

Designing Queries with Power BI Desktop



Agenda

- Course Introduction
- Importing Data using Power Query
- Writing Query Logic in M
- Understanding Query Folding
- Writing Reusable Function Queries



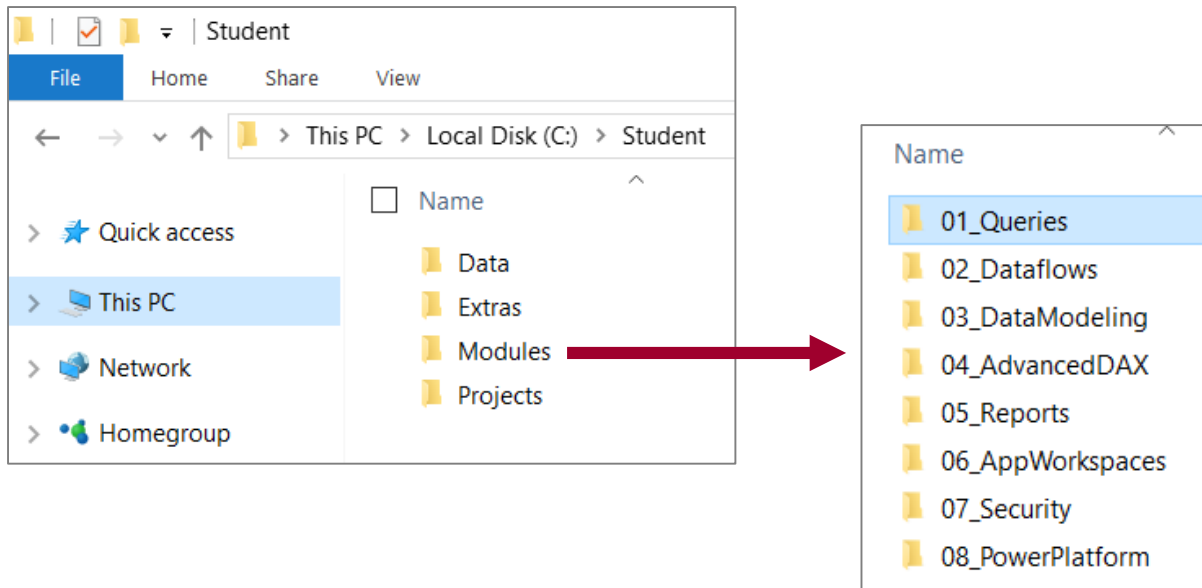
Student Background

- What is your name?
- What are you doing with Power BI?
- Which products/services have you used
 - Excel
 - Access
 - SQL Server, SSRS, SSAS
 - SharePoint and Office 365
 - Tableau
 - Dynamics 365
 - Salesforce
 - Others



Student Files for This Course

- Copy the **Student** folder from the master zip archive
 - Create a new local folder at **C:\Student**
- Each module has folder inside **Student\Modules** folder
 - Slides and lab writeup available through student manual (not in GitHub repository)



What is Power BI?

- What is Power BI?
 - A cloud-based analytics service for licensed subscribers
 - Environment which supports and promotes self-service BI
 - Powerful builder tools for importing, modeling and visualizing data
 - Enterprise-grade platform for deploying reports and dashboards
 - On-premises server product supporting subset of cloud features

Power BI
Service

Power BI
Desktop

Power BI
Report Server

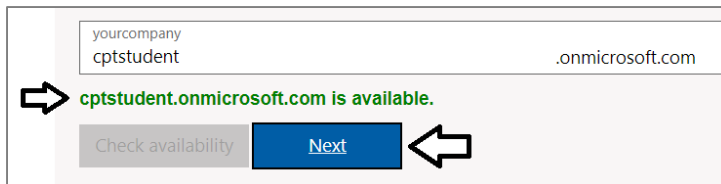


Creating a Power BI Lab Environment

- Sign up for an Office 365 E5 trial account
 - Sign up process creates new Azure AD tenant
 - Tenant created with user account which is Global tenant admin
 - Tenant gets 30-day trial subscription for 25 Office 365 E5 licenses



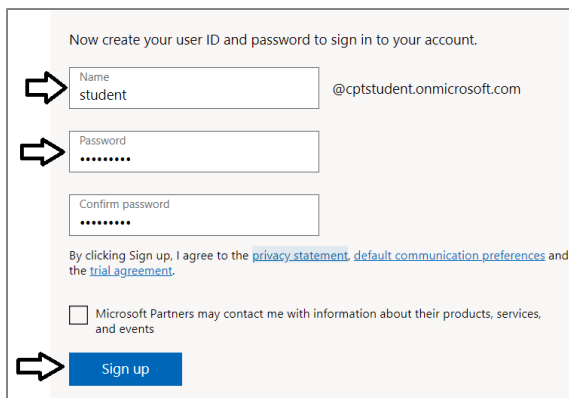
yourcompany
cptstudent .onmicrosoft.com



yourcompany
cptstudent .onmicrosoft.com

→ cptstudent.onmicrosoft.com is available.

Check availability Next



Now create your user ID and password to sign in to your account.

→ Name student @cptstudent.onmicrosoft.com

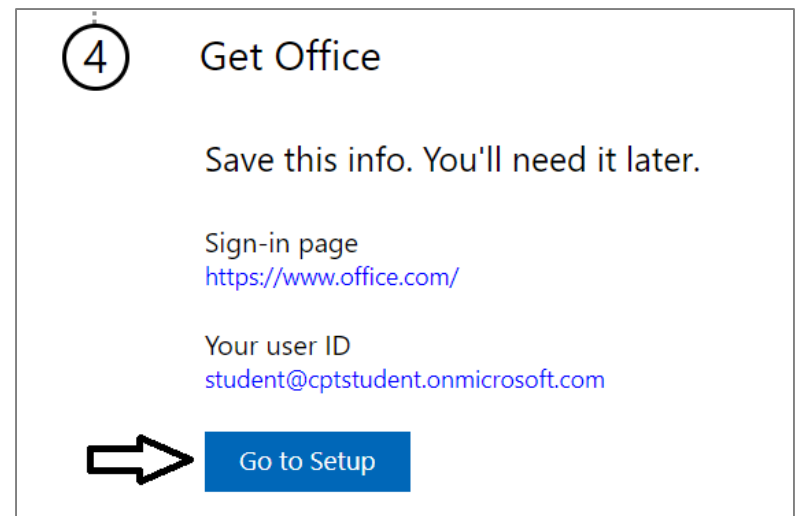
→ Password

Confirm password

By clicking Sign up, I agree to the [privacy statement](#), [default communication preferences](#) and the [trial agreement](#).

☐ Microsoft Partners may contact me with information about their products, services, and events

→ Sign up



④ Get Office

Save this info. You'll need it later.

Sign-in page
<https://www.office.com/>

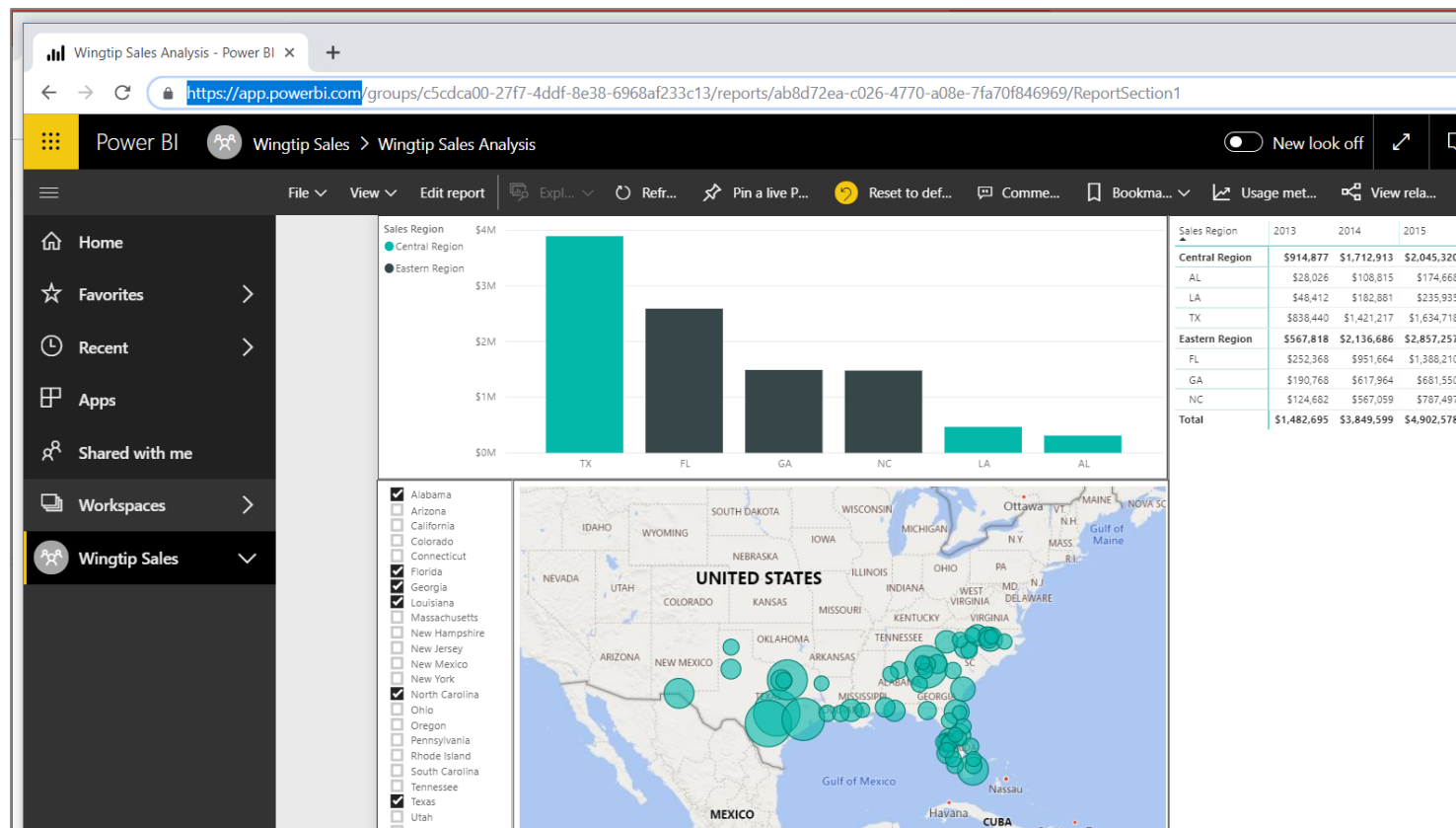
Your user ID
[student@cptstudent.onmicrosoft.com](#)

→ Go to Setup

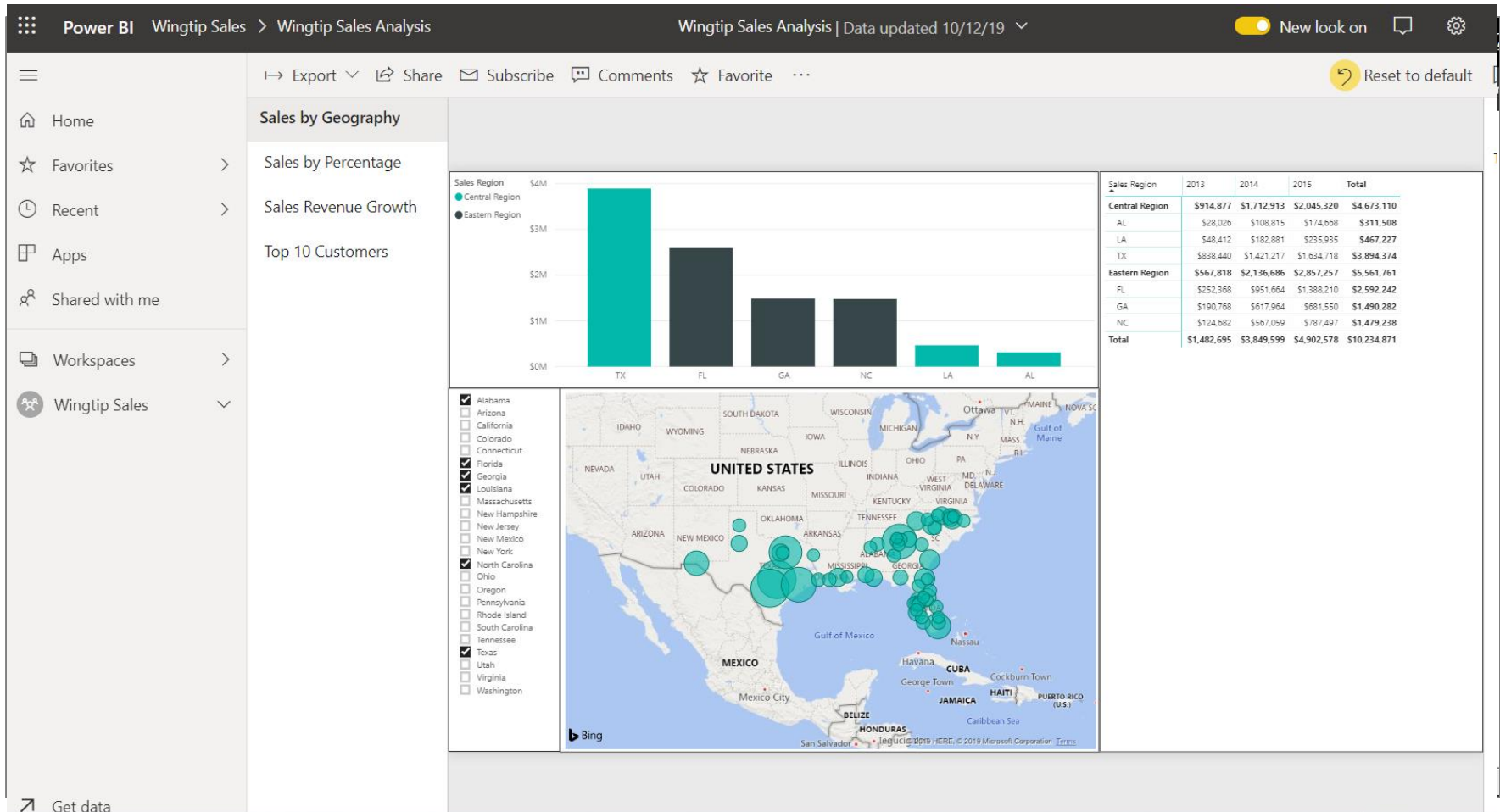


The Power BI Service

- The Power BI Service
 - Provides cloud-based foundation for Power BI platform
 - Provides browser-based portal at <https://app.powerbi.com>



Light Grey is the New Black



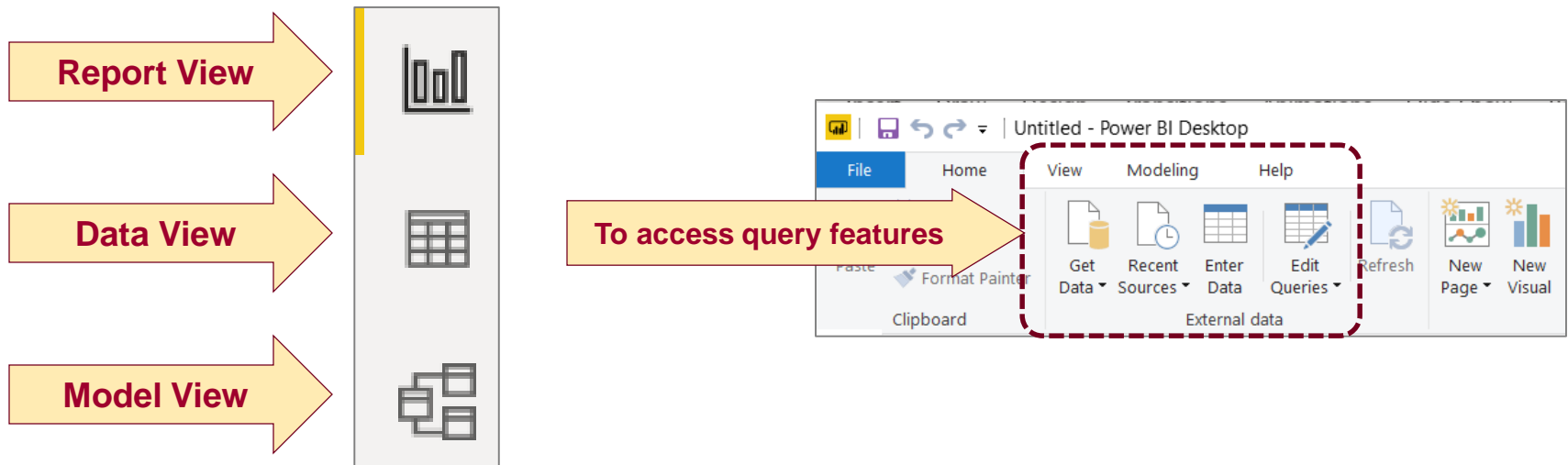
Central Power BI Concepts

- Workspace
 - Secured container for Power BI resources
 - Created as personal workspaces and app workspaces
- Dashboard
 - Consolidated high-level view into reports and datasets
 - Provides great experience on mobile device (*e.g. iPhone, Android, tablet, etc.*)
- Report
 - Collection of one or more pages with tables & visualizations
 - Provides consumer with interactive control through filtering and bookmarks
- Dataset
 - In-memory data model containing one or more tables
 - Used to supply the underlying data to reports and dashboards
- Dataflows
 - Persistent data store used for more complex ETL requirements
 - Not required in most Power BI scenarios



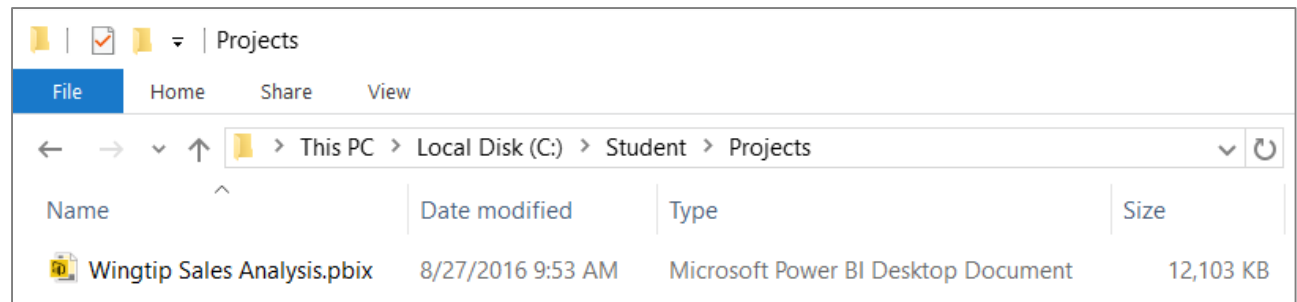
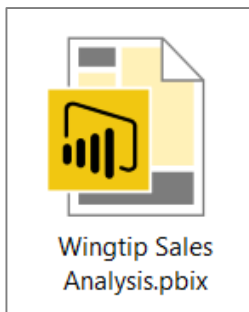
Getting Around in Power BI Desktop

- What do you need to learn to use Power BI Desktop?
 - Query features for importing data
 - Designing data model & writing DAX expressions
 - Designing reports with Power BI Desktop report designer
- Navigating between view modes



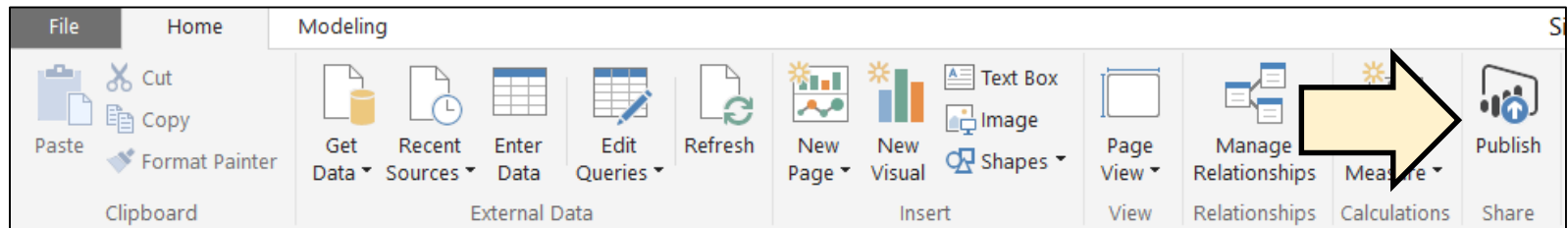
Projects and PBIX Files

- Power BI Desktop projects saved using PBIX files
 - PBIX file contains data source definitions
 - PBIX file contains query definitions
 - PBIX file contains data imported from queries
 - PBIX file contains exactly one data model definition
 - PBIX file contains exactly one report
 - PBIX file never contains data source credentials

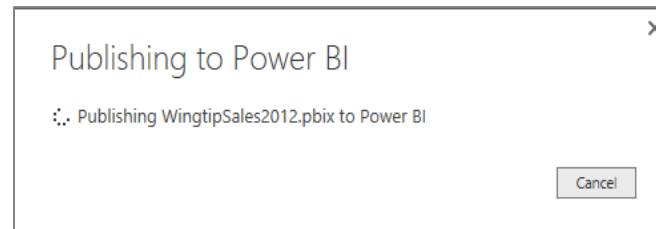
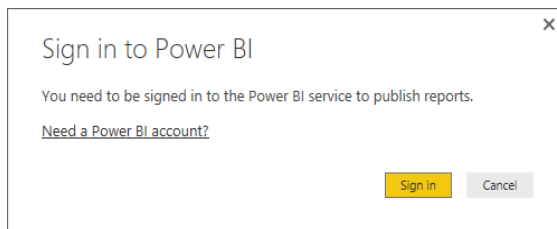


Publishing a Power BI Desktop Project

- Power BI Desktop provides **Publish** command
 - Used to publish project to Power BI service



- Requires logging into your Office 365 account



- Published articles added to target workspace



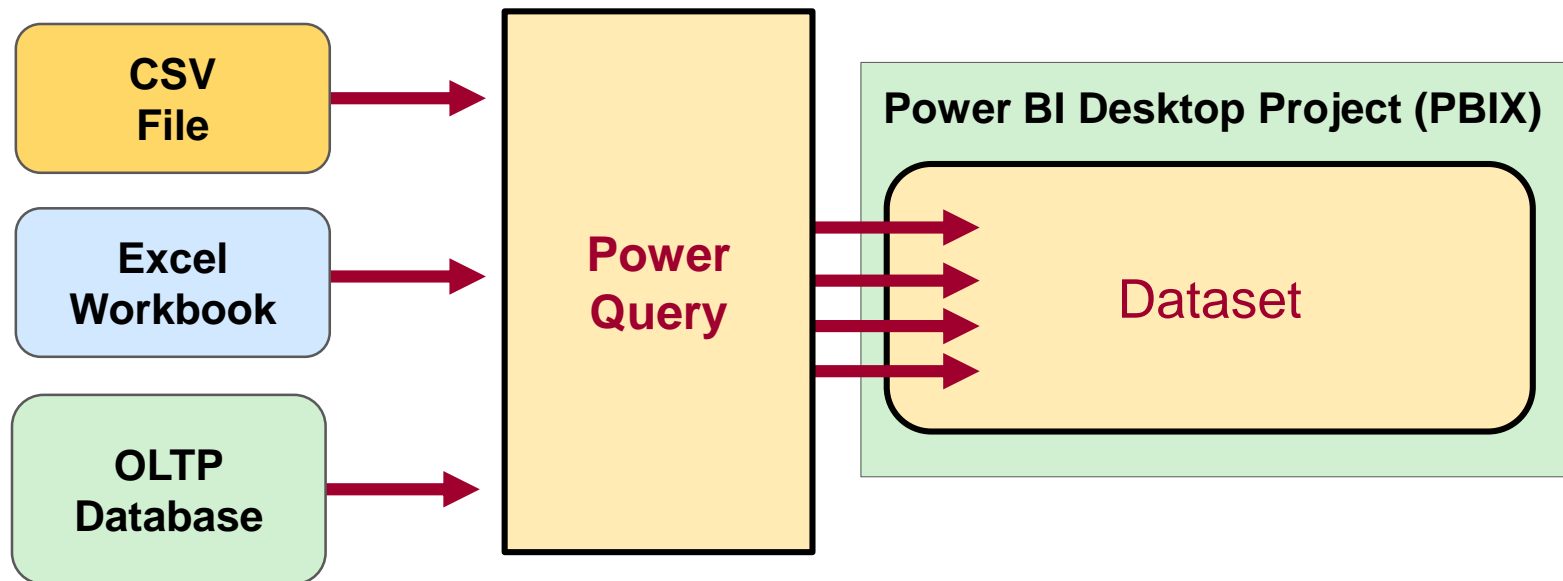
Agenda

- ✓ Course Introduction
- Importing Data using Power Query
 - Writing Query Logic in M
 - Understanding Query Folding
 - Writing Reusable Function Queries



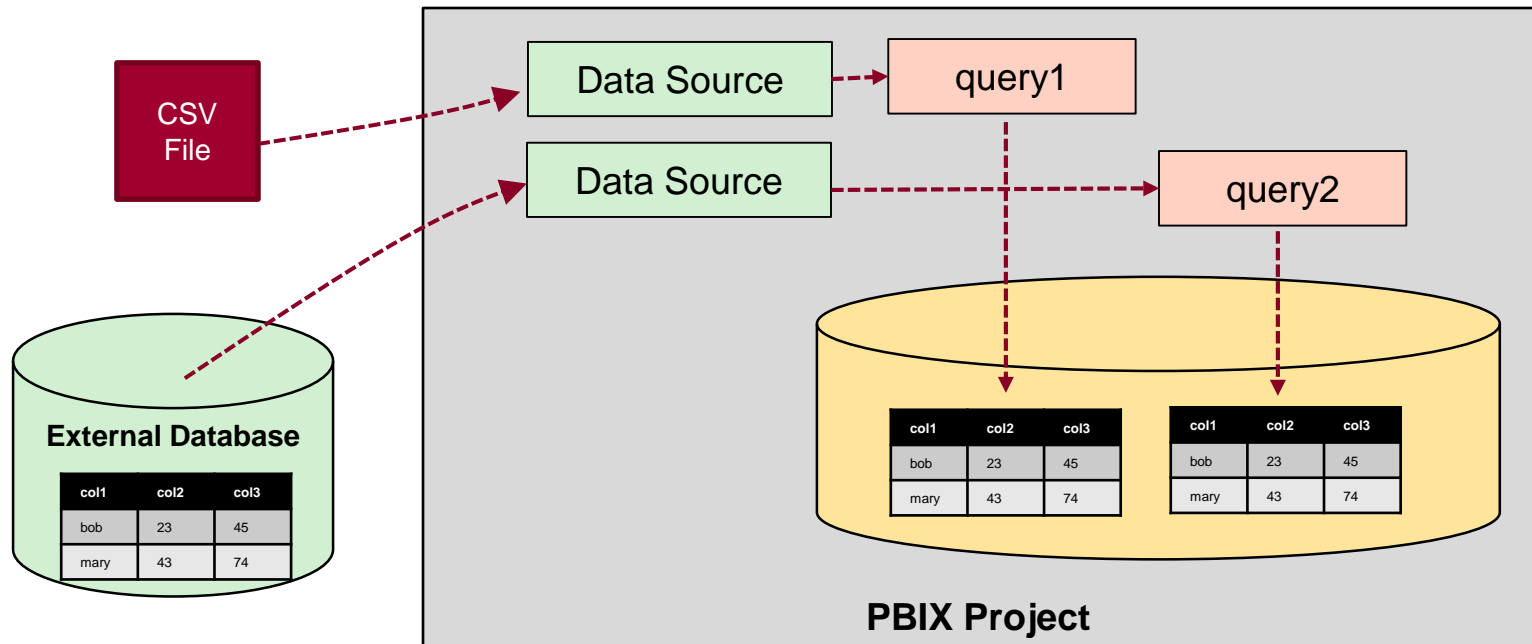
Power Query is an ETL Tool

- 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



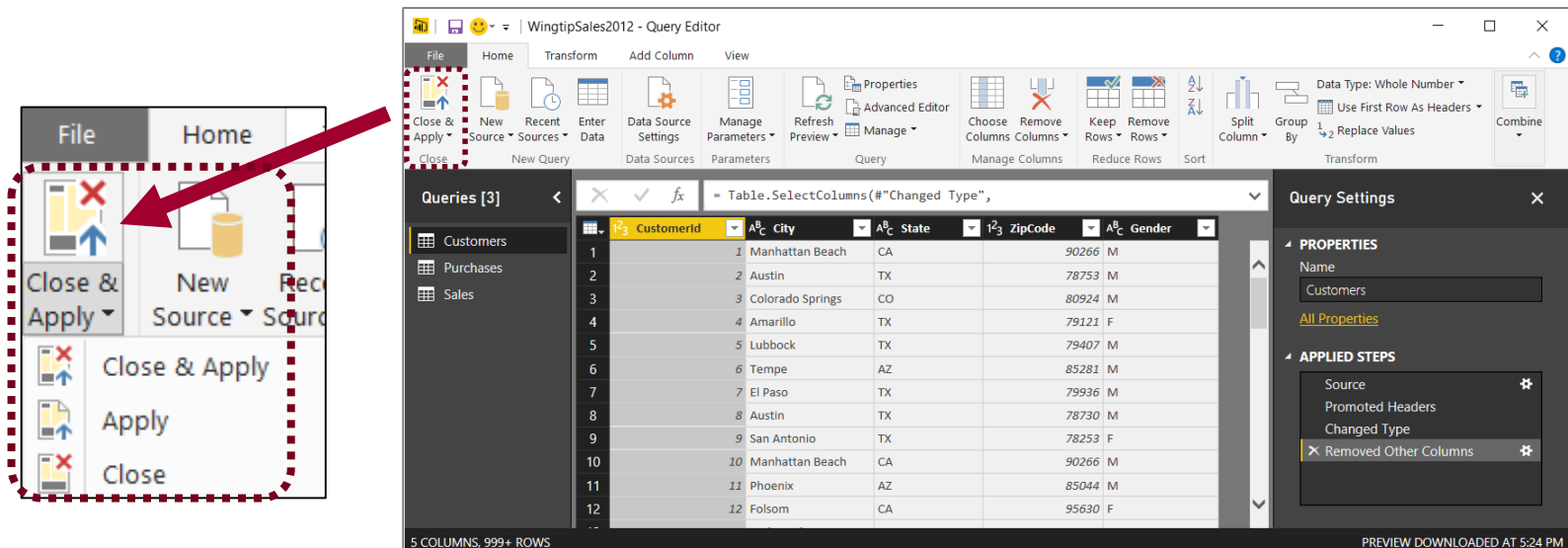
Understanding Query Input and Output

- PBIX project is container for data sources and queries
 - Queries created and saved within scope of Power BI project
 - Queries can pull data from local files
 - Queries can pull data from external content sources
 - Queries main purpose is to load imported data into data model



Query Editor Window

- Power BI Desktop provides separate Query Editor window
 - Provides powerful features for designing queries
 - Displays list of all queries in project on the left
 - Displays **Properties** and **Applied Steps** for selected query on right
 - 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 on the data
 - Each step has formula which can be viewed/edited in formula bar
 - Query starts with Source step to extract data from a data source
 - Additional steps added to perform transform operations on data
 - You can replay query operations one by one by clicking on steps

The screenshot displays the Power BI Query Editor interface. At the top, the ribbon includes 'File', 'Home', 'Transform', 'Add Column', and 'View'. Below the ribbon, the 'Formula Bar' is active, showing the formula: `= Table.ReplaceValue("#Replaced Female Values","M","Male",Replacer.ReplaceText,`. A red dashed box highlights the formula bar, with a callout box labeled 'step formula bar' pointing to it. On the left, the 'Queries [6]' pane lists 'Customers', 'Sales', 'Purchases', 'Products', 'SalesRegions', and 'SalesRegionsSort'. The main area shows a data table with columns: CustomerId, Customer, State, City, Zipcode, and Gender. The table contains 14 rows of data. On the right, the 'Query Settings' pane is open, showing the 'Properties' section with 'Name' set to 'Customers'. Below it, the 'Applied Steps' section is highlighted with a red dashed box, showing a sequential list of steps: Source, Navigation, Removed Other Columns, Merged Columns, Reordered Columns, Replaced Female Values, Replaced Male Values (selected), Changed Type, and Added Conditional Column. A callout box labeled 'sequential list of steps for query' points to this list.

| CustomerId | Customer | State | City | Zipcode | Gender |
|------------|-------------------|-------|------------|---------|--------|
| 1 | Nina Diaz | CA | Eureka | 95501 | Female |
| 2 | Melinda Carter | CA | Napa | 94558 | Female |
| 3 | Pam Miller | CA | Napa | 94558 | Female |
| 4 | Merle Blackwell | CA | Sacramento | 95823 | Female |
| 5 | Ariel Hale | CA | Sacramento | 95818 | Male |
| 6 | Randy Carter | CA | Sacramento | 95818 | Male |
| 7 | Lillie Hinton | CA | Eureka | 95501 | Female |
| 8 | Ladonna Moody | CA | Napa | 94559 | Female |
| 9 | Buddy McKay | OR | Bend | 97701 | Male |
| 10 | Warren Sykes | CA | Sacramento | 95818 | Male |
| 11 | Jan Rutledge | OR | Portland | 97216 | Female |
| 12 | Dallas Lester | OR | Eugene | 97402 | Male |
| 13 | Matthew Zimmerman | OR | Portland | 97220 | Male |
| 14 | Sheryl Hernandez | CA | Sacramento | 95823 | Female |

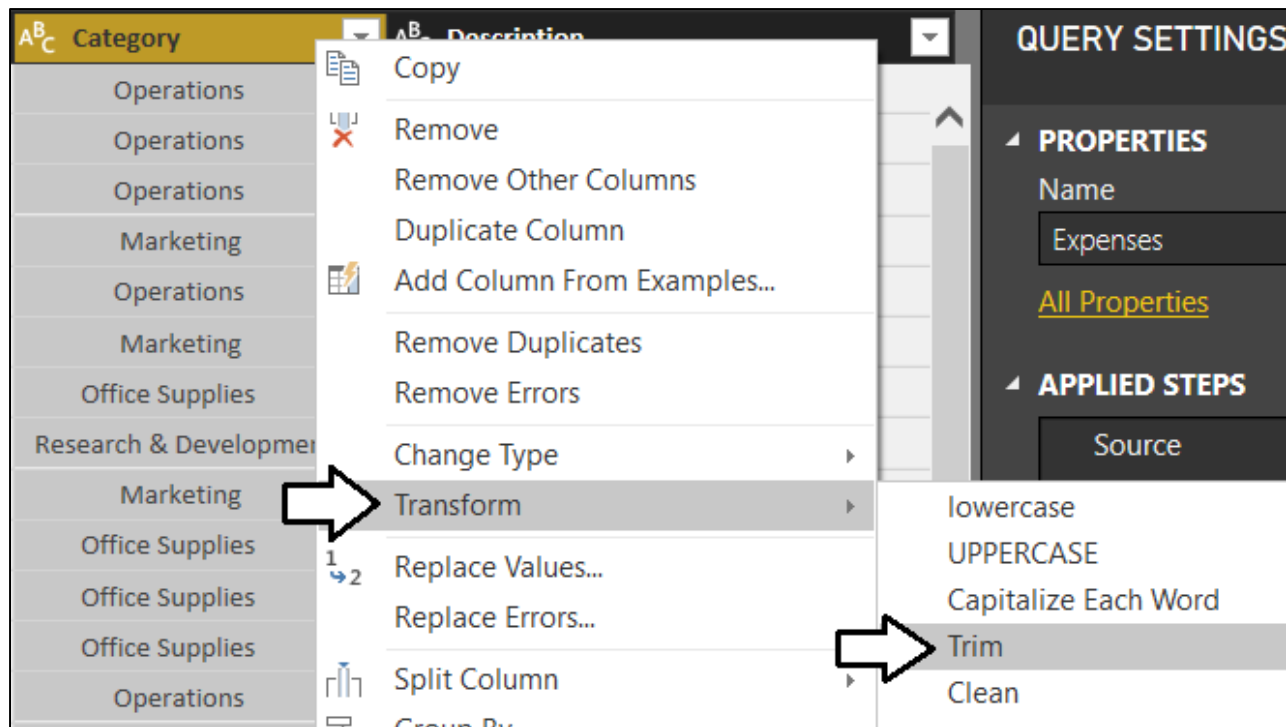
Examples of Basic Power Query Steps

- Rename column
- Convert column type
- Trim and clean column values
- Replace column values
- Format column values
- Expanding related column
- Merging columns
- Splitting columns









Cleaning Data

- Special steps available to clean up string-based data
 - **Transform > Trim** removes whitespace
 - **Transform > Clean** removed non-printable characters



Converting Column Datatypes

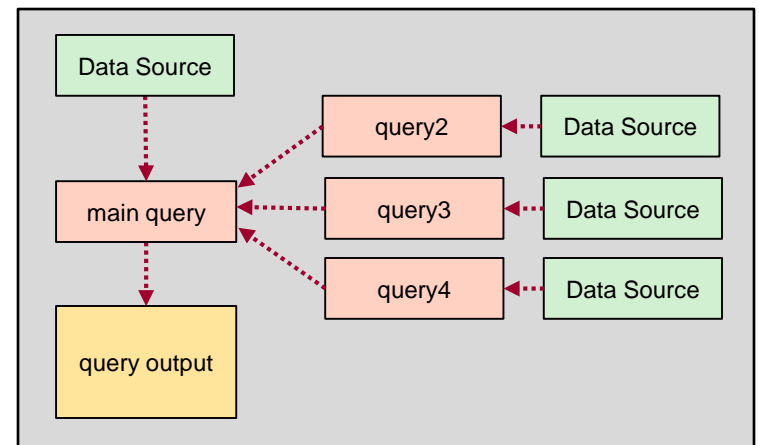
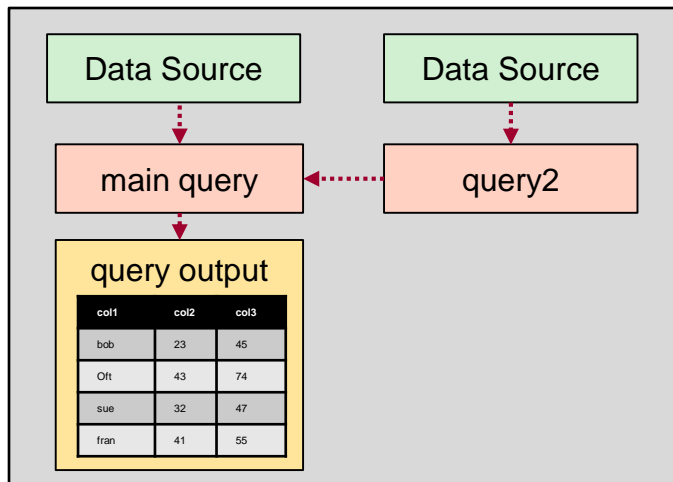
- Transform data to make it more reliable
 - Convert date-time column to date column
- Transform data to make it more efficient
 - Convert decimal to fixed decimal number for currency

|  PurchaseDate |  1 ² ₃ Quantity |  \$ SalesAmount |  \$ ProductCost |
|--|--|--|--|
| 1/28/2012 | 1 | 2.95 | 1.2 Decimal Number |
| 1/28/2012 | 6 | | \$ Fixed Decimal Number |
| 1/28/2012 | 1 | 19.95 | 1 ² ₃ Whole Number |
| 1/28/2012 | 5 | 249.75 |  Date/Time |
| 1/28/2012 | 1 | 2.95 |  Date |



Combining Queries

- Query can be merged or appended with another query
 - Merge operation allows you combine columns from two tables
 - Append operation allows you to combine rows from two tables
- Two queries are combined into single output for loading
 - Load settings of main query determines where output is loaded
 - Secondary query acts as source for main query
 - Secondary query can be created with connection-only load setting



Expanding Related Columns

- Used to pull data from related tables
 - Saves you from performing SQL joins or VLOOKUP

| SalesAmount | Invoices | |
|-------------|----------|-------|
| 119.8 | Value | Value |
| 29.95 | Value | Value |
| 59.9 | Value | Value |
| 399.6 | Value | Value |
| 29.9 | Value | Value |
| 59.8 | Value | Value |

| Id | InvoiceId | ProductId | Quantity | SalesAmount | Invoices | Products |
|----|-----------|-----------|----------|-------------|----------|----------|
| 1 | 1 | 1 | | | | Value |
| 2 | 2 | 1 | | | | Value |
| 3 | 3 | 2 | | | | Value |
| 4 | 4 | 3 | | | | Value |
| 5 | 5 | 3 | | | | Value |
| 6 | 6 | 3 | | | | Value |
| 7 | 7 | 4 | | | | Value |
| 8 | 8 | 5 | | | | Value |
| 9 | 9 | 6 | | | | Value |
| 10 | 10 | 6 | | | | Value |
| 11 | 11 | 7 | | | | Value |
| 12 | 12 | 7 | | | | Value |
| 13 | 13 | 8 | | | | Value |
| 14 | 14 | 9 | | | | Value |

Search Columns to Expand

(Select All Columns)

☐ InvoiceId

☒ InvoiceDate

☐ InvoiceAmount

☐ InvoiceType

☒ CustomerId

☐ Customers

☐ InvoiceDetails

☐ Use original column name as prefix

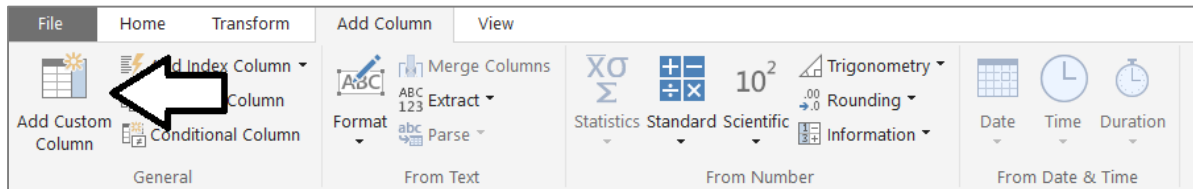
OK Cancel

| Id | InvoiceId | ProductId | Quantity | SalesAmount | InvoiceDate | CustomerId | Products |
|----|-----------|-----------|----------|-------------|-------------|-----------------------|----------|
| 1 | 1 | 1 | 22 | 4 | 119.8 | 1/28/2012 12:00:00 AM | 1 Value |
| 2 | 2 | 1 | 22 | 1 | 29.95 | 1/28/2012 12:00:00 AM | 1 Value |
| 3 | 3 | 2 | 22 | 2 | 59.9 | 1/28/2012 12:00:00 AM | 2 Value |
| 4 | 4 | 3 | 17 | 8 | 399.6 | 1/28/2012 12:00:00 AM | 3 Value |
| 5 | 5 | 3 | 18 | 2 | 29.9 | 1/28/2012 12:00:00 AM | 3 Value |
| 6 | 6 | 3 | 18 | 4 | 59.8 | 1/28/2012 12:00:00 AM | 3 Value |
| 7 | 7 | 4 | 16 | 1 | 2.95 | 1/28/2012 12:00:00 AM | 4 Value |



Adding a Custom Column

- Custom column provide custom logic
 - Logic must be written in M programming language



Add Custom Column

New column name:

Custom column formula:

```
= if [FirstPurchaseDate]=[LastPurchaseDate]  
then "One-time Customer"  
else "Repeat Customer"
```

Available columns:
CustomerId
Customer
State
City
ZipCode
Gender
BirthDate
FirstPurchaseDate
LastPurchaseDate
CustomerType

<< Insert

[Learn about Power BI Desktop formulas](#)

✓ No syntax errors have been detected.

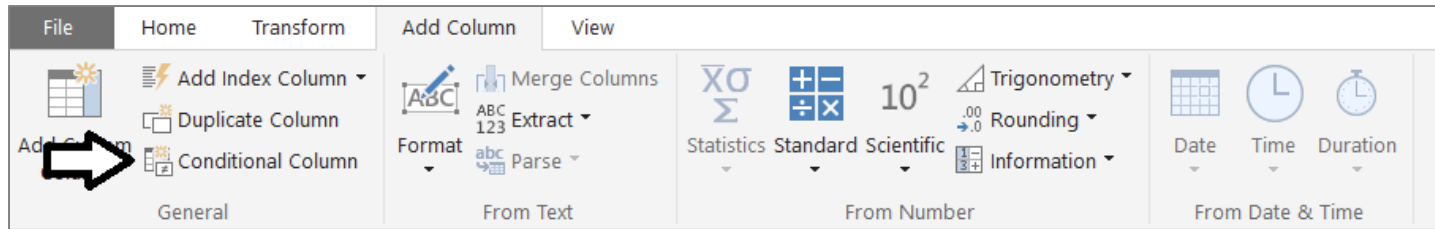
OK Cancel

| FirstPurchaseDate | LastPurchaseDate | CustomerType |
|-------------------|------------------|-------------------|
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/28/2012 | 1/28/2012 | One-time Customer |
| 1/29/2012 | 11/22/2015 | Repeat Customer |
| 1/29/2012 | 10/2/2015 | Repeat Customer |
| 1/29/2012 | 1/29/2012 | One-time Customer |
| 1/29/2012 | 5/6/2015 | Repeat Customer |
| 1/29/2012 | 1/29/2012 | One-time Customer |



Adding a Conditional Column

- Abstracts away need to write M code



Add Conditional Column [X]

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

New column name
Customer Type

| | Column Name | Operator | Value | | Output |
|----|-------------------|----------|------------------|------|-------------------|
| If | FirstPurchaseDate | equals | LastPurchaseDate | Then | One-time Customer |

+ Add Rule

Otherwise

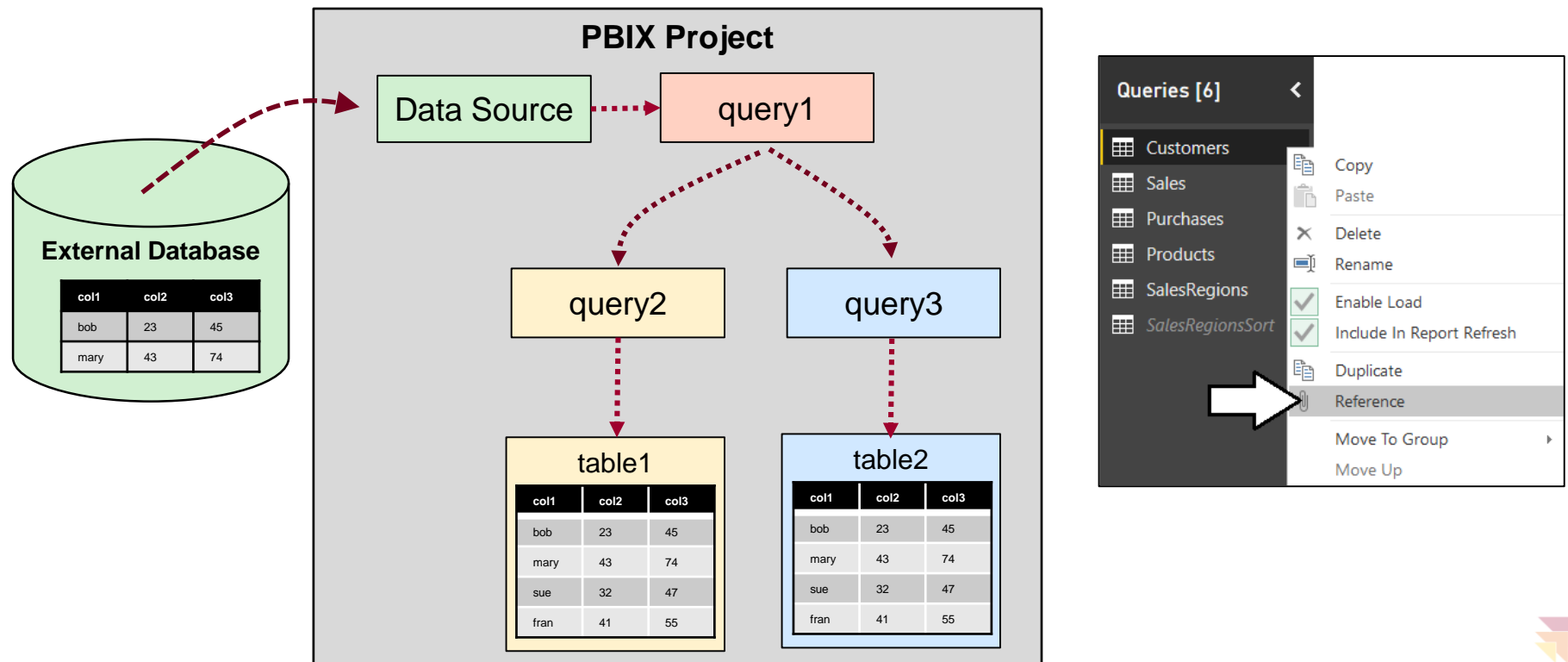
| | |
|---------|-----------------|
| ABC 123 | Repeat Customer |
|---------|-----------------|

OK Cancel



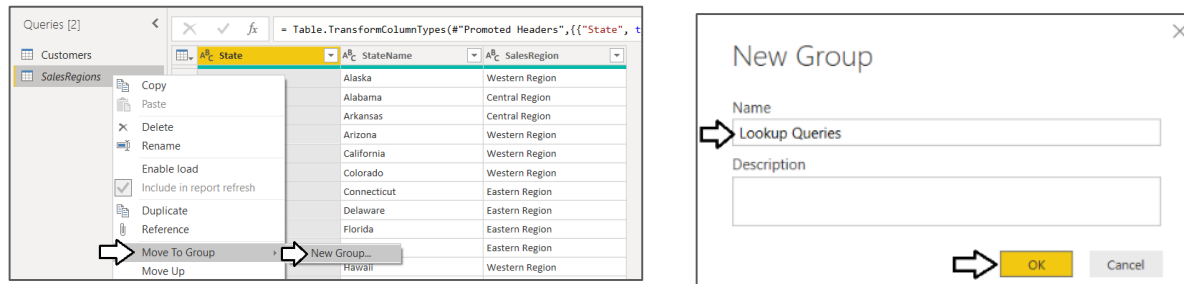
Query Composition

- Query can serve as source for other queries
 - Allows for creation of reusable base queries & query composition
 - Complexity can be hidden in base queries
 - **Reference** command creates new query based on another query

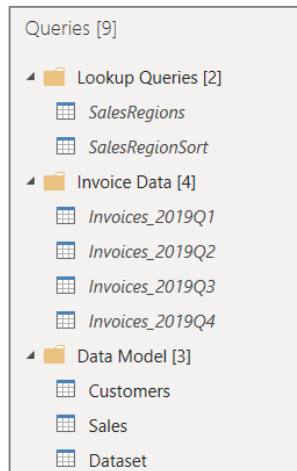


Structuring Queries into Folder Groups

- 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



Agenda

- ✓ Course Introduction
- ✓ Importing Data using Power Query
- Writing Query Logic in M
 - Understanding Query Folding
 - Writing Reusable Function Queries



Advanced Editor

- 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



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



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 uncle" = "are evil",  
  #"kitchen sink" = var1 & #"var 2" & #"Bob's your uncle"  
in  
  #"kitchen sink"
```

APPLIED STEPS

var1
var 2
Bob's your uncle

✕ Kitchen sink



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

The screenshot shows the 'Advanced Editor' interface. On the left, a code editor displays a 'let' statement with four lines of code, each enclosed in a red box. Red arrows point from these boxes to the 'APPLIED STEPS' panel on the right. The code is as follows:

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

At the bottom of the editor, a green checkmark and the text 'No syntax errors have been detected.' are visible.

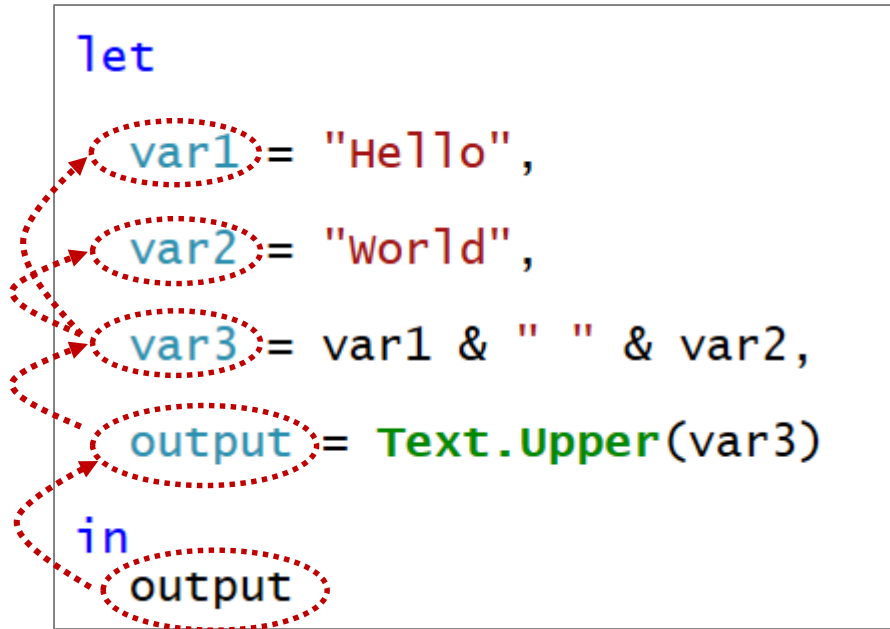
The right-hand panel has two sections:

- PROPERTIES**: Contains a 'Name' field with the value 'Hello World' and a link labeled 'All Properties'.
- APPLIED STEPS**: A list of variables: 'var1', 'var2', 'var3', and 'var4'. 'var4' is highlighted with a grey background and a small 'X' icon to its left.



Flow of Statement Evaluation

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



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



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),
```



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
```



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
```

| | ABC 123 FirstName | ABC 123 LastName |
|---|-------------------------|------------------------|
| 1 | Matt | Masson |
| 2 | Chris | Webb |
| 3 | Reza | Rad |
| 4 | Chuck | Sterling |

ABC
123

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,  
    CustomerTable =  
        #table(CustomerTableType, {  
            { "Matt", "Masson" },  
            { "Chris", "Webb" },  
            { "Reza", "Rad" },  
            { "Chuck", "Sterilicious" }  
        })  
in  
    CustomerTable
```

| | AB C FirstName | AB C LastName |
|---|----------------|---------------|
| 1 | Matt | Masson |
| 2 | Chris | Webb |
| 3 | Reza | Rad |
| 4 | Chuck | Sterilicious |




Using Each with Unary Functions

- Many library functions take function as parameters
 - Function parameters are often unary (*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(_) )
```



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 ) )
```

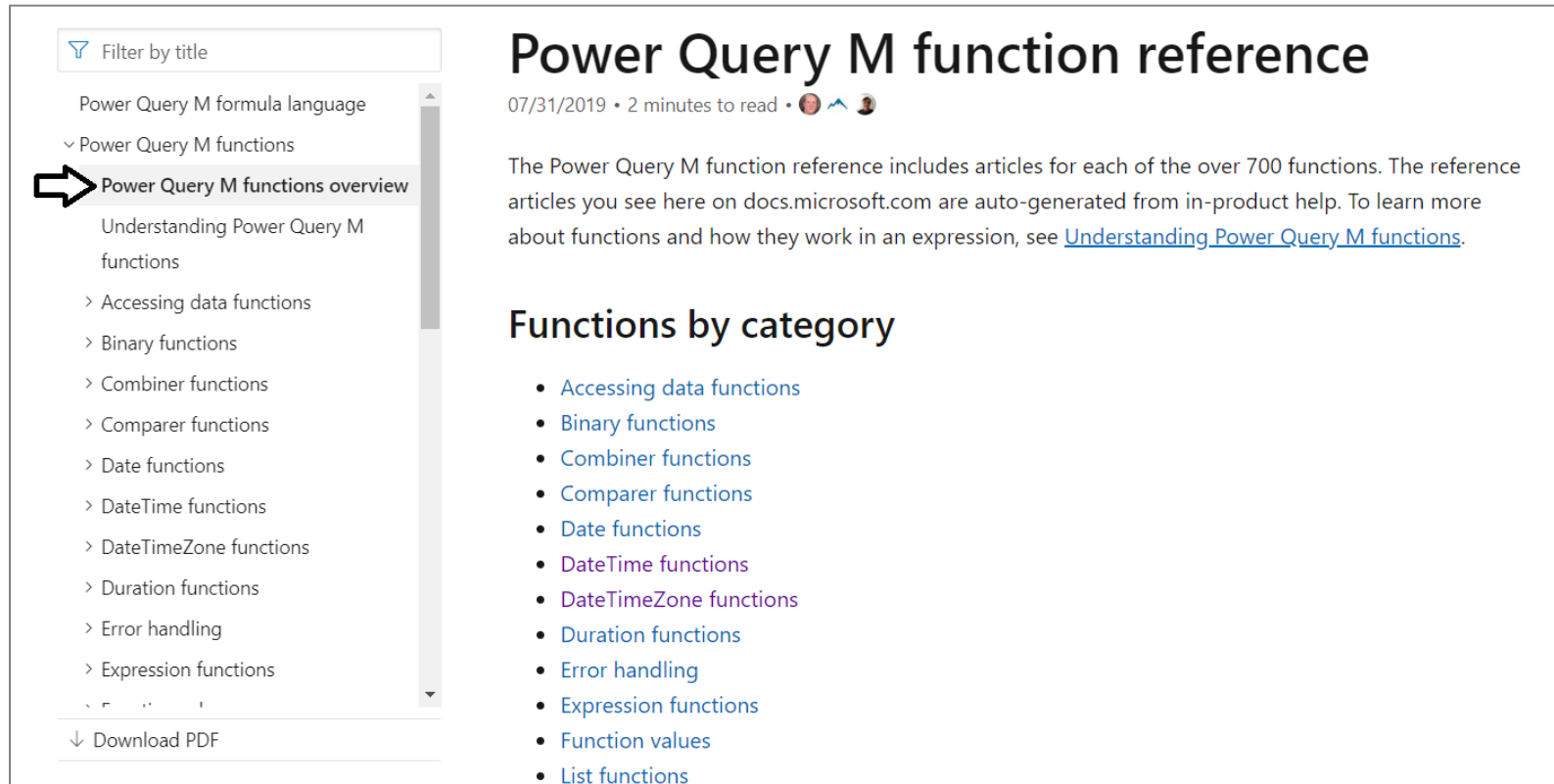
- Expression causing errors replace with value such as `null`

| | ABC Raw Date | ABC 123 Date1 | ABC 123 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 |



M Function Library

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



Filter by title

Power Query M formula language

▼ Power Query M functions

➡ Power Query M functions overview

Understanding Power Query M functions

> Accessing data functions

> Binary functions

> Combiner functions

> Comparer functions

> Date functions

> DateTime functions

> DateTimeZone functions




> Duration functions

> Error handling

> Expression functions

Download PDF

Power Query M function reference

07/31/2019 • 2 minutes to read •   

The Power Query M function reference includes articles for each of the over 700 functions. The reference articles you see here on docs.microsoft.com are auto-generated from in-product help. To learn more about functions and how they work in an expression, see [Understanding Power Query M functions](#).

Functions by category

- [Accessing data functions](#)
- [Binary functions](#)
- [Combiner functions](#)
- [Comparer functions](#)
- [Date functions](#)
- [DateTime functions](#)
- [DateTimeZone functions](#)
- [Duration functions](#)
- [Error handling](#)
- [Expression functions](#)
- [Function values](#)
- [List functions](#)



Agenda

- ✓ Course Introduction
- ✓ Importing Data using Power Query
- ✓ Writing Query Logic in M
- Understanding Query Folding
 - Writing Reusable Function Queries



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(FilteredReaders, 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
```



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+1e+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

- ✓ Course Introduction
- ✓ Importing Data using Power Query
- ✓ Writing Query Logic in M
- ✓ Understanding Query Folding
- Writing Reusable Function Queries



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 !&^",

// 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)
```

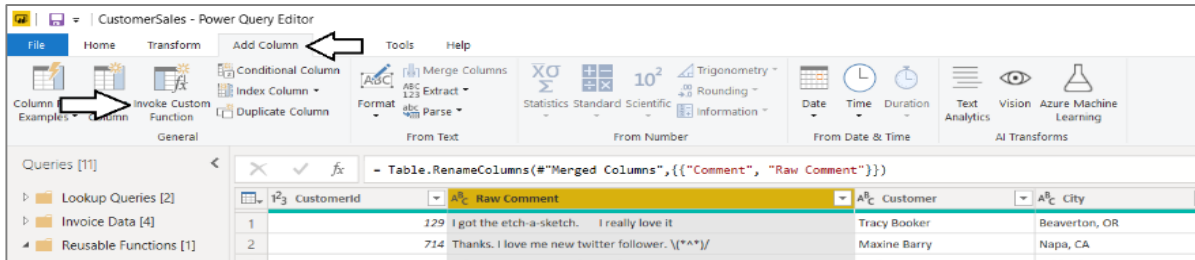


Creating a Function Query

- Requires adding parameter list



Calling a Function Query



Invoke Custom Function

Invoke a custom function defined in this file for each row.

New column name

Function query

input

OK Cancel



Summary

- ✓ Course Introduction
- ✓ Importing Data using Power Query
- ✓ Writing Query Logic in M
- ✓ Understanding Query Folding
- ✓ Writing Reusable Function Queries

