field Names and Data Types from CFN					
23	8	Add step to final table that adds Dynamic Data Types & Field Names					
24	9	Test solution on different folders of data (big and small)					
25	10	Implement solution in Power BI Desktop					
26							
27		**Folders:**	**Big Data Fields:**			**Small Table Field Names:**	
28		19RemoveBlankRowColumnsTextFiles					
29		19-2-BigData	Date	date		Date	date
30			Product	text		Sales	decimal
31			SalesRep	text		Product	text
32			Sales	decimal		City	text
33			City	text			

- Step 10 in Project: Implement solution in Power BI Desktop. Below is a picture of the Power Query Editor in the Start file named "13-M365-BlankRowsColsStart.pbix".



```
= Table.FromRows({{"decimal",type number},{"Currency.Type",
```

	TypeName	DataTypes
1	decimal	Type
2	Currency.Type	Type
3	Int64.Type	Type
4	Percentage.Type	Type
5	datetime	Type
6	date	Type
7	time	Type
8	datetimezone	Type
9	duration	Type
10	text	Type
11	logical	Type
12	binary	Type
13	any	Type
14	nullablenumber	Type

2 COLUMNS, 16 ROWS   Column profiling based on top 1000 rows   PREVIEW DOWNLOADED AT 4:12 PM