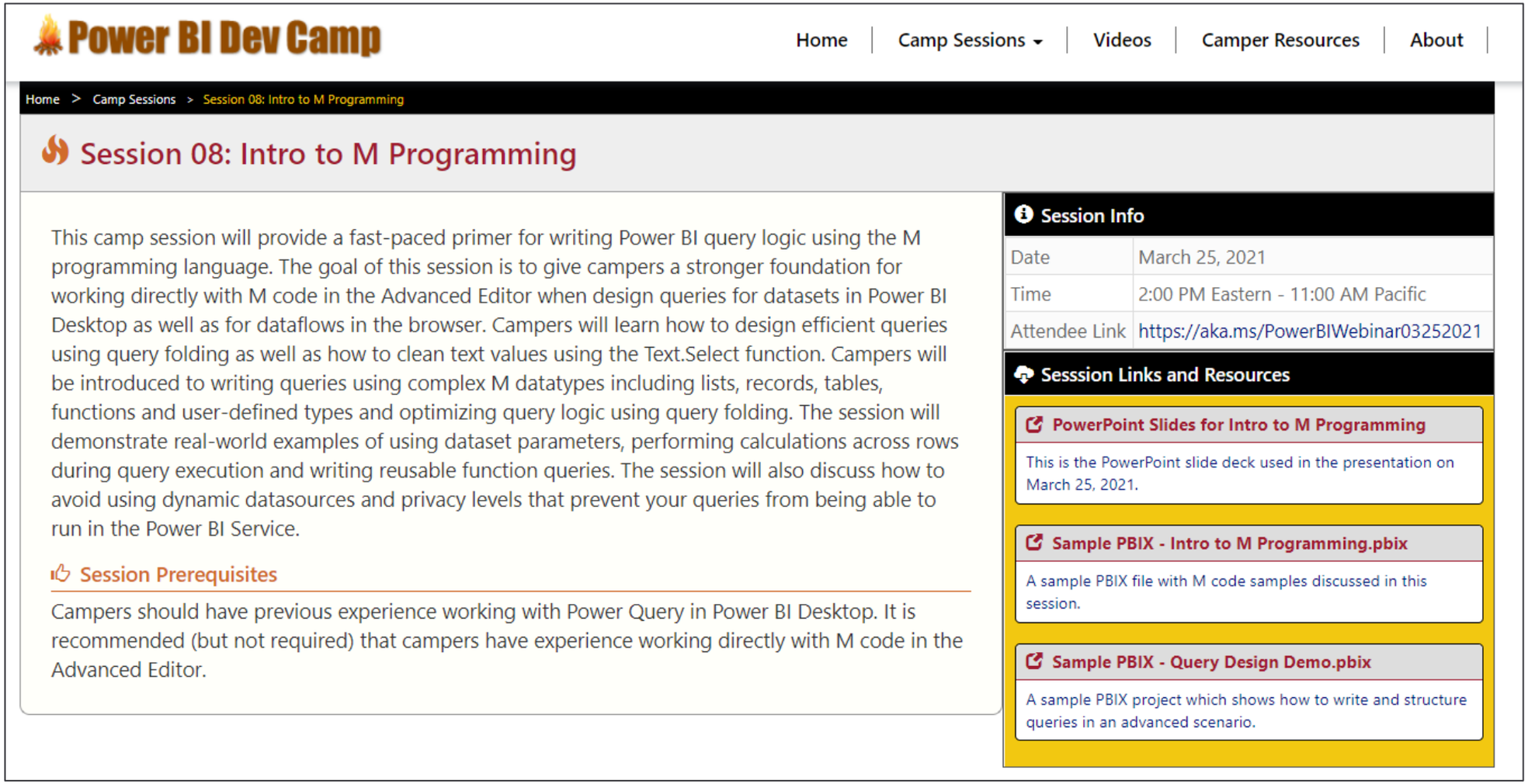
Microsoft Power BI

Intro to M Programming

Ted Pattison
Principal Program Manager
Power BI Customer Advisory Team (PBICAT)

Welcome to Power BI Dev Camp

- Power BI Dev Camp Portal - https://powerbidevcamp.net

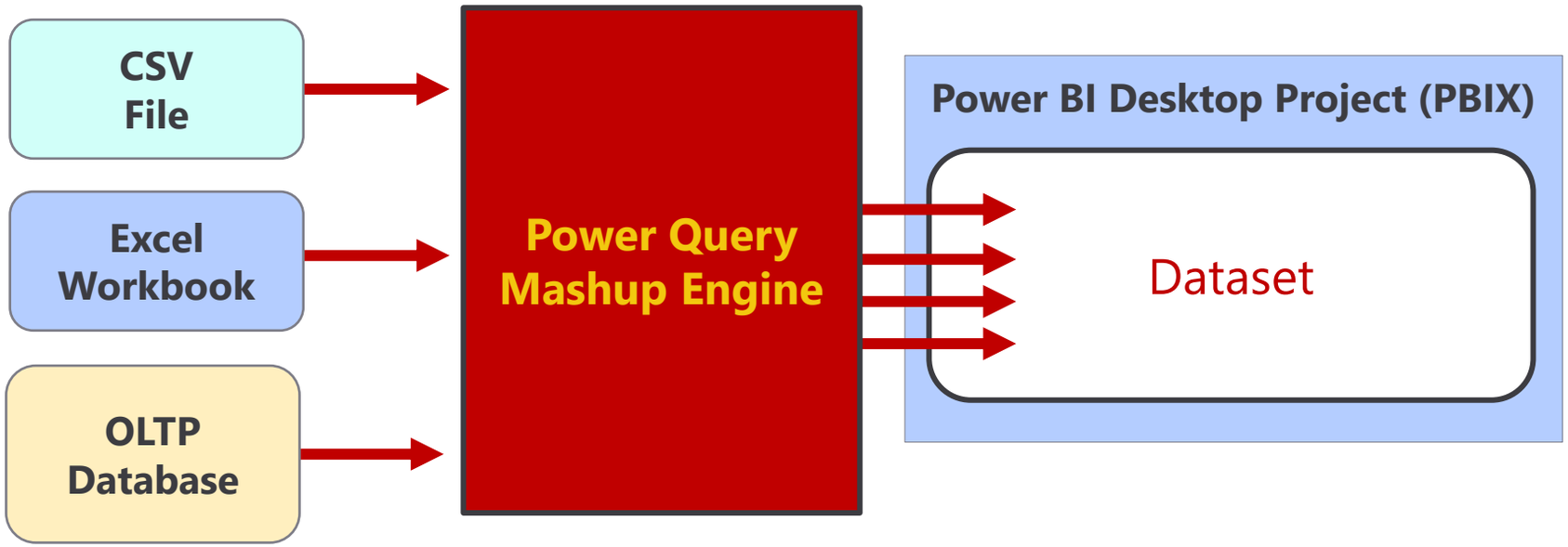


Agenda

- The Power Query Mashup Engine
- M Programming Fundamentals
- Programming Lists, Records and Tables
- Understanding Query Folding
- Choosing Between OData.Feed & Web.Contents
- Writing Reusable Function Queries
- Designing with Query Parameters

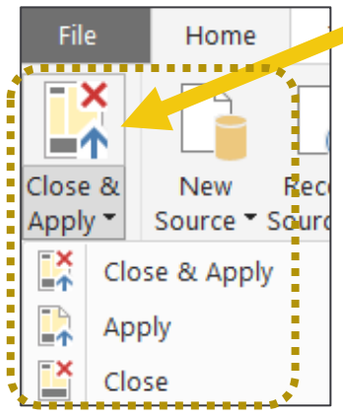
Power Query is an ETL Tool

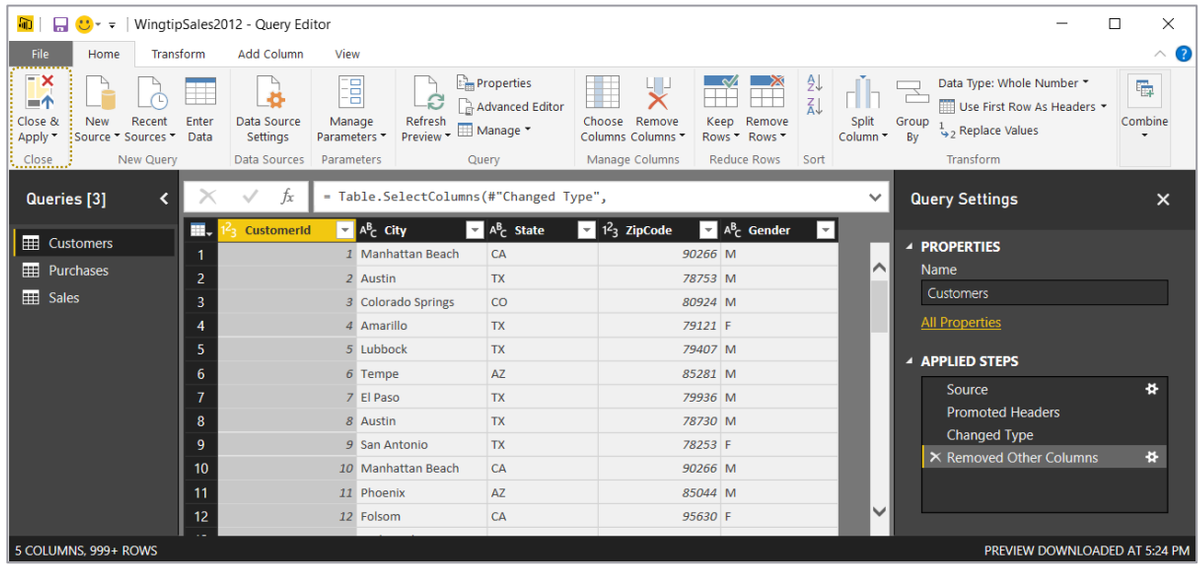
- **ETL process is essential part of any BI Project**
  - Extract the data from wherever it lives
  - Transform the shape of the data for better analysis
  - Load the data into dataset for analysis and reporting



Query Editor Window

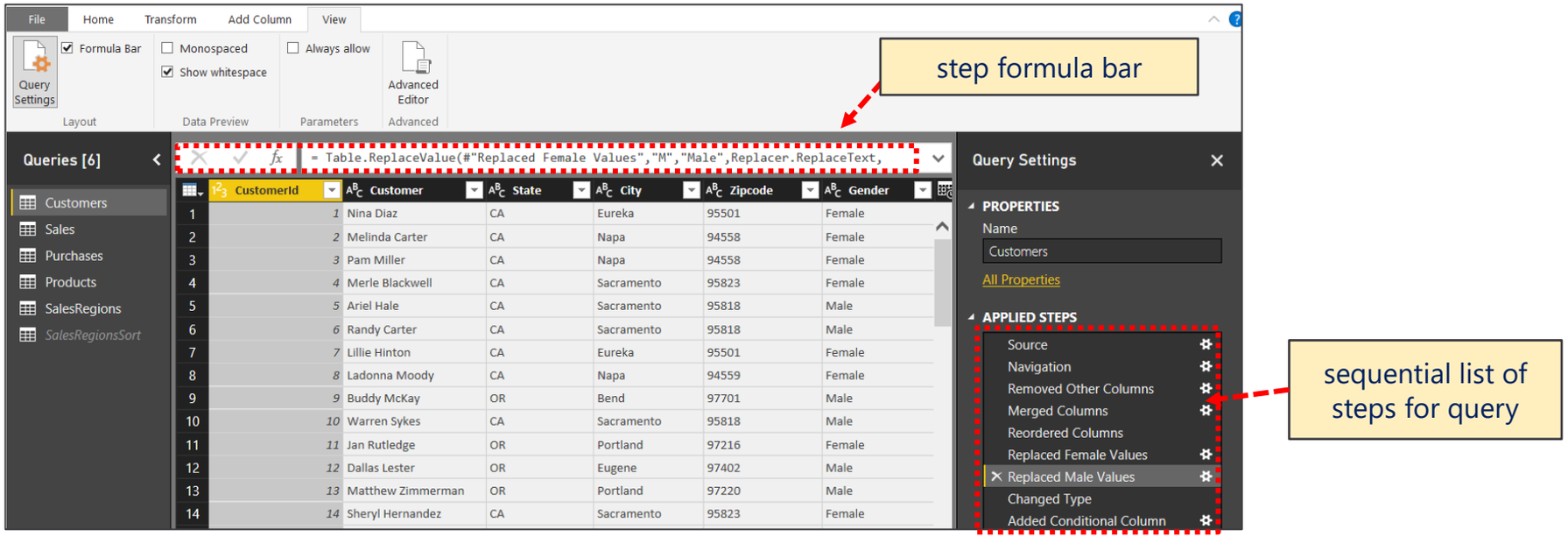
- **Power BI Desktop provides separate Query Editor window**
  - Provides easy-to-use UI experience for designing queries
  - Queries created by creating Applied Steps
  - Preview of table generated by query output shown in the middle
  - Query can be executed using Apply or Close & Apply command





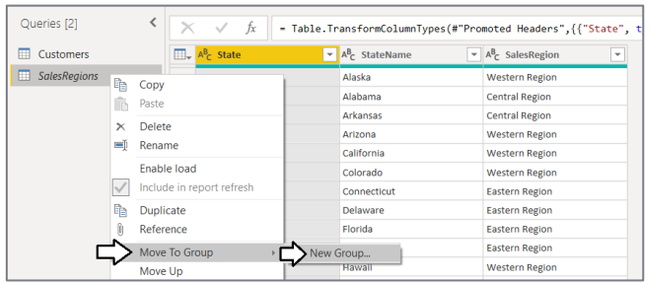
Query Steps

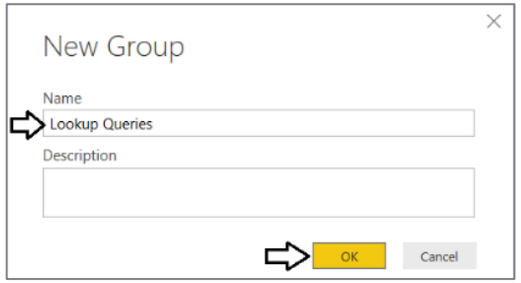
- **A query is created as a sequence of steps**
  - Each step is a parameterized operation in data processing pipeline
  - Query starts with Source step to extract data from a data source
  - Additional steps added to perform transform operations on data
  - Each step is recorded using M (aka Power Query Formula Language)



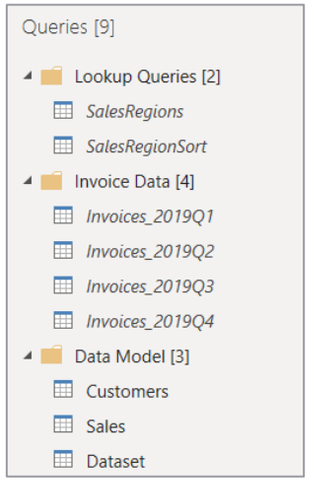
Structuring Queries into Folder Groups

- **Queries can be organized into folder groups**
  - Folder groups can be created for similar types of queries



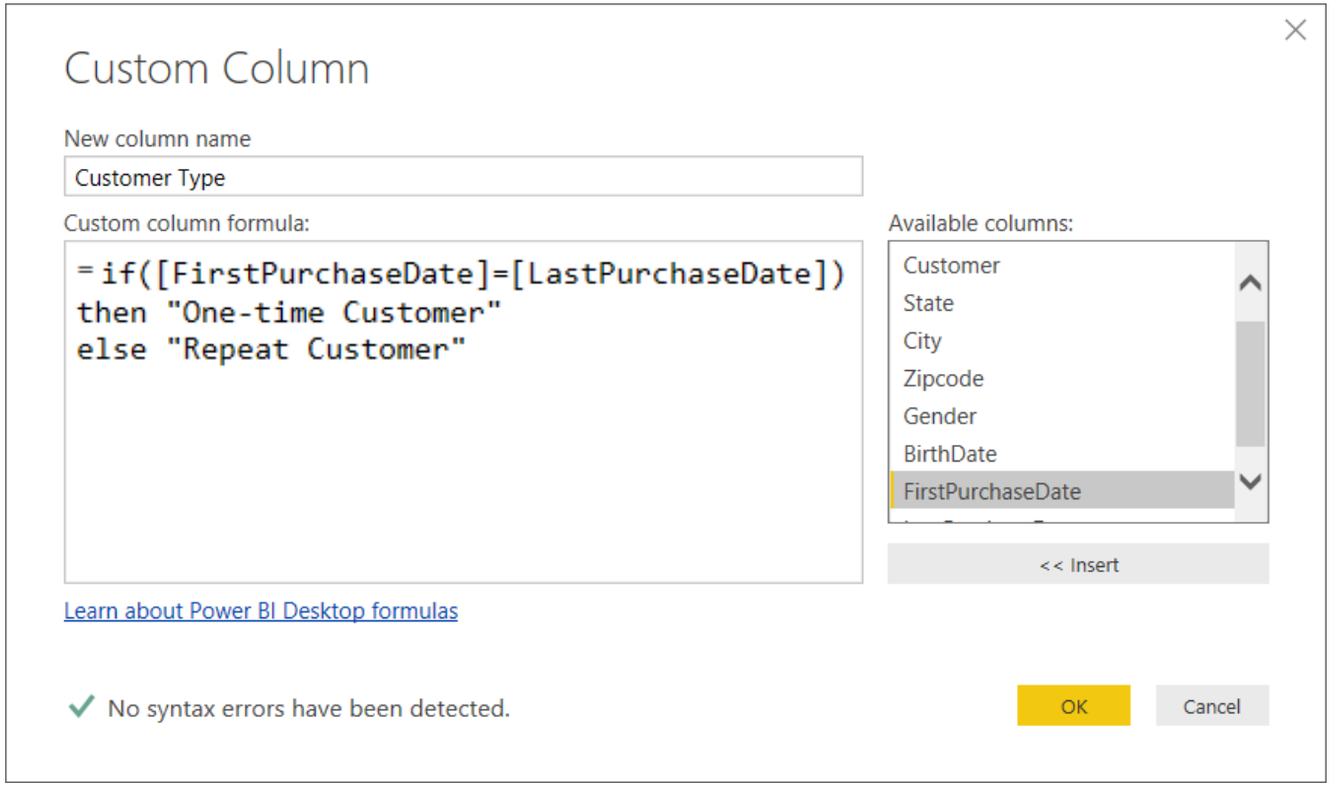


  - Makes it easier to manage project with large number of queries



Custom Column Dialog

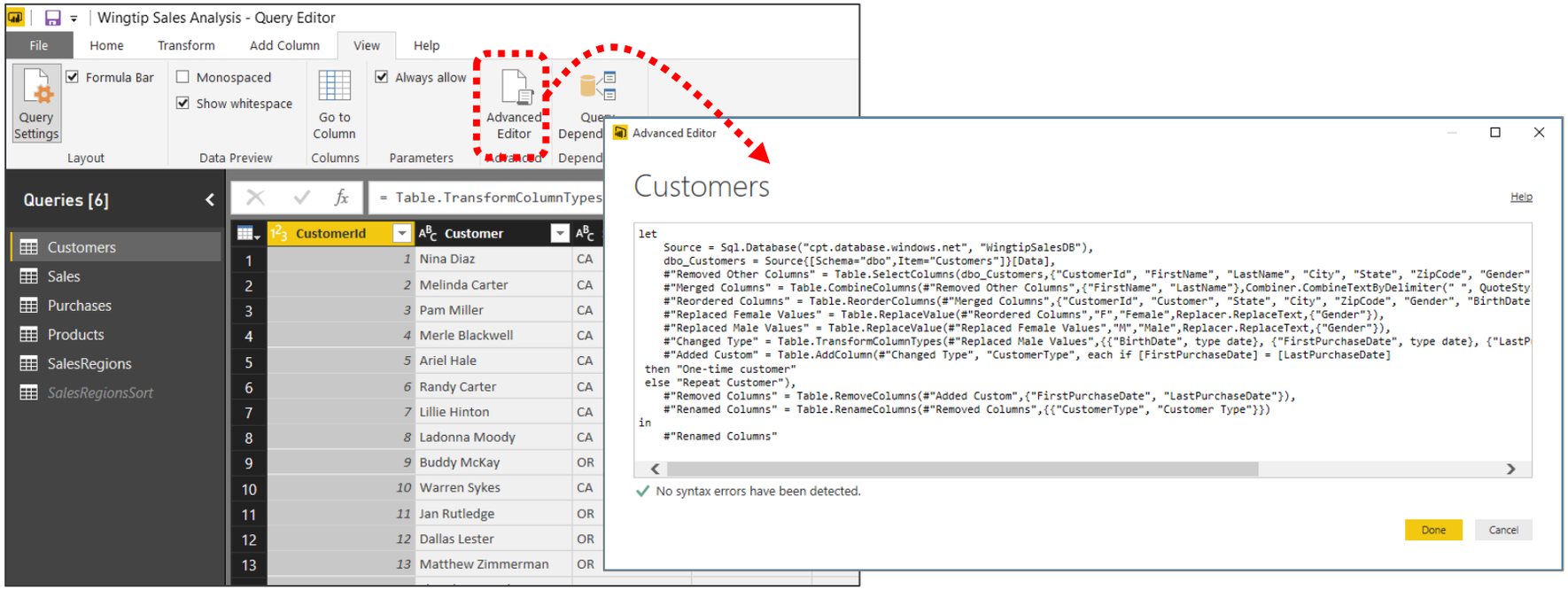
- **You can write M code directly for custom column**
  - The Custom Column dialog provides a simple M code editor



Advanced Editor

or more correctly - The Simple Editor for Advanced Users

- **Power BI Desktop based on "M" functional language**
  - Query in Power BI Desktop saved as set of M statements in code
  - Query Editor generates code in M behind the scenes
  - Advanced users can view & modify query code in Advanced Editor

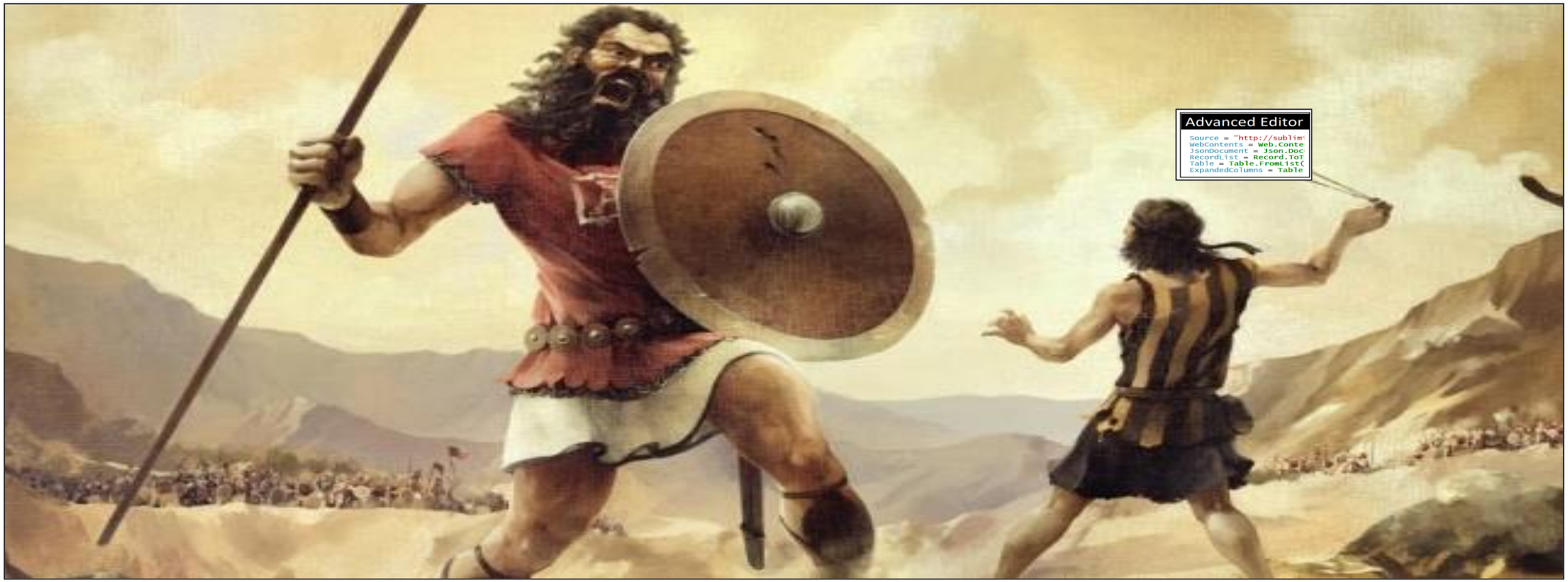


Why Should You Learn M Programming?

- **Accomplish things that cannot be done in query editor**
  - Working with query functions
  - Performing calculations across rows
  - Navigate to SharePoint list by list title instead of GUID with the ID
- **Author queries and check them into source control system**
  - Add query logic in .m files and store them in GitHub, TFS, etc.
  - Ensure query logic is the same across PBIX projects
- **Stay Ahead of the Pack and Win Admiration of Your Peers**
  - People will think you are buddies with Chris Webb!

Why Should You Learn M Programming?

- Take of the biggest of ELT challenges!



#1 Resource for Anyone Serious About Power Query

https://blog.crossjoin.co.uk

Search

CHRIS WEBB'S BI BLOG

Microsoft Power BI, Analysis Services, DAX, M, MDX, Power Query and Power Pivot

HOME ABOUT BOOKS PRIVACY POLICY SPEAKING CONTACT

Parquet File Performance In Power BI/Power Query

MARCH 21, 2021
By Chris Webb
in ADLSGEN2, PARQUET, POWER BI, POWER QUERY
1 COMMENT

There has been a lot of excitement around the newly-added support for reading from Parquet files in Power BI. However I have to admit that I was disappointed not to see any big improvements in performance when reading data from Parquet compared to reading data from CSV (for example, see here) when I first started testing it. So, is Power Query able to take advantage of Parquet's columnar storage when reading data?

FOLLOW BLOG VIA EMAIL

Enter your email address to follow this blog and receive notifications of new posts by email.

Join 15,686 other subscribers

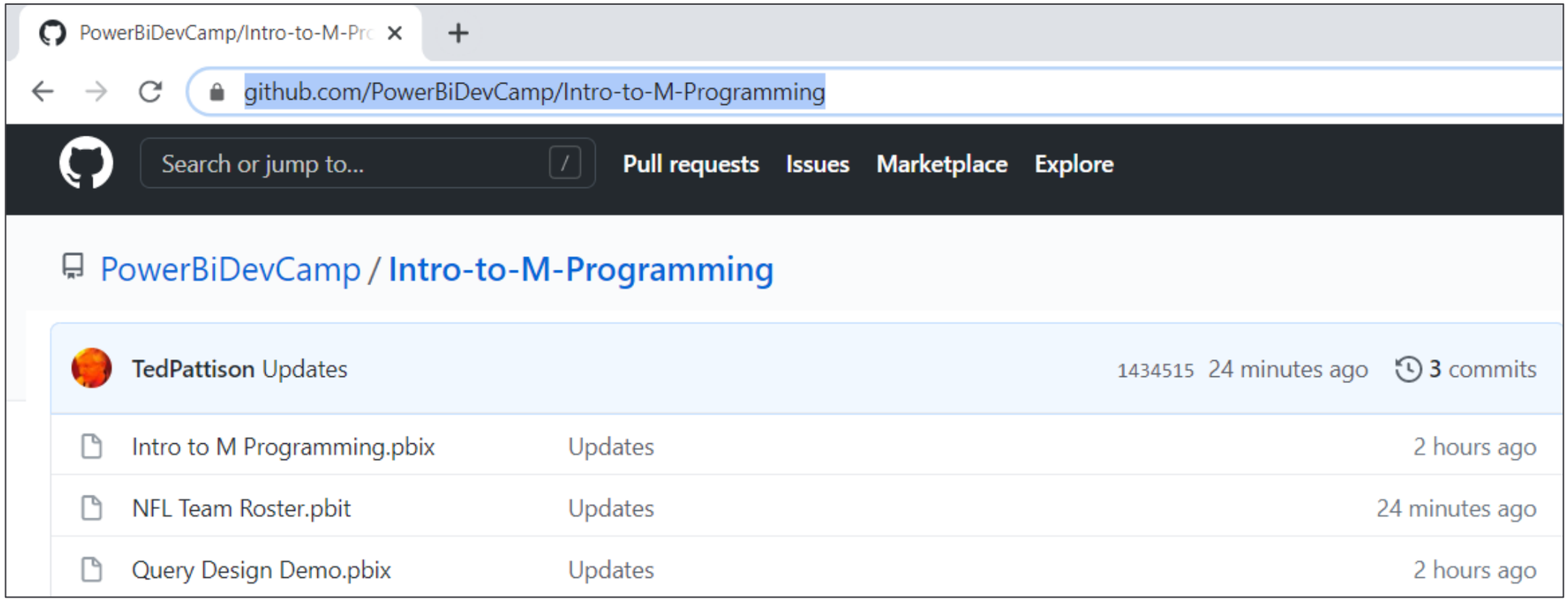
Email Address

Follow

SOCIAL

3 Sample Files Used in This Session

- https://github.com/PowerBiDevCamp/Intro-to-M-Programming



Agenda

- ✓ The Power Query Mashup Engine
- ➢ M Programming Fundamentals
- Programming Lists, Records and Tables
- Understanding Query Folding
- Choosing Between OData.Feed & Web.Contents
- Writing Reusable Function Queries
- Designing with Query Parameters

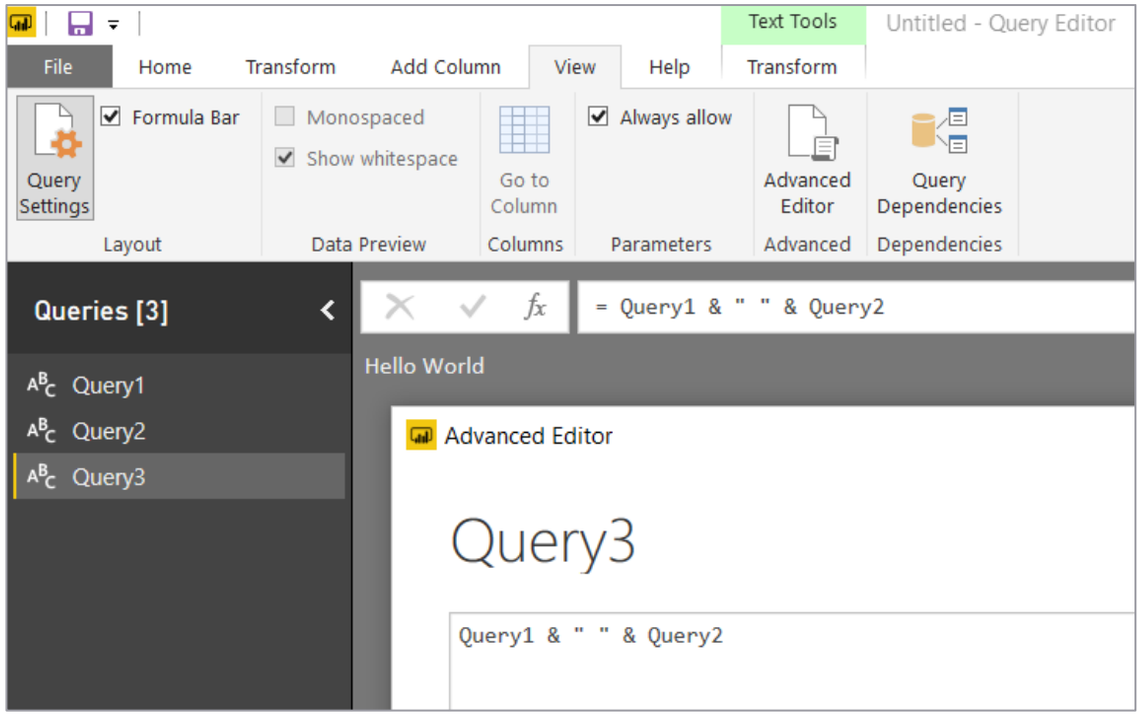
The M Programming Language

- **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

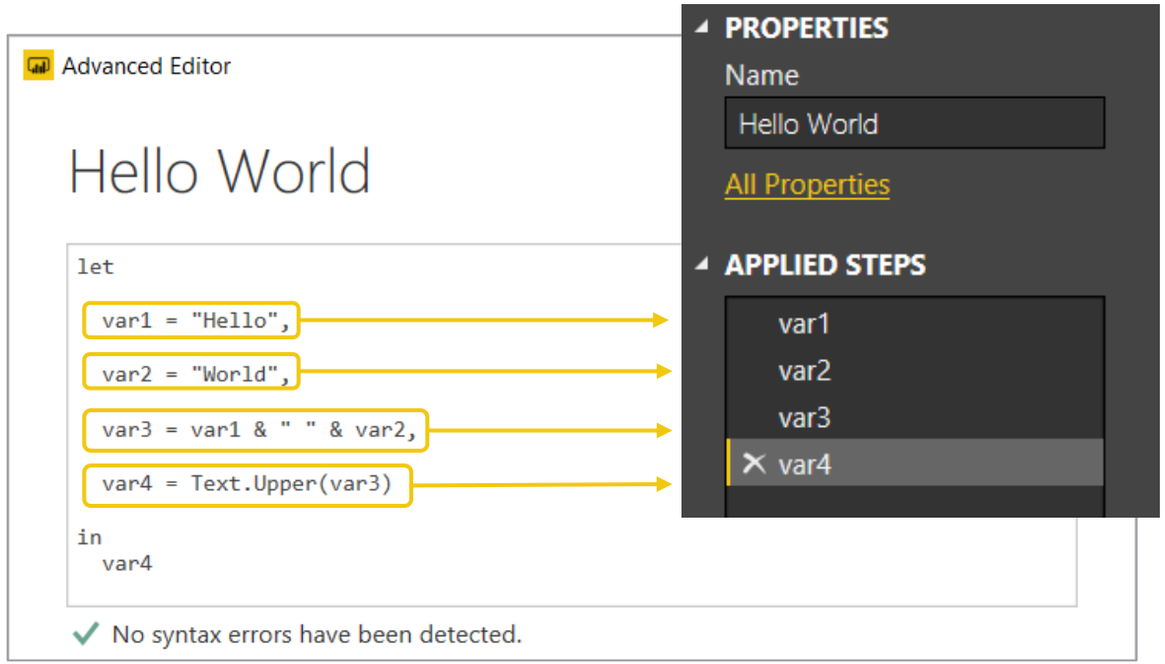
Referencing Other Queries

- **Query can reference other queries by name**
  - Every query is defined with a return type



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



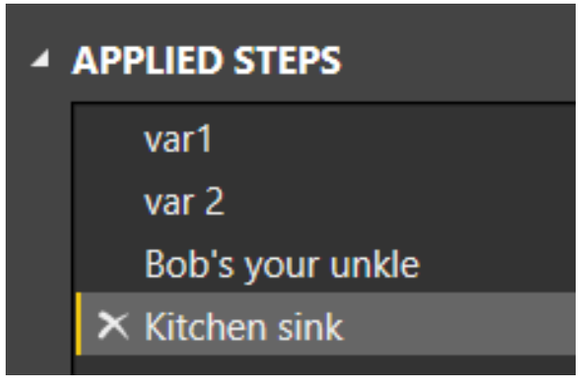
Comments and Variable Names

- **M supports using C-style comments**
  - Multiline comments created using **/* */**
  - Single line comments created using **//**

```
/*
  This is my most excellent query
*/
let

  var1 = 42, // the secret of life
```

- **Variable names with spaces must be enclosed in #" "**
  - Variable names with spaces created automatically by query designer

```
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"
```



Flow of Statement Evaluation

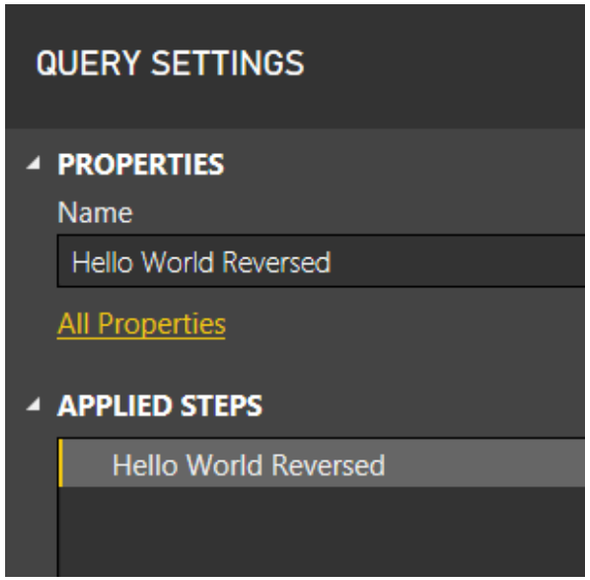
- **Evaluation starts with expression inside `in` block**
  - Expression evaluation triggers other expression evaluation

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

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
```



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
```

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 durection
var5 = #duration(0,1,30,0),
```

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

Agenda

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ➢ Programming Lists, Records and Tables
- ▪ Understanding Query Folding
- ▪ Choosing Between OData.Feed & Web.Contents
- ▪ Writing Reusable Function Queries
- ▪ Designing with Query Parameters

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
```

  - Use **{ }?** 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 charactors
input = "!!My text has some @bad things !&^",

// 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 charactors
set4 = {" ", "-", "_", "."},

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

// call Text.Select to strip out unwanted charactors
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(url, 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

```
let

  CustomersTable = Table.FromRecords({
    [ FirstName="Matt", LastName="Masson"],
    [ FirstName="Chris", LastName="Webb"],
    [ FirstName="Reza", LastName="Rad"],
    [ FirstName="Chuck", LastName="Sterling"]
  })

in
  CustomersTable
```

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

Bad, Bad, Bad ☹

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

Performing Calculations Across Rows

- Requires adding an index column

	Quarter	Sales	Index	Running Total
1	2016-Q1	124	0	124
2	2016-Q2	154	1	278
3	2016-Q3	167	2	445
4	2016-Q4	188	3	633
5	2017-Q1	150	4	783
6	2017-Q2	193	5	976
7	2017-Q3	208	6	1184
8	2017-Q4	234	7	1418

PROPERTIES

Name

Sales Running Total

All Properties

APPLIED STEPS

Source

AddedIndex

AddedCustom

Custom Column

New column name

Running Total

Custom column formula:

```
= List.Sum(List.Range(AddedIndex[Sales], 0, [Index]+1))
```

Using Each with Unary Functions

- **Many library functions take function as parameters**
  - Function passed are often unary functions *(e.g. they accept 1 parameter)*

```
FilteredRows = Table.SelectRows(CustomersTable, (row) => row[CustomerId]<=10  ),
```

- **M provides each syntax to make code easier to read/write**
  - Unary parameter passed implicitly using _ variable

```
FilteredRows = Table.SelectRows(CustomersTable, each _[CustomerId]<=10  ),
```

  - You can omit _ variable when accessing fields inside record

```
FilteredRows = Table.SelectRows(CustomersTable, each [CustomerId]<=10  ),
```

```
AddedColumn =Table.AddColumn(FilteredRows, "Display Name", each [FirstName] & " " & [LastName])
```

  - You must use _ variable when using **each** with a list

```
MyList = { "Item 1", "Item 2", "Item 3" },

MyUpperCaseList = List.Transform(MyList, each Text.Upper(_) )
```

List.Generate

- `List.Generate` accepts 3 function parameters

```
MyList = List.Generate( ()=>1, (item)=>(item<=10), (item)=>(item+1) )
```

- You can use `each` syntax for 2nd and 3rd parameter

```
MyList = List.Generate( ()=>1, each _<=10, each _+1 )
```

- You can optionally split functions out into separate expressions

```
let

  StartFunction =  ()=>10,
  TestFunction = each (_ <= 70),
  IncrementFunction = each (_ + 10),

  MyList = List.Generate( StartFunction, TestFunction, IncrementFunction)

in
  MyList
```

	List
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Agenda

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ Programming Lists, Records and Tables
- ➢ Understanding Query Folding
- ▪ Choosing Between OData.Feed & Web.Contents
- ▪ Writing Reusable Function Queries
- ▪ Designing with Query Parameters

Query Folding

- **Mashup engine pushes work back to datasource when possible**
  - Column selection and row filtering
  - Joins, Group By, Aggregate Operations
- **Datasource that support folding**
  - Relational database
  - Tabular and multidimensional databases
  - OData Web services
- **What happens when datasource doesn't support query folding?**
  - All work is done locally by the mashup engine
- **Things that affect whether query folding occurs**
  - The way you structure your M code
  - Privacy level of datasources
  - Native query execution

Query Folding Example

- When you execute this query in Power BI Desktop…

```
let
  Source = Sql.Database("ODYSSEUS", "WingtipSalesDB"),
  CustomersTable = Source{[Item="Customers"]}[Data],

  // select rows
  FilteredRows = Table.SelectRows(CustomersTable, each ([State] = "FL")),

  // select columns
  ColumnsToKeep = {"CustomerId", "FirstName", "LastName"} ,
  RemovedOtherColumns = Table.SelectColumns(FilteredRows, ColumnsToKeep),

  // rename columns
  ColumnRenamingMap = { {"FirstName", "First Name"}, {"LastName", "Last Name"} },
  RenamedColumns = Table.RenameColumns(RemovedOtherColumns, ColumnRenamingMap)

in
  RenamedColumns
```

- Mashup Engine executes the following SQL query

```
execute sp_executesql
N'select [_].[CustomerId] as [CustomerId],
         [_].[FirstName] as [First Name],
         [_].[LastName] as [Last Name]
  from [dbo].[Customers] as [_]
  where [_].[State] = ''FL'' and [_].[State] is not null'
```

Native Queries

- No query folding occurs after native query

```
let

  DatabaseServer = "cpt.database.windows.net",

  DatabaseName = "WingtipSalesDB",

  SQL = "SELECT CustomerId, FirstName, LastName" &
        " FROM Customers" &
        " WHERE CustomerId <= 10" &
        " ORDER BY LastName, FirstName" ,

  Source = Sql.Database( DatabaseServer, DatabaseName , [Query=SQL] ),

  output = Source

in
  output
```

Search

Chris Webb's BI Blog

Microsoft Power BI, Analysis Services, DAX, M, MDX, Power Query and Power Pivot

HOME ABOUT BOOKS PRIVACY POLICY SPEAKING CONTACT

Query Folding On SQL Queries In Power Query Using Value.NativeQuery() and EnableFolding=true

February 21, 2021
By Chris Webb
in M, Power BI, Power Query

FOLLOW BLOG VIA EMAIL

Enter your email address to follow this blog and receive notifications of new posts by email.

Join 15,686 other subscribers

Email Address

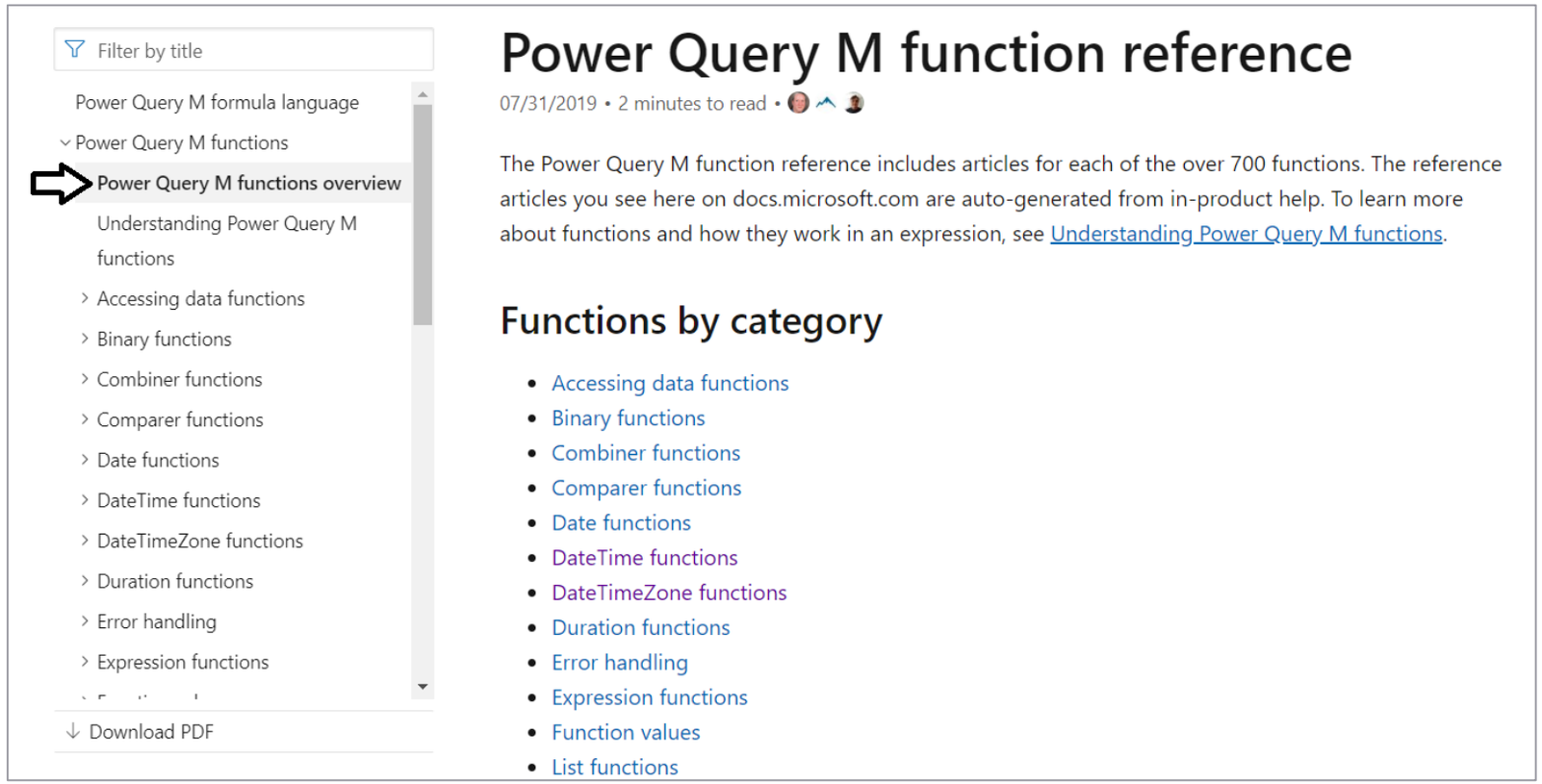
```
let
  Source = Sql.Databases("localhost"),
  AdventureWorksDW2017 = Source
    {[Name = "AdventureWorksDW2017"]}
    [Data],
  RunSQL = Value.NativeQuery(
    AdventureWorksDW2017,
    "SELECT EnglishDayNameOfWeek FROM DimDate",
    null,
    [EnableFolding = true]
  ),
  #"Filtered Rows" = Table.SelectRows(
    RunSQL,
    each (
      [EnglishDayNameOfWeek] = "Friday"
    )
  )
in
  #"Filtered Rows"
```

Agenda

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ Programming Lists, Records and Tables
- ✓ Understanding Query Folding
- ➢ Choosing Between OData.Feed & Web.Contents
- ▪ Writing Reusable Function Queries
- ▪ Designing with Query Parameters

M Function Library

- Check out the Power Query M function reference
  - https://docs.microsoft.com/en-us/powerquery-m/power-query-m-function-reference



Accessing Data using OData.Feed

- **OData.Feed can pull data from OData web service**
  - OData connector assists with navigation through entities
  - OData connector support query folding

```
let
    Source = OData.Feed("http://subliminalsystems.com/api/"),

    // get Customers table
    CustomersTable = Source{[Name="Customers",Signature="table"]}[Data],

    // select columns
    ColumnsToKeep = {"CustomerId", "FirstName", "LastName", "City", "State", "Zipcode", "Gender", "BirthDate"},
    RemovedOtherColumns = Table.SelectColumns(CustomersTable, ColumnsToKeep),

    // select rows
    FilteredRows = Table.SelectRows(RemovedOtherColumns, each [CustomerId] <= 10),

    // perform other transforms
    ReplacedValue = Table.ReplaceValue(FilteredRows,"F","Female",Replacer.ReplaceText,{"Gender"}),
    ReplacedValue1 = Table.ReplaceValue(ReplacedValue,"M","Male",Replacer.ReplaceText,{"Gender"}),
    ChangedType = Table.TransformColumnTypes(ReplacedValue1,{{"BirthDate", type date}}),
    MergedColumns = Table.CombineColumns(ChangedType,{"FirstName", "LastName"},
                                         Combiner.CombineTextByDelimiter(" ", QuoteStyle.None),
                                         "Customer")

in
    MergedColumns
```

- **OData makes extra calls to acquire metadata**
  - Let's look at the execution of this query using Fiddler

Web.Contents

- **Can be more efficient than OData.Feed**
  - You can pass OData query string parameters (e.g. $select)

```
let
    // create REST URI for OData source
    Source = "http://subliminalsystems.com/api/Customers?" &
             "?$select=CustomerId,FirstName,LastName,City,State,Zipcode,Gender,BirthDate" &
             "&filter=(CustomerId+le+10)",

    // create options record for calling Web.Contents
    OptionsRecord =  [Headers=[Accept="application/json;odata=nometadata",
                               #"OData-MaxVersion"="4.0"]],

    // call Web.Content to make call across network
    WebContents = Web.Contents(Source, OptionsRecord),

    // deal with JSON dataset return by Web.Contents
    JsonDocument = Json.Document(WebContents),
    RecordList = Record.ToTable(JsonDocument){1}[Value],
    Table = Table.FromList(RecordList, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    ColumnsToExpand = {"CustomerId", "FirstName", "LastName", "City", "State", "Zipcode", "Gender", "BirthDate"},
    ExpandedColumns = Table.ExpandRecordColumn(Table, "Column1", ColumnsToExpand, ColumnsToExpand),
```

Agenda

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ Programming Lists, Records and Tables
- ✓ Understanding Query Folding
- ✓ Choosing Between OData.Feed & Web.Contents
- ➢ Writing Reusable Function Queries
- ▪ Designing with Query Parameters

Understanding Function Queries

- **Query can be converted into reusable function**
  - Requires editing query M code in Advanced Editor
  - Function query defined with one or more parameters

GetExpensesFromFile

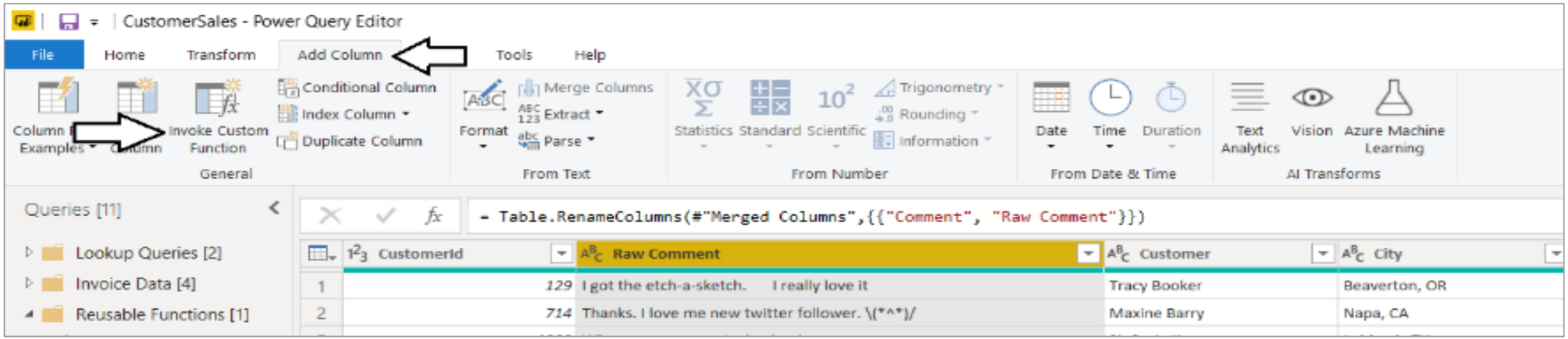
```
(FilePath as text) =>

let
    Source = Csv.Document(Web.Contents(FilePath)
    #"Changed Type" = Table.TransformColumnTypes
```

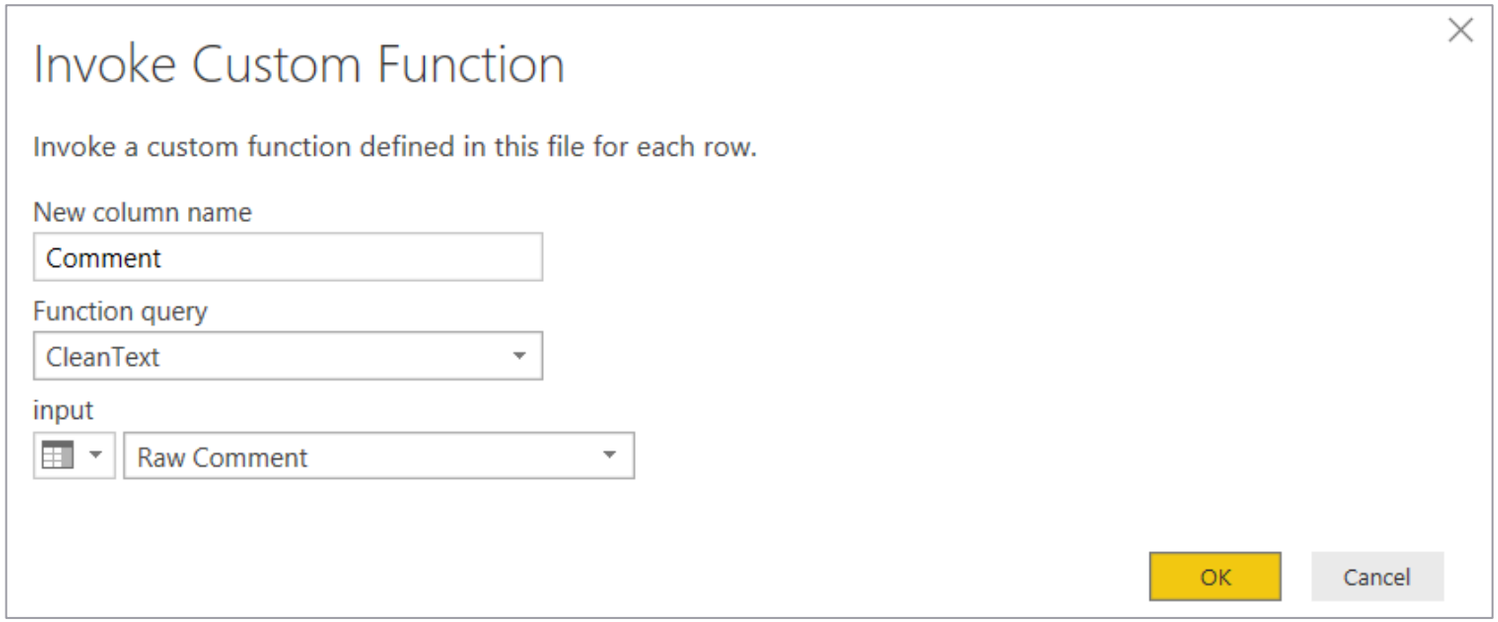
  - Function query can be called from other queries
  - Function query can be called using Invoke Custom Function
  - Function query can't be edited with visual designer

Creating a Function Query

- Requires adding parameter list

Calling a Function Query





Agenda

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ Programming Lists, Records and Tables
- ✓ Understanding Query Folding
- ✓ Choosing Between OData.Feed & Web.Contents
- ✓ Writing Reusable Function Queries
- ➢ Designing with Query Parameters

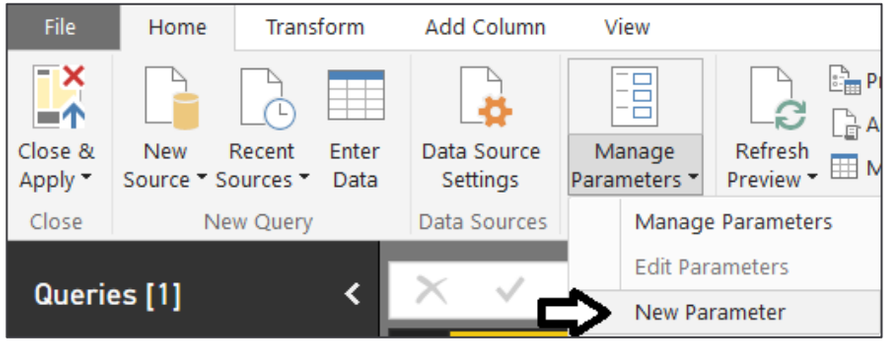
Query Parameters in Power BI Desktop

- **What is a Query Parameter?**
  - Configurable setting for PBIX file
  - Strongly-typed value to which you can apply restrictions
  - Can be referenced from a query
  - Can be referenced from DAX code in data model
  - Query parameters can be updated in Power BI Service

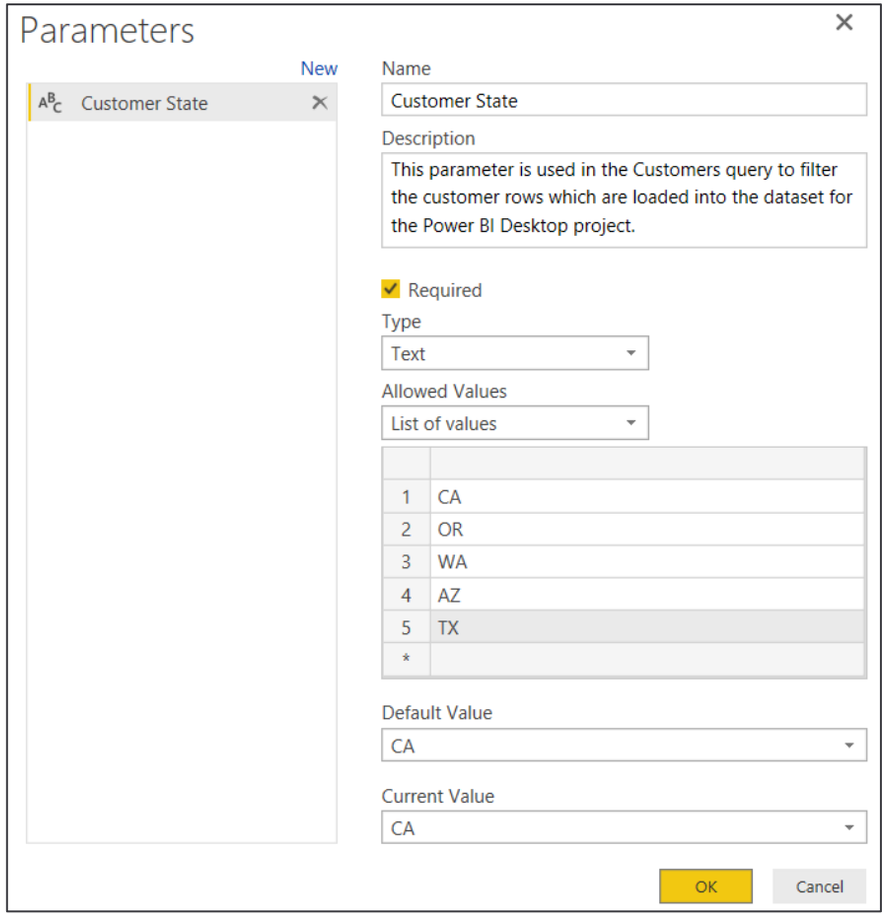
- **Where are Parameters commonly used**
  - To parameterize data source connection details
  - To filter rows when importing data
  - Commonly used together with Template Apps

Creating Query Parameters

- Parameters can be created using Manager Parameters menu

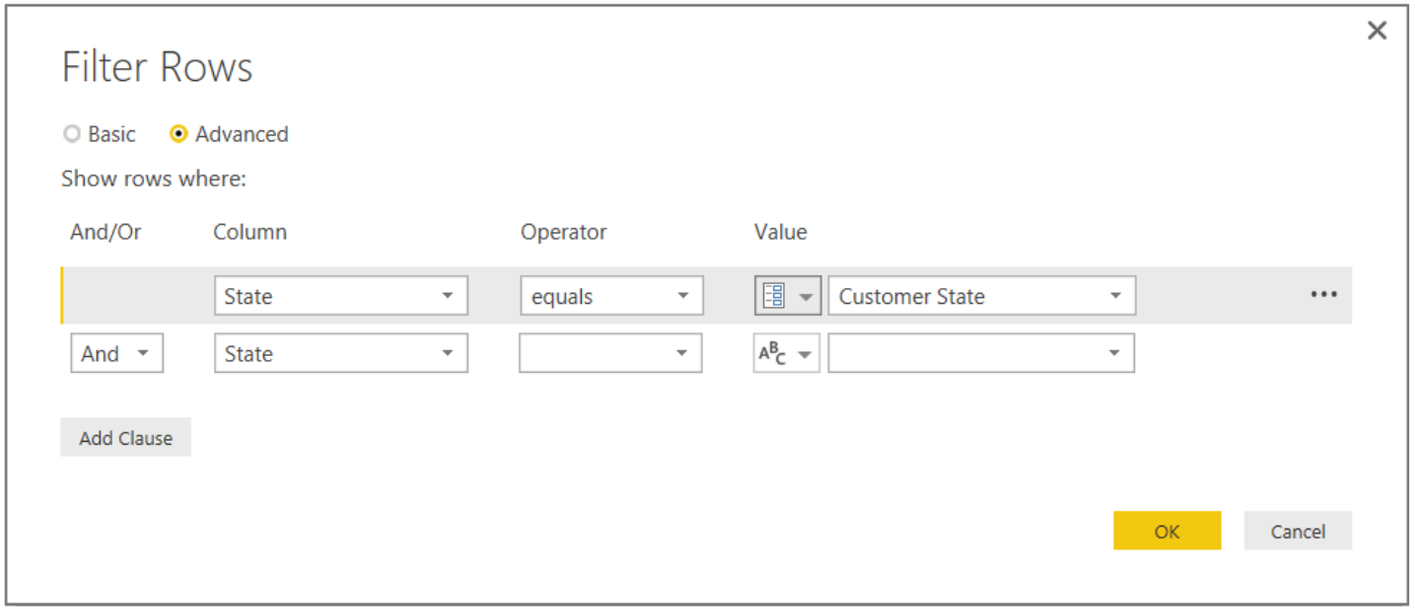


- Parameter properties
  - Name
  - Description
  - Required
  - Allowed Values
  - Default Value
  - Current Value

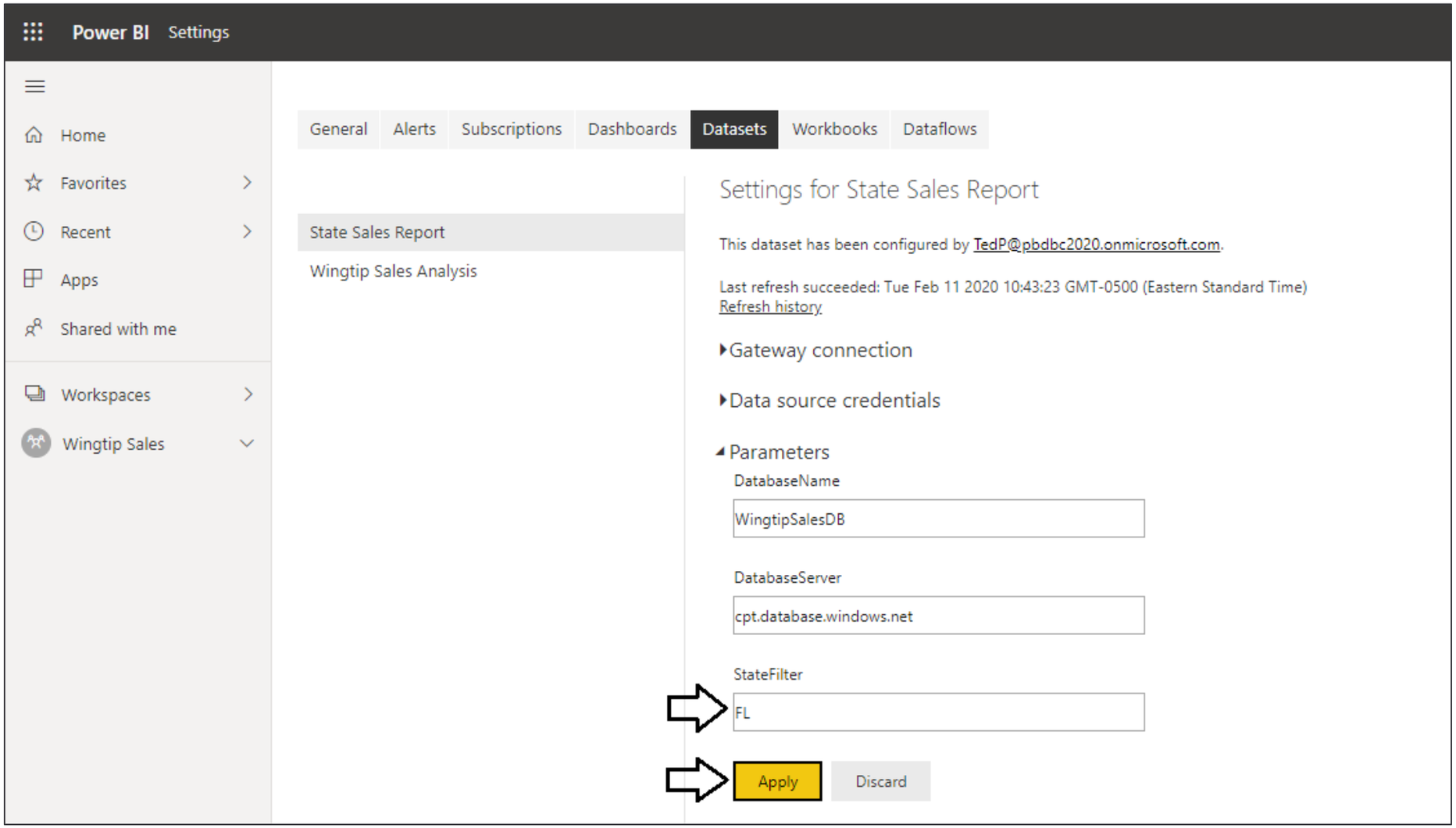


Referencing Parameters in a Query

- **Parameters can be referenced inside query**
  - Next query execution uses current parameter value

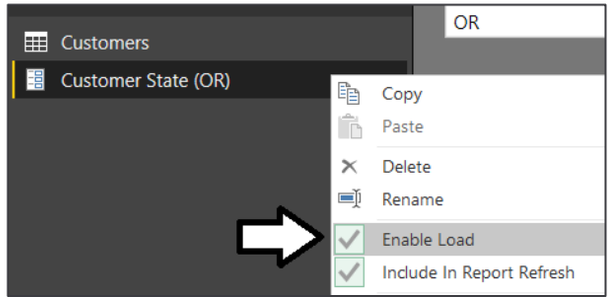


Updating Parameters in the Service

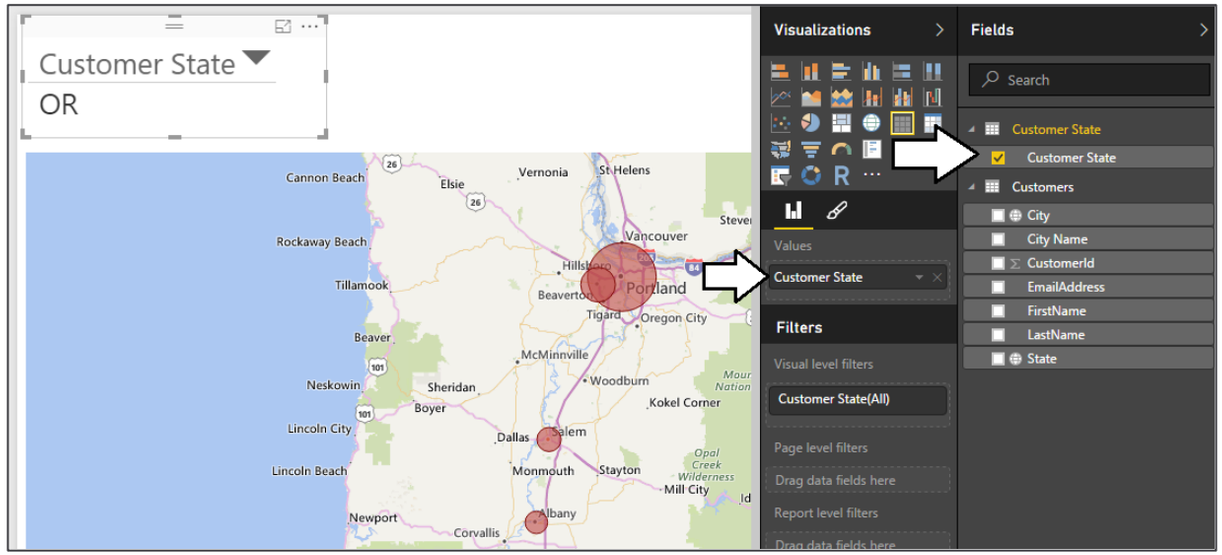


Making Parameters Available to Data Model

- Configure parameter's Enable Load setting

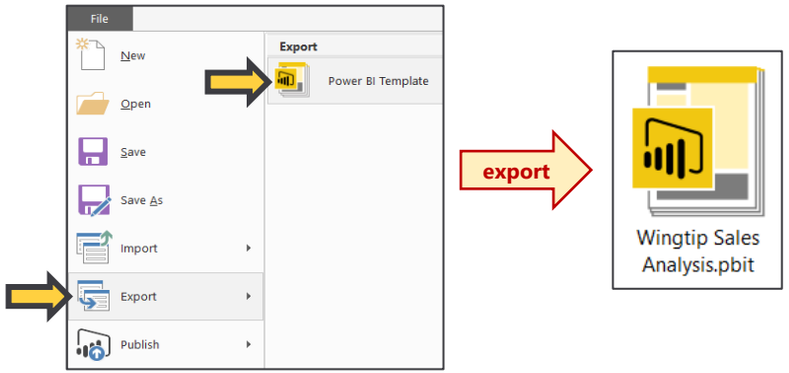


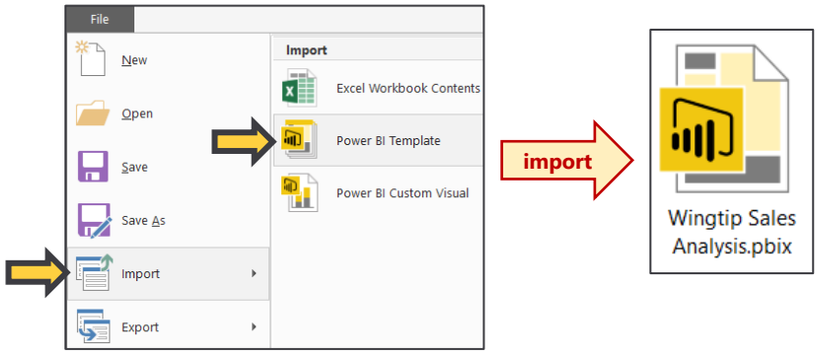
- Parameter becomes visible within fields list in report view



Power BI Project Template Files

- **PBIX project can be exported to project template file**
  - Template file created with PBIT file extension
  - Generated template files contains everything except for the data
  - PBIT template file can be imported to create new PBIX projects
  - Template files are powerful when used together with parameters
- **How are template files used?**
  - Export PBIX project to create a PBIT template file
  - Import the PBIT template file to create a new PBIX project





Summary

- ✓ The Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ Programming Lists, Records and Tables
- ✓ Understanding Query Folding
- ✓ Choosing Between OData.Feed & Web.Contents
- ✓ Writing Reusable Function Queries
- ✓ Designing with Query Parameters

Questions?

Microsoft Power BI