Power BI Advanced

Data Modeling and Data Shaping

José Ramón Iglesias Gamarra

Version: Diciembre 2023

Microsoft Power Platform

COURSE AGENDA

	Introductions and Overview
Module 1	Data Modeling Basics – Getting the Data
Module 2	Getting started with M (Power Query Language) & Relationship between tables
Module 3	Fact Table, introduction to DAX Calculated Columns & Measures
Module 4	Working With Parameters and introduction to DAX
Module 5	Working with functions - DAX CALCULATE Function, custom functions & Time Intelligence Functions
Module 6	Modeling with Power BI & DAX best practice Data Modeling, Optimization techniques
Module 7	Summary
	Wrap-up & Questions

Introduction

Get Involved on Social

Join the Power BI Community
Event Hashtag: AnalyticsAirlift
https://community.powerbi.com/

Follow us on:

- Twitter: @MSPowerBI
- Facebook: Microsoft Power BI

Power Platform Community

1M+ Active community members

400+ Independent User Groups

250K Users submitting ideas

100K Attendees in-person conferences

90K+ User Group members

33K Ideas submitted

Dashboard in a day

782 events in CY19

23,000+ in-person attendees

App in a day

277 events in CY19

6,500+ in-person attendees

Power Platform World Tour

20+ cities in FY20

5,000 projected attendees

Power Platform Champions

What we are Trying to Help with

Budget constraints

Time & Resource constraints

Business expectations

Paper processes

Complex processes

IT/Business partnership

Legacy system maintenance

“Shadow IT” governance

Leverage existing technology

Security & Compliance

Microsoft Power Platform

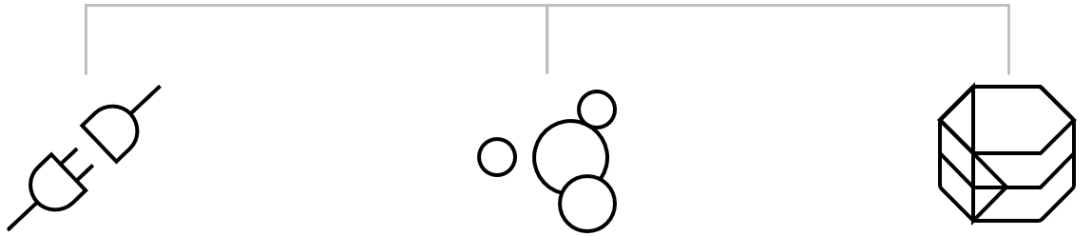
The low-code platform that spans Office 365, Azure, Dynamics 365, and standalone applications

Power BI
Business analytics

Power Apps
Application development

Power Automate
Process automation

Power Virtual Agents
Intelligent virtual agents



Data connectors

AI Builder

Common Data Service

Anatomy of a Dashboard

Any data
Connect to any data source to find the best insights and answer tough business questions.

Share insights with anyone
Share on prem, in the cloud, and mobile devices; even with those outside your organization

Seamless integrations
Directly integrated with Power Automate, Power Apps, Microsoft Teams, Office 365, Dynamics 365 & Azure

Extensibility model
No limits. Embed insights into everyday user apps to build a data culture throughout your org

WYSIWYG design experience
PowerPoint like pixel perfect drag and drop design experience. Start from a template, report is running with live data as you build it

Data visualizations
Select from hundreds of data visualizations or build your own to bring your data story to life

Native intelligence
Ask questions of your data, deliver automated insights; all visualized in the context of your business

Big data support with Azure
Leverage Azure Data Services like Date Lake and SQL Data Warehouse to optimize analysis big data

Module 1 –
Getting the Data & Basic Data Modeiling in Power BI Desktop

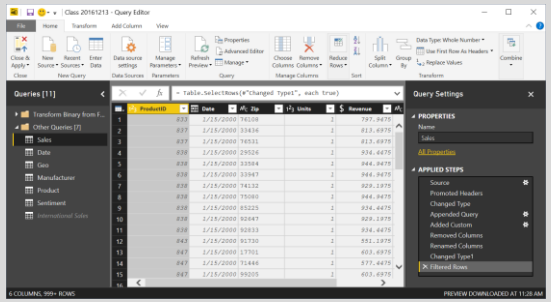
MODULE 1 OBJECTIVES

- Understand what is meant by *data model* in the context of Power BI
- Understand the consequences of data model design decisions
- Understand consequences of Power BI's data type handling

Power BI Desktop Data Flow

Data Sources

Power BI Desktop file (.PBIX)



Query Editor (M)

Data Source Connections

Data Transformations

(Prep data for Data Model)

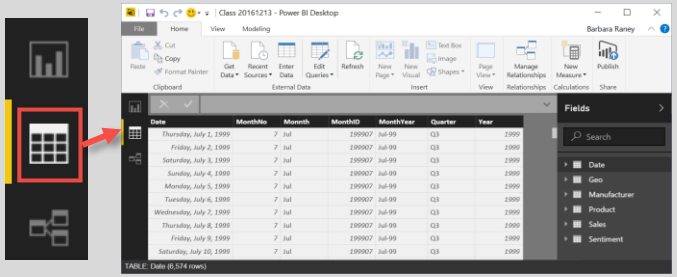
Close & Apply

Close: Closes Query Editor

Apply: Loads data from sources to Data Model

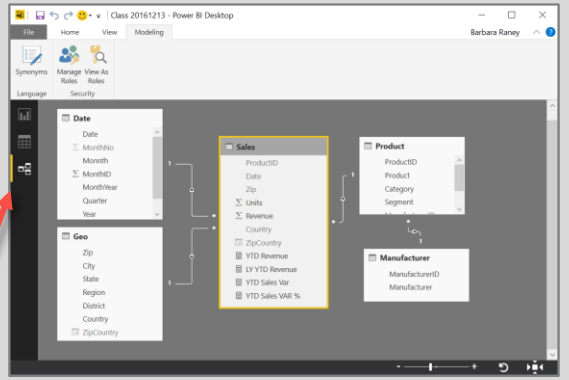
Report

Create Visuals



Data (DAX)

View Tables

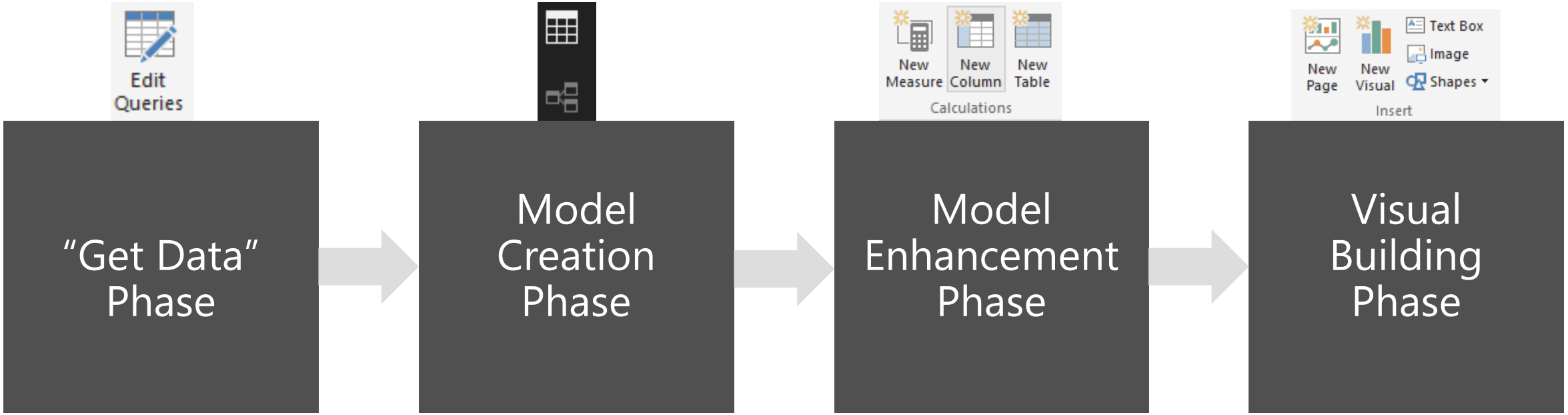


Relationships

See how Tables relate to each other

Power BI Desktop Files

Phases in Building a Power BI Desktop File



Create Query in "M"

Compress data, auto detect relationships

(Automatically done by Tool)

Add calculated columns, measures, add missing relationships

Evaluate Measures and build each visual

What is a Data model?

A Power BI **Data Model** is a **collection of tables with relationships** which enable your business users to easily understand and explore their data to get business insights.

Why is it important to have a Good Data model?

- Improves understandability of the data
- Increases performance of dependent processes and systems
- Increases resilience to change

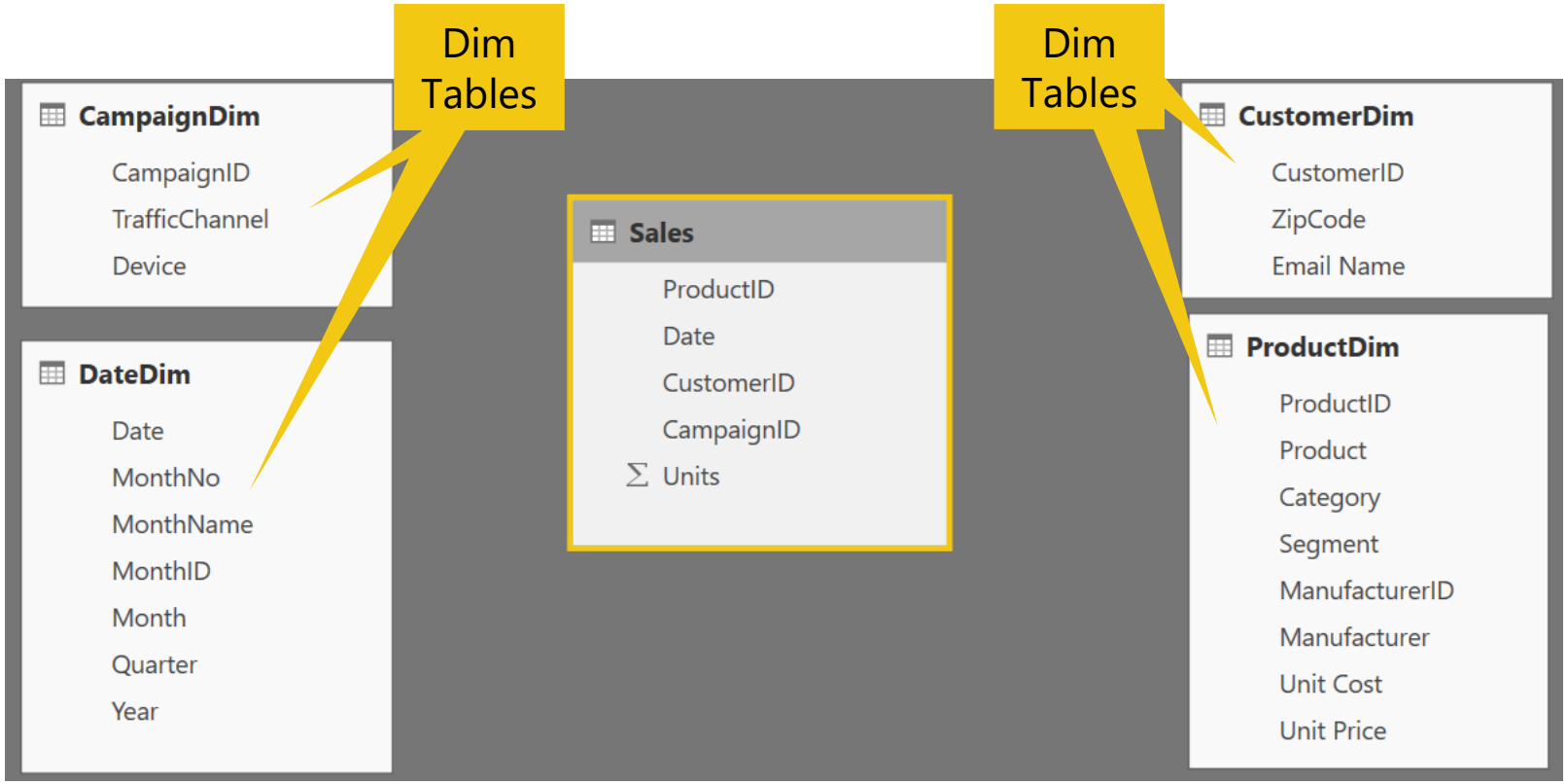
Components of a data model – Fact Table

Fact Table

- Contains Measures
  (or items to be aggregated)
  of a business process
  Examples:
  - Transactions
  - Sales Revenue
  - Units
  - Cost

- Measures are usually sliceable.
  Examples: By Month,
  By Customer

Components of a data model – Dim Table



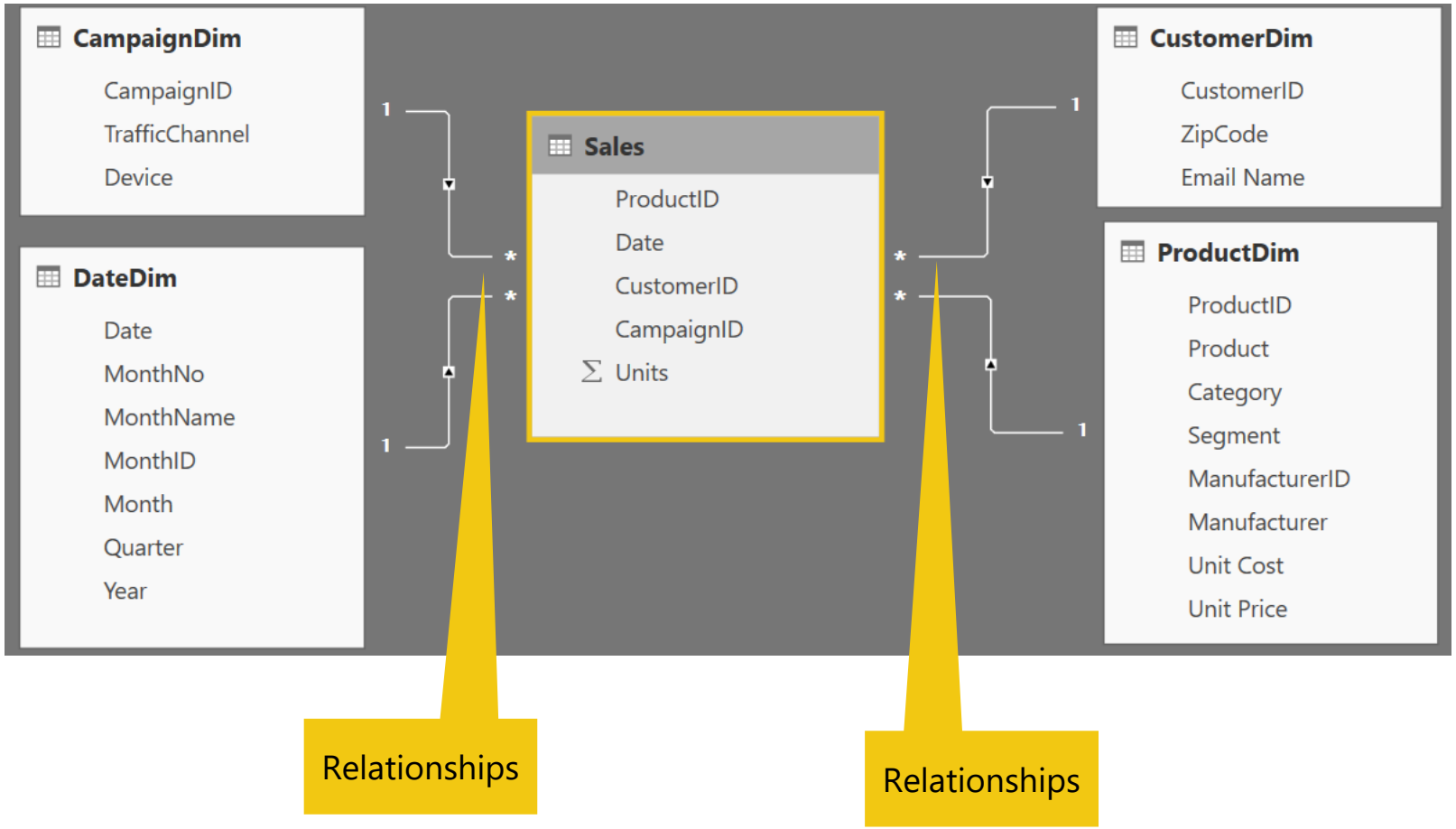
Dim Table

A Dim (or Dimension) table contains descriptive attributes that define how a fact should roll up.

Examples:

By month, By Customer, By Geo

Components of a data model - Relationships



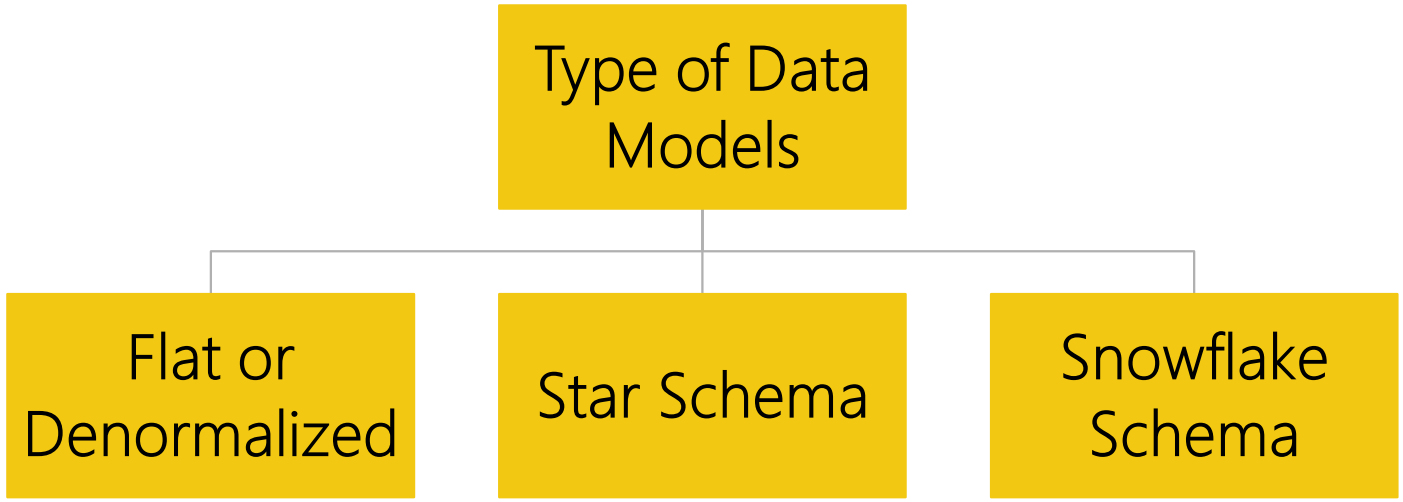
Relationships

- Connection between a 2 tables (usually fact & Dim tables) using columns from each

- 3 kinds of Relationships
  - 1 to Many
  - 1 to 1
  - Many to Many

(with a bridge table)

Data Model Brings Facts and Dimensions Together



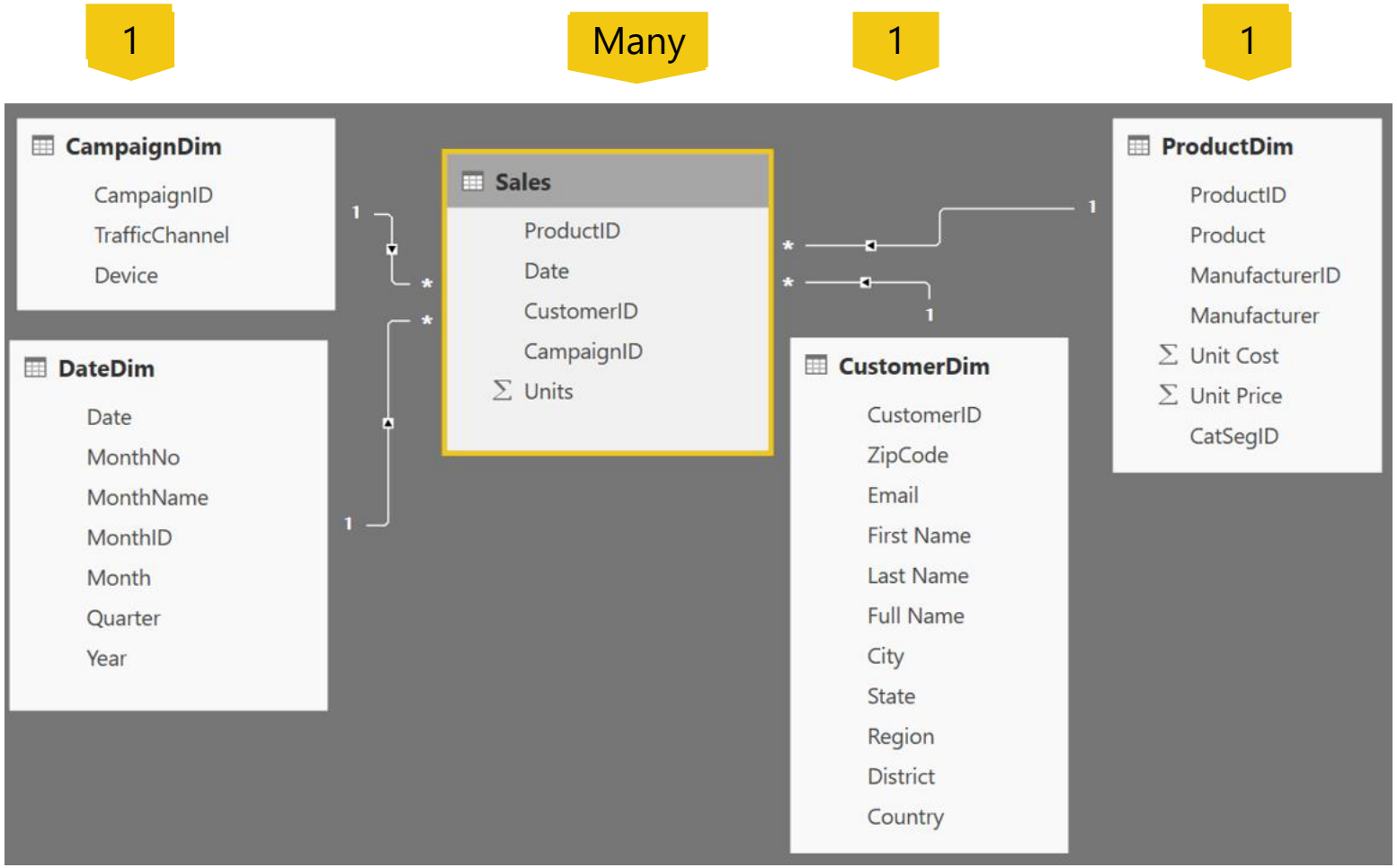
Note: This is not an exhaustive list, but are the most common model types used by Power BI.

Flat or Denormalized Schema

	ProductID	Product	Date	CustomerID	Email	Last Name	First Name	Full Name	CampaignID	Units	CatSegID
1	676	Maximus UC-41	9/25/2011	70283	Farrah.Kent@xyza.com	Kent	Farrah	Farrah Kent	22	1	10
2	585	Maximus UC-50	3/24/2014	70283	Farrah.Kent@xyza.com	Kent	Farrah	Farrah Kent	15	1	10
3	585	Maximus UC-50	11/30/2014	138334	Martha.Mcclain@xyza.…	Mcclain	Martha	Martha Mcclain	8	1	10
4	585	Maximus UC-50	6/21/2015	27193	Hedda.Mcintosh@xyza.…	Mcintosh	Hedda	Hedda Mcintosh	22	1	10
5	585	Maximus UC-50	1/6/2013	238970	Lunea.Walker@xyza.com	Walker	Lunea	Lunea Walker	21	1	10
6	585	Maximus UC-50	3/22/2013	182241	Upton.Page@xyza.com	Page	Upton	Upton Page	17	1	10
7	449	Maximus UM-54	9/25/2011	195385	Drake.Wells@xyza.com	Wells	Drake	Drake Wells	22	1	4
8	449	Maximus UM-54	9/30/2014	168009	Wallace.Bender@xyza.…	Bender	Wallace	Wallace Bender	17	1	4
9	449	Maximus UM-54	8/12/2014	110391	Astra.Erickson@xyza.…	Erickson	Astra	Astra Erickson	20	1	4
10	449	Maximus UM-54	4/16/2014	49327	Echo.Bradley@xyza.com	Bradley	Echo	Echo Bradley	7	1	4
11	449	Maximus UM-54	2/28/2013	65952	Yoko.Gross@xyza.com	Gross	Yoko	Yoko Gross	17	1	4
12	449	Maximus UM-54	6/6/2013	97	Yoshi.Grant@xyza.com	Grant	Yoshi	Yoshi Grant	10	1	4
13	449	Maximus UM-54	5/14/2013	56757	Brian.Carrillo@xyza.…	Carrillo	Brian	Brian Carrillo	10	1	4
14	449	Maximus UM-54	4/9/2015	248715	Mark.Hewitt@xyza.com	Hewitt	Mark	Mark Hewitt	19	1	4
15	449	Maximus UM-54	4/28/2013	248715	Mark.Hewitt@xyza.com	Hewitt	Mark	Mark Hewitt	8	1	4
16	449	Maximus UM-54	3/28/2014	240831	Oscar.Avila@xyza.com	Avila	Oscar	Oscar Avila	18	1	4
17	449	Maximus UM-54	2/26/2014	201004	Duncan.Mcintosh@xyza…	Mcintosh	Duncan	Duncan Mcintosh	19	1	4
18	615	Maximus UC-80	5/14/2012	212645	Jacob.Santiago@xyza.…	Santiago	Jacob	Jacob Santiago	22	1	10
19	615	Maximus UC-80	5/14/2012	70666	Hilary.Collier@xyza.…	Collier	Hilary	Hilary Collier	22	1	10
20	615	Maximus UC-80	5/14/2012	114459	Chester.Mitchell@xyz…	Mitchell	Chester	Chester Mitche…	22	1	10
21	615	Maximus UC-80	5/14/2012	221670	Sage.Yang@xyza.com	Yang	Sage	Sage Yang	22	1	10
22	615	Maximus UC-80	6/3/2012	168009	Wallace.Bender@xyza.…	Bender	Wallace	Wallace Bender	22	1	10
23	615	Maximus UC-80	6/3/2012	154439	Iliana.Dunlap@xyza.c…	Dunlap	Iliana	Iliana Dunlap	22	1	10
24	615	Maximus UC-80	6/4/2012	191391	Joelle.Lee@xyza.com	Lee	Joelle	Joelle Lee	22	1	10

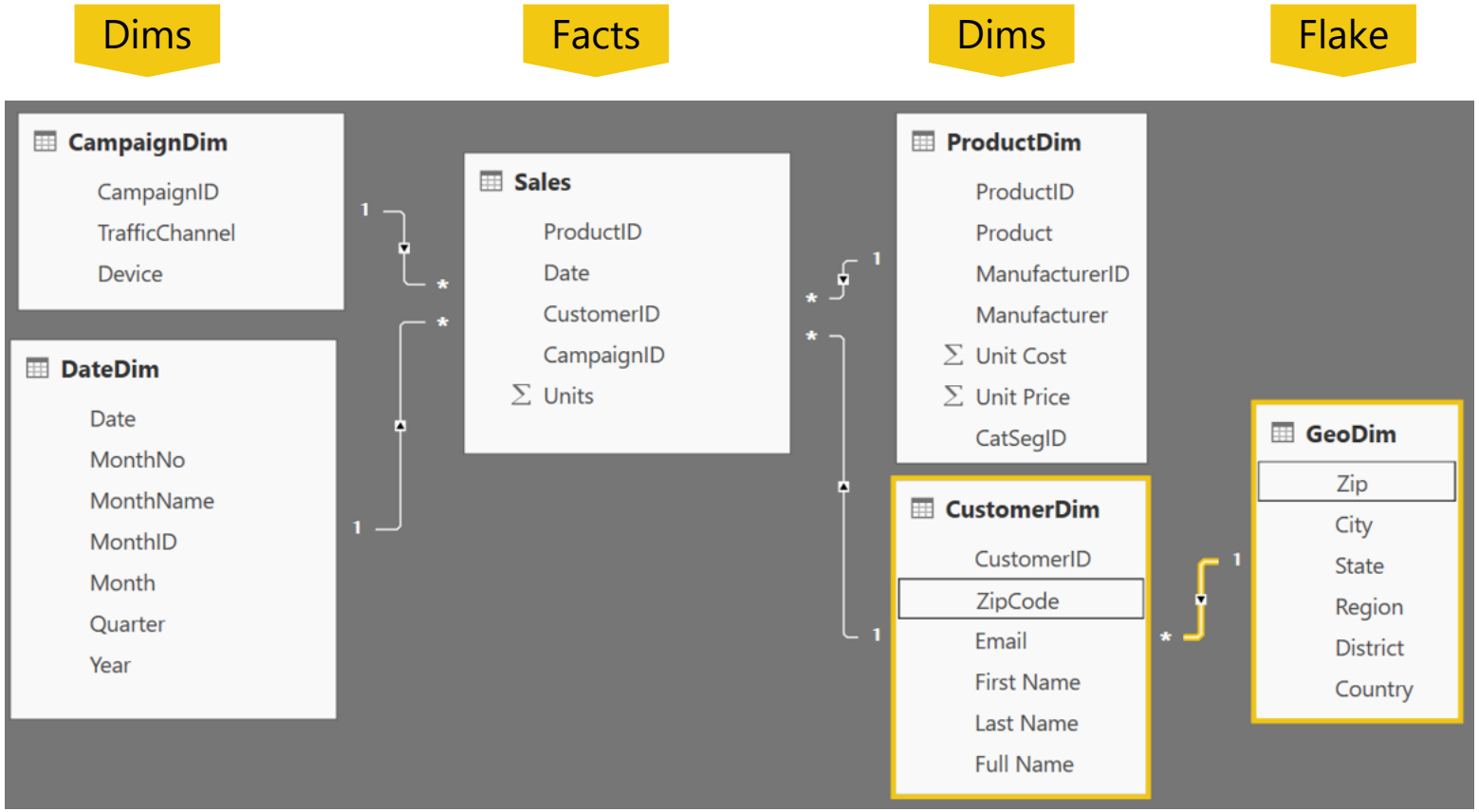
- All attributes for model exist in a single table
- **Highly inefficient**
- Model has extra copies of data > slow performance
- Size of a flat table can grow and quickly “blow up” as data model becomes complex

Star Schema



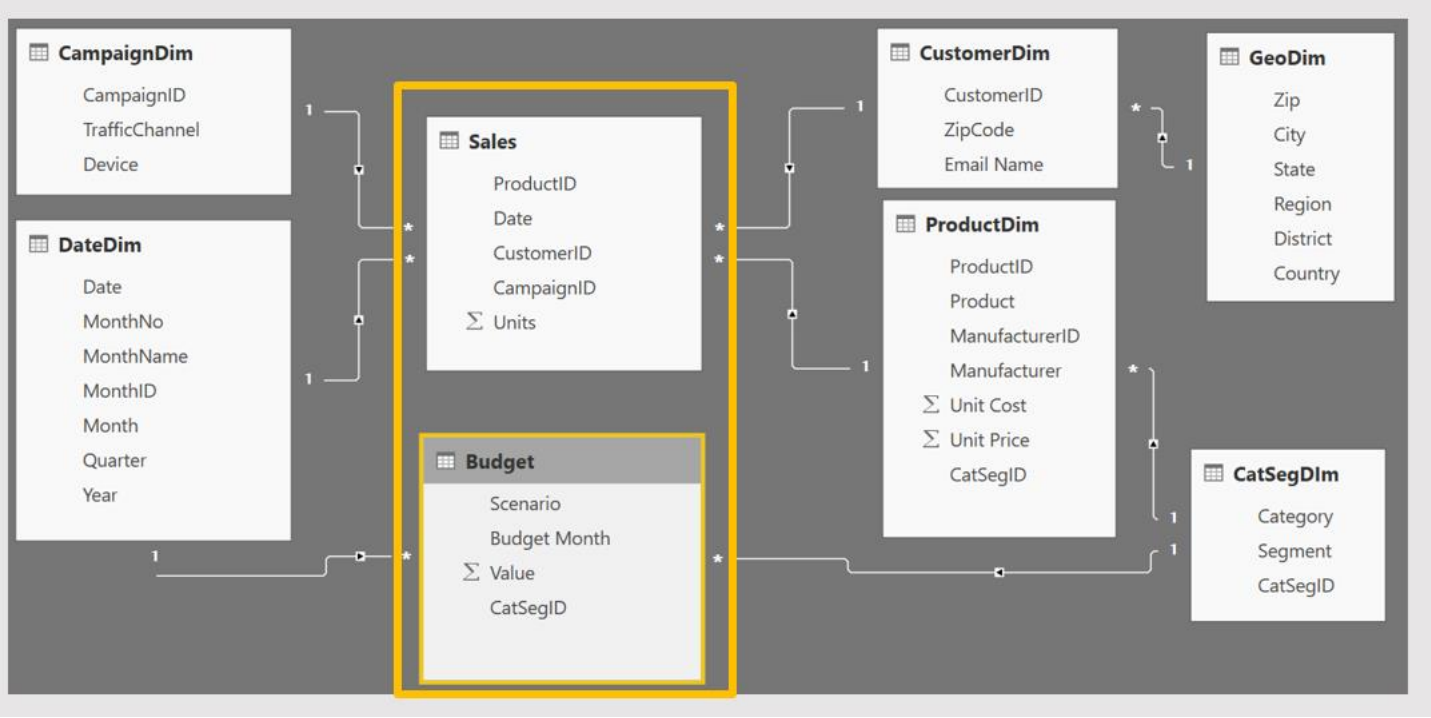
- **Fact** table in the middle
- Surrounded by **Dims**
- Looks like a ‘Star’
- Fact table is the “Many” side of the (one to many) relationship

Snowflake Schema



- Center is a Star schema
- **Fact** table in middle
- Surrounded by **Dims**
- Dims "snowflake" off of other Dims
- If you have many, it looks like a 'Snowflake'
- Dim or Fact tables can be the "Many" side of the relationship

Granularity & Multiple Fact Tables



Sales (Daily by Product)

Budget (Monthly by Product Category & Product Segment)

- Grain (**granularity)** measures the level of detail in a table

  Example:
  One row per **order or per Item**
  Daily or Monthly date grain

- If your facts have **very different granularities**, split them into **Multiple Fact** tables & connect them to shared dimensions at the lowest common granularity.

Data Mode Types in Power BI

How can I tell what Data Model Type I have?

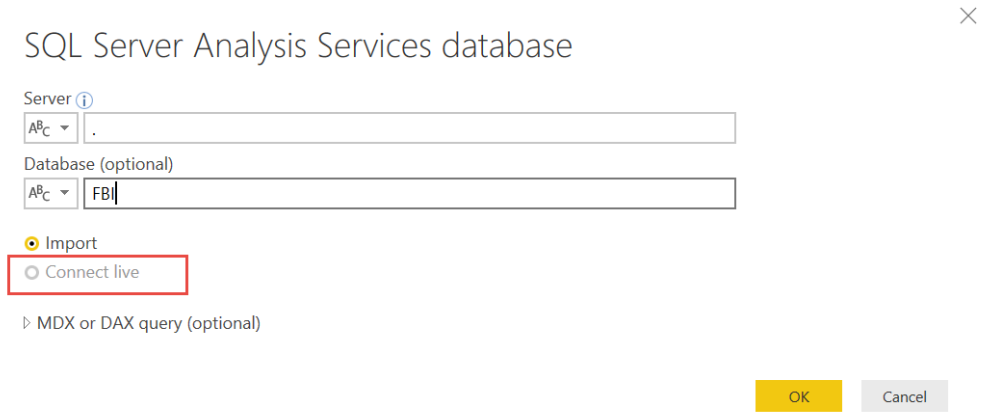
- Live Connect to SQL Analysis Services (SSAS) tabular
  - Report view only available

- DirectQuery to SQL or other relational source
  - Report & Relationship views available

- Import data into Power BI (creates a copy of the data)
  - Report, Data and Relationship views available

Connection: Live Connect

- Live Connect to Multidimensional or Tabular
  - On Premise or Azure
- Only a single connection will be made and all modeling is done in the cube
- You can not add relationships or additional data source
- If allowed, you can add DAX measures



Choosing storage mode: LiveConnect

- Mode used when Power BI model accessing an SQL Server Analysis Services (SSAS) data model
- Can be a server (IaaS) or Azure Analysis Services (PaaS)

- When is it appropriate
  - Organization has already invested significantly in SSAS, have mature models
  - High level of control needed around partitions, data refresh, query scale-out and workload splitting
  - Granular auditing, monitoring and diagnostics
  - Integration with CI/CD or similar automation pipelines
  - Models can't fit in Premium (P3+ models up to 10GB v. no overall model size limit in SSAS)

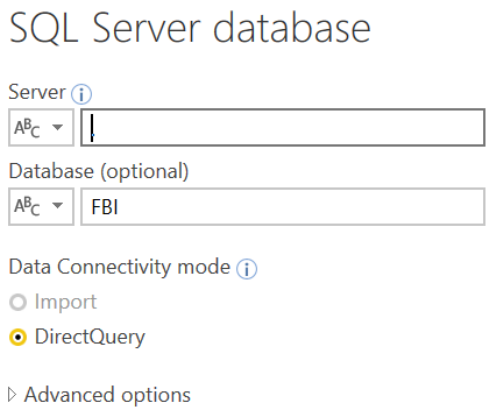
- Considerations
  - Strategically we will be bringing AS features to Premium over time
  - We can enable certain features per customer (e.g. Aggregations in ASAzure)

Connection: Direct Query to Relational Source

- Direct Query to SQL or other relational source
  - On Premise or Azure

- Composite modeling is possible where some data sources are in Direct Query mode and a few are in Import mode

- You can add relationships and DAX



Import Mode

What is unique about Power BI Desktop in Import Mode?

- Columnar database
- In-memory database

Let us understand some of the internals of Power BI Desktop !!

Choosing storage mode: Import vs Direct Query

- Import is your first choice (all in memory = best speed, no DAX limits)

- When is it inappropriate
  - Extremely large data volumes
  - Need near real-time access to data from source
  - Considerable existing investment in external DW or OLAP (modelled, conformed, cleaned, calcs defined etc). SSAS MD, SAP HANA and BW are common.
  - Regulatory and data sovereignty requirements

- Considerations
  - How much source data, how compressible? Rule of thumb is 5x-10x
  - Is Premium an option? (larger datasets supported there)
  - Will blended architecture suffice? (Composite models, Aggregations for summary data)
  - Some limits on DAX in DirectQuery mode (e.g. time intelligence)

Columnar Database

Row Based Database

First Name	Last Name	Sales
John	Smith	$10
Jane	Doe	$25
Hardy	B	$35

- Stores **each row separately** (like a separate file)
- Retrieving multiple columns from a single row is fast
- Retrieving multiple rows from a single column is slower

PBI - Columnar Database

First Name	Last Name	Sales
John	Smith	$10
Jane	Doe	$25
Hardy	B	$35

- Stores **each column separately** (like a separate file)
- Retrieving multiple columns from a single row is slow
- Retrieving multiple rows from a single column is faster
- **Columnar databases are well suited for analytics**

In-Memory Database

Power BI compresses data to conserve space in RAM

Will help you understand the data modeling for performance optimization

Data stored in **RAM (in memory)**
RAM is all electronic – **Read/Write is fast**

(Detailed information in Appendix 2 slides)

Data Types

Numeric Data Types

- Whole Number
- Decimal Number
- Fixed Decimal Number (Floating point stored as integer)
- Boolean

Date/Time Data Types

- Date – Internally stored as an integer
- Time – Internally stored as a fraction between 0 and 1
- Date Time

Other Data Types

- Text
- **Any – You should never see this in a data model. Bad things can happen!!**

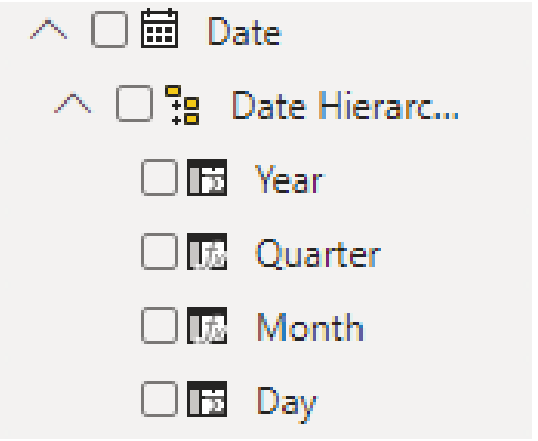
Set your
Data Types
in the
Query Editor

Set your
Data Formats
($ %, etc)
in the Data Model

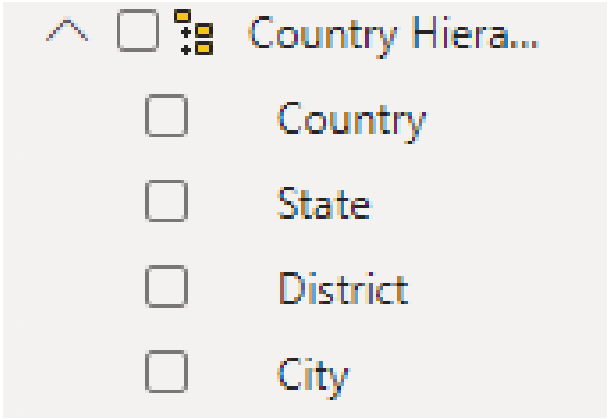
Pro Tip: Data type is different from data format

Hierarchies

- Power BI generates Date hierarchies when dates are added to visuals, this allows the end user to drill from Year, Quarter, Month & Day.

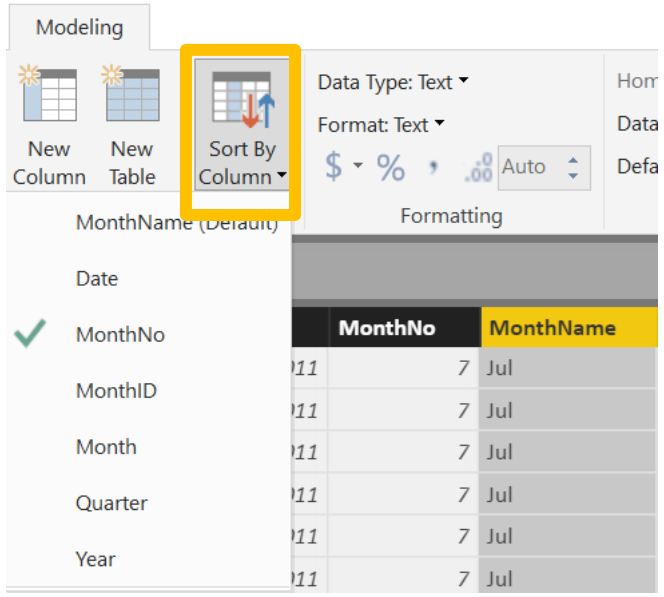


- Users can also create custom hierarchies in the model by dragging a lower level field onto the parent.



Sort By Column

- Enables sorting one text field by another (numeric) field
- Both columns must have the same number of distinct values



Module 1 Lab

1. Open a new Power BI File
2. Make sure your have got 2 Excel files as your data sources on your computer
3. Follow the instructions of Lab 1 in your student handout

MODULE 2:
Getting Started with M – Power Query Language

MODULE OBJECTIVES

Objectives

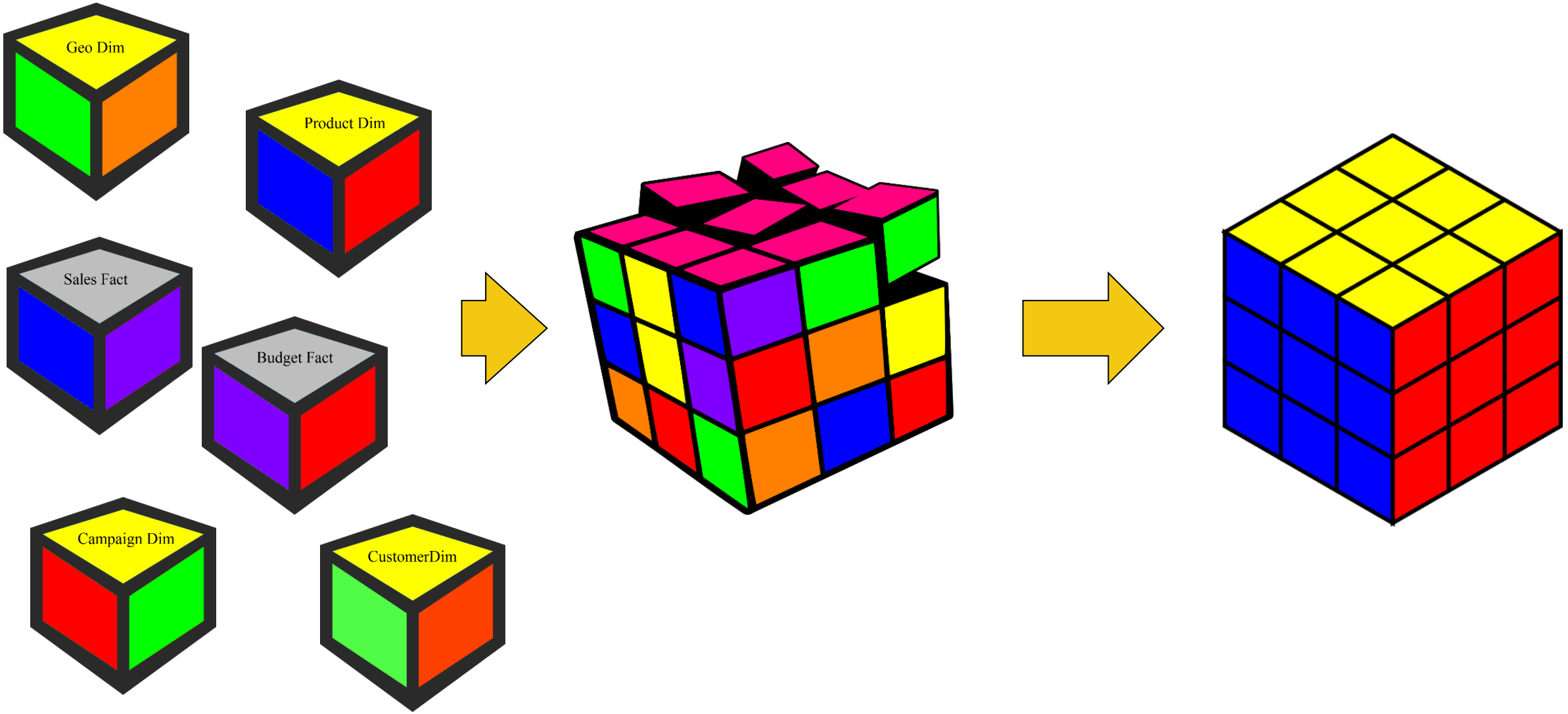
- Understand the basics syntax of the M Language
- Understand function & keyword structure
- Understand how to connect to data
- Understand the basic structure of the Advanced Editor

Agenda

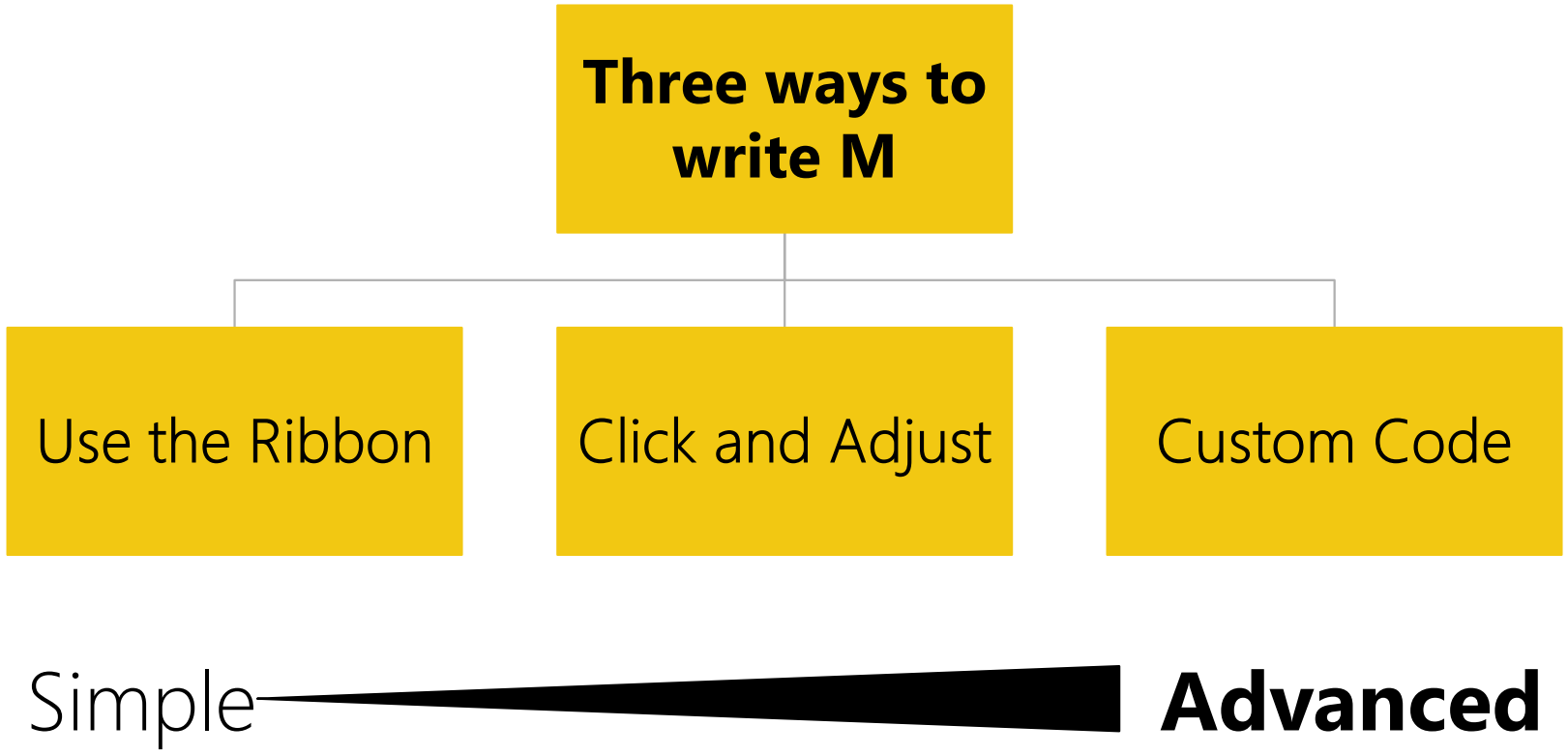
- Power BI M Language Overview
- Basic Data Import
- Introduction to Text functions
- Create and Apply Degenerate Dimensions

Why M?

M has amazing capabilities to transform data to optimize it for the data model



How do I use M?


Three ways to write M
Use the Ribbon
Click and Adjust
Custom Code
Simple
Advanced

Key Concepts of M syntax

M is the **key** to data transformation in Power BI

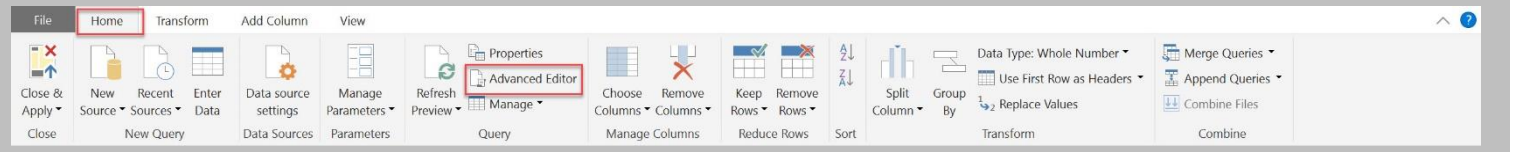
Many transforms are just a click away on the **Ribbons**

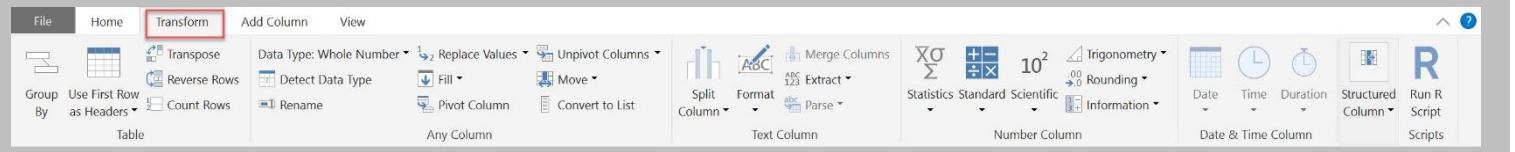
Turn on **Formula Bar** to see M syntax generated by clicks

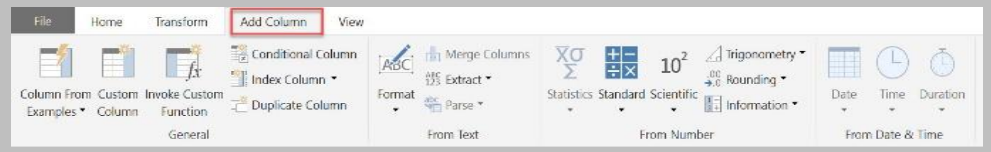
View full syntax in **Advanced Editor**

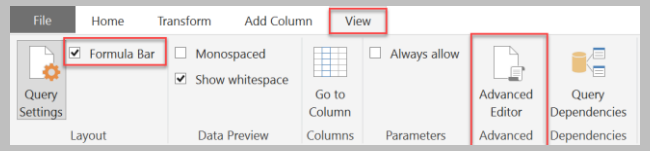
M is Column Based

- Many Transformations occur by adding columns









M is **CaSe SeNsItIvE!**

Key Concepts of M syntax

M Formulas

Foundation of transformations

- **Syntax:**
  Object.Action()
- **Objects:**
  Table, List, Record, Text, Number, Date, etc.
- **Actions (Camel Case):**
  ToList, DayOfWeek, ToNumber, FromText, etc.

M helper Words

little helper words – never capitalize these

- let in each
- if then else
- true false error
- and or not
- try otherwise
- as is meta section shared type

Key Punctuation

- [Column] - To reference a Column
- {List} – use {} to indicate a List

M # Key Words

Used to create new artifacts and literals

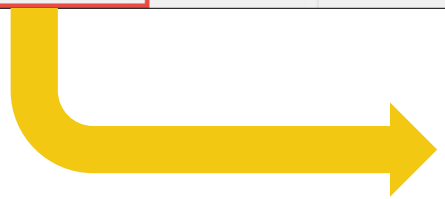
- #date(2016,01,01)
  #date([Year],[Month],[Day])
- #datetime(2016,02,26, 09,15,00)
- #time(09,15,00)
- #datetimezone(2013,02,26, 09,15,00, 09,00)
- #table({"X","Y"},{{0,1},{1,0}})
- #duration(0,1,30,0)
- #binary #infinity #nan #sections #shared

M is **CaSe SeNsItIvE!**

Power Query Formula Language

Text Functions

MyColumn	Text.Start	Text.End	Text.Range (Partial)	Text.Range (Full)	Text.StartsWith (abc)	Text.StartsWith (ABC)	Text.Length	Text.Contains (123)	Text.PostionOf (5)
ABC12345xyz	ABC	5xyz	234	2345xyz	FALSE	TRUE	11	TRUE	7



Position

My Column

M

0	1	2	3	4	5	6	7	8	9	10
A	B	C	1	2	3	4	5	x	y	z

Excel

1	2	3	4	5	6	7	8	9	10	11
A	B	C	1	2	3	4	5	x	y	z

Return "ABC"	Text.Start([MyColumn], 3)	LEFT([MyColumn], 3)
Return "234"	Text.Range([MyColumn], **4**, 3)	MID([MyColumn, **5**, 3)
Find position of "5"	PositionOf([MyColumn], "5") = **7**	SEARCH("5", [MyColumn]) = **8**

Text Functions

MyColumn	Text.Start	Text.End	Text.Range (Partial)	Text.Range (Full)	Text.StartsWith (abc)	Text.StartsWith (ABC)	Text.Length	Text.Contains (123)	Text.PostionOf (5)
ABC12345xyz	ABC	5xyz	234	2345xyz	FALSE	TRUE	11	TRUE	7

M Function Syntax	Excel Syntax	M Output
Text.Start([MyColumn], 3)	LEFT([MyColumn], 3)	ABC
Text.End([MyColumn], 4)	RIGHT([MyColumn], 4)	5xyz
Text.Range([MyColumn], 4, 3) *	MID([MyColumn], 5, 3) **	234
Text.Range([MyColumn], 4) *	MID([MyColumn], 5) **	2345xyz
Text.StartsWith ([MyColumn], "abc")	IF(LEFT([MyColumn], 3)="abc" , TRUE(), FALSE())	FALSE (Excel -> TRUE)
Text.StartsWith ([MyColumn], "ABC")	IF(LEFT([MyColumn], 3)="ABC", TRUE(), FALSE())	TRUE
Text.Length([MyColumn])	LEN([MyColumn])	11
Text.Contains ([MyColumn], "123")	IF(ISNUMBER(SEARCH("123", [MyColumn])),TRUE(), FALSE()) **	TRUE
Text.PositionOf ([MyColumn], "5") *	SEARCH("5", [MyColumn]) **	7 (Excel -> 8)

M is **CaSe SeNsItIvE!**

* M is 0 index based, so Text.Range & Text.PositionOf start counting at 0

** Excel MID & SEARCH are 1 index based – start counting at 1

Want to see all M functions? Add Blank query in Power BI and type: =#shared

if … then … else

- Basic Syntax:
  if <test if true> **then** <result if true> **else** <result if false>
- if's Can be chained or nested
- No parentheses () are *required, except*
- When if test or results have multiple conditions:
  (condition 1 and condition 2)
  (condition 1 or condition 2)
  (condition 1 or (condition 2 and condition 3))
- If multiple if's are used ensure you have the same number of "if", "then", "else"

M is **CaSe SeNsItIvE!**

Simple example (1 each of if, then & else)

```
=if [Column2] = "Key" then [Column2] else "Other"
```

Chained example (3 each of if, then & else)

```
=if Text.EndsWith([Column2],"Key") then [Column2] else
if [Column2] = "TotalCost" then "Cost $" else
if [Column2] = "SalesQuantity" then "Sales #" else null
```

Nested example (3 each of if, then & else)

```
=if Text.EndsWith([Column2],"Key") then
    if [Column2] = "TotalCost" then "Cost $" else
    if [Column2] = "SalesQuantity" then "Sales #" else null
  else "Item #"
```

Multiple conditions example (1 each of if, then & else)

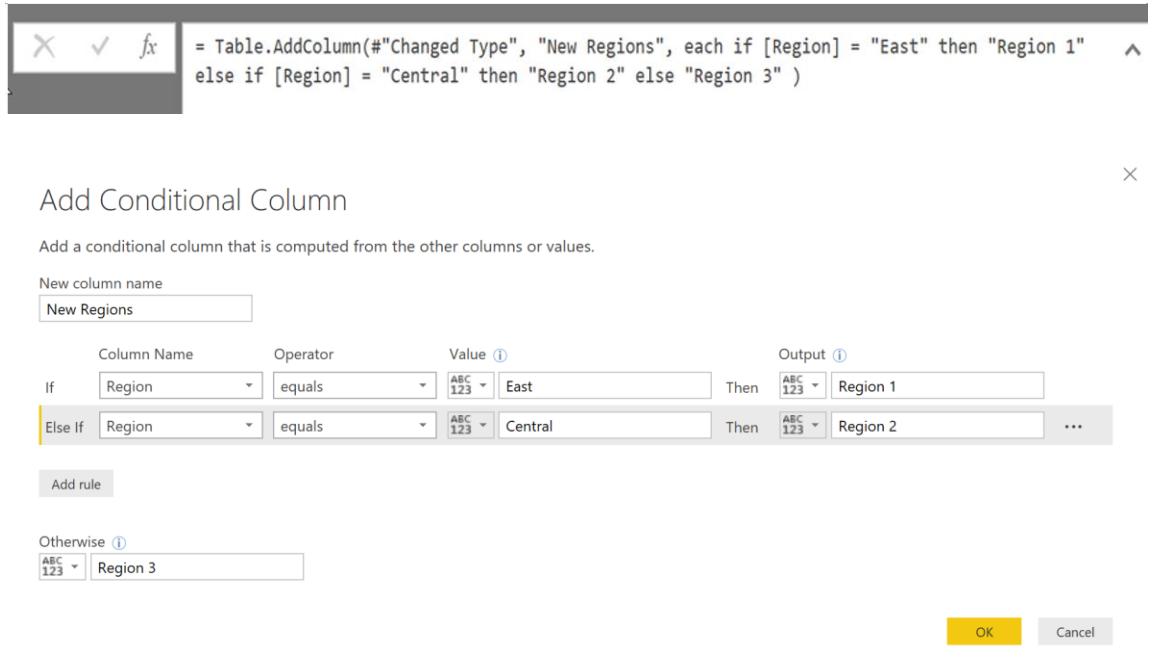
```
=if  (Text.Length([Column2]) = 3 or Text.Length([Column2]) = 2 )
then  true else false
```

if … then … else using Conditional and Custom

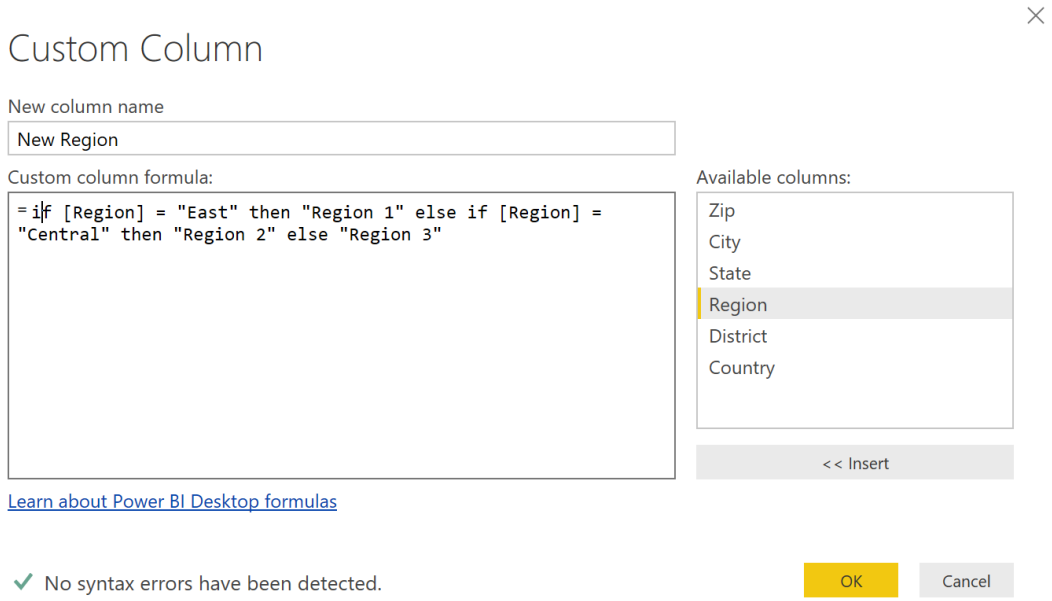
Whether you use Conditional Column or Custom Column, the results are the same

66	12204	Albany, NY, USA	NY	East	District #01	USA	Region 1
67	45710	Albany, OH, USA	OH	East	District #15	USA	Region 1
68	74721	Albany, OK, USA	OK	Central	District #22	USA	Region 2
69	97322	Albany, OR, USA	OR	West	District #34	USA	Region 3
70	97321	Albany, OR, USA	OR	West	District #34	USA	Region 3
71	76430	Albany, TX, USA	TX	Central	District #22	USA	Region 2

1. How it reads in the formula bar when using the Conditional Column format

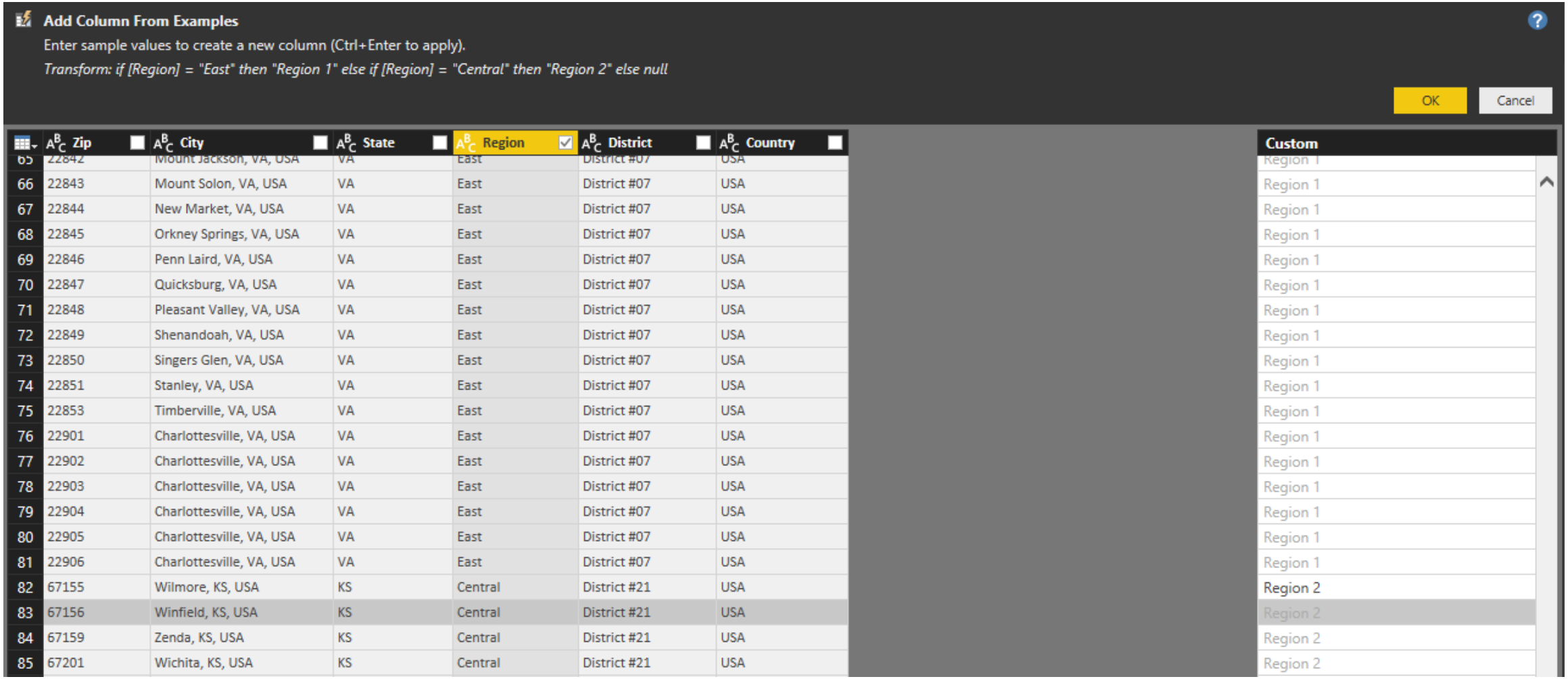


2. When using the Conditional Column format



if … then … else using Columns From Examples

You use Columns From Examples to create if else logic



	Zip	City	State	Region	District	Country	Custom
65	22842	Mount Jackson, VA, USA	VA	East	District #07	USA	Region 1
66	22843	Mount Solon, VA, USA	VA	East	District #07	USA	Region 1
67	22844	New Market, VA, USA	VA	East	District #07	USA	Region 1
68	22845	Orkney Springs, VA, USA	VA	East	District #07	USA	Region 1
69	22846	Penn Laird, VA, USA	VA	East	District #07	USA	Region 1
70	22847	Quicksburg, VA, USA	VA	East	District #07	USA	Region 1
71	22848	Pleasant Valley, VA, USA	VA	East	District #07	USA	Region 1
72	22849	Shenandoah, VA, USA	VA	East	District #07	USA	Region 1
73	22850	Singers Glen, VA, USA	VA	East	District #07	USA	Region 1
74	22851	Stanley, VA, USA	VA	East	District #07	USA	Region 1
75	22853	Timberville, VA, USA	VA	East	District #07	USA	Region 1
76	22901	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
77	22902	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
78	22903	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
79	22904	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
80	22905	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
81	22906	Charlottesville, VA, USA	VA	East	District #07	USA	Region 1
82	67155	Wilmore, KS, USA	KS	Central	District #21	USA	Region 2
83	67156	Winfield, KS, USA	KS	Central	District #21	USA	Region 2
84	67159	Zenda, KS, USA	KS	Central	District #21	USA	Region 2
85	67201	Wichita, KS, USA	KS	Central	District #21	USA	Region 2

Module 2 Lab:
Create and Update Dimension Table using M

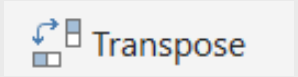
MODULE 3:
Fact Table, Understanding Logic Operands & Functions

MODULE 3 OBJECTIVE

- Understand Transformations
- Understand Join operation
- Create a fact table using transformations

Key Transformations

Transpose



Convert Rows to Columns and Columns to Rows

- **Syntax:**
  Table.Transpose(Previous)
- Column headers are lost
- To transpose column headers, demote them to data prior to transpose

	Color	Letter	Value
1	Red	A	1
2	Red	B	2
3	Blue	C	3
4	Blue	D	4
5	Green	E	5

	Column1	Column2	Column3	Column4	Column5
1	Red	Red	Blue	Blue	Green
2	A	B	C	D	E
3	1	2	3	4	5

Pivot Column

Convert Selected column of values into Column Headers

- **Syntax:**

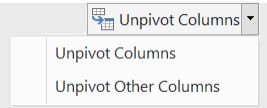
Table.Pivot(Previous,List.Distinct(Previous[Color]), "Color", "Value", List.Sum)

- The values in [Color] column are converted to headers
- The values in the [Value] column are filled in where applicable

	Color	Letter	Value
1	Red	A	1
2	Red	B	2
3	Blue	C	3
4	Blue	D	4
5	Green	E	5

	Letter	Red	Blue	Green
1	A	1	null	null
2	B	2	null	null
3	C	null	3	null
4	D	null	4	null
5	E	null	null	5

Unpivot Columns



Convert Selected Column Headers into a column of values

- **Syntax:**
- Table.UnpivotOtherColumns(Previous, {"Letter"}, "Attribute", "Value")
- The unpivoted column headers become the column [Attribute]
- The unpivoted values become the column [Value]

	Letter	Red	Blue	Green
1	A	1	null	null
2	B	2	null	null
3	C	null	3	null
4	D	null	4	null
5	E	null	null	5

	Letter	Attribute	Value
1	A	Red	1
2	B	Red	2
3	C	Blue	3
4	D	Blue	4
5	E	Green	5

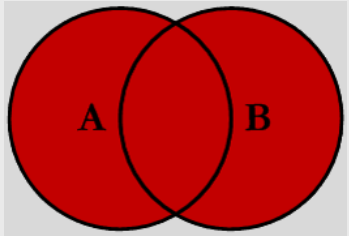
Note: Previous => Name of the previous step in the query

Join Kinds – Merge types

Full Outer (JoinKind.FullOuter)

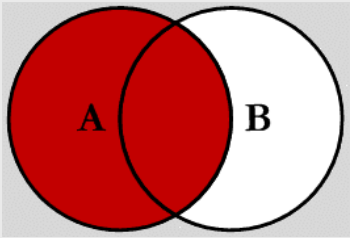
Shows **all of the rows from both queries.**

Append Table= Like SQL "Union All"

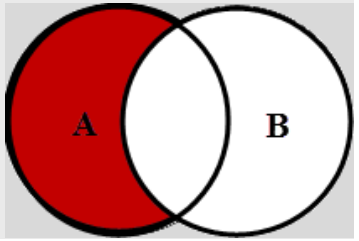


Left Outer (JoinKind.LeftOuter)
Default for Merge

Shows all rows in 1st query (**A**) and "paints" attributes from second query (**B**) based on matches in joined column(s).

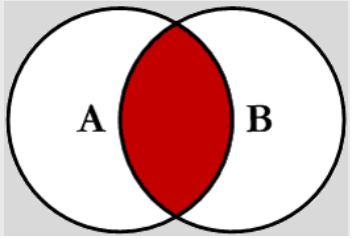


Left Anti (JoinKind.LeftAnti)
Shows ONLY rows in 1st (**A**) query where there is NO MATCH to 2nd query (**B**) based on joined column(s)

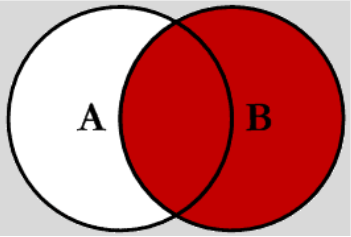


Inner (JoinKind.Inner)

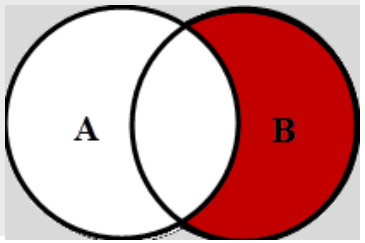
Limits the rows to ONLY those which Match on joined column(s)



Right Outer (JoinKind.RightOuter)
Shows all rows in 2nd (**B**) query and "paints" attributes from 1st query (**A**) based on matches in joined column(s)



Right Anti (JoinKind.RightAnti)
Shows ONLY rows in 2nd (**B**) query where there is NO MATCH to 1st query (**A**) based on joined column(s)



Note: One or Multiple columns can be used to create the join

Join Kinds – Other Combining Queries

Merge

Combines attributes (columns) of two queries based on a joined columns(s)

- Column(s) used for the join must be in both queries (can be one or more)
- Both queries rerun with each execution
- In Database terms, this is a JOIN

Append

"Stack" records of two (or more) queries

- Both (all) queries rerun with each execution
- Like column headers must be named the same
- Non-matching columns will be added to the right as extra colunns

Duplicate

Copies a query at a point in time (Save As)

- Duplicate query can be modified independently without altering the original query
- Executing the 2nd query (duplicate) does NOT execute the 1st (original) query

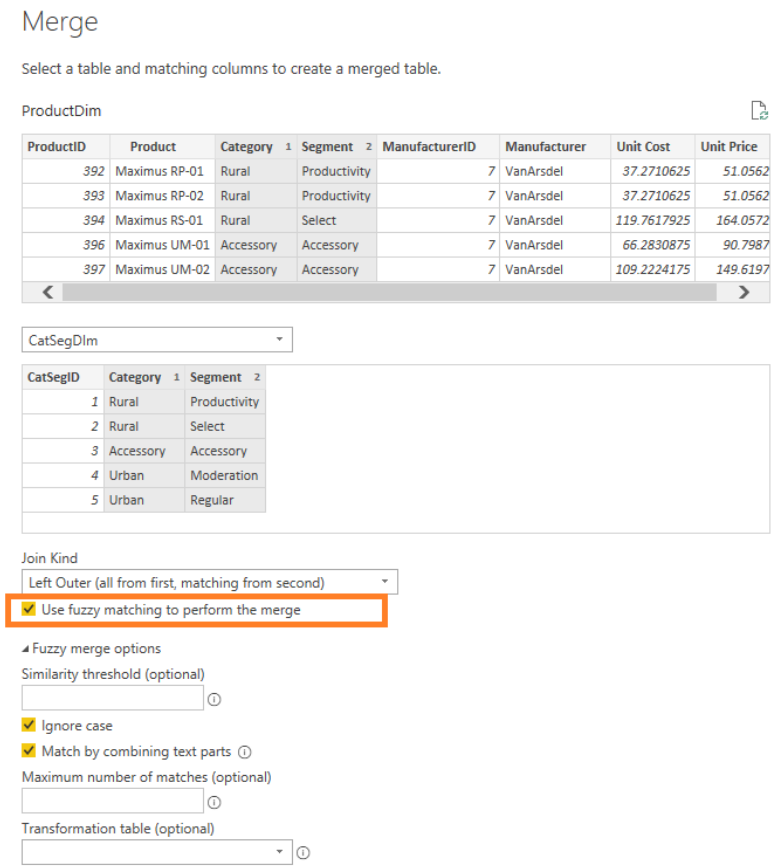
Reference

Uses the output of one query as the INPUT in another query. (Link in a chain)

- All subsequent changes made to 1st query affect the 2nd during refresh
- When the 2nd query executes, it re-executes all the steps in the 1st query

Fuzzy Merge

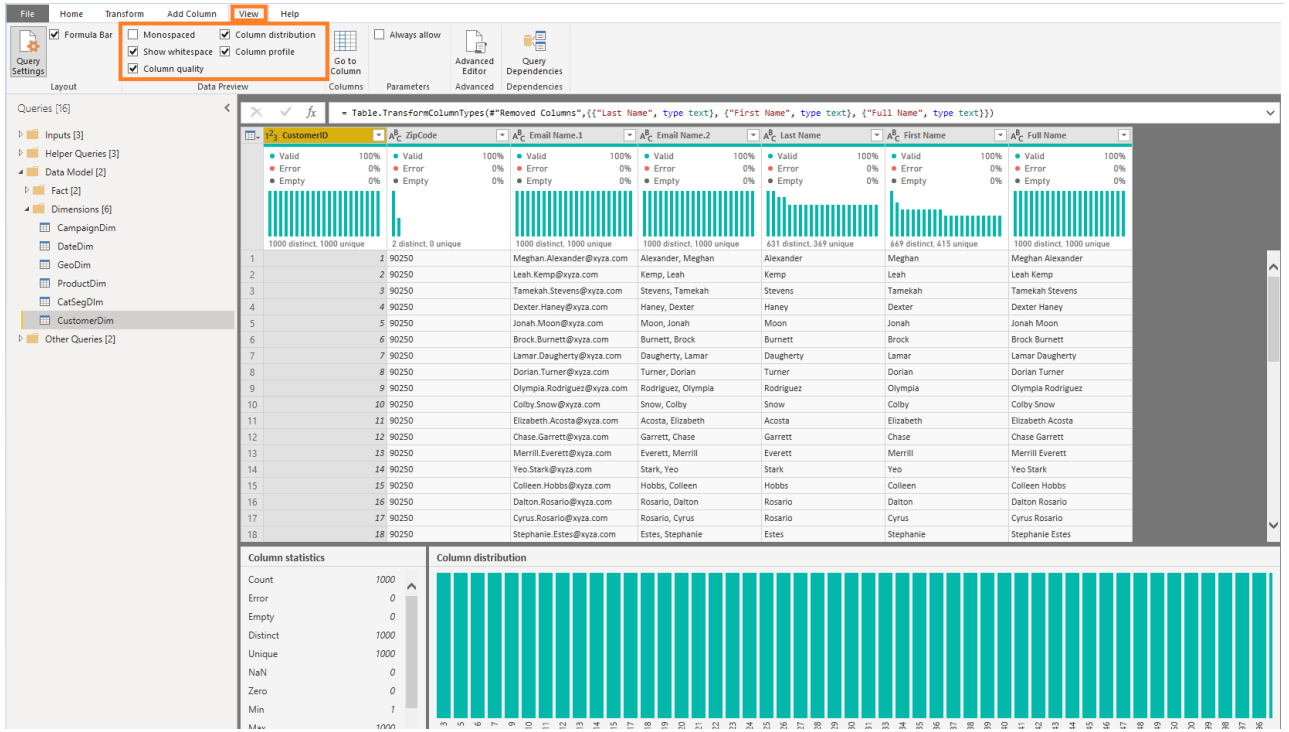
Fuzzy Merge allows you to apply Fuzzy Matching algorithms when comparing columns and try to find matches across tables being merged.



Data Profiling

Data Profiling allows you to easily and quickly understand data distribution

- **Column statistics** – # of errors, empty, valid, duplicated and unique values. Value distribution measures such as Min/Max/Average/Median, etc.
- **Column distribution –** Larger size version of the inline value distribution histograms, also including the ability to Keep or Remove values, which will generate the corresponding Filter Rows step in your query (“Equals” / “Does Not Equal” filters).



RELATED Function

Row Context and Multiple Tables – RELATED Function

```
Sales[COGS] = RELATED(ProductDim[Unit Cost]) * Sales[Units]
```

ProductID	Date	CustomerID	CampaignID	Units	Sales Amount	COGS C
577	10/29/15	164277	10	1	$102.37	$74.73
577	10/29/15	3934	10	1	$102.37	$74.73
577	9/17/15	35	10	1	$102.37	$74.73
577	12/24/15	11	11	1	$102.37	$74.73
577	12/24/15	15	11	1	$102.37	$74.73
577	8/27/15	5	11	1	$102.37	$74.73

ProductID	Product	Category	Segment	ManufacturerID	Manufacturer	Unit Cost	Unit Price
577	Maximus UC-42	Urban	Convenience	7	VanArsdel	74.73	102.37
578	Maximus UC-43	Urban	Convenience	7	VanArsdel	57.48	78.74
579	Maximus UC-44	Urban	Convenience	7	VanArsdel	96.96	132.82
580	Maximus UC-45	Urban	Convenience	7	VanArsdel	60.92	83.45
581	Maximus UC-46	Urban	Convenience	7	VanArsdel	101.54	139.10
582	Maximus UC-47	Urban	Convenience	7	VanArsdel	26.06	35.69

- **RELATED is just like VLOOKUP in Excel**

RELATED Function

RELATED Function Example

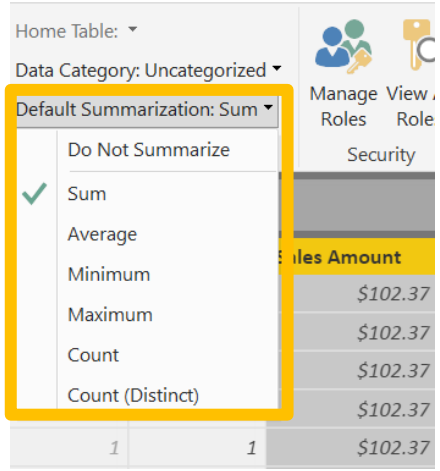
- You *could* follow a chain of relationship from Many side to 1 side using RELATED

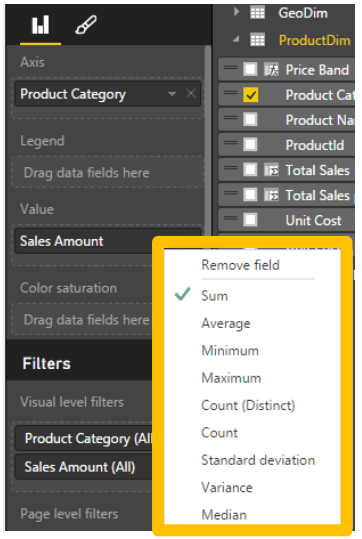
Sales [City State]= RELATED(GeographyDim[City]) & ", " & RELATED(GeographyDim[State])

Default Summarization

What is a Default Summarization?

On the Modeling ribbon





For an individual visualization

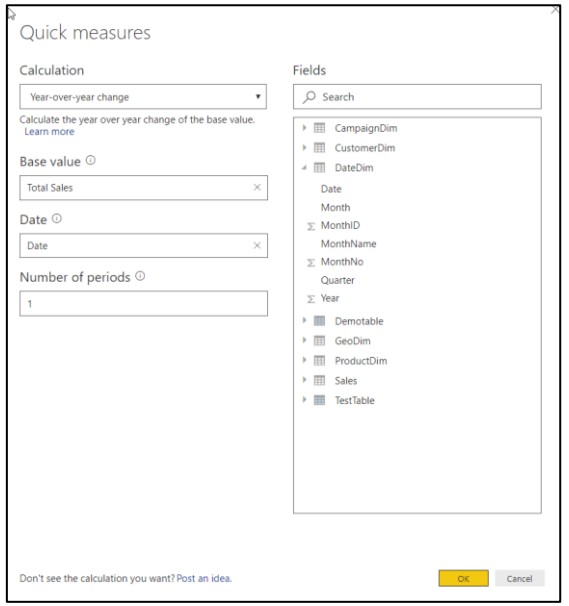
- Sales Amount is automatically summed for each category
- You should set this summarization for all numeric fields in the **Modeling** ribbon

Create Sales amount units by Category

Quick Measures

Quick Measures are wizard driven DAX calculations

- Right+Click a table and select **Quick Measures**
  - Select calculation type and fill-in parameters
  - DAX is generated automatically
  - Great way to learn DAX




```
Total Sales YoY% =
IF(
    ISFILTERED('DateDim'[Date]),
    ERROR("Time intelligence quick measures can only be grouped or filtered by the Power BI-provided date hierarchy or primary date
column."),
    VAR __PREV_YEAR = CALCULATE([Total Sales], DATEADD('DateDim'[Date].[Date], -1, YEAR))
    RETURN
        DIVIDE([Total Sales] - __PREV_YEAR, __PREV_YEAR)
)
```

Category	Total Sales	Total Sales YoY%
Urban	$54,427,851	9.98%
Accessory	$5,991,334	9.06%
Mix	$3,853,181	14.15%
Youth	$1,268,274	3.37%
Rural	$6,500	38.43%
Total	**$65,547,141**	**10.00%**

See Quick Measure gallery: https://community.powerbi.com/t5/Quick-Measures-Gallery/bd-p/QuickMeasuresGallery

Module 3 Lab:

Import and Transform Create additional fact Table

MODULE 4: Parameters & Introduction to DAX

MODULE 4 OBJECTIVE

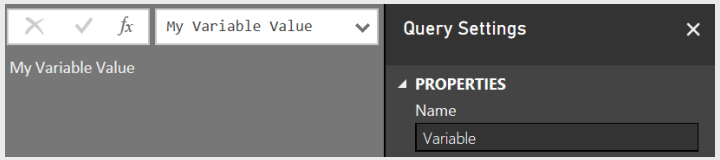
- Understand what is DAX and how can it be used
- Understand working with parameters

Variable and Parameters

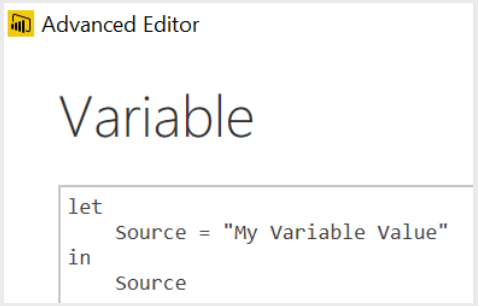
M Variables

Query with a constant value, table or list result

- Create with a new blank query
- Query name is the Variable name to use in other queries
- Type the variable value in the formula bar



- Advanced Editor would look like this:

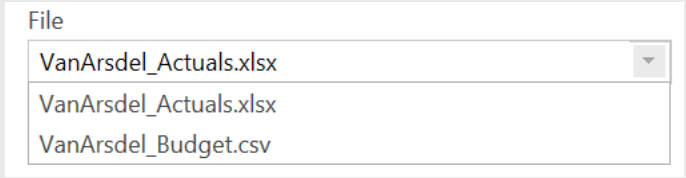


M Parameters

Parameters take Variables to the next level.

- Parameters can be any VALID data type
- The most common are hard keyed single, switchable values

- The values list for a parameter can also come from a query that is formatted as a **LIST**



- Parameter used as Source Input

```
= Excel.Workbook(File.Contents( Path & ActualsFile ), null, true)
```

Parameter Usage

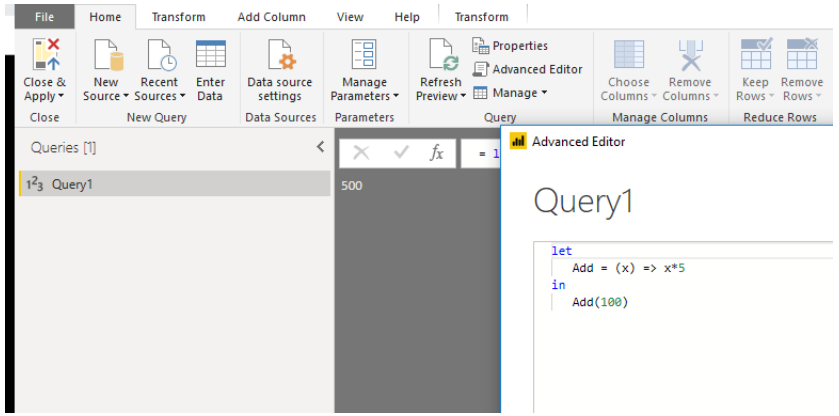
Be Creative!

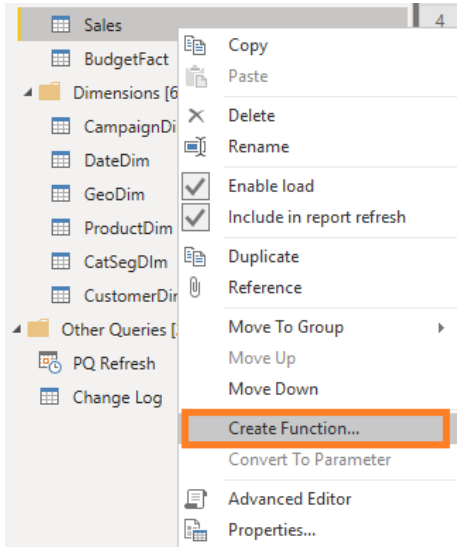
- SQL Server
  - For Database Name
  - For Server (Switch Dev, Test, Prod)
- "Where Clauses" in SQL Queries
  - To pull all Orgs, or just a single one
- SharePoint Team site names
  - If they are based on a template, you can easily share queries
- File Paths and File Names
  - Have a single place where you need to make a change

Custom Functions

Custom Functions help you re-use code

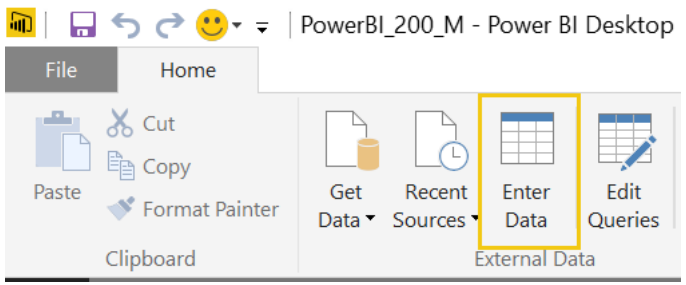
1. Create function using Blank query
2. Create function using Baseline query

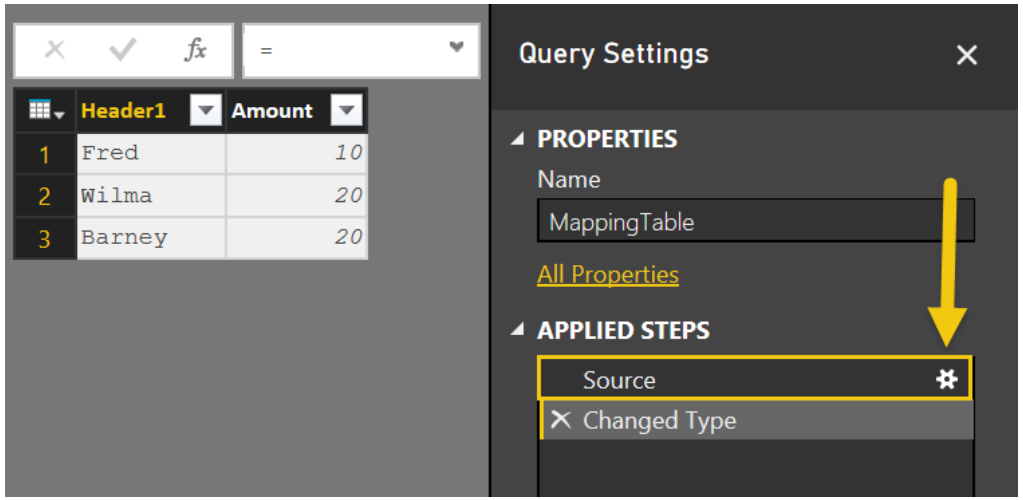


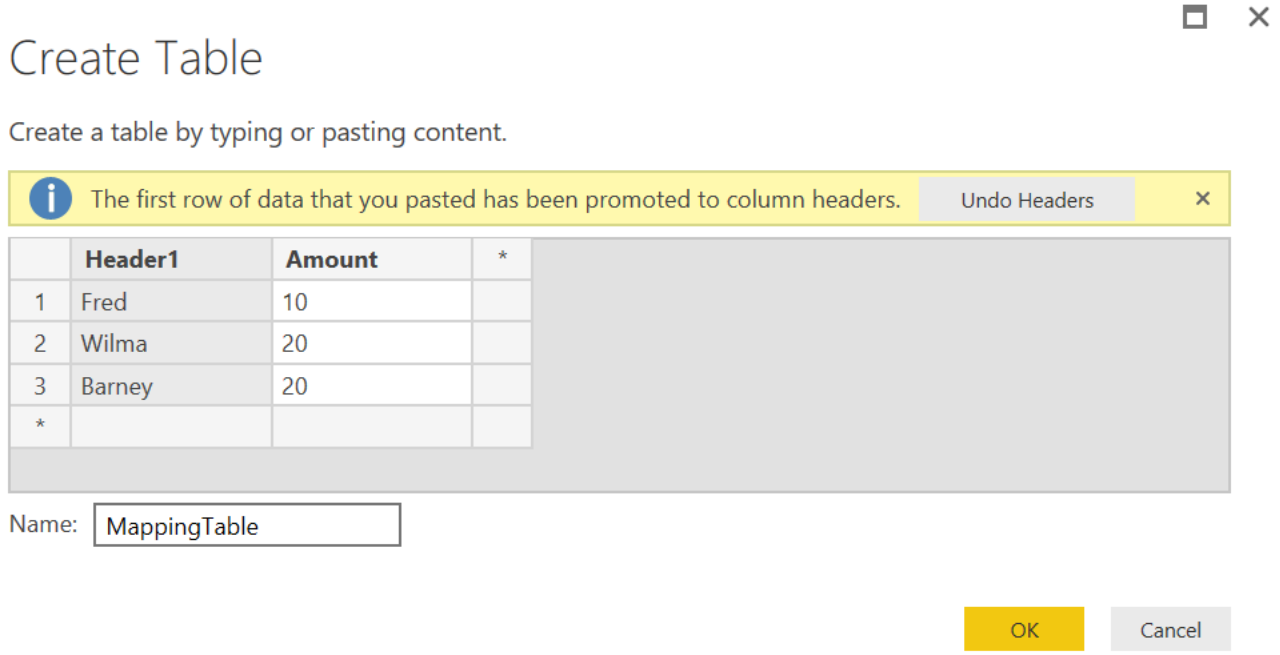


Enter Data

User **Enter Data** to add a table on the fly. They make great mapping tables.







<- To add additional rows or columns later, click the gear icon.

DAX Level Set

- DAX looks similar to Excel functions, but they have key differences
- DAX is a very deep and elegant…
- This class provides a base in DAX, but don’t expect to leave being able to write the most complex DAX patterns--they take practice.

DAX Foundations

Path to DAX Expertise

Evaluation Contexts

CALCULATE

Calculated Columns and Measures

DAX Foundations

Calculated Column

Measure

What is a Calculated Column?

```
Price Band = If(ProductDim[Unit Price] <=25, "Low",If(ProductDim[Unit Price] <=50, "Medium", "High"))
```

ProductID	Product	Category	Segment	ManufacturerID	Manufacturer	Unit Cost	Unit Price	Price Band
577	Maximus UC-42	Urban	Convenience	7	VanArsdel	74.73	102.37	High
578	Maximus UC-43	Urban	Convenience	7	VanArsdel	57.48	78.74	High
579	Maximus UC-44	Urban	Convenience	7	VanArsdel	96.96	132.82	High
580	Maximus UC-45	Urban	Convenience	7	VanArsdel	60.92	83.45	High
581	Maximus UC-46	Urban	Convenience	7	VanArsdel	101.54	139.10	High
582	Maximus UC-47	Urban	Convenience	7	VanArsdel	26.06	35.69	Medium
583	Maximus UC-48	Urban	Convenience	7	VanArsdel	40.18	55.05	High
584	Maximus UC-49	Urban	Convenience	7	VanArsdel	45.22	61.94	High

Calculated Column

Pro Tip: Always refer to a calculated column by its full name -> **TableName[ColumnName]**

Calculated Column

Calculated Column in DAX

```
Price Band = If(ProductDim[Unit Price] <=25, "Low",If(ProductDim[Unit Price] <=50, "Medium", "High"))
```

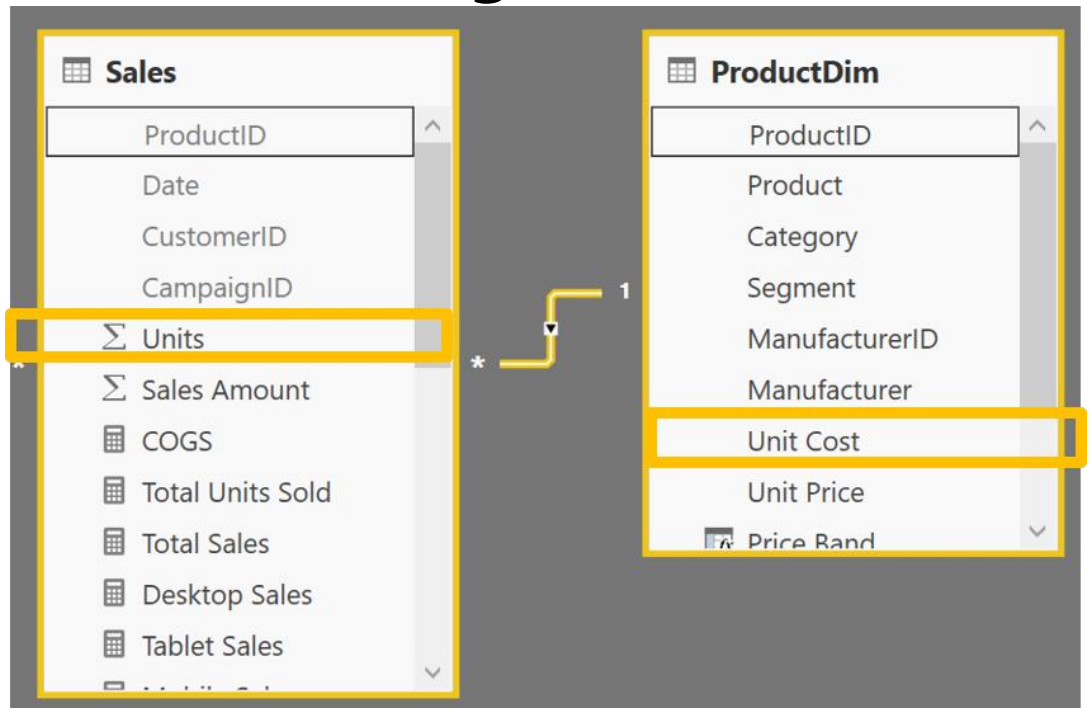
ProductID	Product	Category	Segment	ManufacturerID	Manufacturer	Unit Cost	Unit Price	Price Band
577	Maximus UC-42	Urban	Convenience	7	VanArsdel	74.73	102.37	High
578	Maximus UC-43	Urban	Convenience	7	VanArsdel	57.48	78.74	High
579	Maximus UC-44	Urban	Convenience	7	VanArsdel	96.96	132.82	High
580	Maximus UC-45	Urban	Convenience	7	VanArsdel	60.92	83.45	High
581	Maximus UC-46	Urban	Convenience	7	VanArsdel	101.54	139.10	High
582	Maximus UC-47	Urban	Convenience	7	VanArsdel	26.06	35.69	Medium
583	Maximus UC-48	Urban	Convenience	7	VanArsdel	40.18	55.05	High
584	Maximus UC-49	Urban	Convenience	7	VanArsdel	45.22	61.94	High

Custom Column in "Query Editor"

Note: If given a choice, creating the column in "M" or "Query Editor" will give you better compression.

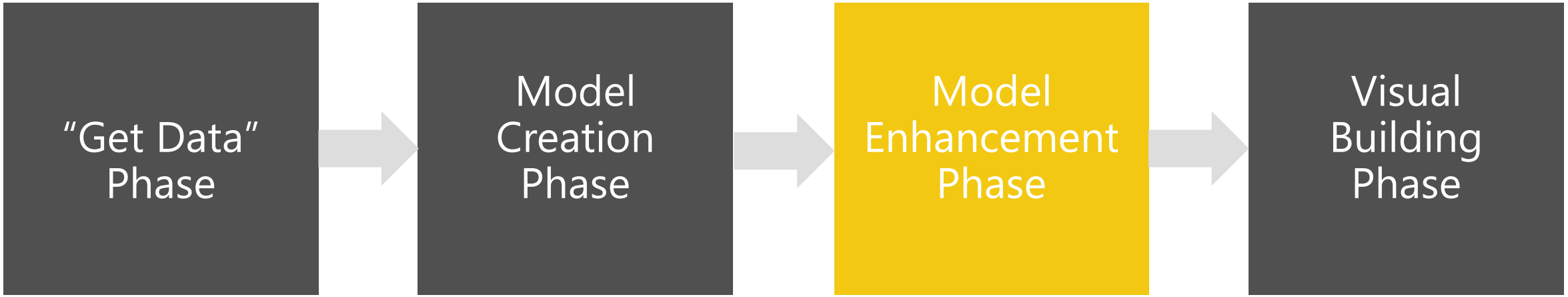
Calculated Column

Calculated Column – Accessing columns from other Tables in model



- **Often you want to access columns from multiple tables to create a Calculated Columns**
- **Let us say you want to calculate COGs, which is Units * Units Cost**
- **Units Cost is in another Table**

When is a Calculated Column Evaluated?



- Create Query in "M"

- Compress data
- Auto detect relationships

- Add calc. columns, Measures
- Add missing relationships

- Evaluate Measures and build each visual

DAX Foundations

Calculated Column

Measure

Measures

What is a Measure?

ProductID	Date	CustomerID	CampaignID	Units	Sales Amount
666	2/24/12	58642	3	1	$81.37
666	2/25/12	208515	3	1	$81.37
666	7/12/12	164032	3	1	$81.37
666	7/12/12	243676	3	1	$81.37
406	6/12/16	31036	16	1	$191.62
406	6/17/16	44688	16	1	$191.62
406	6/17/16	108991	16	1	$191.62

[Total Sales]=SUM(Sales[Sales Amount])

- Measures are created using DAX
- Place your Measures on a Fact table for best results

Pro Tip: When referring to a measure in other calculations, refer to it without a Table name: **[MeasureName]**

Measures

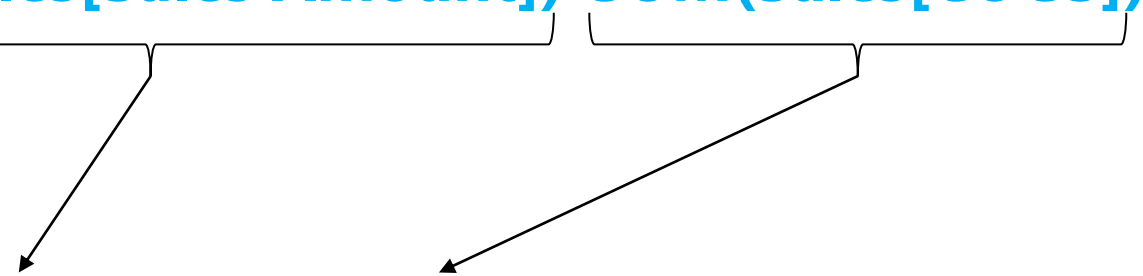
Measure, Use Case 1: Using One Measure in Another

Instead of writing this:

```
[Profit] = SUM(Sales[Sales Amount])-SUM(Sales[COGS])
```

Write this:

```
[Profit] = [Total Sales]- [Total COGS]
```



- Allows re-use of measures
- Formulas are much simpler to read

Measures

Measure, Use Case 2: More Complex Calculations

```
[Profit Margin %] = [Profit] / [Total Sales]
```

- Ratios are calculations that cannot be created using a Calculated Column or Default Summarization
- Use DAX **DIVIDE** for built in error handling

```
[Profit Margin %] = DIVIDE([Profit] , [Total Sales])
```

Measures

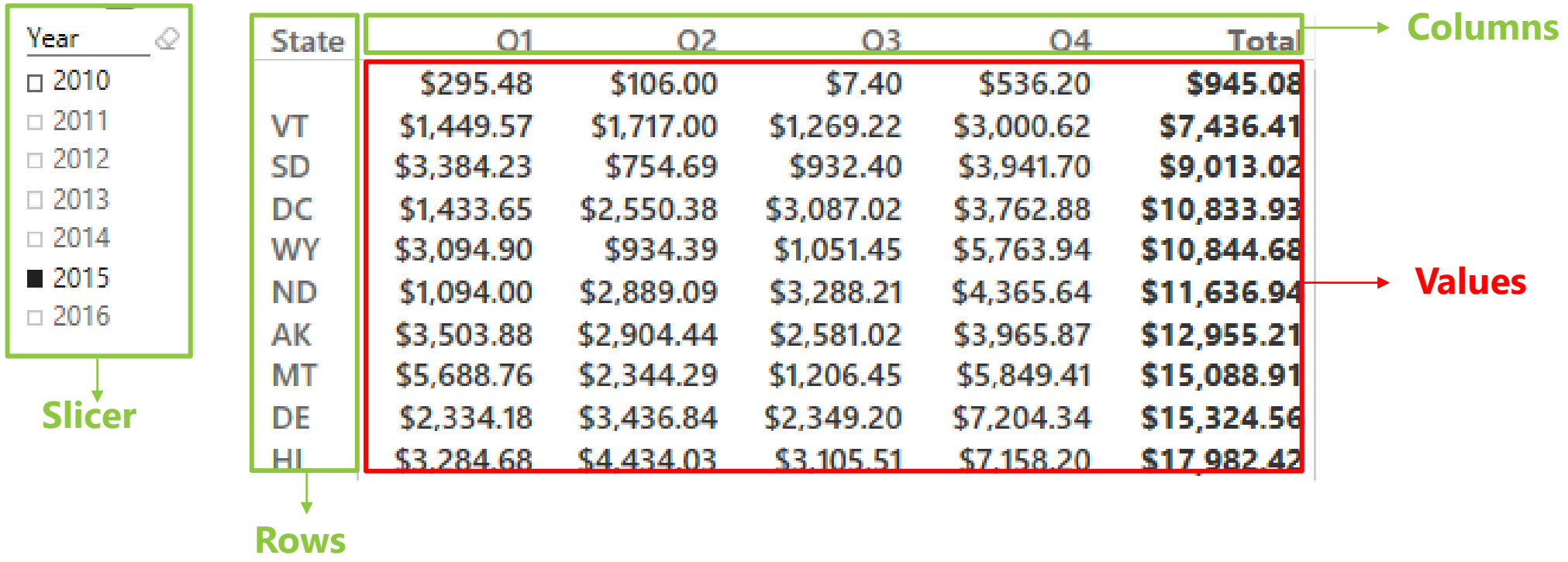
Measure, Use Case 3: More Complex Calculations Using Variables

```
MobileSalesLastYear =
    VAR MobileProducts = FILTER(
                            ALL('CampaignDim'[Device]),
                            CampaignDim[Device]="Mobile"
                        )
    VAR LastYear = SAMEPERIODLASTYEAR('DateDim'[Date])
    RETURN
    CALCULATE(SUM(Sales[Sales Amount]),MobileProducts,LastYear)
```

- Allows re-use of variables
- Formulas are much simpler to read

Calculated Column vs. Measure

Calculated Column vs. Measure - When to Use What



Rule of Thumb for Calculated Column vs Measure

- **Calculated Column** – Use in Page, Report & Visual Filters as well as Slicers, Rows and Columns
- **Measures** - Use in Values section

DAX Foundations

PATH to DAX Expertise

Evaluation Contexts

CALCULATE

Calculated Columns and Measures

DAX Function Types

Scalar Functions

- Scalar functions return a Single value as an output
- Ex. SUM(Sale[Sales Amount])

Table Functions

- Table functions return a Table as an output
- Ex. ALL(GeographyDim)

There are other ways to classify functions – By kind of operation they perform etc.

Evaluation Context

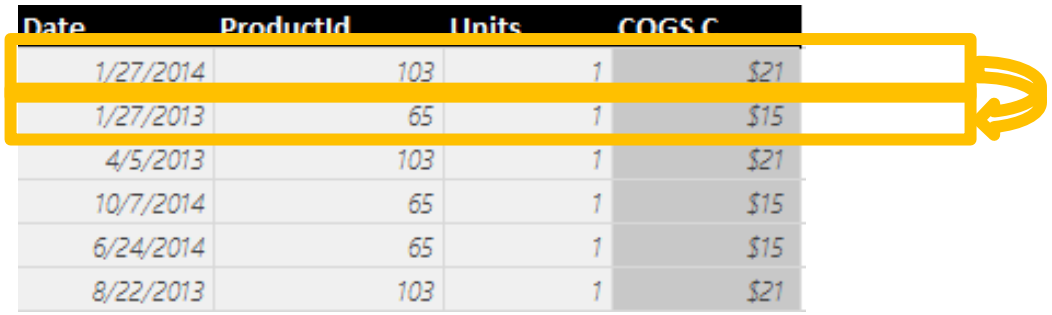
There are two contexts under which calculations are evaluated

Row Context

Filter Context

Row context in Calculated Column

Sales[COGS] = RELATED(ProductDim[Unit Cost]) * Sales[Units]



Date	ProductId	Units	COGS C
1/27/2014	103	1	$21
1/27/2013	65	1	$15
4/5/2013	103	1	$21
10/7/2014	65	1	$15
6/24/2014	65	1	$15
8/22/2013	103	1	$21

- Formula is evaluated row by row
- The context under which formula is evaluated for each row is called “Row Context”

Pro Tip: To accumulate up from Fact to Dimension, use **RELATEDTABLE()**

Evaluation Context

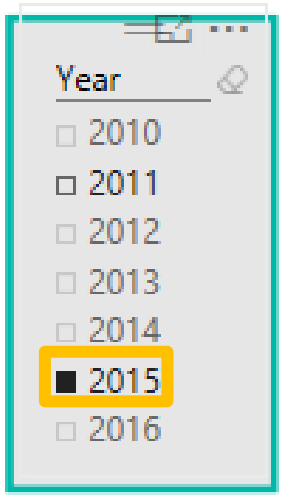
Both Calculated Columns and Measures are always evaluated under two contexts

Filter Context in Measures

Filter Context in a Measure – Example 1

[Total Sales] = SUM(Sales[Sales Amount])

Filter Context for current coordinate Year = 2015, State = **HI**, Quarter = **Q1**



State	Q1	Q2	Q3	Q4	Total
	$295.48	$106.00	$7.40	$536.20	**$945.08**
VT	$1,449.57	$1,717.00	$1,269.22	$3,000.62	**$7,436.41**
SD	$3,384.23	$754.69	$932.40	$3,941.70	**$9,013.02**
DC	$1,433.65	$2,550.38	$3,087.02	$3,762.88	**$10,833.93**
WY	$3,094.90	$934.39	$1,051.45	$5,763.94	**$10,844.68**
ND	$1,094.00	$2,889.09	$3,288.21	$4,365.64	**$11,636.94**
AK	$3,503.88	$2,904.44	$2,581.02	$3,965.87	**$12,955.21**
MT	$5,688.76	$2,344.29	$1,206.45	$5,849.41	**$15,088.91**
DE	$2,334.18	$3,436.84	$2,349.20	$7,204.34	**$15,324.56**
HI	$3,284.68	$4,434.03	$3,105.51	$7,158.20	**$17,982.42**

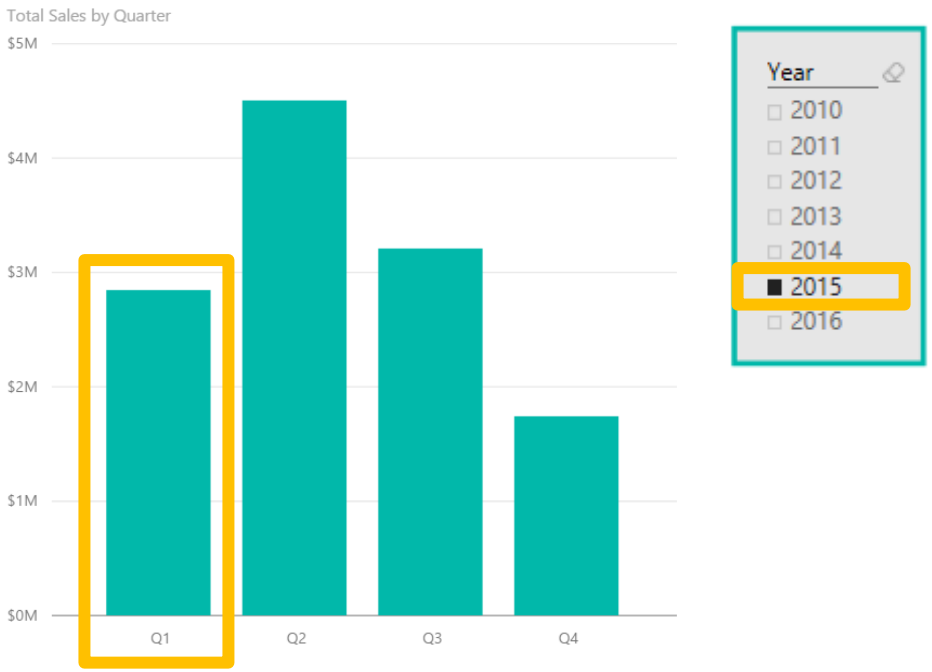
- Formula is evaluated for each "Coordinate" in each visual
- The context for each coordinate is called "Filter Context"

Filter Context in a Measure

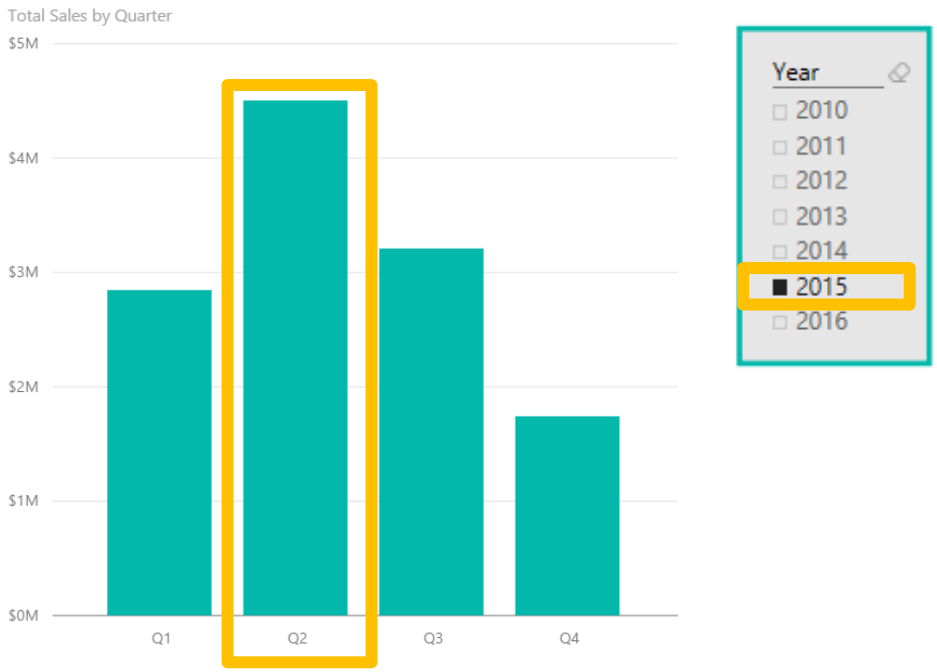
Filter Context in a Measure – Example 2

[Total Sales] = SUM(Sales[Sales Amount])

Filter Context : Year = 2015, Quarter = **Q1**



Filter Context : Year = 2015, Quarter = **Q2**



Filter Context in a Measure

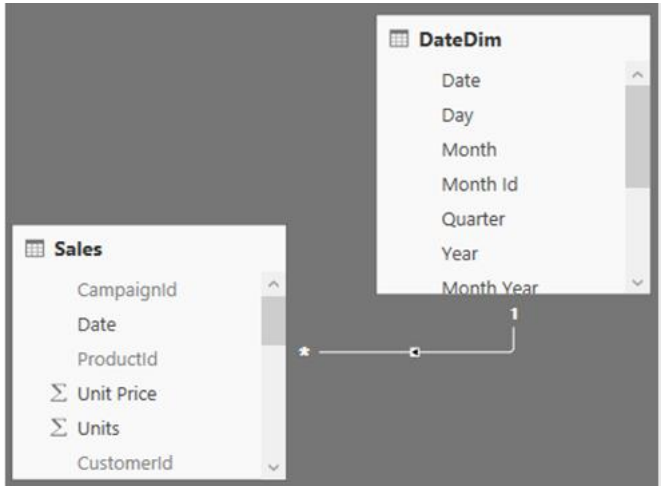
Filter Context in a Measure

```
[Total Sales] = SUM(Sales[Sales Amount])
```

Better definition of above measure:

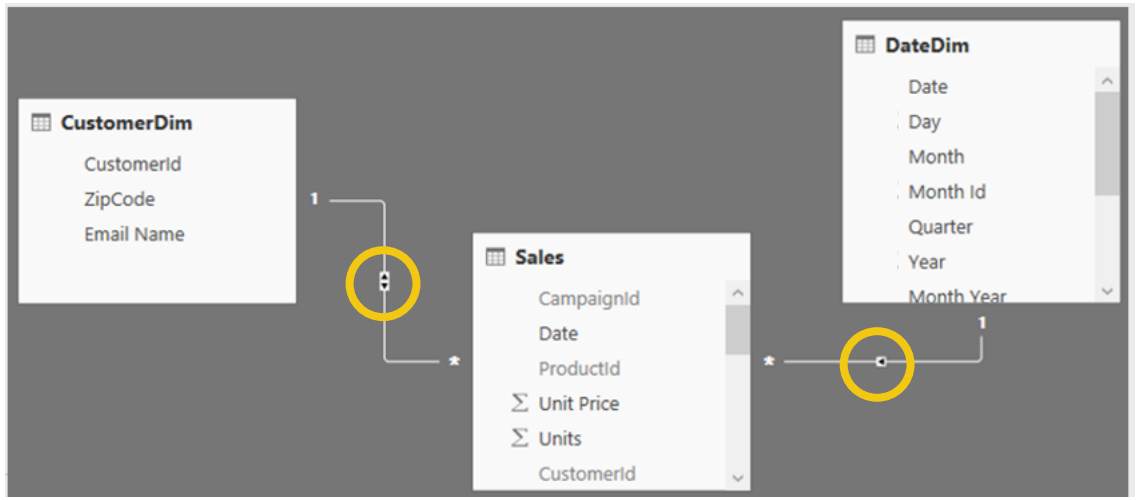
"Total Sales" – SUM of Sales[Sales Amount] column **under a filter context**

Filter Context and Multiple Tables



- Filter context automatically propagates from Dim Table to Fact Table
- Filtering the DateDim Table to Year = 2015 returns only Sales for 2015

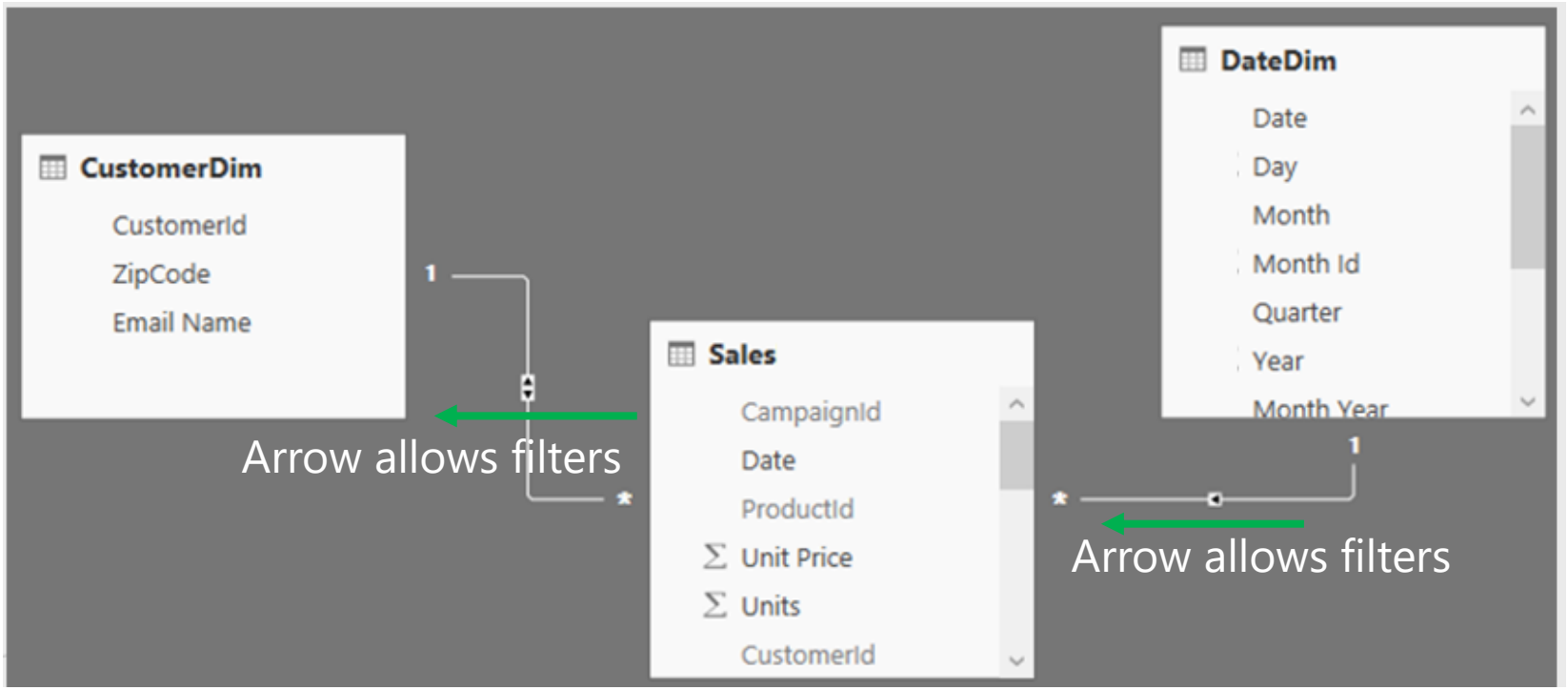
Filter Context and Multiple Tables



- Filters (Filter context) automatically propagate based on direction of arrows in relationships
- Examples
  - Filter goes from DateDim to CustomerDim
  - Filter does not go from CustomerDim to DateDim

Filter Context and Multiple Tables

Filter Context and Multiple Tables – Right Arrow Direction



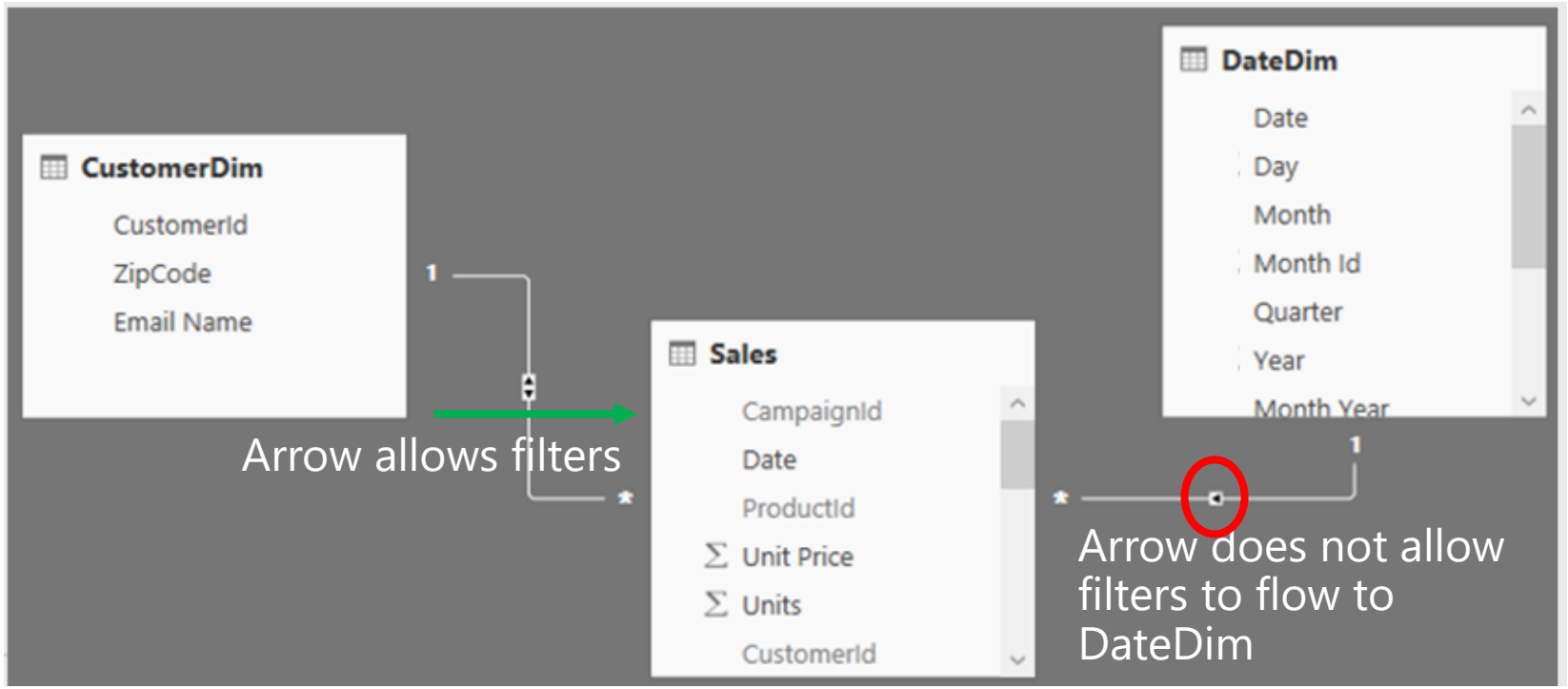
Cross filtering works properly

Month	Total Sales M	Count of CustomerId
Jan	$1,673,394.03	7132
Feb	$431,531.13	2820
Mar	$690,671.10	4017
Apr	$852,018.76	4629
May	$972,018.47	5185
Jun	$907,703.04	4854
Jul	$608,678.35	3680
Aug	$1,355,530.22	6242
Sep	$720,851.83	4186
Oct	$1,117,087.73	5728
Nov	$2,372,763.71	8242
Dec	$2,003,261.11	7683
Total	**$13,705,509.48**	**10000**

- Filter goes from DateDim to CustomerDim
- This is why the above Pivot table works

Filter Context and Multiple Tables

Filter Context and Multiple Tables – Wrong Arrow Direction



Cross filtering does not work

CustomerId	Total Sales M	Count of Month
	$1,985.76	12
00001	$438.34	12
00002	$840.08	12
00003	$1,246.69	12
00004	$706.23	12
00005	$1,653.97	12
00006	$2,170.10	12
00007	$2,308.44	12
00008	$1,517.34	12
00009	$1,184.11	12
00010	$2,221.02	12
00011	$1,646.48	12
Total	**$13,705,509.48**	**12**

- Filter goes from DateDim to CustomerDim
- This is why the Count of Month in the table above is incorrect

Evaluation Context and Multiple Tables

Evaluation Context Multiple Table – Summary and Take Aways

Row Context

- Does not propagate automatically
- Need to use
  RELATED
  RELATEDTABLE

Filter Context

- Propagates automatically
- Depends on direction of arrow in relationship diagram

Applications of Table functions

Table functions can be used 2 ways in Power BI Desktop

- As an input to another DAX function
  - CALCULATE
  - Iterator functions

- Calculated Tables

Basic TABLE functions

Return All Rows	Return Distinct Rows	Return Filtered Rows
ALL & variants	ALL, DISTINCT, VALUES	FILTER

There are more advanced Table functions, which we will not cover

Basic Table functions – Return All Rows

- The **ALL** function - Can take either Table or Columns in a Table as input

ALL with Entire Table

ALL(GeographyDim)

Returns all rows all columns in Table

ALL with One Column

ALL(GeographyDim[Region]))

Returns all unique values of Column

ALL with Multiple Columns

ALL(GeographyDim[Region], GeographyDim[State])

Returns all unique combinations of Column values

Basic Table functions – ALL versions

- There are several forms of the ALL function
  - **ALL**
  - **ALLEXCEPT** - Return all columns in a Table except 1 or more columns
  - **ALLSELECTED** – Return all values in a column selected by users in Slicers
  - **ALLNONBLANKROW** – Return all non-Blank rows

Basic Table Functions – Return Distinct Rows

- **VALUES –** Return all distinct values in a column or Table
  (including blank rows)

- **DISTINCT** - Return all distinct values in a column or Table
  (not including blank rows)

Basic Table Functions – Return Distinct Value

- **HASONEVALUE** - Returns TRUE when the context for columnName has been filtered down to one distinct value only. Otherwise is FALSE

```
Header =  IF ( HASONEVALUE ( ProductDim[Price Band] ),
                    CONCATENATE ("Report Header for Price Band : ",
                    VALUES ( ProductDim[Price Band] )),
               "Overall Report")
```

- **SELECTEDVALUE** - Returns the value when the context for columnName has been filtered down to one distinct value only. Otherwise returns alternateResult.

```
Header (SELECTEDVALUE) =
            VAR selectedPriceBand =SELECTEDVALUE ( ProductDim[Price Band] )
            RETURN
            IF ( ISBLANK ( selectedPriceBand ),  "Overall Report",
              CONCATENATE ( "Report Header for Price Band: ", selectedPriceBand )
            )
```

Basic Table Functions – Return Filtered Set of Rows

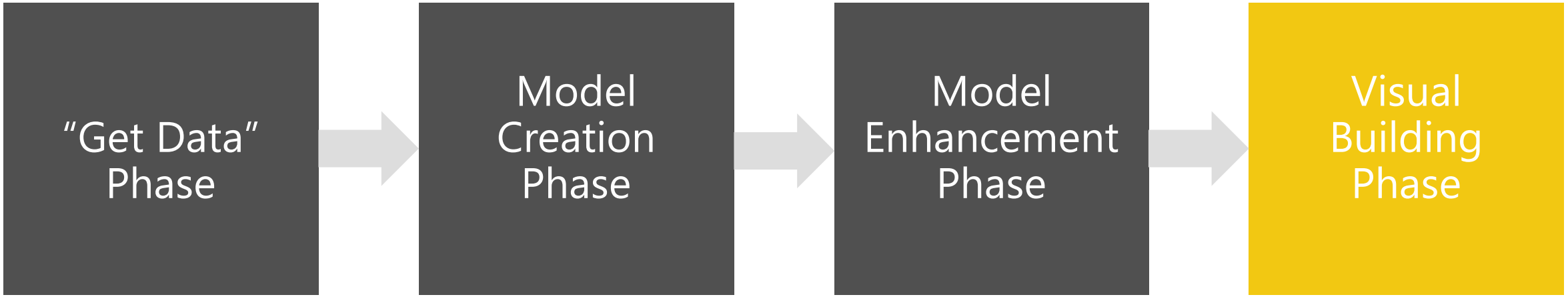
FILTER(ALL(GeographyDim[Region], GeographyDim[State]), GeographyDim[Region] = “Central”)

- Take all unique combinations of GeographyDim[Region], GeographyDim[State]
- Filter down to the rows where GeographyDim[Region] = “Central”

State	Region
CO	Central
MT	Central
OK	Central
UT	Central
IL	Central
IA	Central
WY	Central
SD	Central
ND	Central
NM	West
TX	West
NV	West

State	Region
CO	Central
MT	Central
OK	Central
UT	Central
IL	Central
IA	Central
WY	Central
SD	Central
ND	Central

When is a Measure Evaluated?



- Create Query in "M"

- Compress data
- auto detect relationships

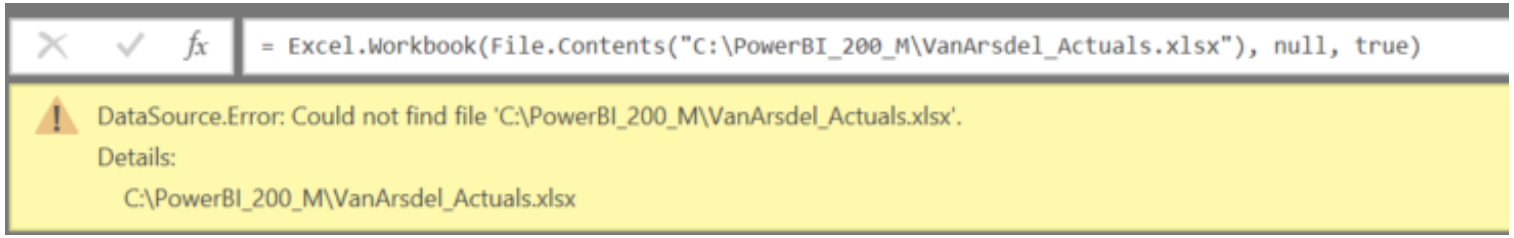
- Add calc. columns, Measures
- Add missing relationships

- Evaluate Measures and build each visual

Module 4 Lab: File Source considerations

All of the queries in our file are dependent on the file being in a specified path

- If you move the source file and try to refresh the queries you will receive an error
  - C:\ to File Share
  - Different folder location
  - To SharePoint
  - To OneDrive



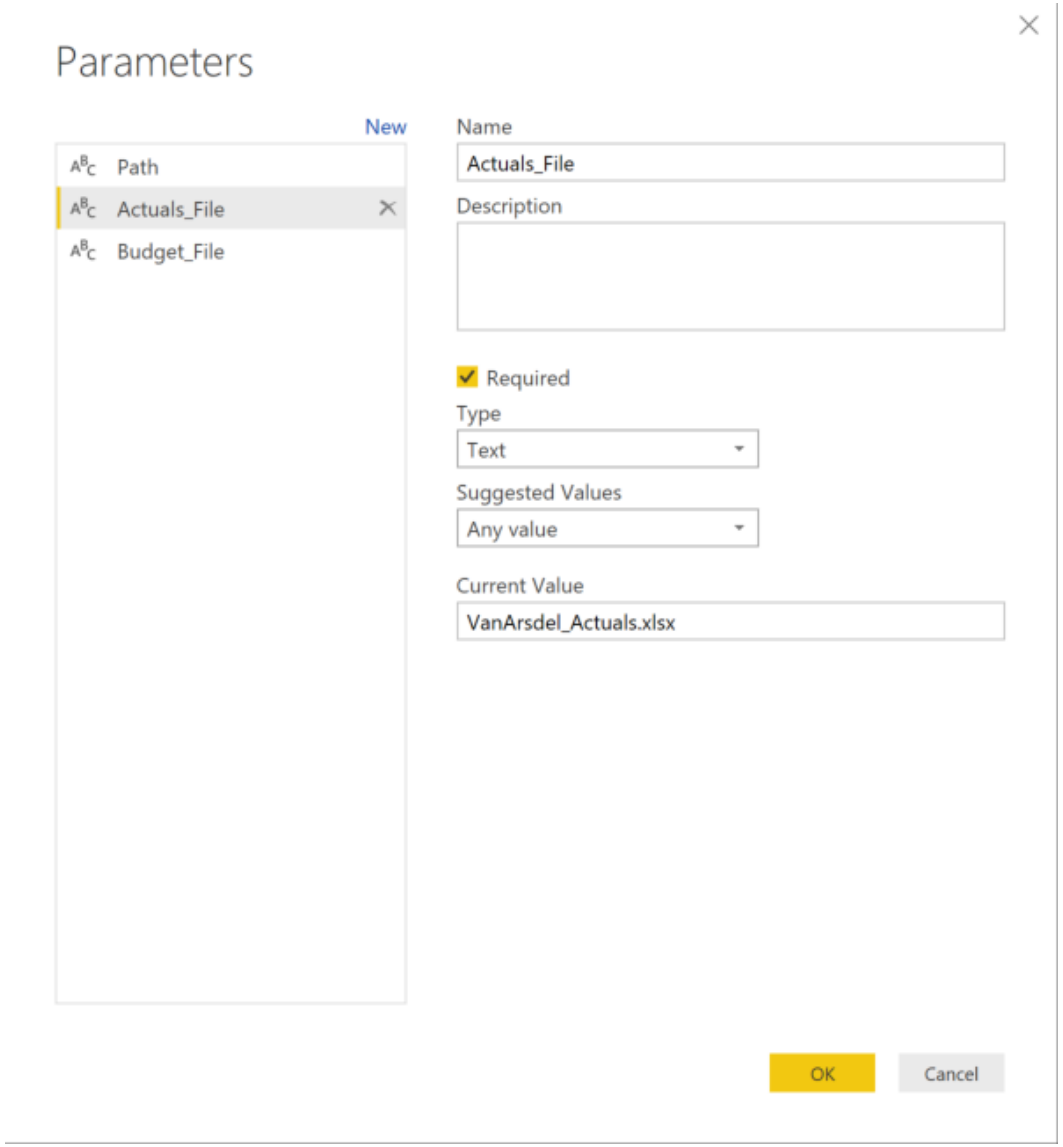
- Each query will need to be changed EACH time the file is moved

Creating Parameters with Dynamic Paths resolves this issue

Module 4 Lab: Create Parameters

Create Parameters

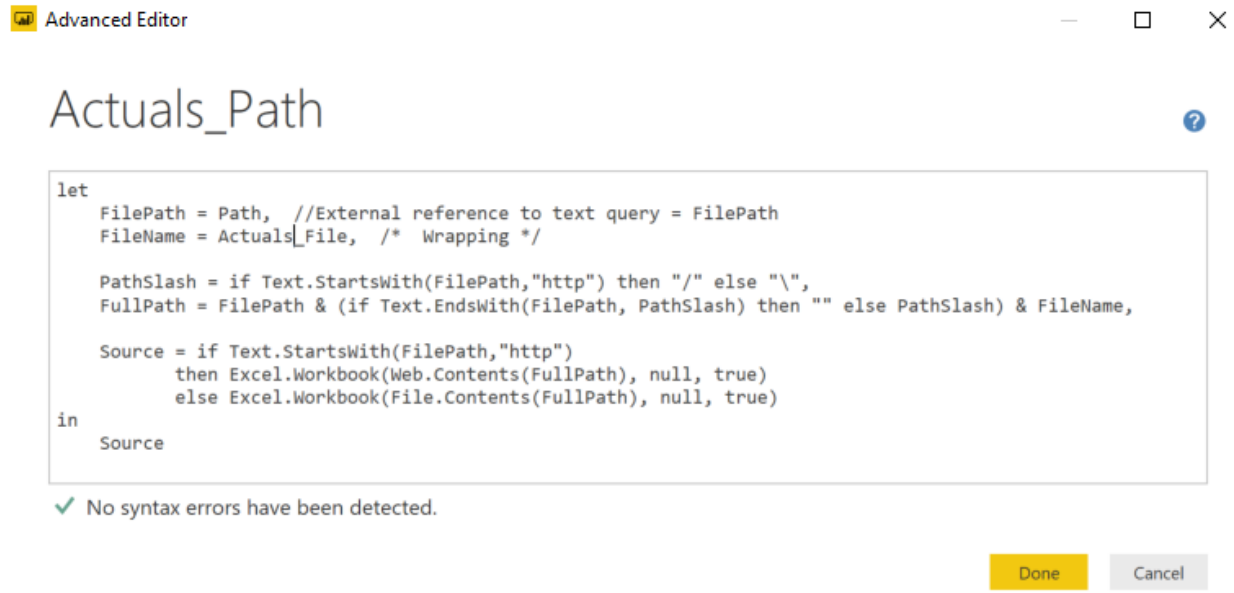
- Actual File
- Budget File
- Path

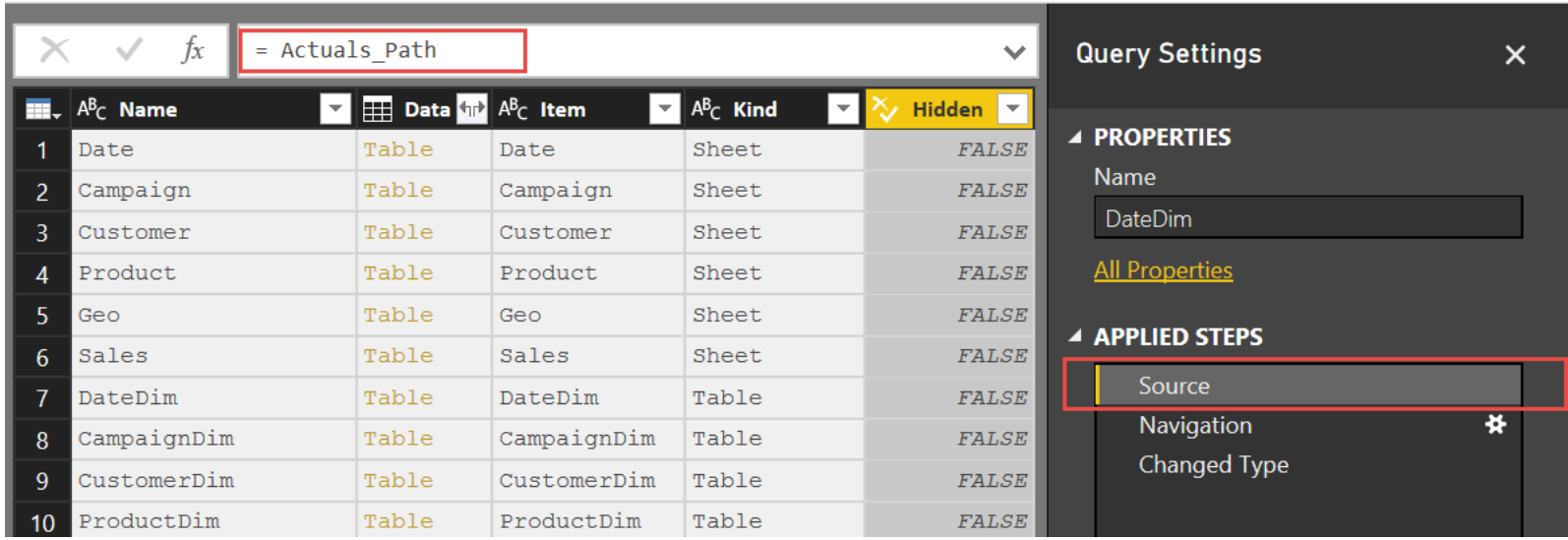


Module 4 Lab: Dynamic Path to Excel Source File

Use blank query to create Dynamic Path

- Uses Blank Query
- Populate Advanced Editor of query with text from file provided
- For each query which uses the Excel source, update the Source (applied step) with the new variable name ***=Actuals_Path***





	Name	Data	Item	Kind	Hidden
1	Date	Table	Date	Sheet	FALSE
2	Campaign	Table	Campaign	Sheet	FALSE
3	Customer	Table	Customer	Sheet	FALSE
4	Product	Table	Product	Sheet	FALSE
5	Geo	Table	Geo	Sheet	FALSE
6	Sales	Table	Sales	Sheet	FALSE
7	DateDim	Table	DateDim	Table	FALSE
8	CampaignDim	Table	CampaignDim	Table	FALSE
9	CustomerDim	Table	CustomerDim	Table	FALSE
10	ProductDim	Table	ProductDim	Table	FALSE

Module 4 Lab: Dynamic Path to Excel Source File

Use the **Actuals_Path** query as a variable in other queries

- In Advanced Editor, update the Excel source to the new Actuals_Path source.

CampaignDim — Original

```
let
    Source = Excel.Workbook(File.Contents("C:\PowerBI_200_M\PowerBI_200_M_Data.xlsx"), null, true),

    CampaignDim_Table = Source{[Item="CampaignDim",Kind="Table"]}[Data],
    #"Changed Type" = Table.TransformColumnTypes(CampaignDim_Table,{{"CampaignId", type text}, {"Traffic Chann
in
    #"Changed Type"
```

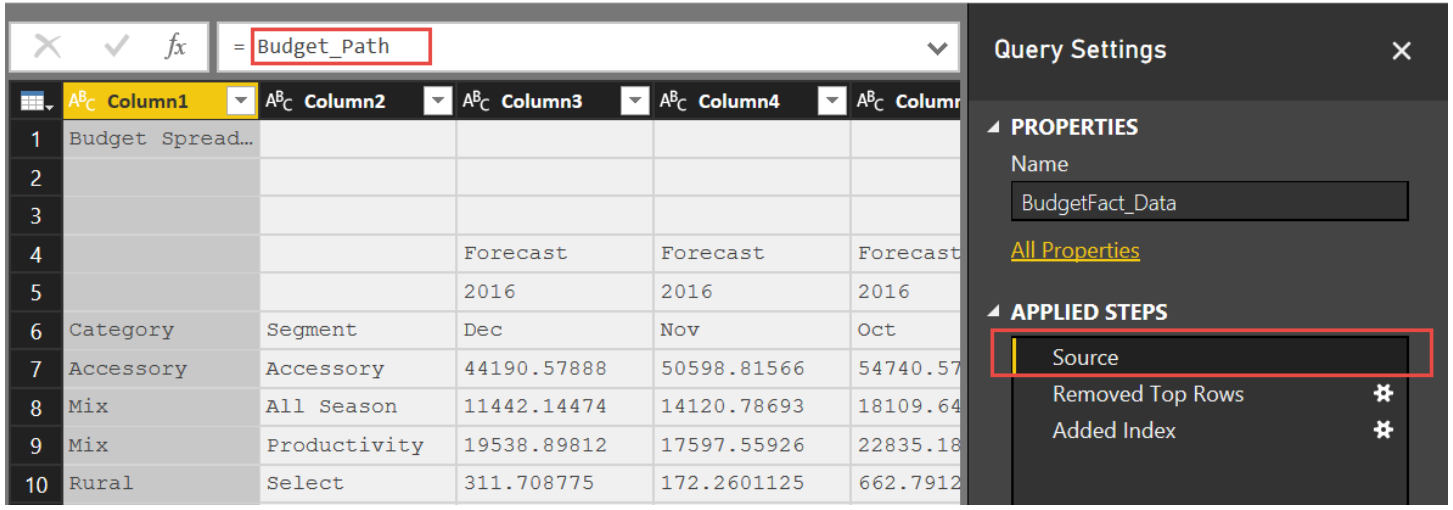
CampaignDim — Updated

```
let
    Source = Actuals_Path,
    Campaign_Sheet = Source{[Item="CampaignDim",Kind="Table"]}[Data],
    #"Changed Type" = Table.TransformColumnTypes(Campaign_Sheet,{{"CampaignID", Int64.Type}, {"Traffic
in
    #"Changed Type"
```

Module 4 Lab: Dynamic Path to CSV Source File

Follow a similar patter to make the Budget CSV file dynamic

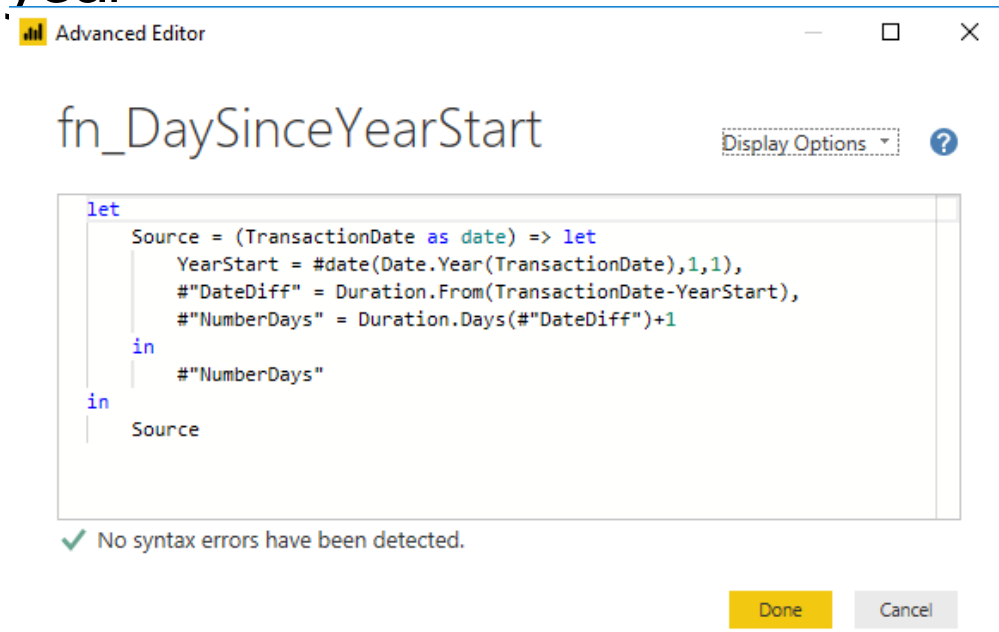
- Uses Blank Query
- Populate Advanced Editor of query with text from file provided
- For each query which uses the Excel source, update the Source (applied step) with the new variable name
  =Budget_Path

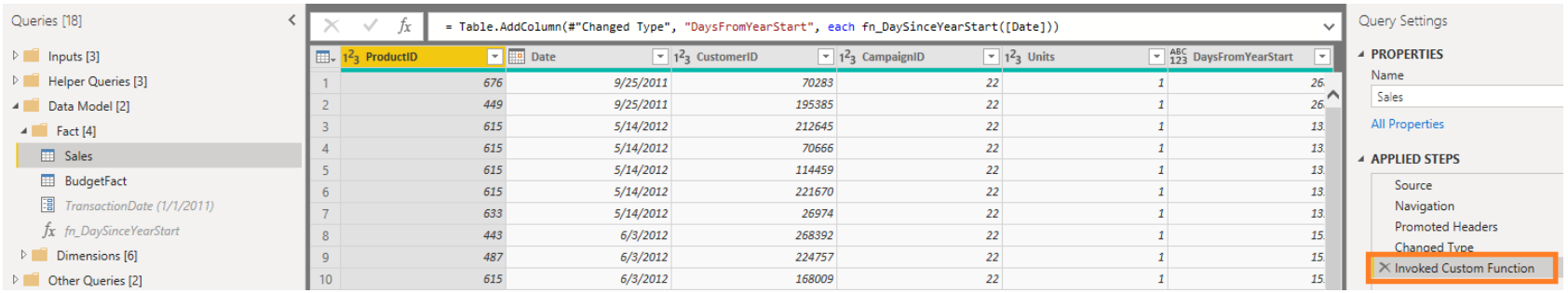


Module 4 Lab: Custom Function

Create a function to get number of days from start of year

- Uses Blank Query
- Populate Advanced Editor of query with text from file provided
- Invoke the custom function from Sales table. This will provide the number of days from start of year for each transaction

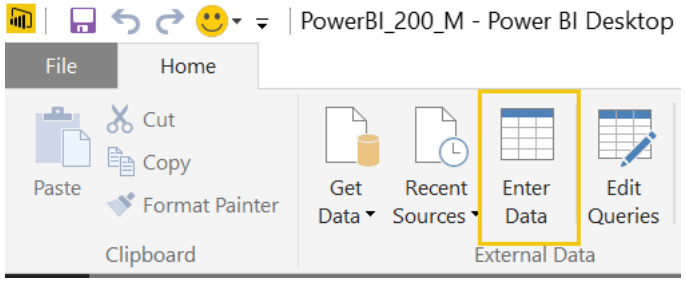




Module 4 Lab: Create a Change Log

Use Enter Data to create a Change Log

- Create a table within query to house the change log
- Name the query **Change Log**
- Update table each time a change is made to the code
- If you decide to load file to data model, then you have the ability to report on current version and last update date



Create Table

Create a table by typing or pasting content.

	Version	Date	Made By	Change	*
1	2.0	12/15/2016	V-Barran (Barbara	Version 2 Updates	
2	2.01	1/3/2017	V-Barran (Barbara	Lab Updates	
*					

Report

Version	Change	Date
2.00	Version 2 Updates	12/15/16
2.01	Lab Updates	1/3/17

Module 4a Lab - bonus

1. Create a MEASURE for Total Units Sold
   HINT: The formula will probably use SUM()

2. Create a CALCULATED COLUMN on the fact table that shows product category and campaign traffic channel combined
   Example: Urban, Organic Search

3. It is fairly easy to see that the CALCULATED COLUMN is working. Create some visuals that allow you to confirm that the Total Units Sold MEASURE is working right

MODULE 5:
Working with Functions, DAX CALCULATE, Custom functions

MODULE 5 OBJECTIVES

- Understand working with functions
- Understand the basics of the CALCULATE formula

DAX Foundations

PATH to DAX Expertise

Evaluation Contexts

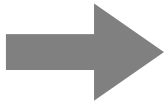
CALCULATE

Calculated Columns and Measures

CALCULATE

Why is CALCULATE Useful?

You create a report of
breakdown of Sales by Month



Typical Business Question:
Provide a break out of this Sales from Desktop

Month	Total Sales
January	$3,379,202
February	$4,434,793
March	$7,848,903
April	$8,175,811
May	$8,133,443
June	$7,847,091
July	$5,736,090
August	$5,739,110
September	$4,755,394
October	$3,746,354
November	$2,968,954
December	$2,781,997
Total	**$65,547,141**

Month	Total Sales	Desktop Sales
January	$3,379,202	$1,451,782
February	$4,434,793	$1,647,766
March	$7,848,903	$2,619,290
April	$8,175,811	$2,540,481
May	$8,133,443	$2,699,799
June	$7,847,091	$2,381,357
July	$5,736,090	$2,243,771
August	$5,739,110	$2,043,244
September	$4,755,394	$1,633,458
October	$3,746,354	$1,203,403
November	$2,968,954	$866,860
December	$2,781,997	$884,017
Total	**$65,547,141**	**$22,215,229**

Here is how you do it with CALCULATE

[Desktop Sales] = CALCULATE([Total Sales], CampaignDim[Device] = "Desktop")

- Use CALCULATE function to create a Measure which filters down to Desktop Sales

Month	Total Sales	Desktop Sales
January	$3,379,202	$1,451,782
February	$4,434,793	$1,647,766
March	$7,848,903	$2,619,290
April	$8,175,811	$2,540,481
May	$8,133,443	$2,699,799
June	$7,847,091	$2,381,357
July	$5,736,090	$2,243,771
August	$5,739,110	$2,043,244
September	$4,755,394	$1,633,458
October	$3,746,354	$1,203,403
November	$2,968,954	$866,860
December	$2,781,997	$884,017
Total	**$65,547,141**	**$22,215,229**

Anatomy of CALCULATE

CALCULATE(Expression, [Filter 1], [Filter 2].....)

Filter Arguments

- EXPRESSION used as the first parameter is essentially the same as a measure
- CALCULATE works differently from other DAX functions
- The second set of arguments, i.e. the “Filter arguments,” are evaluated and applied first
- Then the Expression is evaluated under new “Filter Context"

CALCULATE – Add Filter

CALCULATE – The Most Important Function in DAX

4 Key functions that CALCULATE can do:

- Add Filter
- Ignore Filter
- Update Filter
- Convert Row Context to Filter Context

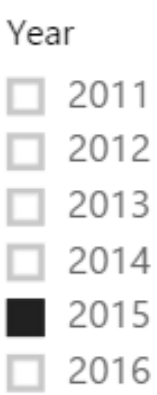
CALCULATE – Add Filter

[Desktop Sales] = CALCULATE([Total Sales], CampaignDim[Device] = "Desktop")

[Tablet Sales] = CALCULATE([Total Sales], CampaignDim[Device] = “Tablet")

[Mobile Sales] = CALCULATE([Total Sales], CampaignDim[Device] = “Mobile")

Month	Total Sales	Desktop Sales	Tablet Sales	Mobile Sales
January	$617,594	$248,081	$113,385	$256,128
February	$846,436	$300,692	$278,821	$266,922
March	$1,382,885	$492,987	$223,870	$334,252
April	$1,512,488	$461,759	$620,238	$404,458
May	$1,589,728	$558,984	$368,121	$511,447
June	$1,402,897	$433,576	$459,494	$313,134
July	$1,122,721	$430,424	$316,463	$375,833
August	$1,222,190	$501,972	$312,637	$404,067
September	$865,028	$308,490	$304,430	$244,142
October	$712,729	$232,041	$246,786	$203,002
November	$562,400	$192,873	$171,329	$169,693
December	$467,428	$148,821	$162,990	$144,679
Total	**$12,304,523**	**$4,310,700**	**$3,578,565**	**$3,627,759**



*When the Device Slicer is selected, only “Total Sales” changes.

CALCULATE – Ignore Filter

CALCULATE – The Most Important Function in DAX

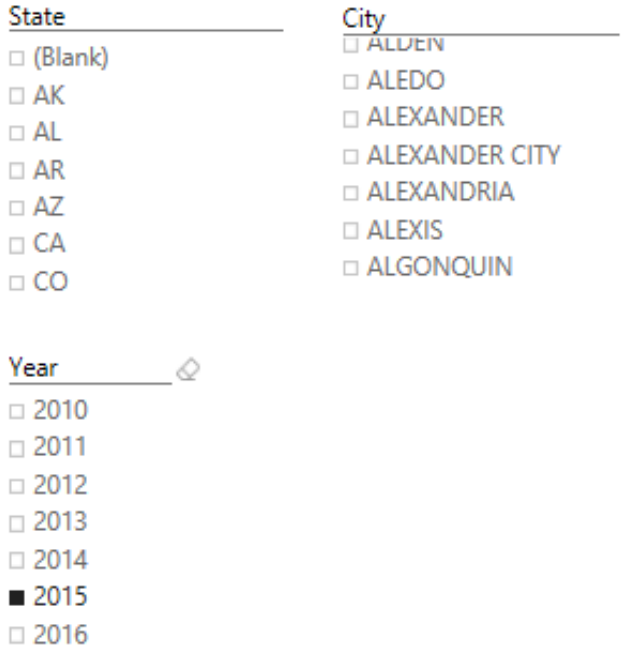
4 Key functions that CALCULATE can do

Add Filter	Ignore Filter	Update Filter	Convert Row Context to Filter Context

CALCULATE – Ignore an Existing Filter

[Total Sales All Geo] = CALCULATE([Total Sales], ALL(GeographyDim))

State	Total Sales	Total Sales All Geo
UT	$482,268	$65,547,141
VA	$1,609,751	$65,547,141
VT	$42,233	$65,547,141
WA	$1,336,132	$65,547,141
WI	$2,297,199	$65,547,141
WV	$599,850	$65,547,141
WY	$351,374	$65,547,141
Total	**$65,547,141**	**$65,547,141**

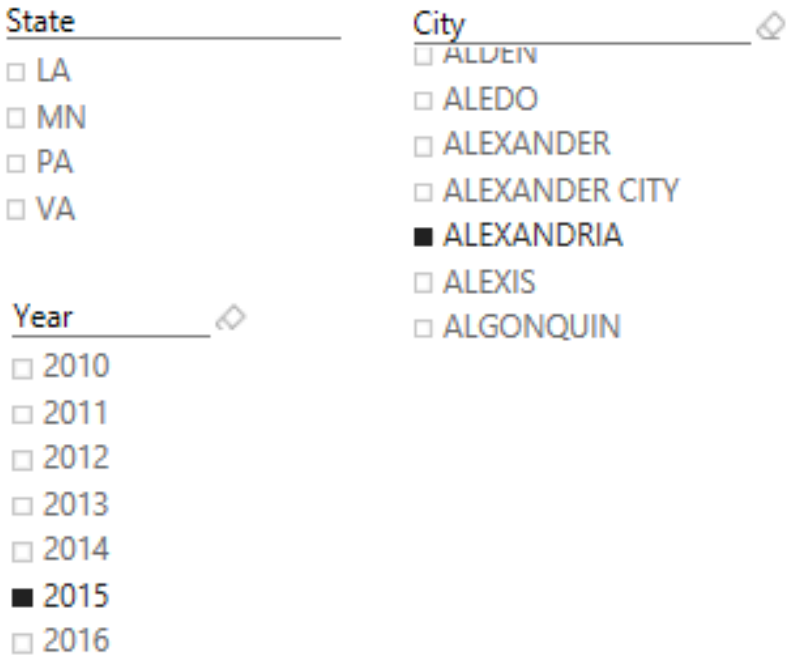


*Ignore filter on ANY column from the GeographyDim table, but allows filters from Year

CALCULATE – Ignore an Existing Filter

[Total Sales All States] = CALCULATE([Total Sales], ALL(GeographyDim[State]))

State	Total Sales	Total Sales All Geo	Total Sales All States
AL	$206	$12,304,523	$15,387
IN	$710	$12,304,523	$15,387
KY	$702	$12,304,523	$15,387
LA	$3,343	$12,304,523	$15,387
MN	$2,545	$12,304,523	$15,387
MO		$12,304,523	$15,387
NE		$12,304,523	$15,387
OH		$12,304,523	$15,387
PA	$283	$12,304,523	$15,387
SD		$12,304,523	$15,387
TN	$144	$12,304,523	$15,387
VA	$7,455	$12,304,523	$15,387
Total	**$15,387**	**$12,304,523**	**$15,387**

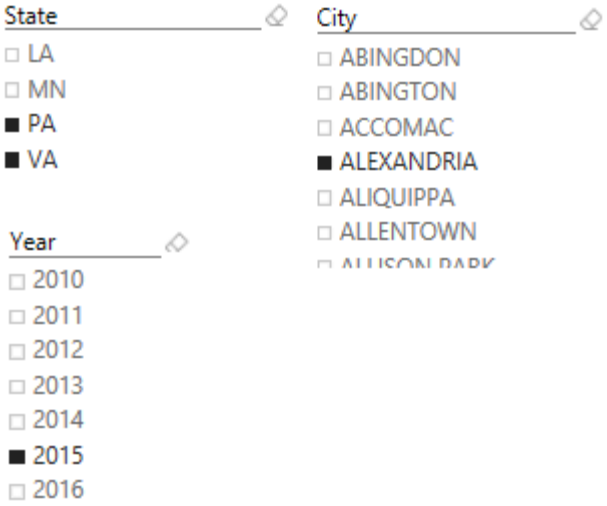


*Ignore filter on the STATE column from the GeographyDim table, but allows filters from Year

CALCULATE – Ignore Existing Filter

[Total Sales All Selected States] = CALCULATE([Total Sales], ALLSELECTED(GeographyDim[State]))

State	Total Sales	Total Sales All Geo	Total Sales All States	Total Sales All Selected States
PA	$283	$12,304,523	$15,387	$7,737
VA	$7,455	$12,304,523	$15,387	$7,737
Total	**$7,737**	**$12,304,523**	**$15,387**	**$7,737**



*Ignore filter on the STATE column from the GeographyDim table, but allows filters from Year

CALCULATE – Update Filter

CALCULATE – The Most Important Function in DAX

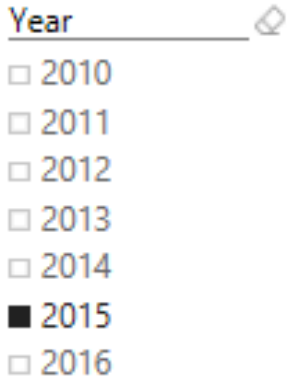
4 Key functions that CALCULATE can do

- Add Filter
- Ignore Filter
- **Update Filter**
- Convert Row Context to Filter Context

CALCULATE – Update Existing Filter

[2014 Sales] = CALCULATE([Total Sales], DateDim[Year] = 2014)

Month	Total Sales	2014 Sales
January	$617,594	$624,956
February	$846,436	$817,549
March	$1,382,885	$1,245,627
April	$1,512,488	$1,400,954
May	$1,589,728	$1,510,563
June	$1,402,897	$1,481,390
July	$1,122,721	$1,281,466
August	$1,222,190	$1,273,948
September	$865,028	$1,201,762
October	$712,729	$916,774
November	$562,400	$714,021
December	$467,428	$575,281
Total	**$12,304,523**	**$13,044,290**



*Ignores filter on the Year Slicer

CALCULATE – Convert Row Context to Filter Context

CALCULATE – The Most Important Function in DAX

4 Key functions that CALCULATE can do

Add Filter	Ignore Filter	Update Filter	Convert Row Context to Filter Context

Let's investigate what we mean by **Filter Context**

DAX Iterator Functions

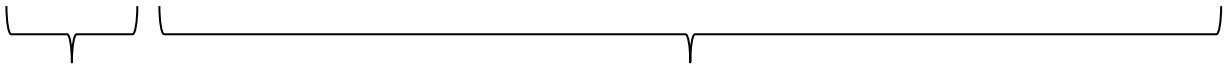
DAX Iterator Functions Take Advantage of Evaluation Context

Iterator Functions

- Creates a *row context* by iterating over a table that you specify
- Ex. SUMX

Table Functions Application – Iterators

```
[COGS] = SUMX(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))
```



Argument 1

Argument 2

Table Functions Application – Iterators

```
[COGS] = SUMX(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))
```

Argument 1

Date	ProductId	Units
1/27/2014	103	1
1/27/2013	65	1
4/5/2013	103	1
10/7/2014	65	1
6/24/2014	65	1
8/22/2013	103	1
11/8/2013	65	1
3/27/2015	103	1
5/26/2013	103	1
12/23/2014	103	1
1/28/2015	103	1
5/21/2015	65	1
7/5/2015	103	1
7/19/2015	65	1

Iterate through each row in Argument 1

Sales

Table Functions Application – Iterators

```
[COGS] = SUMX(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))
```

Argument 2

ProductID	Date	CustomerID	CampaignID	Units
449	7/29/14	128304	1	1
449	7/29/14	89917	1	1
449	7/29/14	128811	1	1
449	7/29/14	59550	1	1
449	7/29/14	207690	1	1
449	7/29/14	121043	1	1

Sales

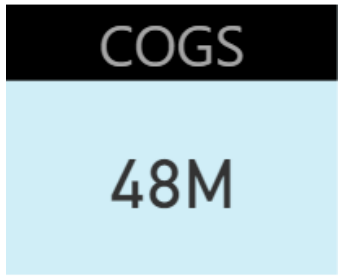
ProductID	Product	Category	Segment	ManufacturerID	Manufacturer	Unit Cost
445	Maximus UM-50	Urban	Moderation	7	VanArsdel	65.13
447	Maximus UM-52	Urban	Moderation	7	VanArsdel	109.59
449	Maximus UM-54	Urban	Moderation	7	VanArsdel	74.73
450	Maximus UM-55	Urban	Moderation	7	VanArsdel	125.32
451	Maximus UM-56	Urban	Moderation	7	VanArsdel	66.49
452	Maximus UM-57	Urban	Moderation	7	VanArsdel	79.71
456	Maximus UM-61	Urban	Moderation	7	VanArsdel	86.23

ProductDim

Row Context in a Measure – Iterator Functions

```
[COGS] = SUMX(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))
```

SUM it up



SUM up list obtained

Iterators

Why Can an Iterator be a Better Approach then a Calculated Column?

- You avoid creating a Calculated Column
- Let us see the impact of a Calculated Column Called COGS on Data model with 100K rows - What if we have 10 M rows?
- Iterators help you avoid several “Intermediate Calculated Columns”

CALCULATE – Converting Row Context to Filter Context (Example 1)

```
Sales velocity Segment = IF(
                          SUMX(RELATEDTABLE(Sales), Sales[Sales Amount])>=200000,
                         “High  Velocity”,
                         “Low Velocity”)
```

```
Sales Velocity (Using CALCULATE) = IF (
                                    CALCULATE(SUM(Sales[Sales Amount])) >= 200000,
                                    "High Velocity",
                                    "Low Velocity")
```

- Another way to do Dynamic Segmentation
- This method does not use Iterators
- Instead it uses CALCULATE to convert Row Context to Filter Context

Other Iterator Functions

AVERAGEX , PRODUCTX, MINX, MAXX– All work the same way as SUMX

RANKX – Works similar to SUMX, but slightly more complex (more options)

Table Functions – Summary and Application

Table functions can be used in 2 ways in Power BI Desktop:

- As an input to another DAX function
  - CALCULATE
  - Iterator functions

- Calculated Tables

CALCULATE is one of the primary places where Table functions are used

CALCULATE – Steps in Evaluating the CALCULATE Function

CALCULATE(Expression, [Filter1], [Filter2].....)

- Step1 : Copy the current filter context
- Step 2: Add new filters if any
- Step 3: Update/ignore existing filters if any
- Step 4: Convert row context to filter context
- Step 5: AND all filter conditions to create new filter context
- Step 6: Evaluate the Expression
- Step 7: Return back to original filter context

Advanced DAX – Time Intelligence

Introducing Time Intelligence – There is an App for that!!

```
[SalesYTD Easier] =

CALCULATE (
    [Total Sales],
    DATESYTD(DateDim[Date])
)
```

- This allows you to write the formula without being a DAX guru!
- Microsoft is continuously improving Time Intelligence functions to make it simple to use

Time Intelligence functions are your friends – They will save you time!

Advanced DAX – Month over Month

```
Total Sales Last Month =

CALCULATE([Total Sales],

    PREVIOUSMONTH(DateDim[Date]))
```

```
MoM =

DIVIDE([Total Sales] - [Total Sales Last Month],

   [Total Sales Last Month])
```

- DAX has several shortcut Time Intelligence functions

Advanced DAX – Monthly Active Users

```
[Monthly Active Users] =

CALCULATE(

    SUMX(VALUES(Sales [CustomerId]),1),

    ALL('DateDim'),

    DATESINPERIOD('DateDim'[Date],

    LASTDATE('DateDim'[Date]), -1, MONTH)

)
```

- Sumx vs DistinctCount of CustomerID
- SUMX can be more performant

Advanced DAX – Time Intelligence

Other Time Intelligence Functions

DATESINPERIOD

DATESYTD

DATESQTD

NEXTMONTH

NEXTYEAR

PREVIOUSYEAR

PREVIOUSMONTH

SAMEPERIODLASTYEAR

PARALLELPERIOD

Pro Tip: Learn about Time Intelligence functions - https://msdn.microsoft.com/en-us/library/ee634763.aspx

KNOWLEDGE CHECK Module 6

- Can I parse advanced DAX formulas?
- What are some standard DAX patterns?
- Which time intelligence functions are built-in to DAX?

KNOWLEDGE CHECK Module 5

- What are the different kinds of evaluation contexts?
- When are filter or a row contexts present?
- Which functions are commonly used to *modify* existing evaluation contexts?

MODULE 5 Lab:

MODULE 6:
Modeling with Power BI & Optimization techniques

MODULE 6 OBJECTIVES

- Gain familiarity with basic data modeling for business scenario
- Learn some best practices for working with Power BI

Data Modeling

- An inefficient model can completely slow down a report, even with very small data volumes

GOALS

- Make the model as small as possible
  There are valid reasons to bend this rule
- Schema supports the analysis
- Relationships are built purposefully and thoughtfully

Move calculations to the source

Scenario

- Many DAX calculated columns with high cardinality

Why is it undesired?

- Calculated columns don’t compress as well as physical columns

Proposed Solution

- Perform calc in Power Query, ideally push down
- Customize source query for non foldable transforms

Remove unused tables and columns

Scenario

- Model contains tables/columns that are not used for reporting/analysis or calculations

Why is it undesired?

- Increases model size
- Increases time to load into memory
- Increases refresh time
- May affect usability

Avoid high precision/cardinality columns

Scenario

- Model contains columns at a higher precision than needed for analysis e.g. datetime in milliseconds, weight to 6 decimal places
- Model contains columns that are highly unique

Why is it undesired?

- Less compression with high precision/cardinality
- Increases time to load into memory
- Increases refresh time

Proposed Solution

- Remove if not needed
- Reduce precision
- Split datetime into date and time

Use integers instead of strings

Why is it undesired?

- Strings use dictionary encoding, integers use run length encoding which is more efficient

Proposed Solution

- Check data types and set to integer if known to be numerical

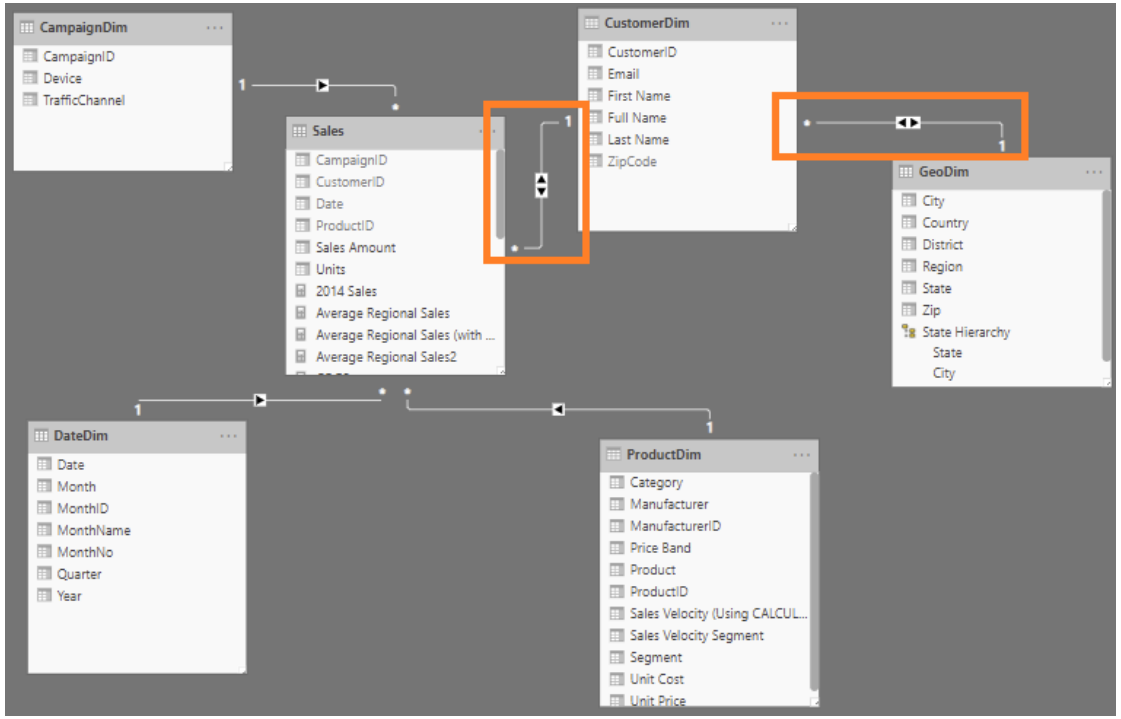
Use integer surrogate keys, pre-sort them

- Power BI compresses rows in segments of millions of rows
- Integers use Run Length Encoding
- Sorting will maximize compression when encoded as it reduces the range of values per segment

Be careful with bi-directional relationships

Scenario

- Most relationships in the model are set to bi-directional



Why is it undesired?

- Applying filters/slicers traverses many relationships and can be slower
- Some filter chains unlikely to add business value

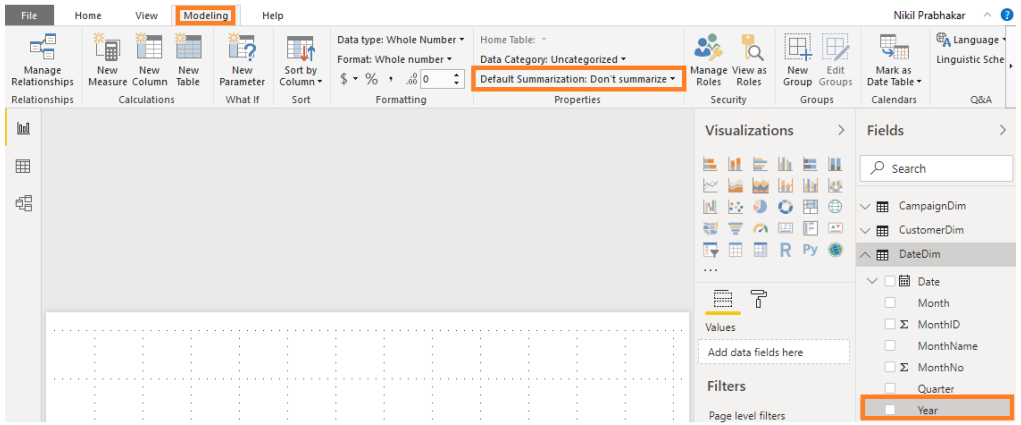
Proposed Solution

- Only use bi-di where the business scenario requires it

Set Default Summarization

Scenario

- Numeric columns in model that are purely informational (e.g. Account ID)
- Default summarization is Sum



Why is it undesired?

- Power BI will try to sum the number when dropped into visuals.
- Detailed tables/matrixes can be slower

Proposed Solution

- Set the default summarization to None

Best Practices – Calculated Columns

Best Practices with DAX Calculated Columns

- Whenever possible, DAX helper columns should be avoided. Each “Helper Column” will consume RAM
- Create a calculated column in the Dim Table as opposed to in the Fact Table
- Move calculated columns to “M” if you can

Use variables instead of repeating measures

- Consider the following DAX expression:

```
Ratio = IF([Total Rows] > 10, SUM(Revenue) /[Total Rows], 0)
```

- Faster DAX:

```
VAR totalRows = [Total Rows];
Ratio = IF(totalRows > 10, SUM(Revenue) / totalRows,0)
```

- In the first expression, since measures are calculated on the fly, the [Total Rows] expression gets calculated twice, first for the condition check and then for the true condition expression
- Instead of calculating the same expression multiple times, the resulting measure value can be stored in a variable and variable reference can be used wherever required

Use DIVIDE() instead of /

- DIVIDE() function has 3rd extra parameter which is returned in case of denominator being zero
- It internally performs check to validate if the denominator is 0
- There is no need to use IF condition along with '/' operator to check for invalid denominator
- DIVIDE() also checks for ISBLANK()
- **Note:** If it is certain that the denominator value would not be 0, then it is better to use '/' operator without any *IF* check since DIVIDE() function would always perform an *IF* check internally

Calculate ratios efficiently

Use (a-b)/b with variables instead of a/b – 1 or a/b*100 – 100

- We can achieve the same performance by using variables and using (a-b)/b to calculate ratio
- If both a and b are blank values, then (a-b)/b would return blank and would be filtered out where as a/b – 1 would return -1 and increase query space

Don’t change blanks to zeros or other values

- Sometimes people replace blanks with zeros or other strings
- Power BI automatically filters out all the rows with blank values from query results
- If the blanks are replaced, the query space is greatly increased

Use SELECTEDVALUE() instead of HASONEVALUE()

- A common pattern is to use HASONEVALUE() to check if there is only one value present for a column after applying slicers and filters and then use VALUES(ColumnName) DAX function to get the single value

- SELECTEDVALUE() performs both the above steps internally and gets the value if there is only one distinct value present for that column or returns blank in case there are multiple values available

Use SELECTEDVALUE() instead of VALUES()

- VALUES() returns error in case it encounters multiple values. Normally, users handle it using Error functions which are bad for performance

- Instead of using that, SELECTEDVALUE() must be used which is a safer function and returns blank in case of multiple values being encountered

Use DISTINCT() and VALUES() functions consistently

- Power BI adds a Blank value to the column in case it finds referential integrity violation
- For direct query, Power BI by default adds blank value to the columns as it does not have a way to check for violations
- Difference :
  - DISTINCT(): Does not return blank which is added due to integrity violation. It includes blank only if it is part of original data
  - VALUES(): It includes blank which is added by Power BI due to referential integrity violation
- The usage of either of the function should be same throughout the whole report
- Power BI recommends to use VALUES() in the whole report if possible and blank value is not an issue

Avoid using IFERROR() and ISERROR()

- IFERROR() and ISERROR() are sometimes used in measures
- These functions force Power BI engine to perform step by step execution of each row to check for errors as there is currently no way which directly states which row returned the error
- FIND() and SEARCH() DAX functions provide an extra parameter which can be passed and is returned in case of the search string not present – avoids use of IFERROR/ISERROR
- Both of this functions are currently also used to check for divide by zero error or along with values to check if more than one values are returned.
- Can be avoided by using the correct DAX functions like DIVIDE() and SELECTEDVALUE() which performs the error check internally and returns the expected results

Use ISBLANK() instead of =BLANK() check

- Use inbuilt function ISBLANK() to check for any blank values instead of using comparison operator “= Blank()”

- ISBLANK() is faster

Use FILTER(ALL(ColumnName))

- To calculate measures ignoring all the filters applied on a column, use All(ColumnName) function along with the FILTER instead of Table or VALUES().

  Eg: CALCULATE([Total Sales], FILTER(ALL(Products[Color]), Color = 'Red'))

- Directly applying filters using expressions and not using FILTER function behaves in the same way as mentioned above and it internally translates to use ALL function in the filter

  Eg: CALCULATE([Total Sales], Products[Color]= 'Red')) ->
  CALCULATE([Total Sales], FILTER(ALL(Products[Color]), Products[Color]= 'Red'))

- It is always better to apply filters at desired column than the whole table

- Always use ALL along with FILTER function if there is no specific need to keep current context

  - https://pbidax.wordpress.com/2016/05/22/simple-filter-in-dax-measures/
  - https://www.sqlbi.com/articles/filter-arguments-in-calculate/

Do not use scalar variables in SUMMARIZE()

- SUMMARIZE() traditionally used to perform grouping of columns and get the resulting aggregations along with it
- It is recommended to use SUMMARIZECOLUMNS() function which is a newer more optimized version
- SUMMARIZE function should only be used to get just the grouped elements of a table without any measures/aggregations associated with it.

  E.g. SUMMARIZE(Table, Column1, Column2)

Avoid using ADDCOLUMNS() in measure expressions

- Measures are calculated in iterative manner by default
- If measure definitions use iterative functions like AddColumns, it create nested iteration which downgrades the performance

Avoid string manipulation in measures

- Slows down measures
- Work is done in calculation engine

KNOWLEDGE CHECK DAX Best Practice

- Which of these are best practice ?
  - isBlank() or comparison operation =Blank()
  - SELECTEDVALUE() or HASONEVALUE()
  - DIVIDE() or IFERROR()
  - Using variables or repeating calculations

Consider subsets for very large models

Scenario

- Large model – hundreds of tables and tens of GB
- Large high grain fact tables – millions to billions

Why is it undesired?

- Aggregating/measures across large facts can affect performance
- Large models become harder to maintain and use ad-hoc

Proposed Solution

- Consider aggregations and composite models features
- Build manual summary tables with smart measures
- Create smaller models for the most common business cases

Designing good data models

Key takeaways to design a good Power BI Desktop data model

- **RAM is precious !!!!!**

Some Tips and tricks to save RAM and increase speed of model

- If a fact table contains an ID field which is unique for each record, **remove it**
  - Ex. Transaction ID

- **Sort columns** before bringing them into a Power BI data model

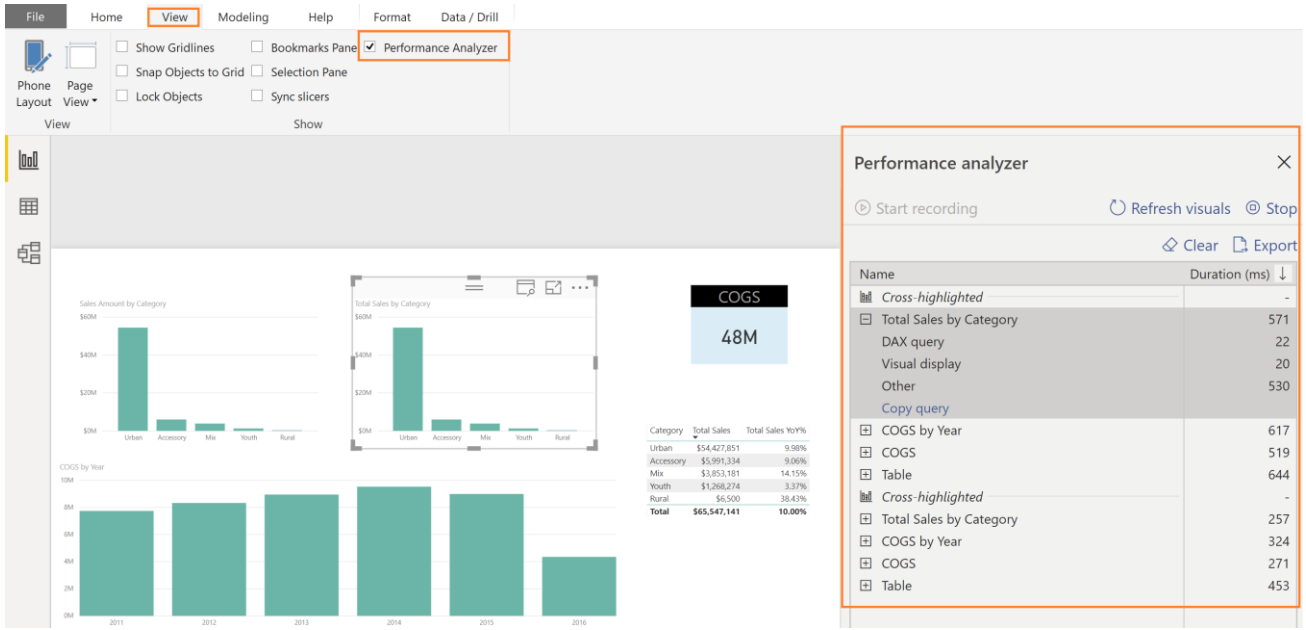
- The DateTime data type is usually not needed, unless you are specifically using the Time component
  - ➢ If you really need Time, **try splitting Date & Time** into two columns - Reduces # of unique values

Star Schema – Good for most Data Models

Performance Analyzer

Using Performance Analyzer:

- You will know how each of your report elements, such as visuals and DAX formulas, are performing
- You can see and record logs that measure how each of your report elements performs when users interact with them, and which aspects of their performance are most (or least) resource intensive



KNOWLEDGE CHECK Module 6

- Which of these help with compression of data and performance?
  - Using integers instead of strings
  - Using high cardinality columns
  - Move calculations to data source
  - Remove unused tables and columns

Module 6 Lab – Basic Best Practices

MODULE 7:
Are we there yet?
Summary Lab and Surveys

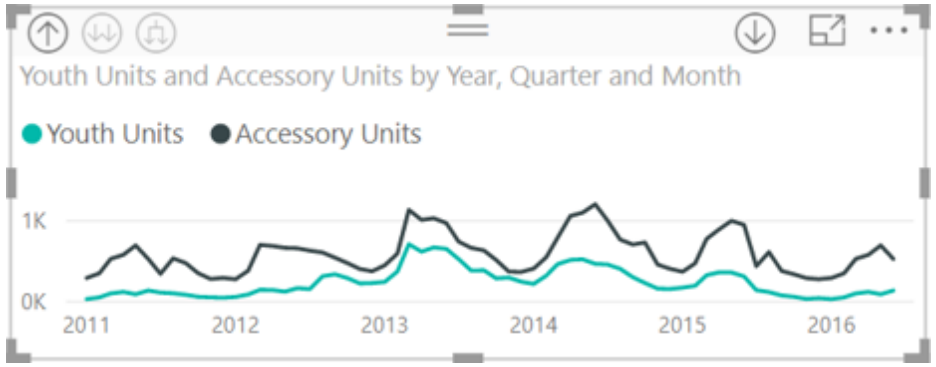
Module 7 Lab

Create a report for the VP in charge of the Youth and Accessory Segments

1. Include a table visualization showing total units sold in the Youth Segment, Accessory Segment, and all other segments; by Campaign Device

Device	Total Units	Youth Units	Accessory Units	Rest of Company Units
Deskop	10806	222	653	9931
Desktop	218680	4933	12412	201335
Mobile	198014	4427	11420	182167
Paper	40524	908	2376	37240
Tablet	207344	5151	12308	189885
Total	**675368**	**15641**	**39169**	**620558**

2. Include a line chart showing total units sold in Youth and Accessory Segments by month



3. BONUS: Use the Unit Cost and Unit Price from the ProductDim table to calculate Sales Amount, Cost of Goods Sold, Profit and build some visuals around them

Power BI Support Resources

Contact Support

Report Errors, Issues – Support.PowerBI.com

Resources *use presentation mode to click the hyperlinks*

- *Community.PowerBI.com – Community Forum*
- *Data Stories Gallery – Get inspired with Data Stories by other Power BI users*
- *R-Visuals Gallery – Get inspired by others use of R for analyzing their data*
- *Visuals.PowerBI.com – Custom PBI visuals and R visuals you can download and use in your story*
- *Power BI Blog - weekly updates*
- *User Voice for Power BI – Vote on (or submit) your favorite new ideas for Power BI*
- *Issues.PowerBI.Com – log issues with the community*
- *Guided Learning Self Service Power BI training*
- *DAX Formula Language – syntax for DAX*
- *DAX Patterns – Great website to learn new patterns for the DAX Language*
- *Power Query Formula Language – syntax for the "Query" language*

Instructors:

Questions?

Appendix

KNOWLEDGE CHECK Module 3

- When is Calculated Column Evaluated?
- What is Default Summarization?
- When is a Measure Evaluated?
- When to use Measures and Calculated Columns?

KNOWLEDGE CHECK ANSWERS Module 1

- What is a *data model* in the context of Power BI?
  - *A data model is a collection of tables and relationships*

- What are some advantages of a star schema over a flat or denormalized model?
  - *Dimension tables save space by reducing the amount of data that needs to be repeated over and over in every row*
  - *Relationships between tables can be leveraged for more complex measures*

- How might you improve the performance of a Power BI model?
  - *Try using a star schema instead of a flat or denormalized model*
  - *Remove unnecessary columns*
  - *Set appropriate data types*

- How does Power BI store DateTime information? What are some consequences of this?
  - *DateTime information is stored as a floating-point decimal number. This means that datetimes are very precise but not very efficient to store.*

KNOWLEDGE CHECK Module 2 – post Lab questions

Which relationships were already detected by Power BI?
Which table is the Fact Table?
Which are the dimensions tables?
Which type and direction are you going to choose?
Which type of schema have you created?

KNOWLEDGE CHECK ANSWERS Module 2

- Which of these help with compression of data and performance?
  - *All of the listed options help with compression of data and performance.*

KNOWLEDGE CHECK ANSWERS Module 3

- When is Calculated Column Evaluated?
  - *At the time of data load/data refresh.*

- What is Default Summarization?
  - *A default summarization is an implicit measure created in the background when you put a numeric field on a visualization. The function used (sum/max/min/avg/…) is based on the numeric field's default summarization setting.*

- When is a Measure Evaluated?
  - *At render time.*

- When to use Measures and Calculated Columns?
  - *It depends ☺. Calculated columns are useful when each row of data should be independently considered (although measures can do this too!) and the result won't change until the next data refresh. Measures should be used everywhere else.*

KNOWLEDGE CHECK ANSWERS Module 5

- What are the different kinds of evaluation contexts?
  - *Filter context and row context*

- When are filter or a row contexts present?
  - *Row contexts are present in iterator functions and calculated column evaluations. Filter contexts are present in pivot tables and other visualizations.*

- Which functions are commonly used to *modify* existing evaluation contexts?
  - *CALCULATE, ALL, etc.*

KNOWLEDGE CHECK ANSWERS Module 6

- Can I parse advanced DAX formulas?
  - *Yes I can!*
- What are some standard DAX patterns?
  - *CALCULATE(…)*
- Which time intelligence functions are built-in to DAX?
  - *Lots of them…YTD, FY, previous month, etc*

KNOWLEDGE CHECK ANSWERS Module 7

- Which of these are best practice ?
  - isBlank() or comparison operation =Blank()
    - *Using isBlank()*
  - SELECTEDVALUE() or HASONEVALUE()
    - *Using SELECTEDVALUE()*
  - DIVIDE() or IFERROR()
    - *Using DIVIDE()*
  - Using variables or repeating calculations
    - *Using variables*

Appendix 2 Power BI Compressing data mechanism

Aggregations

- Using aggregations in Power BI enables interactive analysis over big data. Advantages of aggregation are:
  - Query performance over big data
  - Data refresh optimization
  - Achieve balanced architectures

Compressing Data – Dictionary Encoding

How Power BI Compresses Data – Dictionary Encoding

Sale Id	Color	Sales Amount
390a30e0-dc37	Red	$10
390a30e1-dc37	Green	$25
390a30e2-dc37	Red	$35
390a30e3-dc37	Red	$15
390a30e4-dc37	Red	$25
390a30e5-dc37	Green	$30
390a30e6-dc37	Blue	$10
390a30e7-dc37	Blue	$12
390a30e8-dc37	Blue	$15
390a57f0-dc37	Blue	$18
390a57f1-dc37	Green	$25

Red = 1 **Green = 2** **Blue = 3**

- Create a Dictionary to create an integer value for text string
- Storing 1,2,3 instead of “Red”, “Green”, “Blue” saves memory
- **Dictionary encoding is powerful when there are few unique values** in a column
  - Ex. Color column – Good for dictionary encoding
  - Ex. Sale ID – Bad for dictionary encoding

Compressing Data – Run Length Encoding

How Power BI Compresses Data – Run Length Encoding

Sale Id	Color	Sales Amount	
390a30e0-dc37	Red	$10	1
390a30e1-dc37	Green	$25	2
390a30e2-dc37	Red	$35	1
390a30e3-dc37	Red	$15	1
390a30e4-dc37	Red	$25	1
390a30e5-dc37	Green	$30	2
390a30e6-dc37	Blue	$10	3
390a30e7-dc37	Blue	$12	3
390a30e8-dc37	Blue	$15	3
390a57f0-dc37	Blue	$18	3
390a57f1-dc37	Green	$25	2

Run Length Encoding in Power BI

Where **Red** = **1** **Green** = **2** **Blue** = **3**

- Instead of storing - 1, 2, 1, 1, 1, 2, 3, 3, 3, 3, 2
- It Stores:

1 – 1	(1 instance of One)
1 – 2	(1 instance of Two)
3 – 1	(3 instances of One)
1 – 2	(1 instance of Two)
4 – 3	(4 instances of Three)
1 - 2	(1 instance of Two)

- **Run length encoding** is **very powerful when data is sorted** well and has few unique values

Compression

Practical Example of Compression

Dashboard in a Day Class Data

Sales Fact	420.0 MB
Dimensions	4.4 MB
Int'l Sales	32.4 MB
Total Data	**456.8 MB**

Queries ONLY – No Data Loaded

Query Metadata	**113 KB**

DIAD Complete Data Model

Data Model	**59.4 MB**

Almost 8X Compression!!

Appendix 3 Advanced DAX & Time Intelligence

Advanced DAX

Before we get to Time Intelligence - Let us apply all of the DAX techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date])
    )
)
```

Date	Year	Total Sales	SalesYTD
January 1, 2011	2011	$551	$551
January 2, 2011	2011	$7,366	$7,917
January 3, 2011	2011	$1,873	$9,790
January 4, 2011	2011	$10,113	$19,902
January 5, 2011	2011	$9,660	$29,562
January 6, 2011	2011	$14,450	$44,012
January 7, 2011	2011	$7,883	$51,895
January 8, 2011	2011	$11,793	$63,688
January 9, 2011	2011	$10,341	$74,029
January 10, 2011	2011	$1,374	$75,404
January 11, 2011	2011	$10,950	$86,353
January 12, 2011	2011	$20,217	$106,570
January 13, 2011	2011	$16,812	$123,382
January 14, 2011	2011	$15,215	$138,597
January 15, 2011	2011	$15,841	$154,438
January 16, 2011	2011	$14,391	$168,828
January 17, 2011	2011	$2,423	$171,252
January 18, 2011	2011	$15,712	$186,964
January 19, 2011	2011	$23,557	$210,521
January 20, 2011	2011	$20,912	$231,434

Let us take a super complicated DAX statement and break it down and understand what it means

Before we get to Time Intelligence - Let us apply all of the DAX techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX( DateDim[Date] )
    )
)
```

- In a CALCULATE statement Filter arguments are evaluated first
- The Filter in this case comes from a FILTER function
- FILTER function is an iterator

Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX( DateDim[Date] )
    )
)
```

- In a FILTER statement the input Table is evaluated first
- ALL statement means take all DateDim

Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Iterate through each row in DateDim
- Check for condition based on row context and filter context

Pro Tip: if you need to concatenate two conditions with an **AND** use **&&** for and **OR** use **||**

Let us apply all of data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Now you have a FILTERED list of dates
- Use this to update the filter context (since it is in a CALCULATE statement)

Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Use updated FILTER context to evaluate ‘Total Sales’