

# Working with Power BI

Version 1.0

This courseware is to be distributed only to accompany instructor led training, it is not designed to stand on its own as an effective learning resource.

Students who like to take notes are encouraged to print this guild to write on during lectures and demonstrations.

# Table Of Contents

<b>MODULE 1 BUSINESS INTELLIGENCE .....</b>	<b>4</b>
SECTION 1.1 BI OVERVIEW .....	5
SECTION 1.2 BI COMPONENTS .....	19
SECTION 1.3 EXERCISES.....	29
EXERCISE 1.1 : BRAINSTORMING OPPORTUNITIES FOR BI WITHIN YOUR ORGANIZATION .....	30
<b>MODULE 2 POWER BI.....</b>	<b>31</b>
SECTION 2.1 POWER BI ECOSYSTEM.....	32
SECTION 2.2 POWER BI TOUR .....	52
SECTION 2.3 EXERCISES.....	71
EXERCISE 2.1 EXPLORING POWER BI SAMPLE PROJECTS .....	72
<b>MODULE 3 DATA MODELING .....</b>	<b>74</b>
SECTION 3.1 DATA MODELING OVERVIEW.....	75
SECTION 3.2 FLAT FILES .....	81
SECTION 3.3 RELATIONAL DATABASES.....	85
SECTION 3.4 OLTP DATABASE DESIGN .....	98
SECTION 3.5 OLAP DATABASE DESIGN.....	104
SECTION 3.6 DATA CUBE MODELING.....	132
SECTION 3.7 POWER BI MODELING .....	135
SECTION 3.8 DATA TRANSFORMATION.....	159
SECTION 3.9 EXERCISES.....	169
EXERCISE 3.1 NORTHWIND MODELING – EXPLORING THE NORTHWIND DATABASE.....	170
<b>MODULE 4 POWER QUERY GUI EXPERIENCE .....</b>	<b>176</b>
SECTION 4.1 POWER QUERY .....	177
SECTION 4.2 POWER QUERY EDITOR TOUR.....	187
SECTION 4.3 SIMPLE TRANSFORMATIONS .....	194
SECTION 4.4 EXERCISES.....	201
EXERCISE 4.1 NORTHWIND MODELING – IMPORTING AND TRANSFORMING OUR DATA WITH POWER QUERY .....	202
EXERCISE 4.2 NORTHWIND MODELING – MODELING AND VISUALIZING OUR NEW DATA .....	206
EXERCISE 4.3 NORTHWIND MODELING – A FIRST GLANCE AT RELATIONSHIPS AND COLLAPSING CATEGORIES INTO PRODUCTS ..	207

<b>MODULE 5 POWER QUERY FORMULA LANGUAGE (M) .....</b>	<b>208</b>
SECTION 5.1 INTRODUCING M .....	209
SECTION 5.2 DISSECTING A LET EXPRESSION.....	214
SECTION 5.3 M LANGUAGE LEXICON .....	224
SECTION 5.4 M LANGUAGE REFERENCE .....	243
EXERCISE 5.1 CONTOSO M – PART 1 – SHAPING FLAT FILE DATA WITH M.....	246
EXERCISE 5.2 CONTOSO M – PART 2 – BUILDING DIMENSIONS .....	249
EXERCISE 5.3 CONTOSO M – PART 3 – RELATIONSHIPS AND REPORTING .....	254
<b>MODULE 6 INTRODUCING DAX .....</b>	<b>256</b>
SECTION 6.1 DEFINING DAX.....	257
SECTION 6.2 DAX SYNTAX FUNDAMENTALS.....	268
SECTION 6.3 CALCULATED COLUMNS.....	295
SECTION 6.4 MEASURES.....	304
SECTION 6.5 EXERCISES.....	312
EXERCISE 6.1 ONLINE REVIEW OF FUNCTION REFERENCE .....	313
EXERCISE 6.2 NORTHWIND MODELING – SIMPLE CALCULATED COLUMNS .....	314
EXERCISE 6.3 NORTHWIND MODELING – WORKING WITH PERCENTAGES .....	315
EXERCISE 6.4 NORTHWIND MODELING – CONTEXT BASED AGE CALCULATIONS .....	318
EXERCISE 6.5 NORTHWIND MODELING – FLATTENING MANY-TO-MANY RELATIONSHIPS .....	320
<b>MODULE 7 ADVANCED DAX CONCEPTS.....</b>	<b>321</b>
SECTION 7.1 FILTERING.....	322
SECTION 7.2 CALCULATED TABLES.....	331
SECTION 7.3 ITERATORS .....	334
SECTION 7.4 ROW LEVEL SECURITY .....	339
SECTION 7.5 EXERCISES.....	343
EXERCISE 7.1 NORTHWIND MODELING – RESOLVING GRANULARITY ISSUES .....	344
EXERCISE 7.2 NORTHWIND MODELING – ROW LEVEL SECURITY.....	347
<b>MODULE 8 DATE AND TIME INTELLIGENCE.....</b>	<b>348</b>
SECTION 8.1 DATES AND DATE TABLES .....	349
SECTION 8.2 DAX TIME INTELLIGENCE FUNCTIONS.....	366
SECTION 8.3 EXERCISES.....	371
EXERCISE 8.1 NORTHWIND MODELING – COMPUTING A DATE TABLE .....	372
EXERCISE 8.2 NORTHWIND MODELING – USEFUL DATE HIERARCHY .....	375
EXERCISE 8.3 NORTHWIND MODELING – RUNNING TOTALS.....	377
EXERCISE 8.4 NORTHWIND MODELING – PARALLEL PERIOD COMPARISON.....	378

# Module 1

## Business Intelligence

## Section 1.1

### BI Overview

## 1.1.1 BI Overview

---

- Helps organizations analyze historical and current data, so they can quickly uncover actionable insights for making strategic decisions
- Tools make this possible by processing large data sets across multiple sources presenting findings in visual formats that are easy to understand and share

## 1.1.2 History of Business Intelligence

---

- The earliest known use of the term *Business Intelligence* is in Richard Millar Devens' Cyclopædia of Commercial and Business Anecdotes (1865)
  - ◊ Devens used the term to describe how the banker Sir Henry Furnese gained profit by receiving and acting upon information about his environment, prior to his competitors
- In 1989, Howard Dresner (later a Gartner analyst) proposed *business intelligence* as an umbrella term to describe “concepts and methods to improve business decision making by using fact-based support systems.”
  - ◊ It was not until the late 1990s that this usage became widespread
- BI has been criticized as a marketing buzzword in the context of the “big data” surge
  - ◊ Some argue that BI is merely an evolution of business reporting together with the advent of increasingly powerful and easy-to-use data analysis tools

## 1.1.3 BI Evolution

---

- Although large companies have been using BI techniques for decades, numerous factors have accelerated its use in nearly every business
  - ◊ Businesses are using more software, collecting more information
  - ◊ Hard drives keep getting larger and less expensive
  - ◊ Computers keep getting faster
  - ◊ Tooling has matured as more businesses embrace spending money on the BI product ecosystem
- Tooling has become less expensive

## 1.1.4 BI for different scopes

---

- BI can be applied at different levels within an organization
  - ◊ Organizational BI
  - ◊ Team BI
  - ◊ Personal BI

## 1.1.5 Organizational BI

---

- At the organizational level, it's common to have entire IT teams designated to analyzing business information to make big decisions or influence strategy
  - ◊ It might make sense for a logistics company to compare staffing costs and labor hours with shipping numbers, to help make decisions about how to price an upcoming opportunity
- Power BI can be used for this, although there are still advantages to using traditional BI solutions

## 1.1.6 Team BI

---

- At a team level or within a department, BI can be used to identify performance within systems that are unique to that team
  - ◊ Customer Service might use Power BI to analyze service ticket statistics, to help show upper management how performance has been reduced by 30% after losing a recent employee, helping to raise budget priority
- Team level BI can be used to help managers within the team gamify their performance with team specific KPIs

## 1.1.7 Personal BI

---

- Before Products like Power BI, the feasibility of Personal BI was limited
- Using Excel or another spreadsheet was about as advanced as was practical
- Examples:
  - ◊ Creating a personal dashboard by combining various sources of tasks
    - \* Unread emails
    - \* Project Management Software
    - \* Files in a Folder
    - \* Numbers from a Customer Relationship Management Portal

## 1.1.8 BI is a Process

---

- Analyzing Data is the goal, but there is a lot of thought and work that needs to go into getting data to a point where it can be analyzed
- Steps might include:
  - ◊ Collecting data
  - ◊ Cleaning data
  - ◊ Consolidating data
  - ◊ Augmenting data
  - ◊ Shaping data
  - ◊ Visualizing data
- Power BI is an all-in-one ecosystem of tooling to help with this process

## 1.1.9 Power BI Ecosystem

---

- The Power BI Ecosystem consists of multiple products, tools, technologies, and languages
  - ◊ Power BI Desktop
  - ◊ Power BI Service
  - ◊ Power BI Mobile
  - ◊ Power BI Embedded
  - ◊ SQL Server Analysis Services
  - ◊ Power BI Report Server
  - ◊ Power BI On-Premises Data Gateway
  - ◊ xVelocity Analytical Engine
  - ◊ Power Query Editor
  - ◊ Power Query Formula Language M
  - ◊ DAX - Data Analytics eXpression Language
  - ◊ Direct Query
  - ◊ Connect Live
- We will discuss these in more detail in a later module

## 1.1.10 Power BI is an Industry Disruptor

---

- Microsoft Power BI is an industry disruptor
  - ◊ Companies are saving thousands of dollars switching from expensive suites and are finding that the tooling brings them considerably more value
- More Powerful
  - ◊ All of the tooling that you need to ingest and prepare data for analysis
  - ◊ Interactive visualizations enable you to explore data interactively and fast - without waiting minutes for a new query to execute each time you navigate through a visualization
- Less Expensive
  - ◊ Power BI Desktop is FREE
  - ◊ Power BI Service is hard to beat with subscriptions to host and share interactive data models and reports starting at \$10/user/month

## 1.1.11 Not a replacement for enterprise BI

---

- OLAP Databases collect data that changes over time
- OLAP Databases help to centralize BI efforts
- Data Cubes are hosted from disk, which enables terabyte sized data
- Data Lakes can store raw data from numerous sources that can then be shaped and queried later

## 1.1.12 BI has become essential

---

- BI used to give businesses an edge
- It was a secret weapon, used by the most successful businesses to get ahead
- Now that everyone is embracing it, businesses have little chance to survive without making smarter, more informed decisions

## 1.1.13 Discussion

---

- What are some ways that you are already using Business Intelligence at your organization?
- What are some of the tools that you are currently using?
- What are some of the BI problems that you were hoping to solve when you decided to take this class?
- What are some companies, other than your own, that have given themselves an edge by building Business Intelligence into their company culture?

## Section 1.2

### BI Components

## 1.2.1 The source of Digital Information

---

- In business, the source of information is usually an event taking place
- Any number of things can happen, and frequently when something happens, a record of that thing is made digitally
- Usually, records are made with software

## 1.2.2 Common Business Software Types

---

- Customer Relationship Management
- Point-Of-Sale Retail Software
- E-Commerce Retail Software
- Financial Accounting Software
- Warehouse Management Systems
- Enterprise Resource Planning
- Content Management Systems
- Employee Timecard Software
- HR Hiring Software

## 1.2.3 How software stores data

---

- Most software products use industry standards or popular products to persist the information they are collecting
  - ◊ This gives us some common options when it comes to making that information available to BI
- How we might access the data may vary

## 1.2.4 Getting the data

---

- What options are available differ by software product
- Common options include:
  - ◊ Querying the database directly
  - ◊ Export data using reporting features
  - ◊ Access an applications API

## 1.2.5 Querying the Database Directly

---

- Frequently we can write queries directly against a database schema

- ◊ Pros:

- \* Data is usually going to be real-time.
    - \* When possible, sometimes the easiest option.
    - \* Query Languages like SQL can perform a lot of the transformation needed
    - \* The Database has clearly defined data types and precision

- ◊ Cons:

- \* Can slow down production systems
    - \* Not always clear what fields represent
    - \* Software Upgrades can change the database schema

## 1.2.6 Export Data using Reporting Features

---

- Many software applications have the ability to Export data as CSV Files, Text Files, Flat Files, Excel Files, or as structured JSON or XML Files
  - ◊ Pros:
    - \* Queries to assemble data from various sources and include important fields have already been designed
    - \* Usually resilient to software updates
  - ◊ Cons:
    - \* Sometimes a manual process
    - \* Files can become large and cumbersome to keep and work with
    - \* Many files do not store meta-data like column names, data types, size limits, and other important schema detail

## 1.2.7 Access data through an application's API

---

- Lets you access data through the application itself
- Difference Protocols exist
  - ◊ REST
  - ◊ GraphQL
  - ◊ SOAP
  - ◊ ODATA
  - ◊ Many more...
- Common when interacting with Software-as-a-Service
  - ◊ Pros
    - \* Accessing Data in a way that developers of a specific product designed for
    - \* Resilient to Changes made to the software and its database over time
    - \* Data is usually real-time
  - ◊ Cons
    - \* Not all APIs are created equally
    - \* These can sometimes be slower with large datasets
    - \* There are sometimes query quotas in place

## 1.2.8 Modeling Data for Analysis

---

- After you have your data, it needs to be cleaned, merged, augmented, and re-shaped for easy analysis
- This process is called modeling, and will be covered in detail in a future module

## 1.2.9 Section Discussion

---

- Where does your business information originate?
  - ◊ What events take place that might translate to a record, or generate a document?
- What software keeps records of various events?
- How do you access that information for analysis?

## Section 1.3

## Exercises

## Exercise 1.1 : Brainstorming Opportunities for BI within your organization

---

1. Pair up with other individuals within your organization.
2. Brainstorm a few ways that BI might be used within your organization at each of the following three levels.
  - Organization
  - Team (Consider multiple teams or departments)
  - Personal (Both for yourselves, and other roles within your organization)
3. Highlight some of your ideas for group discussion.

# **Module 2**

# **Power BI**

## Section 2.1

### Power BI Ecosystem

## 2.1.1 Ecosystem

---

- The Power BI Ecosystem consists of multiple programs, tools, technologies, and languages
  - ◊ Power BI Desktop
  - ◊ Power Query Editor
  - ◊ Power BI Service
  - ◊ SQL Server Analysis Services
  - ◊ Power BI Report Server
  - ◊ Power BI Mobile
  - ◊ Power BI Embedded
  - ◊ Power BI On-Premises Data Gateway
  - ◊ xVelocity Analytical Engine
  - ◊ Data Analysis eXpression Language
  - ◊ Direct Query
  - ◊ Connect Live

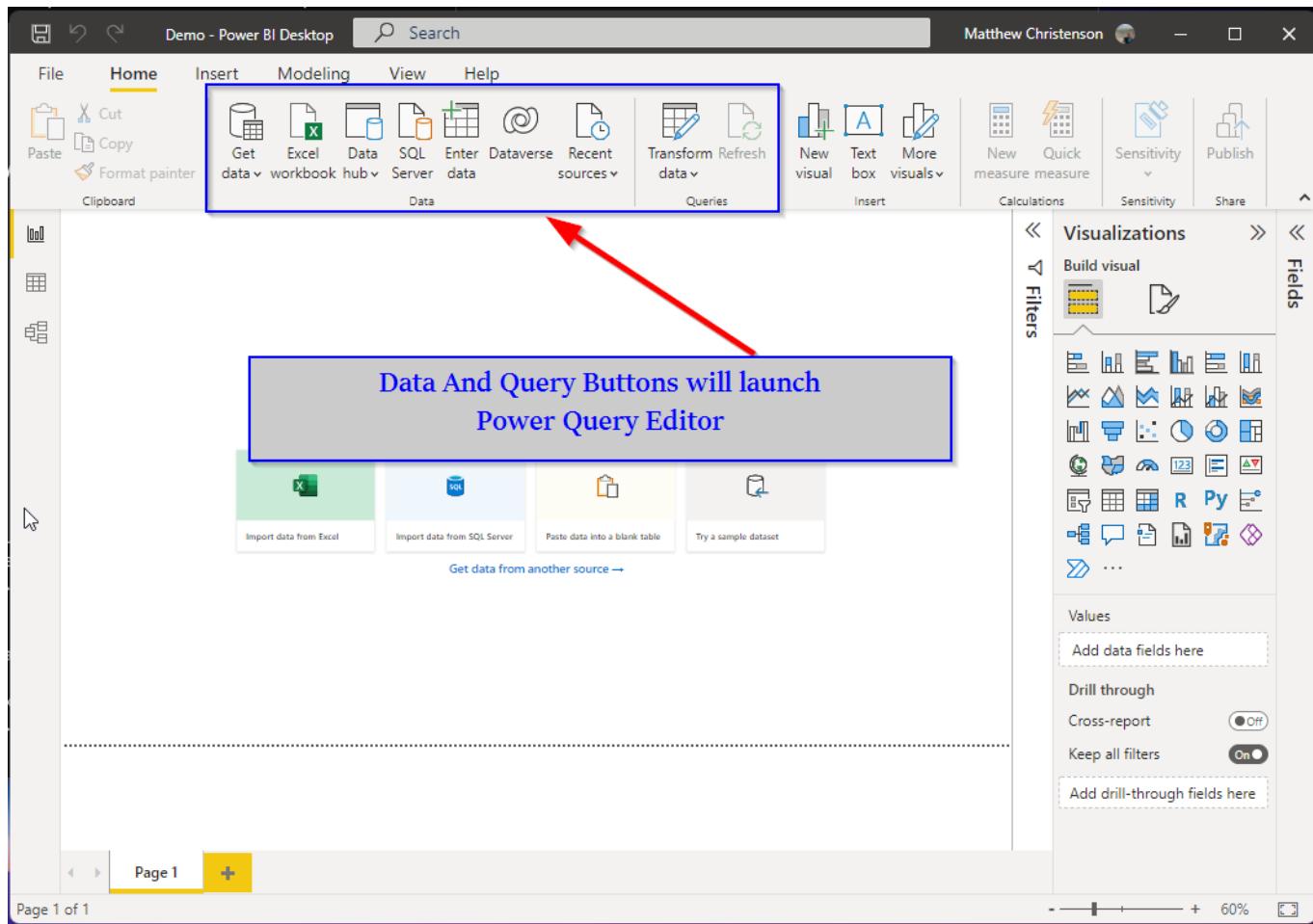
## 2.1.2 Power BI Desktop

---

- Free Powerful Desktop Application for working with Power BI Data Models and authoring reports.
- Hosts the data in-memory using a powerful database engine called xVelocity
- Creates .pbix files that can be shared with others
- Data within the .pbix files needs to be “refreshed” or it will go stale
  - ◊ Can be problematic when sharing files, as users often have different credentials to remote data sources

## 2.1.3 Power Query Editor

- Power Query Editor is Embedded into Power BI, and represents tooling used to bring data into the Power BI Project to be modeled



## 2.1.4 Power BI Service

---

- Pro and Premium Paid Subscriptions to the Power BI Service, a website-based service that makes sharing Power BI Reports with others easier
- We will be exploring the Power BI Service in a later module, but the majority of this course will be focused on authoring within Power BI Desktop

## 2.1.5 SQL Server Analysis Services

---

- In Multidimensional Mode, hosts Data Cubes which are a more additional approach to Business Intelligence
  - ◊ Primary advantage is that these do not fit into memory so their size can be more significant
- In Tabular Mode, can be used to host xVelocity databases
  - ◊ This is the same as what Power BI Desktop and Power BI Service use to host the ‘model’
  - ◊ You can license and run this yourself, keeping your data behind your firewall but available to many within your organization
- Use the “Connect Live” option in Power BI desktop to connect to a hosted model, this offloads the storage and most of the processing to the SSAS server

## 2.1.6 Power BI Report Server

---

- Publish Power BI reports on-premises
- Requires Premium Product Licensing, or SQL Server Enterprise
- Enables you to apply governance on your own terms, keeping all your data behind your company's firewall

## 2.1.7 Power BI Mobile

---

- View reports hosted with the Power BI Service in a mobile friendly way
- Connects to Power BI Service, or Power BI Report Server

## 2.1.8 Power BI Embedded

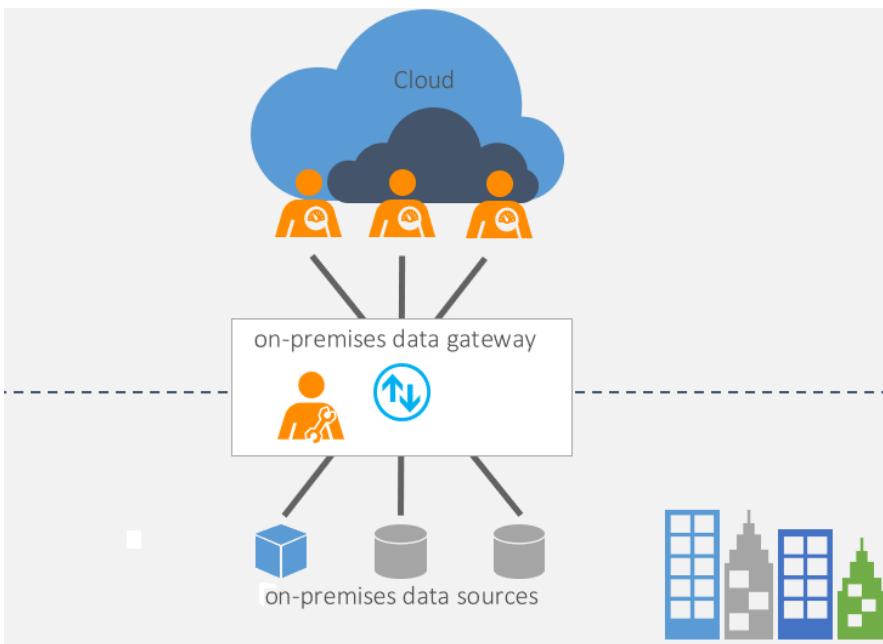
---

- Enables you to embed Power BI visualizations into custom software
- Embed Content for Customers enables the ability to visualize data without logging into a Power BI Account
  - ◊ Requires “Capacity” licensing
- Embed Content for your organization
  - ◊ Either all users will need licensing, or a “Capacity” license will be required

## 2.1.9 Power BI On Premises Data Gateway

---

- Acts as a bridge to provide quick and secure data transfer between on-premises data (data that isn't in the cloud) and several Microsoft Cloud based services
  - ◊ This includes:
    - \* Power BI
    - \* Power Apps
    - \* Power Automate
    - \* Azure Analysis Services
    - \* Azure Logic Apps



Source: <https://docs.microsoft.com/en-us/power-bi/connect-data/media/service-gateway-onprem/on-premises-data-gateway.png>

- Supports a “standard mode” which allows data to be made available to reports shared with multiple Power BI Service users
- Supports a “personal mode” which allows a single user to connect to data that cannot be shared with others
- For more information see:

<https://powerbi.microsoft.com/en-us/gateway/>

## 2.1.10 xVelocity Analytical Engine

---

- Commonly referred to by its codename, VertiPaq
- Known as Power Pivot from the perspective of Excel
- DAX was written specifically as an Expression Language for xVelocity
- In-Memory Columnar database
  - ◊ Stores and hosts the data model
- Used by multiple BI related Microsoft Products
  - ◊ Power BI Desktop
    - \* Hosts your model on the desktop
  - ◊ Power BI Service
    - \* Hosts your model in the cloud
  - ◊ SQL Server Analysis Services Tabular Engine
  - ◊ Power Pivot for Excel

## 2.1.11 Columnar Databases

---

- Columnar databases are ideal for analytical reporting
  - ◊ Computers read and write from disks in large chunks, often 64k bytes at a time
  - ◊ Traditional database engines read and write a row's data grouped together, resulting in unnecessary columns being read and discarded
- A Columnar database stores data grouped by the column it belongs to, instead of the row it belongs to
  - ◊ This is useful in scenarios where you plan to read every row to perform an evaluation - but are only interested in a small subset of the available columns
  - ◊ Such operations perform much less work because it is not reading and then discarding data from columns that it is not interested in

## 2.1.12 Compression in a Columnar Database

---

- Columnar databases allow for powerful compression
  - ◊ Run-length encoding is used to compress sorted data
    - \* Adding 1,000,000 values of 1,000 distinct numbers might become 1,000 multiplication problems and 1,000 addition problems
    - \* 2000 operations is considerably less than 1,000,000
  - ◊ Dictionary encoding is used to compress unsorted data that has a lot of repeated values
    - \* Useful for text
  - ◊ Value encoding is used to store numeric values that are close in value to each other, but not close to 0
    - \* This allows the computer to store data with less bits per value
- The processing time to process the compression and decompression is frequently less than the time needed to store and retrieve the uncompressed data

## 2.1.13 In-Memory Databases

---

- Because of compression, large datasets can now be hosted entirely in memory, instead of on a disk
- RAM is considerably faster than disk access
- RAM is also typically smaller and more expensive than disk space
- This does mean that Power BI's usefulness is limited to data sets that can fit in memory.

## 2.1.14 Data Analysis eXpression Language

---

- DAX Stands for Data Analytics eXpression Language

Developed by Microsoft as the expression language native to the in-memory columnar database used for information analysis

- Used to add meaning to the data that you are analyzing through formulas, or expressions
- Adds four types of Calculations to our Model
  - ◊ Calculated Columns
  - ◊ Measures
  - ◊ Calculated Tables
  - ◊ Row Filters
- Used everywhere the xVelocity engine is used
- Can now be used when defining Data Cubes in SSAS Multidimensional mode

## 2.1.15 Direct Query

---

- DirectQuery is an engine that translates DAX formulas into relational SQL Queries
  - ◊ Using DirectQuery means that you are not making use of an in-memory columnar database
  - ◊ You will lose nearly all of the performance benefits of xVelocity if you use DirectQuery
- Limited DAX support
  - ◊ Calculated Tables are not allowed
  - ◊ Calculated Columns are limited to data within the row
  - ◊ You can still use DAX to write measures as those are calculated on-the-fly with each query as the data comes through
- All Model Tables must come from the same database
- Use this when you require real-time information from the source database, as it is updated
  - ◊ This approach is often not practical
  - ◊ Luckily, real-time data is rarely needed in analytics
    - \* When it is needed, the size of the datasets are often limited

## 2.1.16 Connect Live

---

- Enables you to connect to an xVelocity Database that has been hosted on Power BI Service, or with Microsoft SQL Server Analysis Services

## 2.1.17 Composite Model

---

- Beginning in 2018, Power BI began supporting Composite Models
- Composite Models allow for a hybrid of In-Memory data, and data provided via DirectQuery or Connect Live
- This provides the flexibility of using imported sources for smaller data sets used to group or filter larger data sets
- Implementing aggregations tables can improve performance
- More Information:

<https://docs.microsoft.com/en-us/power-bi/guidance/composite-model-guidance>

[Composite model guidance in Power BI Desktop]

## 2.1.18 Imported Data verse Direct Query and Connect Live

---

- Imported Data is brought into Power BI Desktop through Power Query, and resides in memory
  - ◊ Data is not real-time, needs to be explicitly refreshed
  - ◊ Limited to what can fit into memory
- Direct Query data is accessing data from a relational database.
  - ◊ Navigating around reports is slow as data refreshes from the source
  - ◊ Data is real-time
- Connect Live is accessing a rich data model designed around data analysis (Multidimensional or Tabular).
  - ◊ Data is not usually real-time, but is hosted at a centralized location and may not need to be explicitly refreshed (may be updated on a schedule)

## Section 2.2

### Power BI Tour

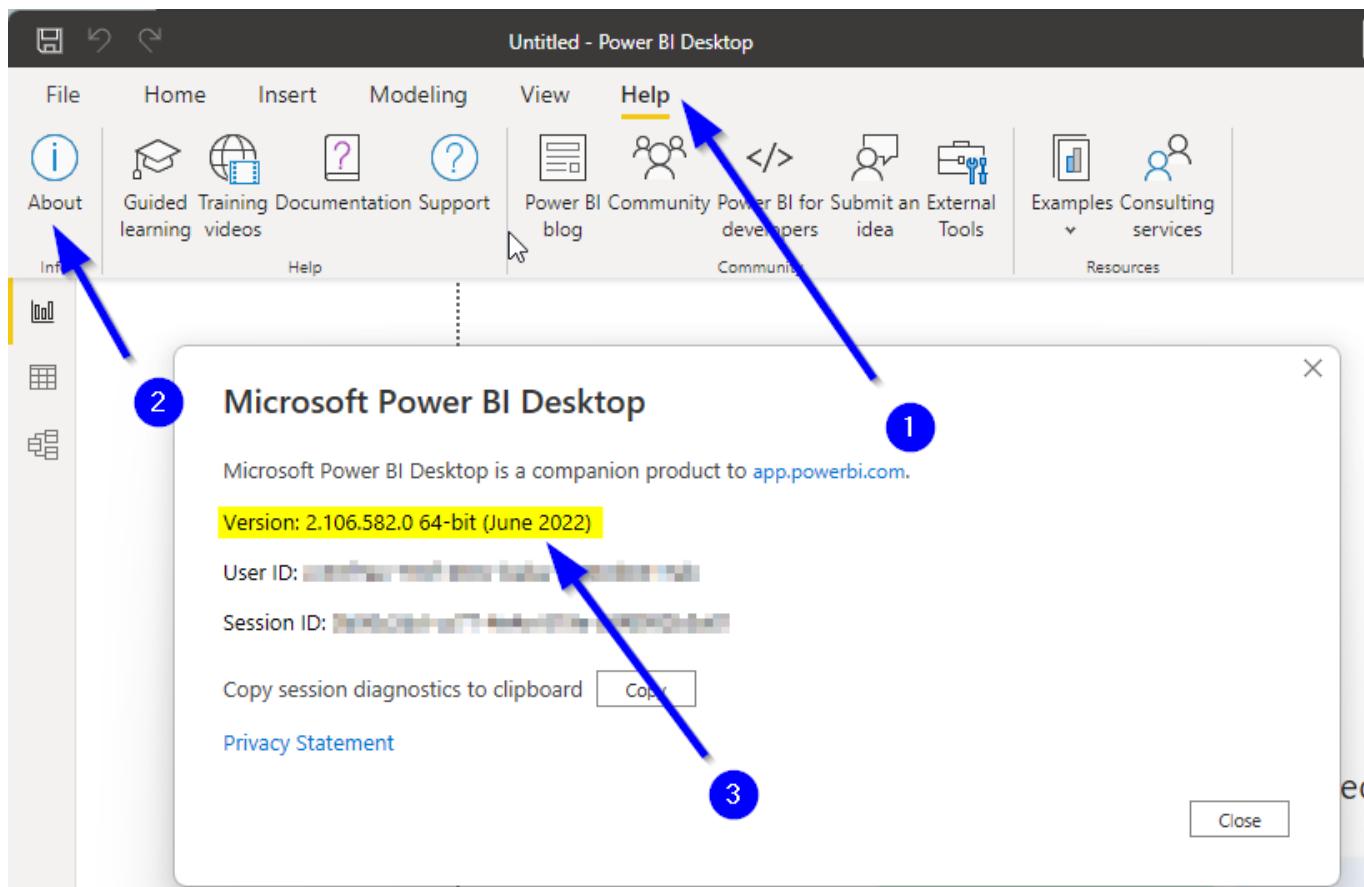
## 2.2.1 Power BI Versioning Strategy

---

- Microsoft has been very aggressive about releasing updates to Power BI
- The course has been revised over time
- The best strategy for sharing files, is to always be on the latest version of Power BI
- If somebody with a newer version of Power BI shares a file with you, you will not be able to open it until you upgrade

## 2.2.2 Checking your version of PBI Desktop

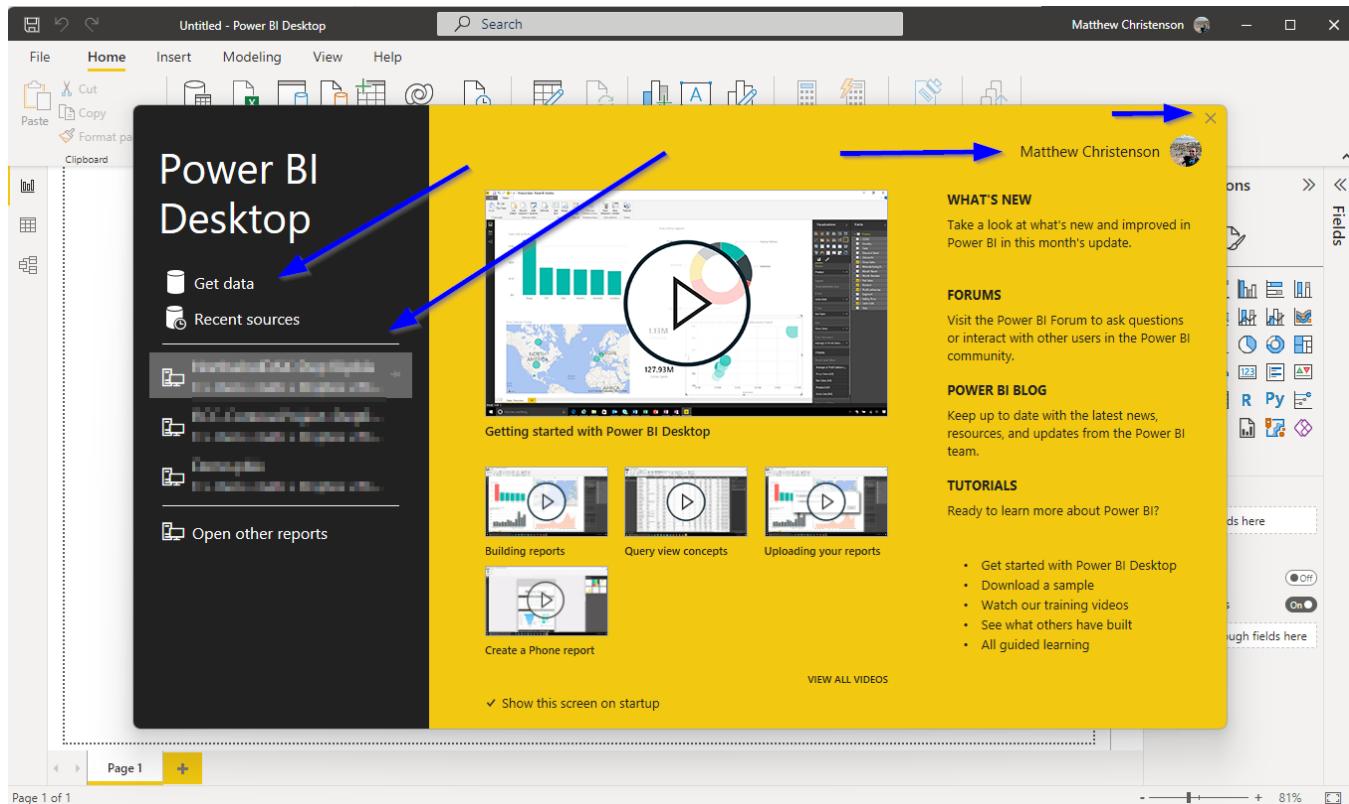
- The following illustration demonstrates how to check your version



1. Navigate to the *Help* Ribbon
2. Select *About*
3. Observe your installed version

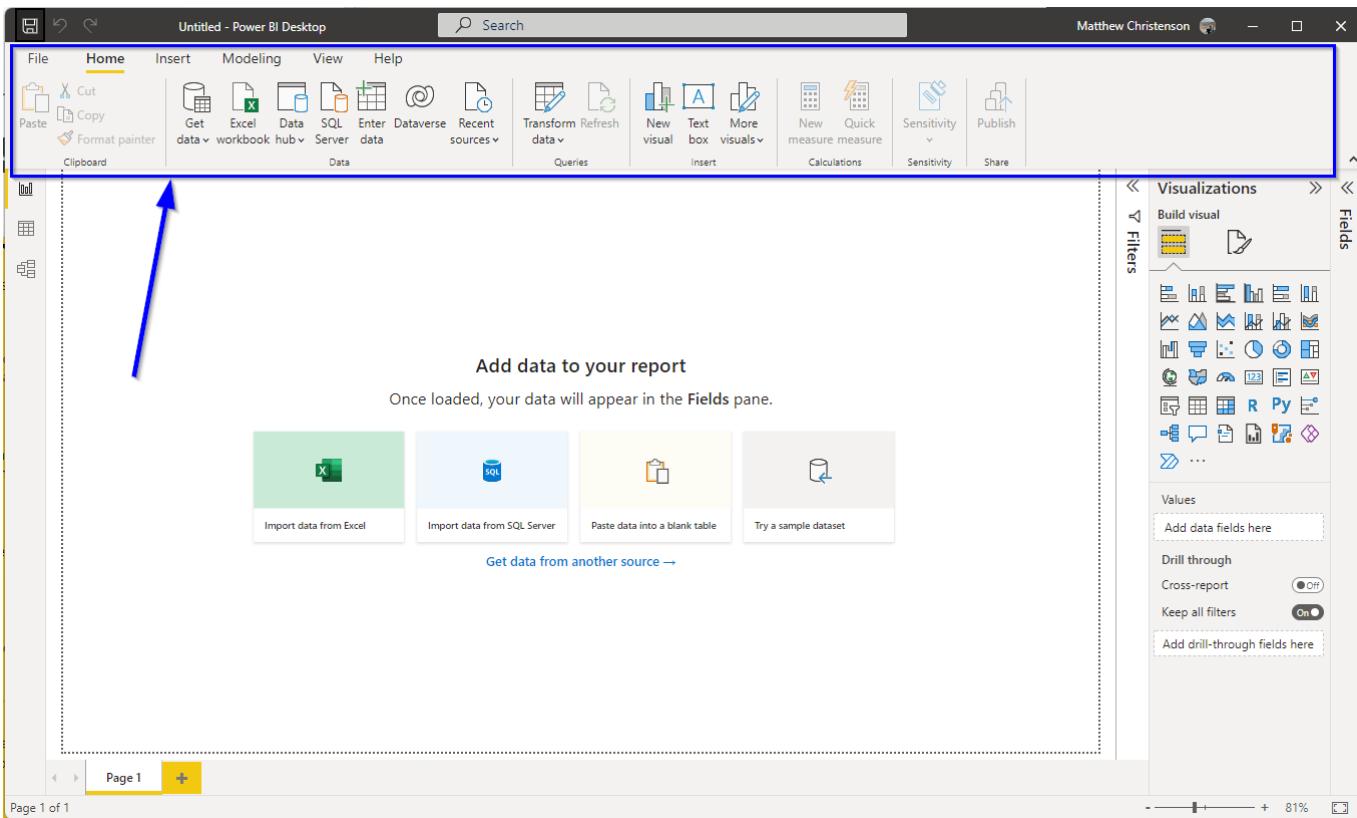
## 2.2.3 Splash Screen

- When the application first loads, a splash screen will appear
    - ◊ This screen can be used to:
      - \* Quickly open a recent project
      - \* Quickly begin a new project with the Get Data link
      - \* Quickly begin a new project with the Recent Sources link
      - \* Jump to Training Content
      - \* Read News related to Power BI
      - \* Log in, or observe the logged in user
      - \* Access community resources like Forums and Blogs



## 2.2.4 The Ribbon

- This area of the Desktop interface is called the Ribbon



- The Tabs on the Ribbon are different depending on the selected view, and in some cases what is selected contextually within that view

## 2.2.5 The View Selector

---

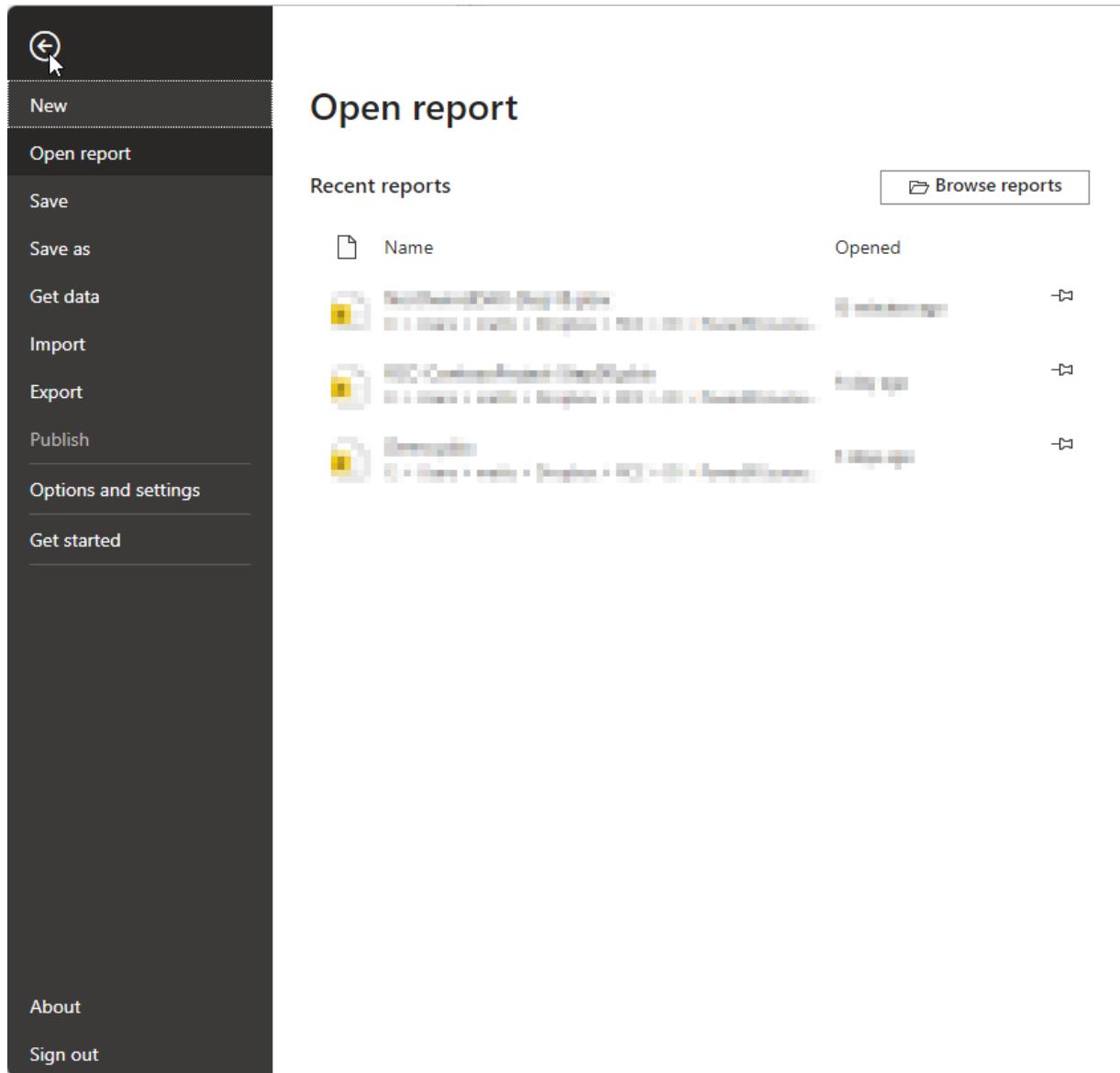
- The View Selector enables the selection of one of three views
  - ◊ Report View
    - \* Used to design and view visual report elements
  - ◊ Data View
    - \* Used to view data
    - \* Some modeling actions can be performed
  - ◊ Model View
    - \* Used to configure and see the relationship between datasets
    - \* Some modeling actions can be performed
- The Selected view will determine what options are seen in the Ribbon
- The Selected view will determine how the body portion of the application will look and work

## Working with Power BI – An Instructor-Led Course

The screenshot shows the Power BI Desktop interface with the 'Home' tab selected. On the left, there's a vertical ribbon bar with icons for Save, Cut, Copy, Format painter, Paste, Get data from workbook hub, Data, SQL Server, Enter data, Data, Transform Refresh data, New visual, Text box, More visuals, Calculations, Sensitivity, and Publish. Below the ribbon, three green boxes highlight the 'Report View', 'Data View', and 'Model View' sections. The 'Report View' section contains the text 'Add data to your report' and 'Once loaded, your data will appear in the Fields pane.' It includes four buttons: 'Import data from Excel', 'Import data from SQL Server', 'Paste data into a blank table', and 'Try a sample dataset'. Below these buttons is a link 'Get data from another source →'. The 'Data View' and 'Model View' sections also have arrows pointing towards the 'Report View' section. To the right, there's a 'Visualizations' pane with various chart icons and a 'Fields' pane with settings for 'Values' (Add data fields here, Drill through, Cross-report, Keep all filters) and 'Drill-through fields'.

## 2.2.6 The Ribbon: File Tab

- The File Tab will bring up a file menu, where you can manage which project you are working with, and access Options and Settings

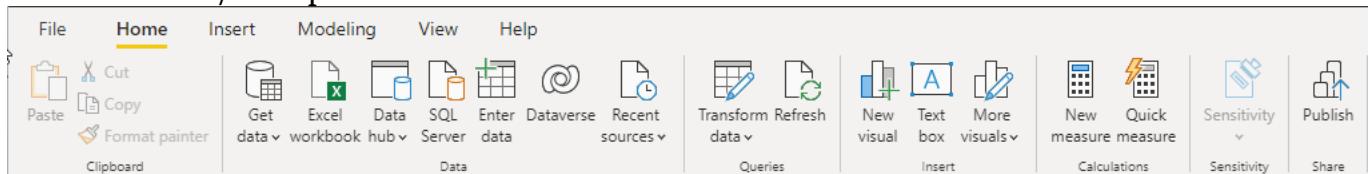


## 2.2.7 The Ribbon: Home Tab

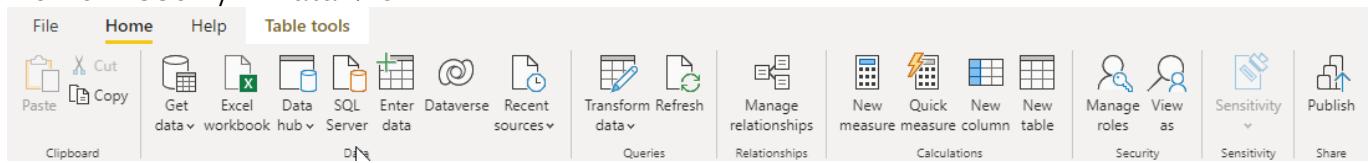
---

- The Home Tab contains general actions broken down into categories
- Categories and actions available will depend on what View is selected

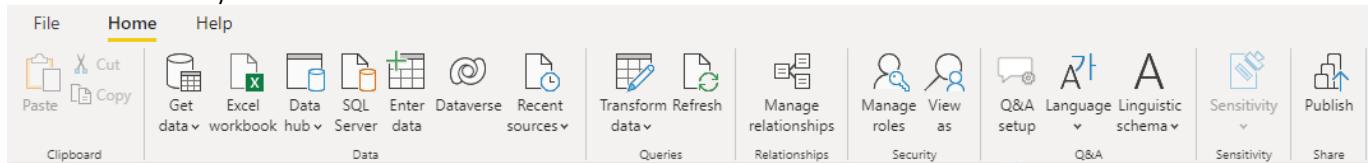
Home Ribbon /w Report View



Home Ribbon /w Data View

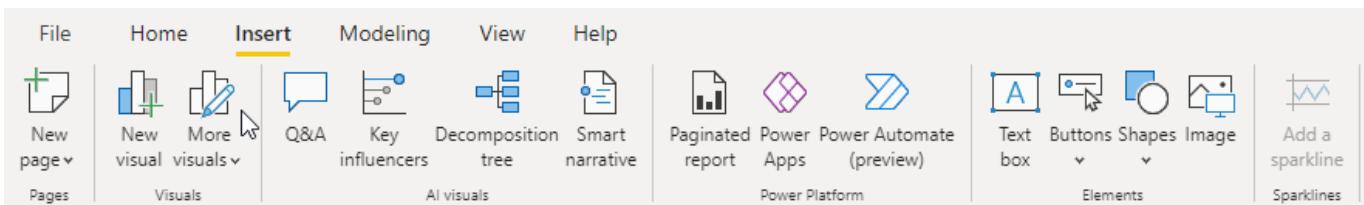


Home Ribbon /w Model View



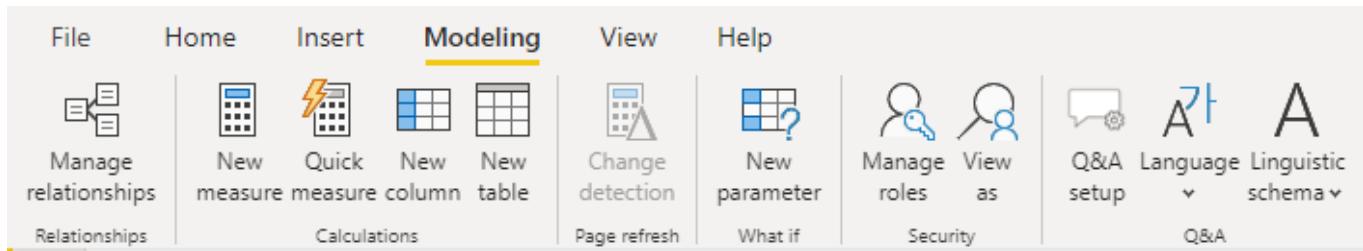
## 2.2.8 The Ribbon: Insert Tab

- The Insert Tab is only available on the Report View
- The Insert Tab various items that can be inserted into a report page



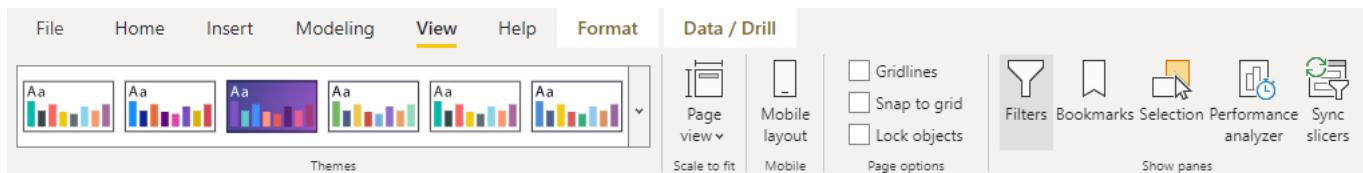
## 2.2.9 The Ribbon: The Modeling Tab

- The Modeling Tab is only available on the Report View
- The Modeling Tab offers a variety of shortcuts to alter the model
  - ◊ Most of these actions are accessible contextually from one of the others views using context menus (right clicks) on specific objects



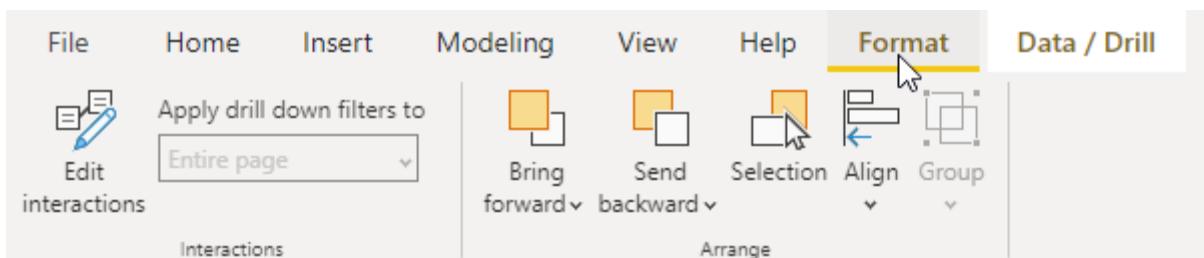
## 2.2.10 The Ribbon: View Tab

- The View Tab is only available on the Report View
- The View Tab contains various actions to help control the styling of your reports, or what features are visible in the editor



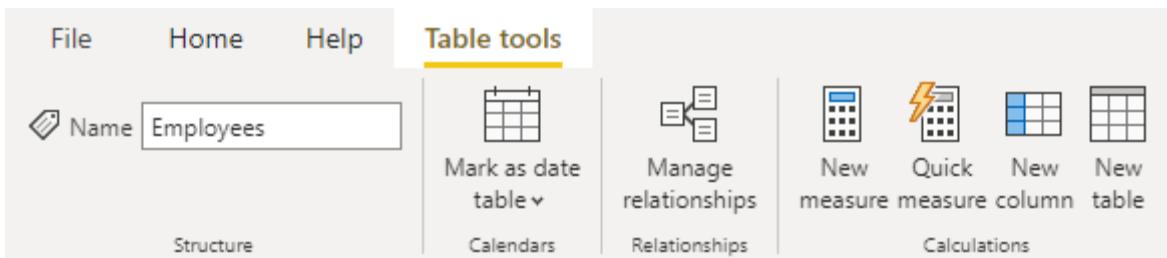
## 2.2.11 The Ribbon: Format Tab

- The Format Tab is only available on the Report View
- The Format Tab becomes visible when a specific report component is selected, and offers actions contextually relevant to the selected component



## 2.2.12 The Ribbon: Table Tools Tab

- The Table tools Tab is only available on the Data View
- The Table tools Tab provides actions that can be performed to the selected table



## 2.2.13 The Ribbon: Data / Drill Tab

- Only available on the Report View
- Only available when a specific control is selected
- Offers actions contextually relevant to the selected component
- Offers actions to drill down into the data the current component is visualizing

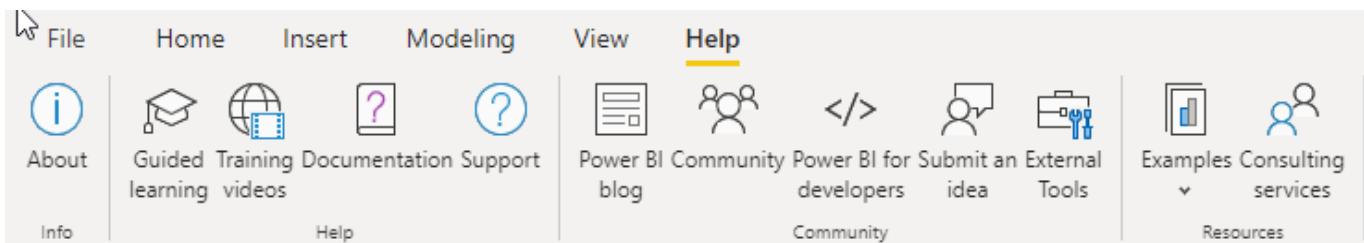
The screenshot shows the Microsoft Power BI ribbon with the 'Data / Drill' tab selected. The ribbon tabs are: File, Home, Insert, Modeling, View, Help, Format, Data / Drill (highlighted in yellow), and Table tools. Below the ribbon, a data grid displays a hierarchical breakdown of sales data. The grid shows columns for Year (2019), Quarter (Qtr 2), Month (June), Gender (2), and various age groups (15 to 35 years). The data is presented in a table format with totals for each category. The 'Drill actions' section of the ribbon is visible, showing options like 'Switch to next level', 'Drill up', 'Expand next level', 'Drill down', 'Drill through', 'Groups', and 'Find anomalies'. A 'Drill on Rows' dropdown menu is open at the bottom right.

Total										
Total										
Total										
<b>Female</b>	<b>\$38,213</b>	<b>\$21,150</b>	<b>\$24,170</b>	<b>\$17,838</b>	<b>\$35,976</b>	<b>\$137,347</b>	<b>\$137,347</b>	<b>\$137,347</b>	<b>\$137,347</b>	<b>\$137,347</b>
15 to 20 years	\$10,462	\$4,950	\$5,273	\$3,873	\$7,846	\$32,404	\$32,404	\$32,404	\$32,404	\$32,404
21 to 25 years	\$13,723	\$8,089	\$8,280	\$6,146	\$14,025	\$50,263	\$50,263	\$50,263	\$50,263	\$50,263
26 to 30 years	\$10,268	\$5,755	\$7,851	\$5,338	\$9,373	\$38,585	\$38,585	\$38,585	\$38,585	\$38,585
31 to 35 years	\$3,760	\$2,356	\$2,766	\$2,481	\$4,732	\$16,095	\$16,095	\$16,095	\$16,095	\$16,095
<b>Male</b>	<b>\$82,071</b>	<b>\$43,552</b>	<b>\$50,122</b>	<b>\$38,810</b>	<b>\$87,584</b>	<b>\$302,139</b>	<b>\$302,139</b>	<b>\$302,139</b>	<b>\$302,139</b>	<b>\$302,139</b>
15 to 20 years	\$15,029	\$8,124	\$10,434	\$7,877	\$14,708	\$56,172	\$56,172	\$56,172	\$56,172	\$56,172
21 to 25 years	\$31,511	\$16,596	\$19,549	\$15,372	\$35,901	\$118,929	\$118,929	\$118,929	\$118,929	\$118,929
26 to 30 years	\$24,771	\$13,122	\$14,478	\$11,173	\$26,712	\$90,256	\$90,256	\$90,256	\$90,256	\$90,256
31 to 35 years	\$10,760	\$5,710	\$5,661	\$4,388	\$10,263	\$36,782	\$36,782	\$36,782	\$36,782	\$36,782
<b>Total</b>	<b>\$120,284</b>	<b>\$64,702</b>	<b>\$74,292</b>	<b>\$56,648</b>	<b>\$123,560</b>	<b>\$439,486</b>	<b>\$439,486</b>	<b>\$439,486</b>	<b>\$439,486</b>	<b>\$439,486</b>

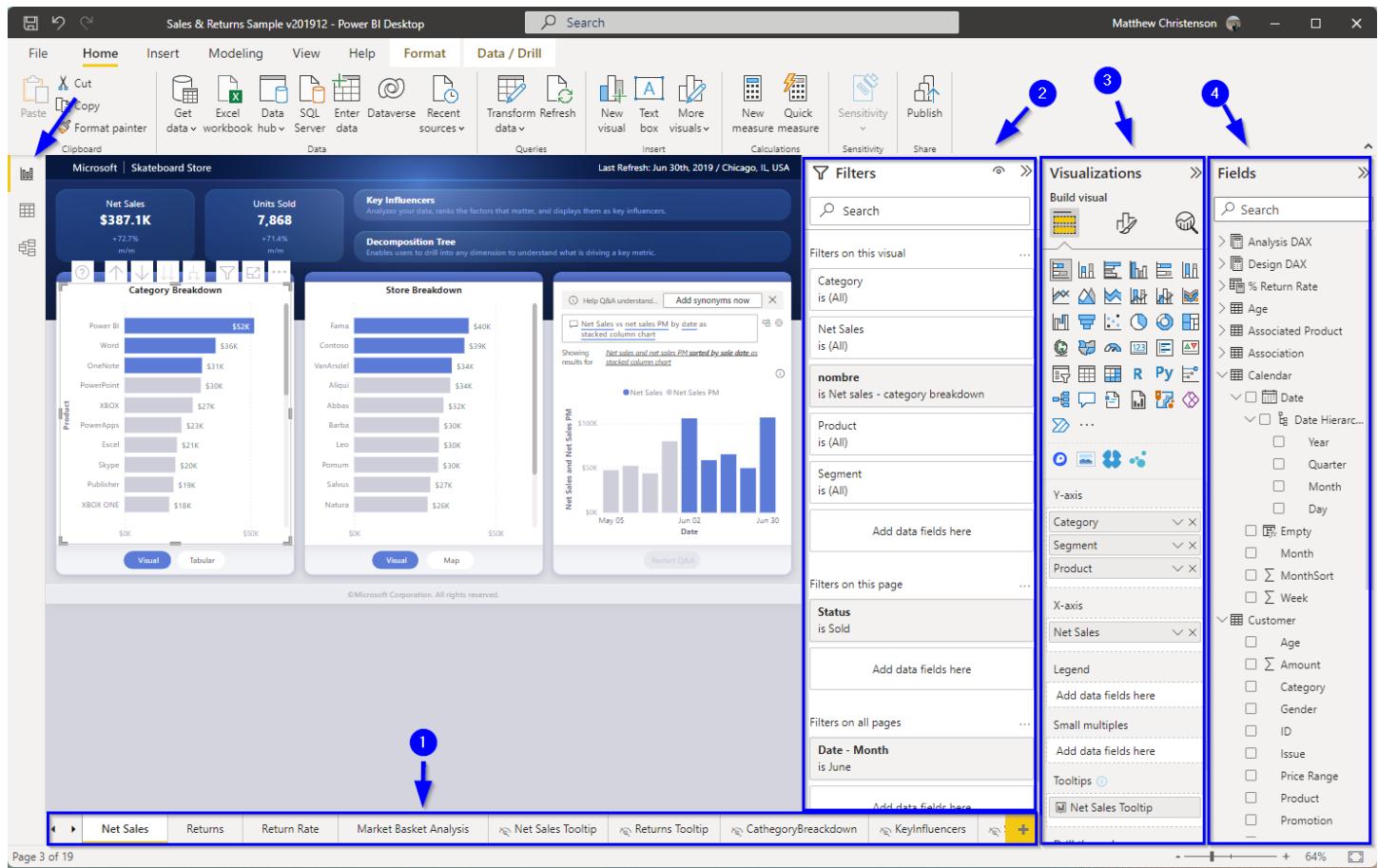
## 2.2.14 The Ribbon: Help Tab

---

- The Help Tab is available on all three of the Report, Data, and Model views
- The Help Tab offers the same options regardless of the view



## 2.2.15 Report View Body



1. The Pages area enables you to create multiple report pages and navigate around them
2. The Filters Pane enables you to modify what filters are applied both at the page and visualization level for the selected visualization
3. The Visualizations Pane enables you to add new visualizations to the report surface, or to change the selected visualization's fields and other properties
4. The Fields Pane enables you to select fields that will be included in the selected visualization

## 2.2.16 Data View Body

- The Data View Body allows you to visualize the raw data in its tables
- You can select columns in the body visual, or select them by expanding a table in the fields pane, and then modify them using the ribbon or context (right click) menus

Sales & Returns Sample v201912 - Power BI Desktop

File Home Help Table tools Column tools

Name Date Format \*Wednesday, Marc... Summarization Don't summarize

Data type Date Data category Uncategorized Sort by column

Structure Formatting Properties Groups Relationships Calculations

**Fields**

Search

> Details

> Issues and Promotions

> Product

> Sales

Σ Amount

Date

ID

ProductID

Status

StoreID

Σ Unit

> STable

> Store

> Tooltip Info

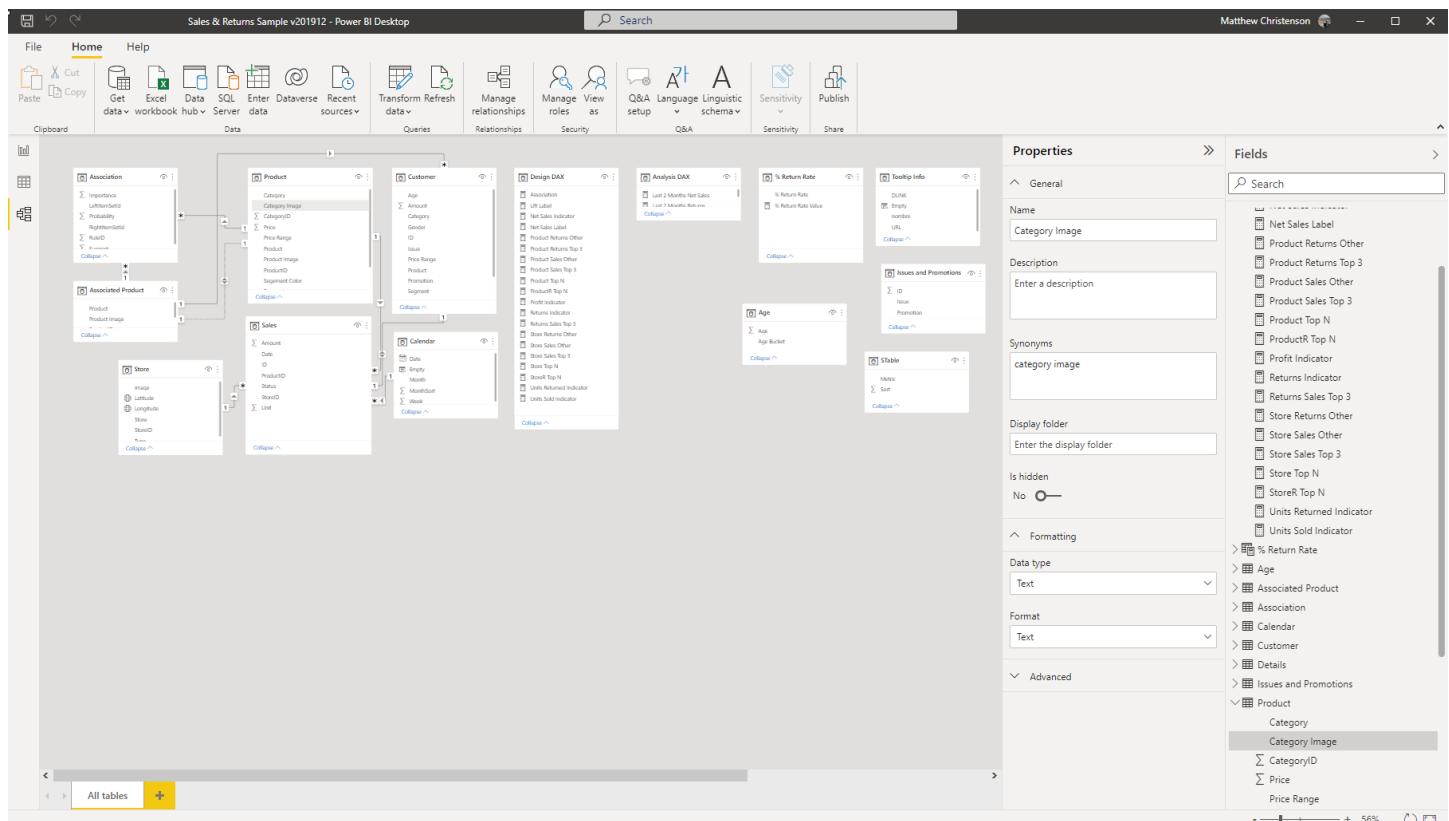
> Tooltip Info2

ProductID	StoreID	Status	ID	Unit	Amount	Date
12	5	Sold	36510074431	1	\$40	Sunday, January 13, 2019
12	5	Sold	35261794581	1	\$40	Sunday, January 13, 2019
12	5	Sold	23806384593	1	\$40	Sunday, January 6, 2019
12	5	Sold	333594642117	1	\$40	Sunday, January 13, 2019
12	5	Sold	360210029131	1	\$40	Sunday, January 13, 2019
12	5	Sold	407034971140	1	\$40	Sunday, January 20, 2019
12	5	Sold	324608468151	1	\$40	Sunday, January 13, 2019
12	5	Sold	496794428168	1	\$40	Sunday, January 20, 2019
12	5	Sold	243750127253	1	\$40	Sunday, January 6, 2019
12	5	Sold	466795008257	1	\$40	Sunday, January 20, 2019
12	5	Sold	340236122286	1	\$40	Sunday, January 13, 2019
12	5	Sold	456379396312	1	\$40	Sunday, January 20, 2019
12	5	Sold	226336721380	1	\$40	Sunday, January 6, 2019
12	5	Sold	217671494393	1	\$40	Sunday, January 6, 2019
12	5	Sold	274455362542	1	\$40	Sunday, January 6, 2019
12	5	Sold	417420029562	1	\$40	Sunday, January 20, 2019
12	5	Sold	549818912619	1	\$40	Sunday, January 27, 2019
12	5	Sold	389819215651	1	\$40	Sunday, January 13, 2019
12	5	Sold	394655426679	1	\$40	Sunday, January 13, 2019

Table: Sales (29,875 rows) Column: Date (26 distinct values)

## 2.2.17 Model View Body

- The Model View Body enables you to visualize the relationships between tables
- Additional pages can be added to focus on a smaller subset of tables
  - This is helpful when you have multiple fact/data tables



## Section 2.3

## Exercises

## Exercise 2.1 Exploring Power BI Sample Projects

---

In this exercise, we will learn to navigate around the Power BI Desktop Interface by exploring a few of the freely available Power BI Desktop Sample projects provided by Microsoft.

For your convenience, the sample projects that we will be exploring have already been downloaded and are available with the files provided to you by your facilitator, however it's important to note that you can access these yourselves at the following URL:

<https://docs.microsoft.com/en-us/power-bi/create-reports/sample-datasets>

As you go through this exercise, please take your time, and do not feel rushed. This is an opportunity for you to start thinking about questions that you have that we will likely answer as we complete this course.

1. Open Power BI Desktop.
  
- 2 . Open the file found at:  
  {LABFILES}\StarterFiles\PowerBISamples\Adventureworks Sales.pbix
  
3. Navigate to the Data View.
  
4. Observe the Table Names in the Fields Pane.
  
5. Expand the Sales Table.  
\* What do you think each means?
  
6. Notice the icons next to each of the Fields in the Table.  
\* What do you think that means?
  
7. Notice the icon that looks like a crossed off Eye next to only some of the fields  
\* What do you think that means?
  
8. Take a few minutes to explore other tables and fields.
  
9. Navigate to the Models Tab
  
10. Observe how the various tables are related and consider the following questions. Don't worry if you don't have answers yet, we will be exploring each of these subjects later.
  - \* What table seems to be at the center?
  - \* What kind of shape does this layout seem to make?

- \* What is different about the fields in the Center Table, compared to the outer tables?

11. Navigate to the Report View

12. Take a few minutes to view each of the Pages in the Report View. Consider the following questions:

- \* What Data Points are being visualized in each of the Pages?
- \* What happens when you click on various components of the visuals?

13. Go into the main Classroom Chat and let your instructor know that you have finished exploring AdventureWorks Sales.pbix, and that you are about to explore a few more samples.

14. Select 2 more sample files from the PowerBISamples folder and explore each of them in a similar way. Spend 5-10 minutes on each sample. The purpose of this exercise is to create questions in your head, and help you begin to think about ways that Power BI can be used to Analyze Data within your organization.

- \* Take your time and feed your creative mind with ideas and questions!

15. Go into the main Classroom Chat and let your instructor know that you have finished!

# Module 3

## Data Modeling

## Section 3.1

### Data Modeling Overview

## 3.1.1 Modeling Basics

---

- A Data Model is a shape
- Some modeling is simple
  - ◊ Relational Databases only cover very basic modeling, the shape and datatype of the tables
- Some modeling is more advanced
  - ◊ Advanced Calculations
  - ◊ Explicit Relationship Rules
  - ◊ Attribute Hierarchies
  - ◊ Data Formatting Preferences
- Modeling is a set of guidelines that one might use to shape data
- Modeling is shaping our data in a way that works best for our needs
  - ◊ Different needs mean different models

## 3.1.2 Modeling Happens in Stages

---

- ◊ Bring in the data
- ◊ Transform it
  - \* Combine Tables
  - \* Split Tables
  - \* Change Column Names
  - \* Change Table Names
  - \* Change Data Types

Some models, like Power BI or Data Cubes, allow for more human oriented types like “Percentage”, or “Currency”, where more database centric models would only support “Decimal” or “Float”
  - \* Clean up bad data by
    - Removing it
    - Fixing it

- Augment it

- ◊ Add Calculations
  - \* Simple Calculations include data from a single row, or from a simple lookup
  - \* Advanced calculations might take into consider data from many other rows
- ◊ Add Attribute Intelligence
  - \* Defining Hierarchies
  - \* Specifying Date Attributes

### 3.1.3 Modeling is Transforming & Describing

---

- A Source Model is transformed into a Destination Model through a series of steps
- Understanding your source data is essential to the modeling process!
- Understanding an ideal destination model is just as essential!

## 3.1.4 Source Models

---

- Understanding about various modeling strategies is helpful to understand our starting point
- Source data may come from
  - ◊ Flat Files
  - ◊ OLTP Databases
  - ◊ OLAP Databases
  - ◊ APIs
  - ◊ JSON Files
  - ◊ XML Files

## 3.1.5 An ideal Power BI Model

---

- An ideal model for Power BI is for data to be organized into a Star, or Snowflake schema
- OLAP databases, also referred to as Data Warehouses, are built using Star Schemas, so these are an ideal data source!
- Frequently OLAP Databases are not available
  - ◊ OLAP Databases require significant planning and design – projects that take months and numerous participants
- Instead, we have two common extremes
  - ◊ Broken into many tables
    - \* Relationships from a measurable number might be either one-to-many or many-to-one
    - \* Relationships form circular loops
  - ◊ Flattened into a single table
    - \* Relationship data is lost and needs to be inferred based on groupings

## **Section 3.2**

### **Flat Files**

## 3.2.1 Flat Files

---

- Frequently, the only source of data that we have access to is a file of extracted data
  - ◊ This is common when data is coming from another organization
- This often comes as a single set of columns and rows
  - ◊ Multiple tables have been “Flattened” by joining them all together into one wide and long table containing many duplicated values
- Sometimes, keys have been excluded making it difficult to rebuild dimensions
  - ◊ In this case you’ll need to identify what can be used as a key
  - ◊ If no single column can be used as a key, you can create one by combining multiple columns
    - \* Power BI Models do not have the ability to specify composite keys as relationships between tables

## 3.2.2 Demo: Exploring a Flat File

---

- Lets take a look at some flat files together!

### 3.2.3 Discussion

---

- What are some examples of Flat files used within your organization?
- Where do Flat Files come from?
- Is an Excel file a Flat File?

## Section 3.3

### Relational Databases

### 3.3.1 Relational Databases

---

- Relational Databases are made up of multiple tables
- A row in one table may relate to a row in another table
  - ◊ This relationship is frequently recorded as a foreign key
    - \* Foreign keys are constraints, not relationships
    - \* Relationships are specified with each query, using join statements
    - \* The model of relational databases does not have an explicit way to define relationships between tables, but the side effect of foreign keys is “good enough”

### 3.3.2 Tables

---

- Tables are made up of
  - ◊ Columns
  - ◊ Rows
  - ◊ Keys
  - ◊ Indexes

### 3.3.3 Columns

---

- Columns define the shape of the data in a table
- Columns must have
  - ◊ A name
  - ◊ A Datatype
    - \* A datatype may or may not have a precision
- Columns can optionally be configured to allow null values

### 3.3.4 Rows

---

- A row is a record of data within a table
- The row's shape depends on the columns configured for that table

### 3.3.5 Keys

---

- There are two important types of keys to understand in Relational Databases
  - ◊ Primary Keys
  - ◊ Foreign Keys

## 3.3.6 Primary Keys

---

- Each table should have one primary key
- A primary key is used to uniquely identify a row in a database
- If a primary key consists of multiple columns, it is also referred to as a composite key
  - ◊ This is frequent in “resolving tables” used to represent many to many relationships

### 3.3.7 Foreign Keys

---

- A Foreign Key is a column in a table that references a record in another table
- This is what builds a *Relationship* between two tables
- We'll look at some examples in just a moment

### 3.3.8 Relationship Properties

---

- Relationships in Relational Databases have two considerations:
  - ◊ Cardinality
  - ◊ Existence

### 3.3.9 Relationship Cardinality

---

- The three possible ways that a relationship may exist between two tables:
  - ◊ One to One
  - ◊ One to Many
    - \* Or Many to One depending on the order of tables
  - ◊ Many to Many
    - \* Many to Many relationships require 3 tables to work

### 3.3.10 Relationship Existence

---

- Existence is whether a record is required on either side of a relationship
- Example
  - ◊ Is it required that an order has a customer?
    - \* In an E-commerce system, the answer is probably yes
    - \* In a Point-of-sale system, the answer may be no
  - ◊ Is it required that a customer has an order?
    - \* Probably not immediately
    - \* But can you really call them a customer if they have never bought anything?!

### 3.3.11 Demonstration: Exploring Relational Databases

---

- Lets explore a few Relational Databases:
  - ◊ AdventureWorks, AdventureWorksDW
  - ◊ WideWorldImports, WideWorldImportsDW
  - ◊ Northwind
- We'll try to find:
  - ◊ 1 to 1 relationships
  - ◊ 1 to many relationships
  - ◊ Many to many relationships
- We'll explore the databases with a few tools
  - ◊ Microsoft SQL Server Management Studio
  - ◊ DBeaver Community Edition
- We'll see how a SQL SELECT Statement that joins many tables could be used to transform data into a flat file!

### 3.3.12 Discussion

---

- Which applications that you use in your organization utilize relational databases to persist data?
- Is querying that database direction to extract data a possibility?
- Why?

## Section 3.4

# OLTP Database Design

## 3.4.1 OLTP Systems

---

- Most of the software products we use persist data into OLTP Databases
- An OLTP Database is still a Relational Database
  - ◊ It follows more defined design guidelines
- OLTP stands for Online Transactional Processing
- OLTP enables real-time execution of large numbers of database transactions by large numbers of people
- OLTP transactions change data with inserts, updates, and deletes of records of rows
  - ◊ Operations are grouped together so that they succeed or fail as a unit
    - \* Referred to as the transaction's atomicity, or indivisibility

## 3.4.2 Characteristics of OLTP Systems

---

- OLTP Systems do the following:
  - ◊ Process a large number of relatively simple transactions
  - ◊ Enable multi-user access to the same data, while ensuring data integrity
  - ◊ Emphasize very rapid processing, with response times measure in milliseconds
  - ◊ Provide Indexed data sets
  - ◊ Are designed around 24/7/365 availability
  - ◊ Database systems designed for frontline workers, or customer self-service applications

### 3.4.3 Database Normalization

---

- To achieve their goals, OLTP Database tables are normalized
- Normalization is a set of rules or guidelines that should be followed when designing tables in a relational database system
- Let's Explore Database Normalization together:
- <https://www.w3schools.in/DBMS/database-normalization>

## 3.4.4 Demonstration: Explore AdventureWorks

---

- Lets take another look at the AdventureWorks Database and identify where data is normalized

## 3.4.5 Discussion

---

- Do you have OLTP Databases within your organization?
- What software systems do they support?
- How does database normalization help to meet the needs of a transaction-based software product?

## Section 3.5

# OLAP Database Design

## 3.5.1 OLAP Databases

---

- OLAP stands for Online Analytical Processing
- An OLAP Database is a Relational Database with its columns and rows structure in a way that makes data analysis easier
- OLAP Databases are frequently referred to as Data Warehouses

## 3.5.2 Data Warehouses

---

- The term Data Warehouse means different things in different contexts
- Some are referencing multiple databases when using the term
- Others are specifying a specific OLAP Database
- Other terms used in the space are Data Marts, and Data Lakes, but both are technically different
  - ◊ Data Marts focus on a single functional area of an organization, and is considered a subset of a Data Warehouse
    - \* In this context the Data Warehouse might span multiple databases
  - ◊ Data Lakes generally contain unstructured data
    - \* Follow “Schema on Read” instead of “Schema on Write”
- In this course, we’ll simply use the term OLAP Databases for clarity

### 3.5.3 OLAP Modeling

---

- OLAP Databases do not follow the same normalization rules that generally govern OLTP Databases
  - ◊ Design focuses on fast retrieval of data
  - ◊ Duplicated data is acceptable
  - ◊ Calculated data is acceptable
  - ◊ Update speed is not a concern
- Many OLAP Databases are updated just once per night based on a schedule, and can be brought down during the processing time
  - ◊ This is adequate for most organizations
- Occasionally you'll see databases updated hourly, or every 5-10 minutes
  - ◊ These are typically very high-volume databases, where catching up a full day's updates might be too resource intensive

- Real-time OLAP is a pipe dream for most organizations
  - ◊ To achieve close to real-time data, custom solutions are needed, usually designed around event pipelines and processing
- Concurrent changes to data is not a concern
  - ◊ Data is generally updated by a single scheduled process
  - ◊ Frequently a maintenance window makes it acceptable to take the database off-line for general use during the update process
- Accommodates data that changes over time
  - ◊ OLAP might capture historic data that OLTP discards or overwrites
- Star Schemas, or Snowflake Schemas are used
  - ◊ This process is only partially normalized

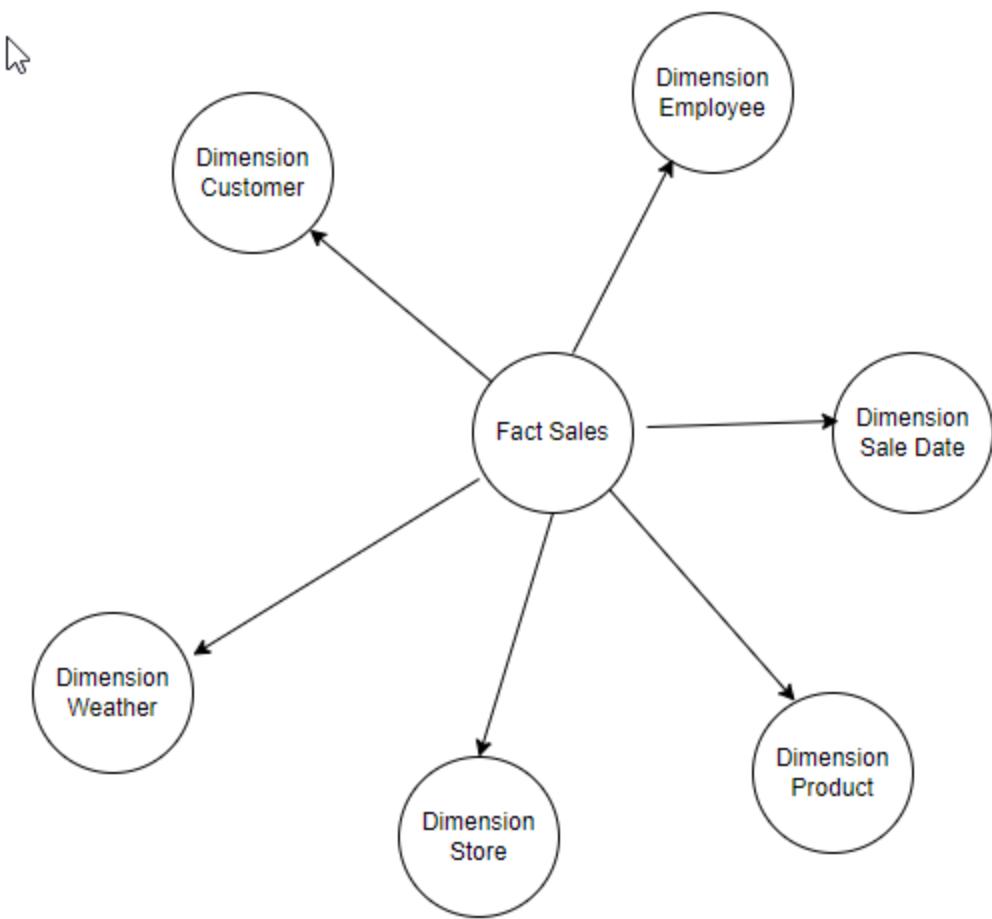
## 3.5.4 Star Schema

---

- A Star Schema is the goal of an OLAP Modeler
- There are two types of tables in a Star Schema
  - ◊ Fact Tables
    - \* A Fact Represents some kind of event that would have measurable quantities that one would like to analyze
  - ◊ Dimension Tables
    - \* A Dimensions represents a way by which one might want to segment facts into groups

## 3.5.5 Star Schema Visual

---

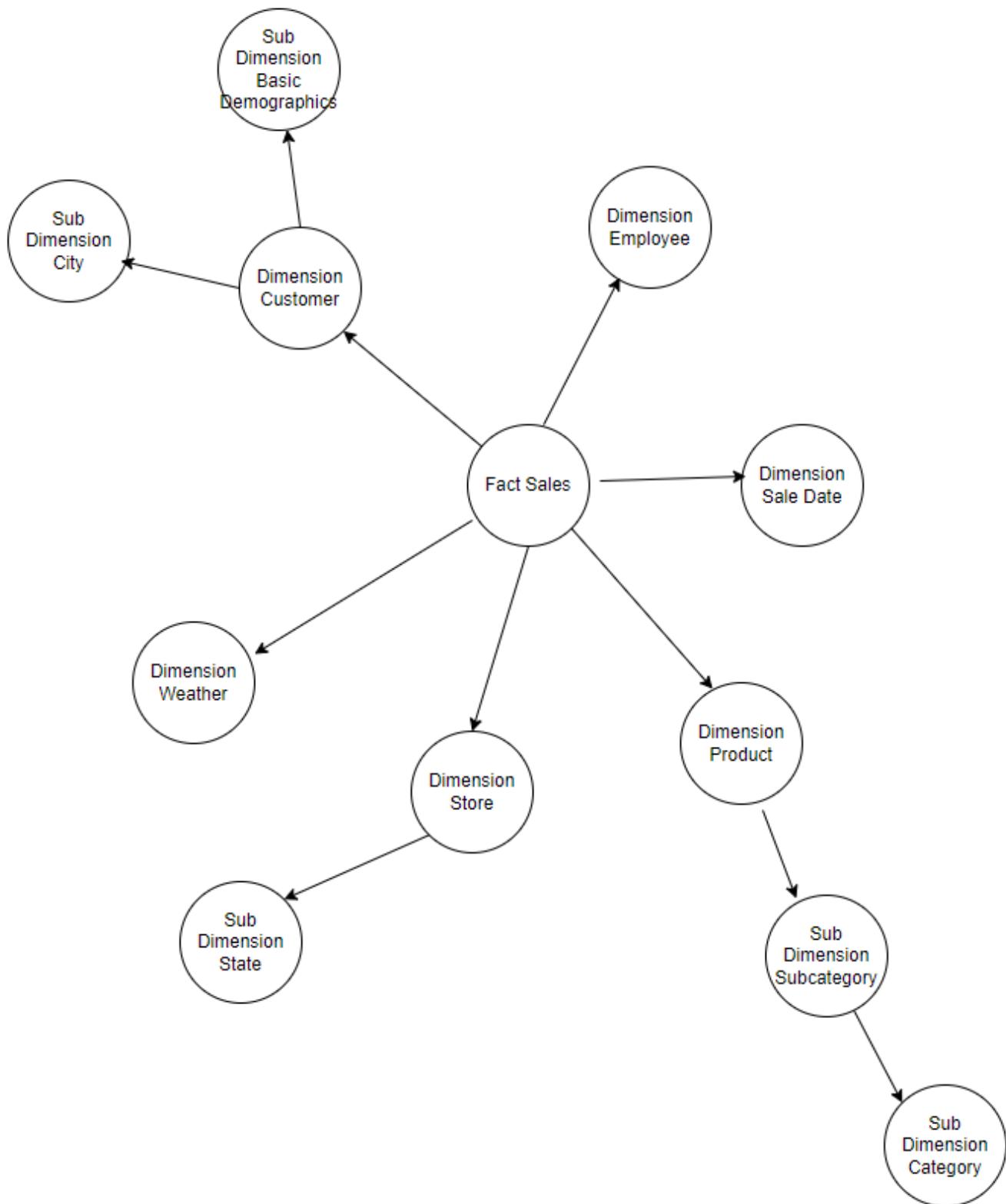


## 3.5.6 Snowflake Schema

---

- Sometimes some normalization is added to dimension tables
- This is called a Snowflake Schema

## 3.5.7 Snowflake Schema Visual



## 3.5.8 Fact Tables

---

- A row in a fact table generally represents some kind of business event
  - ◊ Examples:
    - \* An Order has been placed
    - \* A Product has been Shipped
    - \* Inventory has been moved
    - \* A Day passed at a Call Center
    - \* Someone has filled out a survey
    - \* A user experienced an error
- Have Measure columns
  - ◊ A Measure is almost always a numeric value that will be summarized or aggregated
- Have Dimensional Columns
  - ◊ Represent ways by which the measures might be aggregated
  - ◊ These will be foreign keys to records in Dimension Tables
  - ◊ Sometimes called Key Columns

- May have reference columns
  - ◊ These are not measures that would be aggregated, and do not reference rows in a separate dimension table
  - ◊ These are sometimes used to support auditing, reference the source of the data, or help ETL routines that populate these records
  - ◊ These are generally not used for analysis purposes
- Frequently have large volumes of rows
  - ◊ Could easily be millions of records or more

## 3.5.9 Measures

---

- These are the key metrics by which business decisions are made
- These are numeric values that are summarized
  - ◊ Examples
    - \* Quantity
    - \* Unit Price
    - \* Sales Amount
    - \* Tax Rate
    - \* Number of Calls
    - \* Number of Orders
- Can be non-numeric values that are counted
  - ◊ Ideally these are represented as booleans so that summarizing them is counting them
    - \* Example
    - \* Has Shipped
    - \* 1 for yes, 0 for no
    - \* Was Late
    - \* 1 for yes, 0 for no

- Should be carefully chosen for fact tables
  - ◊ If a measure is not going to be needed for a business decision, it should not be included
- Text should be avoided
  - ◊ Not only is the only possible aggregation to count them, but text takes a lot more bits to store than an integer or decimal type

## 3.5.10 Example Fact Table

---

- This is the FactInternetSales table from AdventureworksDW

dbo.FactInternetSales	
Columns	
ProductKey	(FK, int, not null)
OrderDateKey	(FK, int, not null)
DueDateKey	(FK, int, not null)
ShipDateKey	(FK, int, not null)
CustomerKey	(FK, int, not null)
PromotionKey	(FK, int, not null)
CurrencyKey	(FK, int, not null)
SalesTerritoryKey	(FK, int, not null)
SalesOrderNumber	(PK, nvarchar(20), not null)
SalesOrderLineNumber	(PK, tinyint, not null)
RevisionNumber	(tinyint, not null)
OrderQuantity	(smallint, not null)
UnitPrice	(money, not null)
ExtendedAmount	(money, not null)
UnitPriceDiscountPct	(float, not null)
DiscountAmount	(float, not null)
ProductStandardCost	(money, not null)
TotalProductCost	(money, not null)
SalesAmount	(money, not null)
TaxAmt	(money, not null)
Freight	(money, not null)
CarrierTrackingNumber	(nvarchar(25), null)
CustomerPONumber	(nvarchar(25), null)
OrderDate	(datetime, null)
DueDate	(datetime, null)
ShipDate	(datetime, null)

## 3.5.11 Dimension Tables

---

- A dimension table supports the dimension columns found in the fact tables, with descriptive attributes
- A Fact Table's Dimension column is a foreign key to a record found in the associated Dimension table
- Dimension tables usually support nouns that describe the event that a fact row represents
  - ◊ Examples
    - \* Product
    - \* Employee
    - \* Date
    - \* Customer

## 3.5.12 Dimension Table Columns

---

- Two types of columns are found in dimension tables

## 3.5.13 Dimension Table Key Columns

---

### ◊ Surrogate Keys

- \* Usually the primary key of the table that uniquely identifies a row
- \* Generally auto-numbered specifically for this database
- \* Column names usually suffixed with ‘Key’ i.e. ProductKey

### ◊ Business Keys

- \* Usually the unique identifier of the object this row represents in the source system.
- \* For example the ProductId
- \* Sometimes suffixed with ‘BizKey’ i.e. ProductIdBizKey

## 3.5.14 Dimension Table Attribute Columns

---

- Usually text, describing the fact

- ◊ Examples

- \* Color

- Red
    - Green
    - Blue

- \* Product Name

- Apple
    - Pear
    - Carrot
    - Beef

- \* Product Category

- Fruit
    - Vegetables
    - Meat

- Sometimes fits into a natural Hierarchy

- ◊ Year -> Month -> Day

- ◊ Country -> Region -> Town

- ◊ Product Category -> Product Name

## 3.5.15 Dimension Granularity

---

- Determining and understanding the granularity of an attribute is critical
  - ◊ Date Dimension Questions:
    - \* Are sales facts going to need to be analyzed hourly? Daily? Weekly?
    - \* If Hourly totals are not used in analysis, this would be extraneous information and too granular
    - \* It would not be feasible to describe down to the second or minute
    - \* Although time-clock analysis might be able to justify down to the minute - this would usually mean different Date dimensions and Time dimensions
  - ◊ Geography Dimension Questions:
    - \* Would we aggregate sales per City? Per Postal Code? Per Street?
    - \* Usually Street and Unit number is too granular
- Choosing the proper granularity for a dimension will prevent unnecessary data from being stored, and speed up analysis and data population routines

## 3.5.16 Slowly Changing Dimensions

---

- For each attribute in a dimension table, a modeler needs to ask: Is it important to track this change over time?
- Type 1 SCDs, we will overwrite the attributes, potentially changing how we describe historic facts.
- Type 2 SCDs, we will create a new row in the dimension table leaving old facts pointing to the old record, and new facts pointing to the latest record
  - ◊ Sometimes these are referred to as “historical attributes”
- Type 2 SCDs will require the use of surrogate keys, because the original business key will not uniquely describe the record within the table

## 3.5.17 SCD Supporting Columns

---

- A Dimension table with at least one Type 2 SCD will require additional columns so that ETL routines can lookup the correct record to associate new facts with
- Could be a single column that stores a bit named “Most Recent”
  - ◊ This is best for performance as this can be indexed or partitioned out to speed up surrogate key lookups during ETL routines
- Could be a pair of columns representing a period of validity, for example RowBeginDate, RowEndDate

## 3.5.18 Surrogate Keys

---

- Surrogate Keys are used as primary keys in dimension tables to uniquely identify a row
- Original Primary Keys from source OLTP systems won't usually work, and in this context are referred to as "Application Keys" or "Business Keys"
  - ◊ They will not be unique in a table if any attribute column represents a type 2 slowly changing dimension
- Surrogate keys are usually integers which only used four bytes per row

## 3.5.19 Date Dimension Tables

---

- Date is a common dimension used to summarize data
- A date dimension table is usually created to define periods of date / time
- Frequently there is one entry per day
  - ◊ If further granularity is needed, multiple entries per day might be created, for example one entry per hour
    - \* More than one entry per day can make some common date based calculations more difficult
    - \* An alternative might be separate Date dimensions and Time dimensions, with exactly 24 entries (one per hour) or 1,440 entries (one per minute) in the DimTime table, and with one entry per day in the DimDate table
- Date Tables might have columns to represent both fiscal and regular calendars
- Date Tables might have Boolean or Integer columns that provide holiday information for dates
  - \* Such as “Is Holiday”, “Is Mothersday”, “DaysUntilHoliday”
- Date Tables generally have descriptive text for day names, month names, as well as their numbers

## 3.5.20 Date Dimension Table: Example

---

- The Adventureworks Database uses the following columns in it's Date Table
  - \* DateKey
  - \* FullDateAlternateKey
  - \* DayNumberOfWeek
  - \* EnglishDayNameOfWeek
  - \* SpanishDayNameOfWeek
  - \* FrenchDayNameOfWeek
  - \* DayNumberOfMonth
  - \* DayNumberOfYear
  - \* WeekNumberOfYear
  - \* EnglishMonthName
  - \* SpanishMonthName
  - \* FrenchMonthName
  - \* MonthNumberOfYear
  - \* CalendarQuarter
  - \* CalendarYear
  - \* CalendarSemester
  - \* FiscalQuarter
  - \* FiscalYear
  - \* FiscalSemester

## 3.5.21 Date Dimension Table: Example Data

---

- Values in each column are quite predictable

	DateKey	FullDateAlternateKey	DayNumberOfWeek	EnglishDayNameOfWeek	SpanishDayNameOfWeek	FrenchDayNameOfWeek	DayNumberOfMonth
1	20050101	2005-01-01	7	Saturday	Sábado	Samedi	1
2	20050102	2005-01-02	1	Sunday	Domingo	Dimanche	2
3	20050103	2005-01-03	2	Monday	Lunes	Lundi	3
4	20050104	2005-01-04	3	Tuesday	Martes	Mardi	4
5	20050105	2005-01-05	4	Wednesday	Miércoles	Mercredi	5
6	20050106	2005-01-06	5	Thursday	Jueves	Jeudi	6
7	20050107	2005-01-07	6	Friday	Viernes	Vendredi	7
8	20050108	2005-01-08	7	Saturday	Sábado	Samedi	8
9	20050109	2005-01-09	1	Sunday	Domingo	Dimanche	9
10	20050110	2005-01-10	2	Monday	Lunes	Lundi	10
11	20050111	2005-01-11	3	Tuesday	Martes	Mardi	11
12	20050112	2005-01-12	4	Wednesday	Miércoles	Mercredi	12
13	20050113	2005-01-13	5	Thursday	Jueves	Jeudi	13

## 3.5.22 Date Dimension Justification

---

- It can be difficult to understand why you would need a database table to represent dates
  - ◊ One point is that some information like fiscal year or holiday information might be different for each organization, or even for the same day!
    - \* The fourth of July for a US sale may be recorded as a holiday
    - \* The fourth of July for a UK sale may be recorded as not a holiday
- Another point is that by creating these columns, we can now create indexes on them that would make filtering against them more efficient
  - ◊ All “Tuesday” data will be grouped together in an index on “Tuesday”

## 3.5.23 Demonstration: Explore Adventureworks DW

---

- Look for an example of duplicate data in dimension tables
  - ◊ Compare SQL Queries that extract the same values from each of:
    - \* OLTP:
      - Adventureworks2017.Sales.Customer
      - Adventureworks2017.Person.Person
      - Adventureworks2017.Person.Address
      - Adventureworks2017.Person.StateProvince
    - \* OLAP
      - Adventureworks2017DW.DimCustomer
      - Adventureworks2017DW.DimGeography
  - Look for an example of Calculated data in dimension tables
  - Explore the Date Table
  - Create some Diagrams to visualize relationships

## 3.5.24 Discussion

---

- Do you have OLAP databases available within your organization?
  - ◊ How did those databases get designed?
  - ◊ How are they populated?
  - ◊ Where are they currently being consumed?
- How could your organization benefit from creating OLAP Databases?

## Section 3.6

### Data Cube Modeling

## 3.6.1 Data Cubes

---

- Data Cubes are usually built on top of OLAP Databases, and start with the same principals of Fact and Dimension Tables
- Data Cubes are sometimes built from OLTP Sources, but this often results in the duplication of work, and tools like Power BI might be more practical
- Data Cubes have many similarities to Power BI Data Models
  - ◊ Complex Calculations, Human oriented Data Types, and Attribute Hierarchies are a few
- Different than Power BI Data, this data is stored on the disk
- Microsoft's product to host Data Cubes is Microsoft SQL Server Analysis Services
- Reference:
  - ◊ [Cubes in Multidimensional Models | Microsoft Docs](#)

## 3.6.2 Cube External Reference Materials

---

- We don't cover Cubes in any detail here because it's a large subject worthy of its own class, but for the curious, Microsoft's Reference is the best next resource:
  - ◊ [Cubes in Multidimensional Models | Microsoft Docs](#)

## Section 3.7

# Power BI Modeling

## 3.7.1 Power BI Models

---

- Within the context of Power BI, models are most frequently called “Power BI Models”
  - ◊ They use the xVelocity Engine and are essentially the same as models in Excel’s Power Pivot, and in SQL Server Analysis Services
  - ◊ This engine is sometimes referenced as “Power Pivot”, which it is called in Excel
- Power BI Modeling happens in the Power BI Desktop Interface
  - ◊ Importing data using Power Query M
  - ◊ Configuring relationships using Model View
  - ◊ Configuring Data Types and preferences
  - ◊ Writing DAX Expressions

## 3.7.2 Table Modeling

---

- Ideal Power BI Models take on a similar shape to OLAP databases
  - ◊ Fact Tables are frequently referred to as Data Tables
  - ◊ Dimension Tables are frequently referred to as Lookup Tables

### 3.7.3 Star and Snowflake Schema Rules

---

- It is up to you to enforce Star and Snowflake Schema Rules
- Most difficult Data Analysis problems are rooted in a flawed model
- Do not directly relate a Data Table with another Data Table
  - ◊ Data Tables should be related only to Lookup Tables
- Do not directly relate a Lookup Table with another Lookup Table unless it follows snowflake rules
- Relationships between Data and Lookup Tables should be Many (data records) to One (Lookup Record)

## 3.7.4 Power BI Model Relationships

---

- Relationships in Power BI Models are defined explicitly
  - ◊ Relationships have
    - \* Cardinality (explicitly)
    - \* Filter Direction (explicitly)
    - \* Existence (implicitly)

## 3.7.5 Data Tables

---

- Synonymous with Fact Tables
- Contain Measures - numeric values that will be summarized
- A record generally describes an event, a thing that happened
- The Center Table of a Star Schema

## 3.7.6 Data Table Naming

---

- Data Tables are best named with Plural names representing the thing that happened, such as Sales, Purchases, Shifts, Transfers, Logins, ArticleViews
- Do not prefix tables with Fact, it ends up looking weird on reports and does not work as well with Natural Language Queries and Q&A

## 3.7.7 Lookup Tables

---

- Essentially the same as Dimension Tables
- Contain Descriptive attributes that describe facts
- Are ways by which you would group data to look at fact aggregates
- A record generally describes a noun
- The outer table of a Star Schema

## 3.7.8 Lookup Table Naming

---

- Lookup Tables tend to work better if they are named with non-plural names, for example Product, Employee, Currency, and Customer all make good Lookup Table names
- Don't prefix these with Dim or Dimension, it makes reports look less elegant
- Non-plural names will work better with the Natural Language Queries feature of Power BI

## 3.7.9 Columns

---

- Columns are assigned Datatypes
- Columns can have Formatting Specified
- Columns can have Sort Order configured to point to a different column
- Columns can have default summarization types assigned
- Columns can be placed into Groups, and Bins

## 3.7.10 Column Names

---

- You would have a tough time getting 10 data engineers in a room and having any 2 of them completely agree when it comes to column naming
- There are some guidelines that will make your life a little easier when it comes to building powerful visualizations
- Avoid Ambiguity by adding table names to column names
  - ◊ Examples:
    - \* Supplier City
    - \* Order Ship-To City
    - \* Order Billing City
    - \* Customer City
- Remove unnecessary words like *Name*, or *Company* where they are obvious
  - ◊ Examples:
    - \* Customer Name: Donald Duck
    - \* Customer: Donald Duck
    - \* Shipping Company: ACME Logistics
    - \* Shipper: ACME Logistics

- Add words like Amount and Percent to qualify numeric values
  - ◊ Examples:
    - \* Discount Amount
    - \* Discount Percent
- Use spaces where a user would expect to see spaces
- Use words with all their vowels present
- Use correct business terminology
  - ◊ Examples:
    - \* **Net Sales Amount** is a generally accepted term
    - \* **Total Sales Value** is something that may sound fine at first but..
      - Total of what? At what level of grouping?
      - By Value, do you mean the amount of a Sale, such as a promotion?
        - Use Quick Measures and templates and suggestions found on-line for ideas
- Use Consistent naming between reports and throughout the organization
  - ◊ Reference a Wiki with a description of report terms used organizationally so that users can be sure they have a clear understanding of what a value represents
- Ask peers if they have ideas for better names!

## 3.7.11 Column Datatypes

---

- Numeric Types

- Decimal Number
- Fixed Decimal Number
- Whole Number

- Date / Time Types

- Date / Time
- Date
- Time
- Date / Time / Timezone
- Duration

- Text

- True / False

- Binary

## 3.7.12 Column Numeric Formatting

---

- All three numeric types can have Formatting configuration applied
  - ◊ Can be specified as a Currency (multiple currencies)
  - ◊ Can be specified as a *General* number
  - ◊ Can be specified as being a percentage
  - ◊ Can be specified to be displayed with thousands separators (commas)
  - ◊ Can be specified to be displayed in Scientific Notation
  - ◊ Can be specified to be rounded to a specific decimal place

## 3.7.13 Date / Time column Formatting

---

- Date / Time columns can be assigned preferred text formatting rules

WS

Date formats

\*3/14/2001 (Short Date)

\*Wednesday, March 14, 2001 (Long Date)

Wednesday, March 14, 2001 (dddd, mmmm d, yyyy)

March 14, 2001 (mmmm d, yyyy)

Wednesday, 14 March, 2001 (dddd, d mmmm, yyyy)

14 March, 2001 (d mmmm, yyyy)

3/14/2001 (m/d/yyyy)

3/14/01 (m/d/yy)

03/14/01 (mm/dd/yy)

03/14/2001 (mm/dd/yyyy)

01/03/14 (yy/mm/dd)

2001-03-14 (yyyy-mm-dd)

14-Mar-01 (dd-mmm-yy)

14/03/2001 (dd/mm/yyyy)

March 2001 (mmmm yyyy)

2001-03 (yyyy-mm)

March 14 (mmmm d)

01 (yy)

2001 (yyyy)

## 3.7.14 Column Data Categories

---

- Data Categorization can enable specialized Visualization User Interface behavior
- Options include:
  - ◊ Uncategorized
  - ◊ Geography Classifications:
    - \* Address
    - \* Place
    - \* City
    - \* County
    - \* State or Province
    - \* Postal Code
    - \* Country
    - \* Continent
    - \* Latitude
    - \* Longitude
  - ◊ Hyperlink Classifications:
    - \* Web URL
    - \* Image URL
  - ◊ Barcode

## 3.7.15 Measure Summarization

---

- When a column from a Data Table is added to a Visualization, that column's value cannot be displayed directly because there is more than one value that must be turned into a single value
- We turn many values into a single value by Summarizing them together

- There are multiple forms of Aggregations

- ◊ Don't Summarize

- \* Indicates that this value does not make sense to be displayed

- ◊ Sum

- \* All of the values added together

- ◊ Average

- \* The Average, or Mean, of the values in the set

- ◊ Minimum

- \* The smallest value found in the set

- ◊ Maximum

- \* The largest value found in the set

- ◊ Count

- \* The Total number of Non-Blank values

- ◊ Count (Distinct)

- \* The total number of unique values found, not including Blank

- The Default Summarization, is Sum

## 3.7.16 Column Sort Orders

---

- Columns can have a Sort By column defined
- Examples of this being useful include:
  - ◊ Sorting a Display Name that contains a First and Last name, by a separate Last Name column
  - ◊ Sorting a Month Name column by a Month Number column so that January appears before July
  - ◊ Sorting discretized data like Infant, Child, Teen, Young Adult, Adult, Senior based on their logical sort order

## 3.7.17 Changing Default Summarization

---

- The Default Summarization for Numeric Columns is Sum
- This works great for Quantities, Amounts, etc.
- Measures like Popularity that contains a 0-5 value would not make sense to Sum
  - ◊ An Average would be a better Aggregation for this column
- This controls the implicit measures created for each column when selecting them for a visualization
  - ◊ You can override this within the visualization
- You can also add more measures explicitly to the model
  - ◊ A Year column may not have an obvious default, but two custom measures might be created using Minimum and Maximum with better names:
    - \* First Year
    - \* Most Recent Year
- We will cover measures in detail when we cover DAX

## 3.7.18 Column Grouping and Binning

---

- Certain visualizations benefit from Grouping columns together, or Binning columns together
- [Use grouping and binning in Power BI Desktop - Power BI | Microsoft Docs](#)

## 3.7.19 Navigational Hierarchies

---

- Navigational Hierarchies can be created that specify a drill-down order to be added to visualizations
- Some Hierarchies are obvious and follow natural parent child relationships
  - ◊ Year > Quarter > Month > Day
  - ◊ Country > Region > County > City
- Some Hierarchies do not have as natural relationships, this are frequently ordered by those with the fewest options to the most options, but ultimately depends on your visualization objectives
  - ◊ Gender > Marital Status -> Age Bracket
  - ◊ Category -> Size -> Color

## 3.7.20 Model Calculations

---

- Similar to Data Cubes, Power BI Models can contain advanced calculations
  - ◊ These include
    - \* Calculated Columns
    - \* Calculated Tables
    - \* Measures
    - \* Row Filters
- We will cover these in depth when we discuss Data Analysis eXpressions (DAX)

## 3.7.21 Demonstration

---

- Changing Column Names
- Changing Column Datatypes
- Changing Column Formatting
- Changing Sort Order Configuration
- Changing Default Aggregation

## Section 3.8

### Data Transformation

## 3.8.1 Transformation Pipelines

---

- When building a new model, transformation happens in stages
- In Power BI transformation is done with:
  - ◊ Power Queries
  - ◊ Configuration through the Graphical User Interface
  - ◊ Adding DAX Expressions

## 3.8.2 Transformation Order

---

- The order that transformations takes place might vary from case to case, but generally you'll want to do thinks in this order:
  1. Bring in RAW data to a staging table
  2. Remove the columns you won't need
  3. Remove or repair data that does not conform to column types
  4. Remove data that is out of the scope of analysis
  5. Define the column types
  6. Define Column Names
  7. Filling Unknowns with Lookups
  8. Adding Row Level Calculations
    - a. Add row level totals
    - b. Add formatted variations
    - c. Pulling Parent Data into Children
    - d. Pulling Child Aggregates into Parents

### 3.8.3 Cleaning Column Data

---

- Address Correction Services
- Correcting City Spellings
- Data Abnormalities:
  - ◊ How many years have you been teaching?
    - \* 4
    - \* 16
    - \* 12 year
    - \* 2011

## 3.8.4 Filling Unknowns with Lookups

---

- Pulling in Weather Data
- Pulling in Geological detail
  - ◊ How many feet above sea level?
  - ◊ What is the population of this region?

## 3.8.5 Adding Formatted Variations

---

- You might want to add a “Display Name” column from a First and Last name

## 3.8.6 Pulling Parent Data into Children

---

- You may want the Products Table to contain the Category Name instead of referencing a separate Category Table

## 3.8.7 Pulling Children Aggregates into Parents

---

- You may want an **Order** table to have an **[Order Total]** field
- You may want an **Employee** Table to have a **[Number of Sales Regions]** field

## 3.8.8 You may want to create Discretized Fields

---

- You might transform the person's birthdate into one of the following:
  - ◊ Free Child (3 and younger)
  - ◊ Child (4-12)
  - ◊ Teen (13-18)
  - ◊ Young Adult (19-21)
  - ◊ Adult (21+)
  - ◊ Senior (60+)

## 3.8.9 Discussion

---

- What are some examples of transformations that you’re likely to do of your organization’s data?
- What are some ways that you can think of to make this whole process of transformation easier?

## **Section 3.9**

### **Exercises**

# Exercise 3.1 Northwind Modeling – Exploring the Northwind Database

---

## Introduction

The Northwind Database is a sample database provided by Microsoft. The database was originally created for Microsoft SQL Server 2000. You can find the database online at this address:

<https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs>

This database contains data from a fictional company named Northwind Traders.

This is an OLTP database that has been normalized. Normalization has separated data into many different tables and eliminated as much duplicated data as possible.

In a larger Organizational BI Project, you might go through the process of creating a more ideal OLAP database to drive data analytics. Creating an OLAP database is a time consuming process, and this set of exercises is designed to show you how Power BI can be used to simulate much of the work that would go into creating an OLAP database, saving a lot of time.

In real-world projects, it makes sense to take a little bit of time to research the source data that we'll be working with. As we do this research keeping records of our findings will help keep us organized and effective.

This exercise will give you the opportunity to get to know the Northwind database and make some notes that we will be using in later exercises to create a useful Power BI Model.

## Exploring Northwind

1. Open SQL Server Management Studio.
2. When the Connect to Server dialog appears, enter the appropriate values to connect to the Database Engine that hosts the Northwind Database.
3. In the Object Explorer expand the tree to navigate to:  
Databases -> Northwind -> Tables
4. Observe the tables provided in the database.
  - \* Take a moment to carefully consider each table.

- \* What do you believe a record in the table represents?
5. We wrote a query to report the total number of records in each table. Being a sample database, there are not a lot of records.
- | TableName                    | RowCount |
|------------------------------|----------|
| [dbo].[Categories]           | 8        |
| [dbo].[CustomerCustomerDemo] | 0        |
| [dbo].[CustomerDemographics] | 0        |
| [dbo].[Customers]            | 91       |
| [dbo].[Employees]            | 9        |
| [dbo].[EmployeeTerritories]  | 49       |
| [dbo].[Order Details]        | 2155     |
| [dbo].[Orders]               | 830      |
| [dbo].[Products]             | 77       |
| [dbo].[Region]               | 4        |
| [dbo].[Shippers]             | 3        |
| [dbo].[Suppliers]            | 29       |
| [dbo].[Territories]          | 53       |
6. Note the ratio of records between tables. Is this ratio consistent with what you thought the data represented?
7. Note that the CustomerCustomDemo and CustomerDemographics tables do not contain any records. For this reason, we will not be using these two tables in our analysis.
8. Expand the Keys folder under the table in Object Explorer
- \* What is the Primary Key of this table?
  - \* What are the Parent Tables of this table?  
(Hint: What tables do foreign keys reference?)
9. Expand the Columns folder under the table in Object Explorer

10. Take a moment to carefully consider each column

- \* What does a value in this column likely represent?
- \* What values might you see in this column?
- \* Is this column a part of the Primary key to this table?
- \* Is this column a part of a foreign key that points to a record in another table?
- \* Would this column be valuable to bring into our Power BI Model?
- \* Yes or no?
- \* Why?
- \* Do you believe this column is appropriately named for Data Analysis purposes?
- \* If not, what might be a more appropriate name?
- \* Does this column represent what might become a Measure in our Power BI Model?
- \* Does this column represent what might become a descriptive attribute in a Dimension or Lookup table in our Power BI Model?
- \* How might this column be formatted in a Power BI Model?
- \* Can you think of any calculations where this field may become used?

11. Use SQL Server Management to Create a Database Diagram for the Northwind Database, adding every table except for CustomerCustomDemo and CustomerDemographics.

12. Take a few moments to arrange the tables visually in a way that makes sense to you. Try to avoid any relationship lines overlapping or crossing over other tables.

13. Save your Diagram. If you are using a shared database, prefix the name of the diagram with your initials to help avoid naming conflicts with other students.

Planning our transformations: Keep it or Drop it?

14. Using Microsoft Excel, open the file named

{LabFiles}\StarterFiles\ExploreNorthwind\NorthwindTablesAndColumns.xlsx

15. Save the file as

{LabFiles}\MyWork\NorthwindModeling\NorthwindTablesAndColumns\_Step\_00\_YourName.xls

16. Review the following guidelines for column inclusion

- \* Any Primary Key columns will need to stay, keep those.
- \* Any Foreign Key columns that point to a table in the model will need to be used to define a relationship between tables in the model, keep those.
- \* Any column that represents a name may be useful as we're likely to want to aggregate on that, and see the value in reports. Keep those.
- \* Any column that is a longer descriptive column like Description is probably not going to be any more useful than the name at a summary level. Remove those.
- \* Any column that represents binary data, like a photo, will not be used in this project, remove those.
- \* When it comes to geography, we will keep larger grains like Region, Country, Postal Code, or Zip Code, but we do not need to keep columns that store a street address like Address because we will not be aggregating at that granularity of geography.
- \* Columns that contain dates that we would want to report on, such as a Ship Date or Order Date are important, keep those.
- \* Columns that contain Dates like Birth Date or a Hire Date might be useful as we can compute an age or list employees based on the duration they have been at the company, keep those.
- \* Columns that contain dates that are there for auditing purposes such as Last Updated On are not useful to us, remove those.
- \* Columns that represent phone numbers, email addresses, and other ancillary contact information is not useful to us, remove those.

17. Pass through the spreadsheet once, and fill out the following columns

using the guidelines listed above:

- \* Primary Key?
- \* Foreign Key?
- \* Keep it?

18. Save the file as

{LabFiles}\MyWork\ExploreNorthwind\NorthwindTablesAndColumns\_Step\_01\_YourName.xlsx

Planning our Confirmation: Changing Column Names

19. Creating intuitive names will make down stream modeling and visualizing considerably easier, let's take a moment to consider new names for each of our columns.

20. Here are a few guidelines to consider when naming things:

- \* AddSpacesBetweenWordsSoThatYourReportsDoNotLookLikeThis, although this rule would not apply to Key columns, such as ProductID, as these would not be displayed in reports.
- \* Give most columns a name that is unique across all tables to remove ambiguity
  - If the Product Table has a column named “Name”, and the Category Table has a column named “Name”, and you bring those two columns together in a report, you’ll have the Name and Name. Not useful.
- \* At a first thought you might name these Product Name and Category Name, but you can actually further simplify these to be Product and Category, as these are the primary descriptors for these tables, they make natural labels.
- \* For columns like Country, provide context in its name, for example is it Customer Country, or Store Country or Supplier Country? This may feel redundant because nearly every column ends up getting prefixed with its table name, but in reports you often only see the column name by itself.
- \* Add the word Amount to numeric values that will be aggregated and represent money this will help to add clarity with columns like DiscountAmount and DiscountPercent.

21. Using the above guidelines, fill out the New Name? Column in the spreadsheet to suggest new names.

22. Save the file as

{LabFiles}\MyWork\ExploreNorthwind\NorthwindTablesAndColumns\_Step\_02\_YourName.xlsx

23. Pass through the spreadsheet one more time and identify candidates for

- \* Measures (numeric values that would be aggregated)
- \* Dimensional Attributes (descriptive Text that could be used to group aggregated values by).

24. Pass through the spreadsheet one more time, this time identifying if it will make sense to apply formatting to a numeric column, such as marking it as a currency or percentage.

25. Pass through the spreadsheet one more time, this time brainstorming for possible calculations that a column might later become a part of. Be creative here!

26. Write down any notes or concerns that you might have about your understanding of a column that you may want to further research in the notes column.

**27. Save the file as**

{LabFiles}\MyWork\ExploreNorthwind\NorthwindTablesAndColumns\_Step\_03\_YourName.xlsx

**28. Email this spreadsheet to your instructor! Please use the subject heading “*Working with Power BI: NorthwindTablesAndColumns*”**

## Module 4

# Power Query GUI Experience

## Section 4.1

### Power Query

## 4.1.1 Power Query

---

- Power Query is used to bring data into Power BI
  - ◊ Power Query was designed to be approachable to users who do not have any coding experience
  - ◊ It was designed to support a diverse set of data sources
  - ◊ There are over 700 transformations
- Power Query Editor is a Graphical User Interface
  - ◊ Wizards and Dialogs can be used to bring data in from a data source
  - ◊ Various tooling on the Ribbon can be used to perform basic transformations to data as it is brought in, before being made available to the Power BI Model
- Power Query Formula Language M is the underlying code that describes where data comes from and what happens to it
- The Advanced Editor can be used to write your own Expressions

## 4.1.2 Power Query Steps

---

- Power Queries are broken down into steps
  - ◊ Each of which has a name and performs a transformation
  - ◊ Some steps, such as changing Data Types, can operate on multiple columns at the same time
- Together the steps form a simple Pipeline

## 4.1.3 Power Query Automates Manual Processes

---

- Excel has been used to clean and filter data prior to importing it into reporting solutions
- When presented with a new Excel file and a new set of data, users had to manually repeat the steps they performed to clean and prepare data
- Power Query serves to automate this kind of data preparation work
- Power Query can be thought of as a macro language
- When your source data changes, you can Refresh Data to fetch updates and re-run each of the steps again

## 4.1.4 Power Query Uses

---

- Power Query is great for
  - ◊ Extracting Data from a variety of sources
  - ◊ Shaping data in a variety of ways
  - ◊ Merging data
  - ◊ Transforming data
  - ◊ Cleaning data
  - ◊ Calculating or Manipulating values
- Not the best tool for
  - ◊ Auditing or logging
  - ◊ Strict data type enforcement
  - ◊ Non-linear data prep
  - ◊ Optimized for ease-of-use over speed
  - ◊ Starts to break down when the number of steps exceeds 20
  - ◊ Performance or Scalability
- Other products might be considered for more complex ETL pipelines
  - ◊ Microsoft SQL Server Integration Services is one popular option

## 4.1.5 Products that use Power Query

---

- Power Query is used in a variety of Microsoft Products

<https://docs.microsoft.com/en-us/power-query/power-query-what-is-power-query>

## Where can you use Power Query?

The following table lists Microsoft products and services where Power Query can be found.

Product	M engine <sup>1</sup>	Power Query Desktop <sup>2</sup>	Power Query Online <sup>3</sup>	Dataflows <sup>4</sup>
Excel for Windows	Yes	Yes	No	No
Excel for Mac	Yes	No	No	No
Power BI	Yes	Yes	Yes	Yes
Power Apps	Yes	No	Yes	Yes
Power Automate	Yes	No	Yes	No
Power BI Report Server	Yes	Yes	No	No
Azure Data Factory	Yes	No	Yes	Yes
SQL Server Integration Services	Yes	No	No	No
SQL Server Analysis Services	Yes	Yes	No	No
Dynamics 365 Customer Insights	Yes	No	Yes	Yes

<sup>1</sup> M engine	The underlying query execution engine that runs queries expressed in the Power Query formula language ("M").
<sup>2</sup> Power Query Desktop	The Power Query experience found in desktop applications.
<sup>3</sup> Power Query Online	The Power Query experience found in web browser applications.
<sup>4</sup> Dataflows	Power Query as a service that runs in the cloud and is product-agnostic. The stored result can be used in other applications as services.

## 4.1.6 Dataflows

---

- We will be working with Power Query within the context of Power BI Desktop, where the tooling is tightly coupled with a project
- A dataflow decouples the data transformation layer from the modeling and visualization layer in a Power BI Solution
- Dataflows utilize Power Query in an Application Agnostic way, on the cloud
- Power BI Desktop can consume cloud hosted Dataflows

<https://docs.microsoft.com/en-us/power-query/dataflows/overview-dataflows-across-power-platform-dynamics-365>

## 4.1.7 Power Query Formula Language M

---

- Much of what Power Query accomplishes can be performed by using the Graphical User Interface (GUI): Power Query Editor
- As you interact with the Power Query Editor, it is automatically writing queries using Power Query Expression Language
- Power Query Formula Language is informally known as M
  - ◊ We will refer to it as M, or M Language throughout this course
  - ◊ While researching, we found Microsoft to be very inconsistent with how they work M into their documentation
- Understanding how to write your own M Language queries will enable you to perform more advanced tasks with Power Query
  - ◊ Often you'll start with Queries written by Power Query Editor's GUI, and then use the advanced editor to modify them to perform more advanced tasks

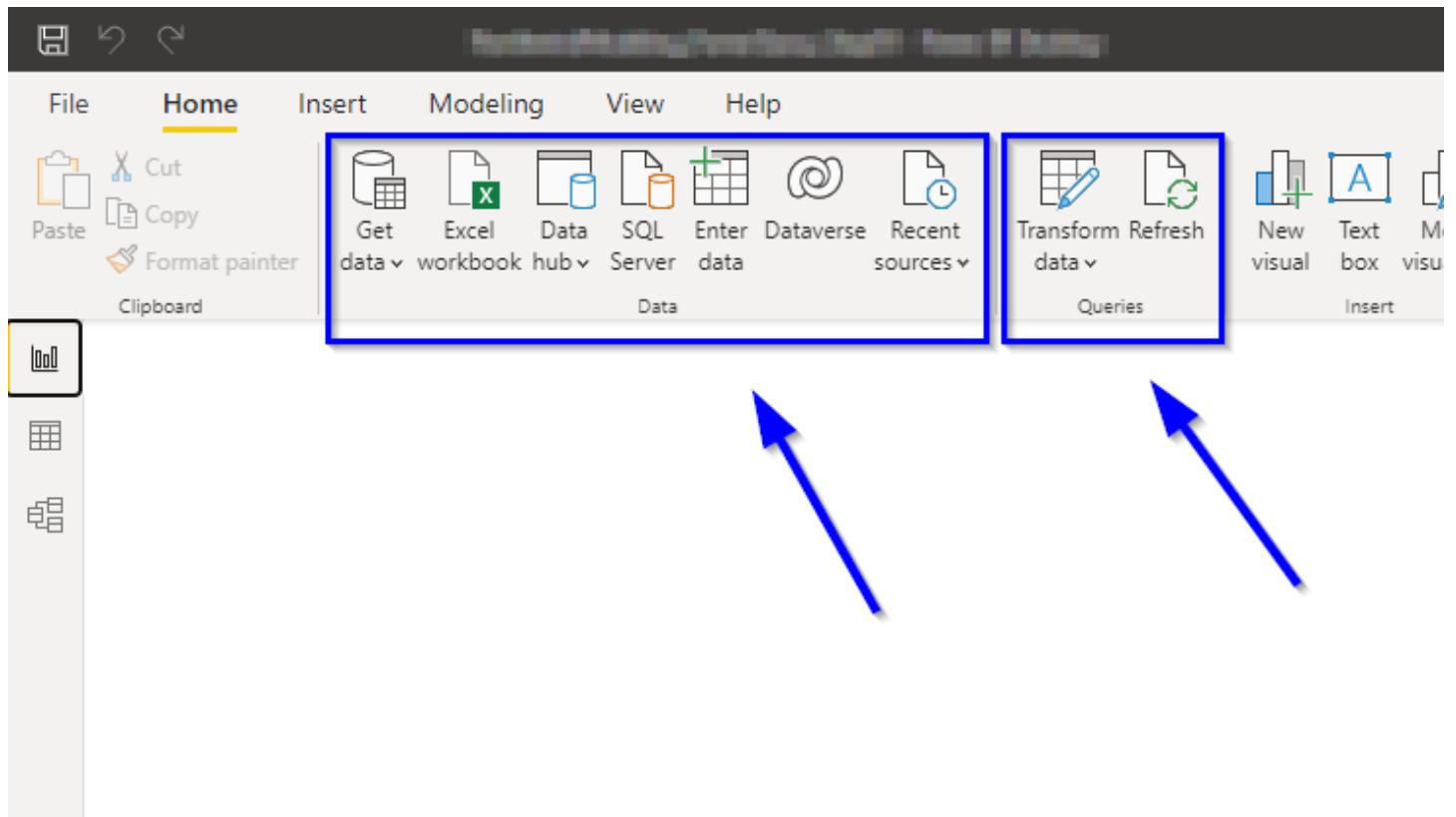
- This module focuses on learning to navigate around Power Query Editor, and using the GUI to get data and perform common transformations
  - ◊ We will cover M Language in the next module.

## Section 4.2

### Power Query Editor Tour

## 4.2.1 Getting to the Power Query Editor

- The Data and Queries sections of the Home Ribbon contains shortcuts related to Power Query



## 4.2.2 Supported Data Sources

---

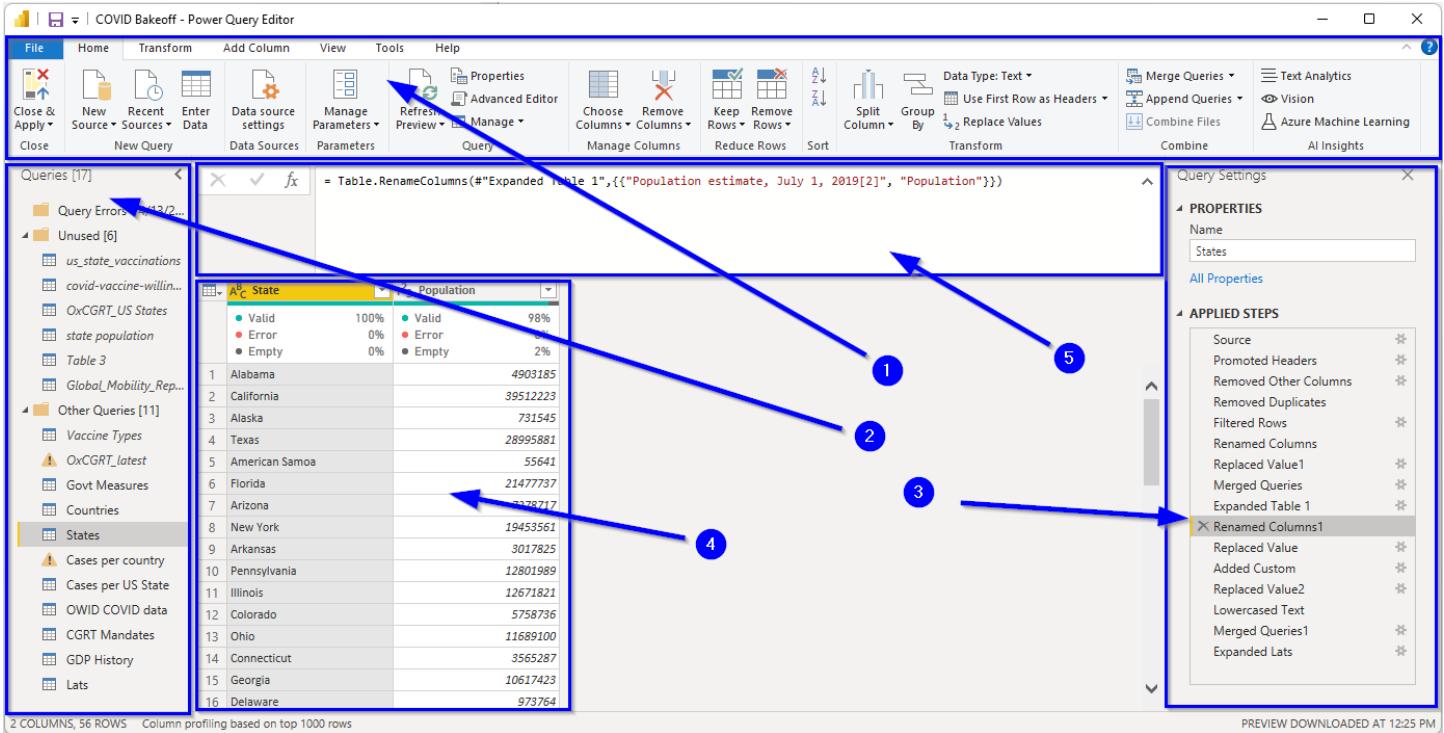
- The Data Sources supported by Power Query can be categorized as follows:
  - ◊ Databases
    - \* On-premises or cloud accessible
    - \* Opportunity to write Database native queries
  - ◊ Files
    - \* Stored locally or in OneDrive
  - ◊ Web
    - \* Url Sources
    - \* OData feeds
  - ◊ Azure
    - \* Cloud Databases
    - \* Blob (file) storage
    - \* Services
  - ◊ Online Services
    - \* Software-as-a-Service sources
  - ◊ Power Platform
    - \* Dataverse
    - \* Datasets
    - \* Dataflows

## 4.2.3 Power Query Editor

---

- The Power Query Editor is the Graphical User Interface used to edit queries

## 4.2.4 Power Query Editor: Image Tour



1. The Ribbon Pane contains many useful shortcuts to contextually bring in or transform data
2. The Queries List Pane has a list of all queries already in your project
  - Queries can be organized into folders if you have a lot of them
3. The Query Settings Pane has a list of Steps that make up your Query
  - Steps can be reordered or removed in this pane
  - A gear appears next to some steps that enable you to modify it using a custom graphical dialog
4. The body region allows you to preview the data after the selected step has been applied
  - You can also right click on columns to see a context menu, double click to rename, and drag to reorder
5. The Expression Editor allows you to edit the code based expression behind the selected step
  - You can expand this from a single line to multiple lines to make editing easier

## 4.2.5 Demonstration: Power Query Editor Tour

---

- Let's pull in some data and shape it!
  - ◊ Getting Data
  - ◊ Removing Columns
  - ◊ Renaming Columns
  - ◊ Changing Column Types
  - ◊ Formatting Columns
  - ◊ Creating Calculated Fields

## 4.2.6 Demonstration: Exploring Sample Project Queries

---

- Open up and explore the Power Query Editor and demo navigating around using the following files:

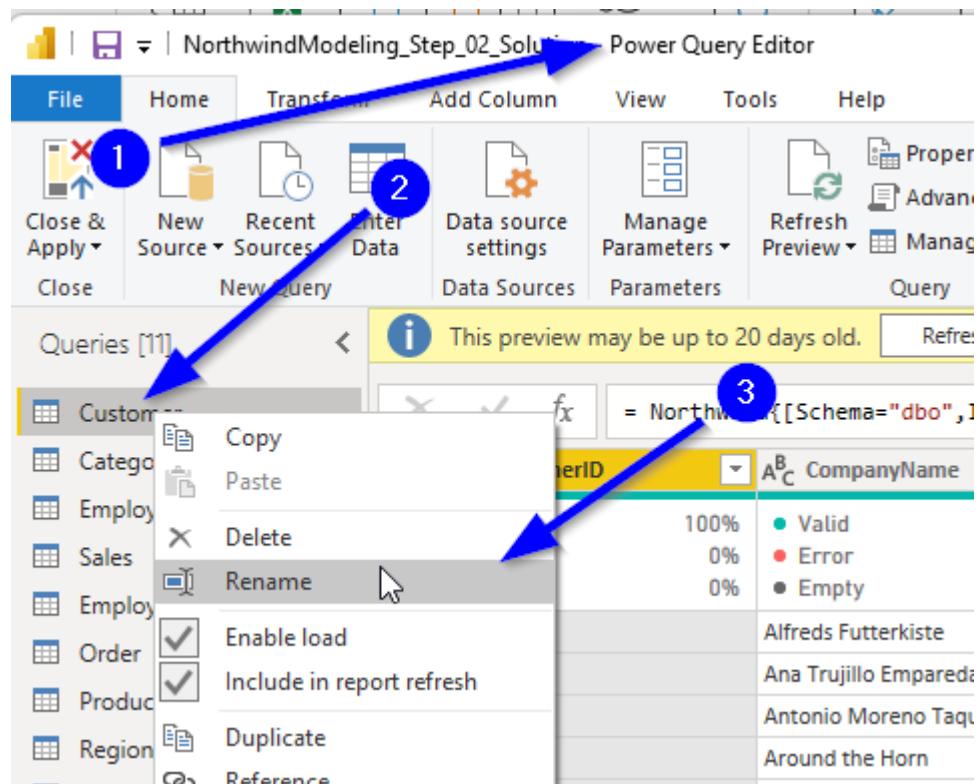
{LabFiles}\StarterFiles\PowerBISamples\Sales & Returns Sample v201912.pbix

{LabFiles}\StarterFiles\PowerBISamples\COVID Bakeoff.pbix

## Section 4.3

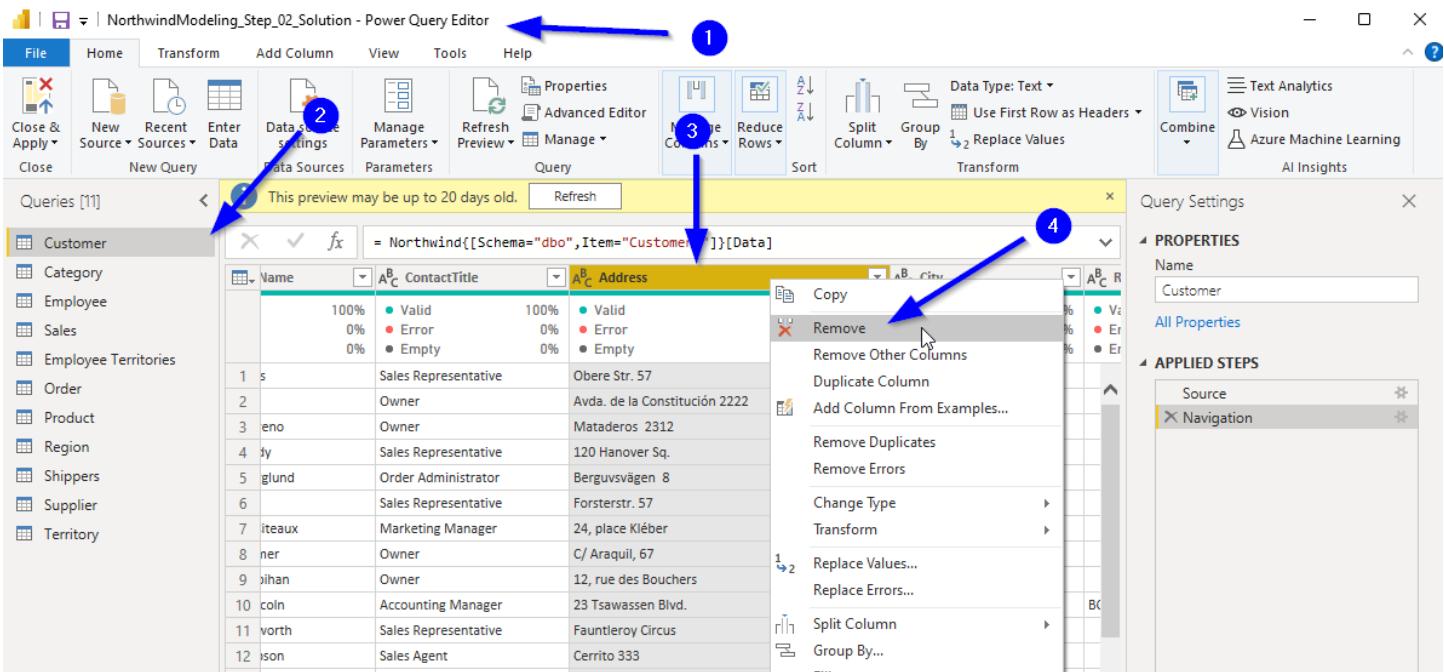
### Simple Transformations

## 4.3.1 Changing a Query/Table Name



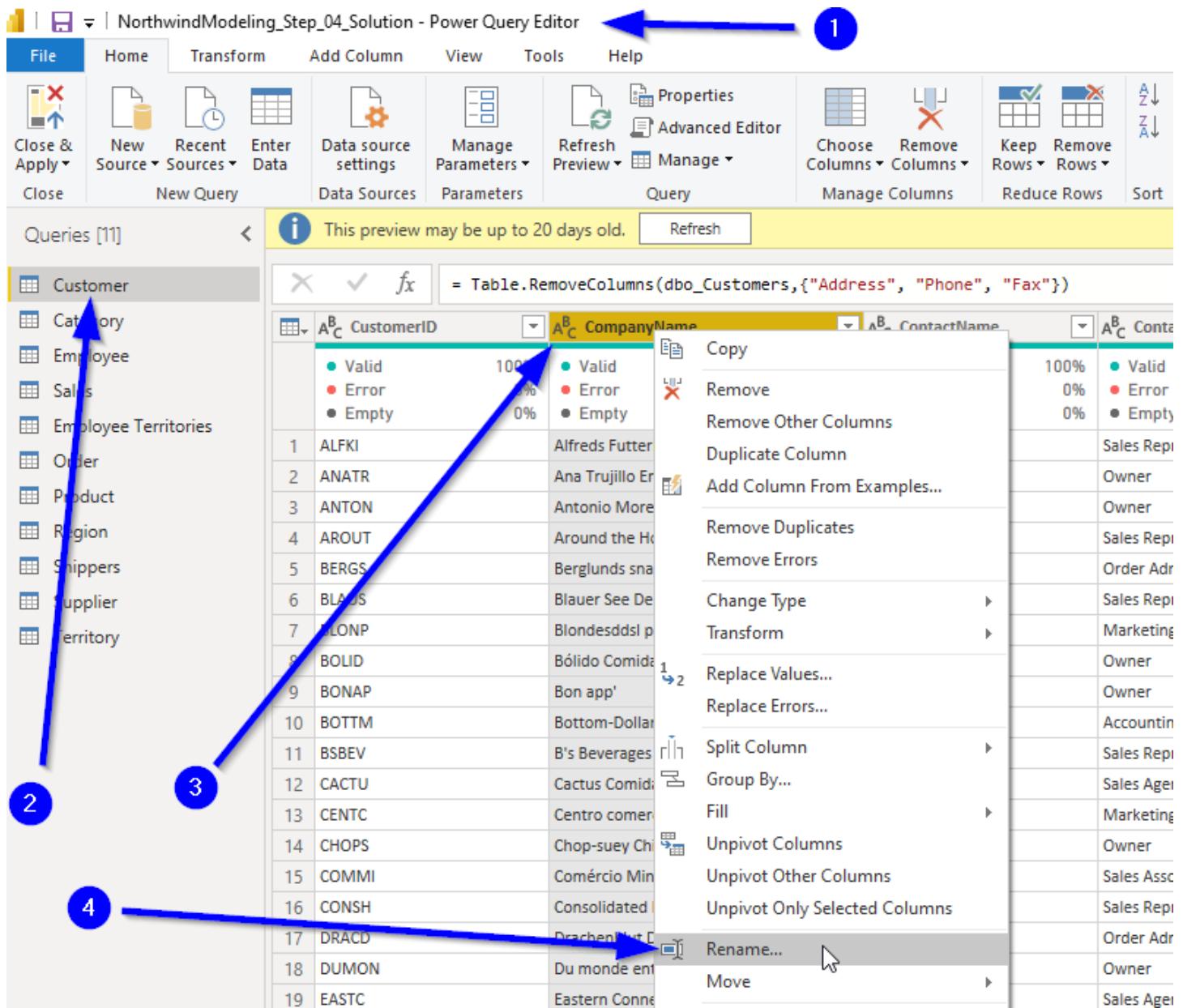
1. Using Power Query Editor
2. Select the Query (table) that you want to change, and **Right Click** to open the context menu
3. Select Rename

## 4.3.2 Removing Columns



1. Using Power Query Editor
2. Select the Query that contains the Column you want to remove
3. Select and then right click on the column you want to remove, to open the context menu
4. Select Remove

## 4.3.3 Renaming Columns



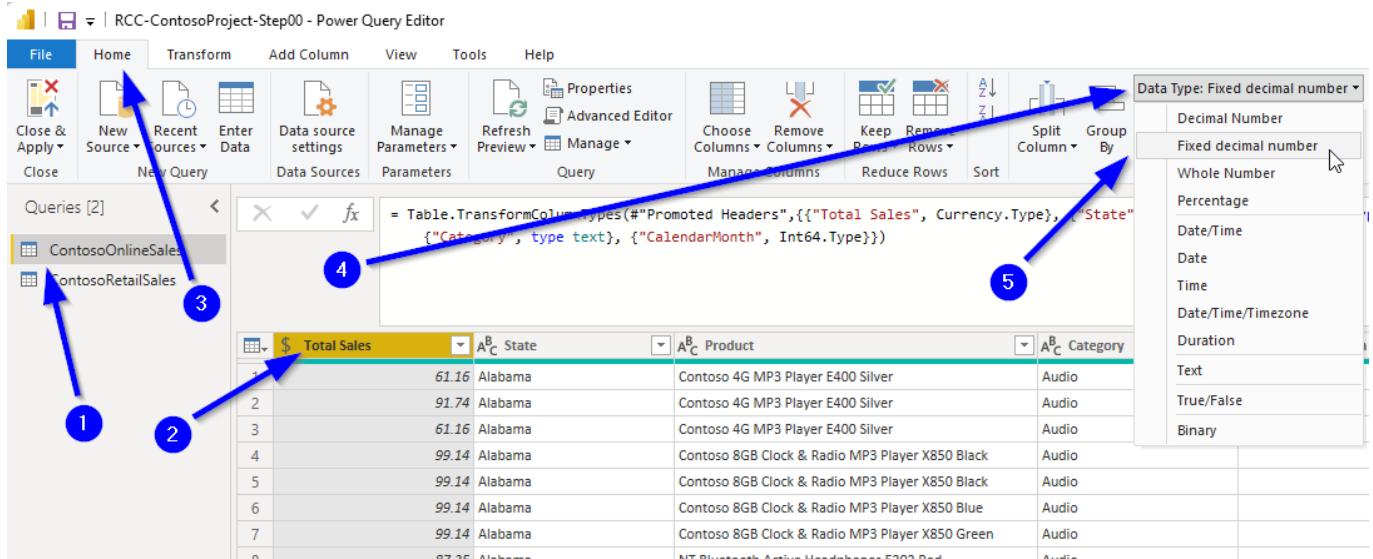
1. Using Power Query Editor
2. Select the Query that contains the Column that you wish to rename
3. Select and then right click on the column you want to rename, to open the context menu
4. Select Rename

## 4.3.4 Changing a Columns Data Type

---

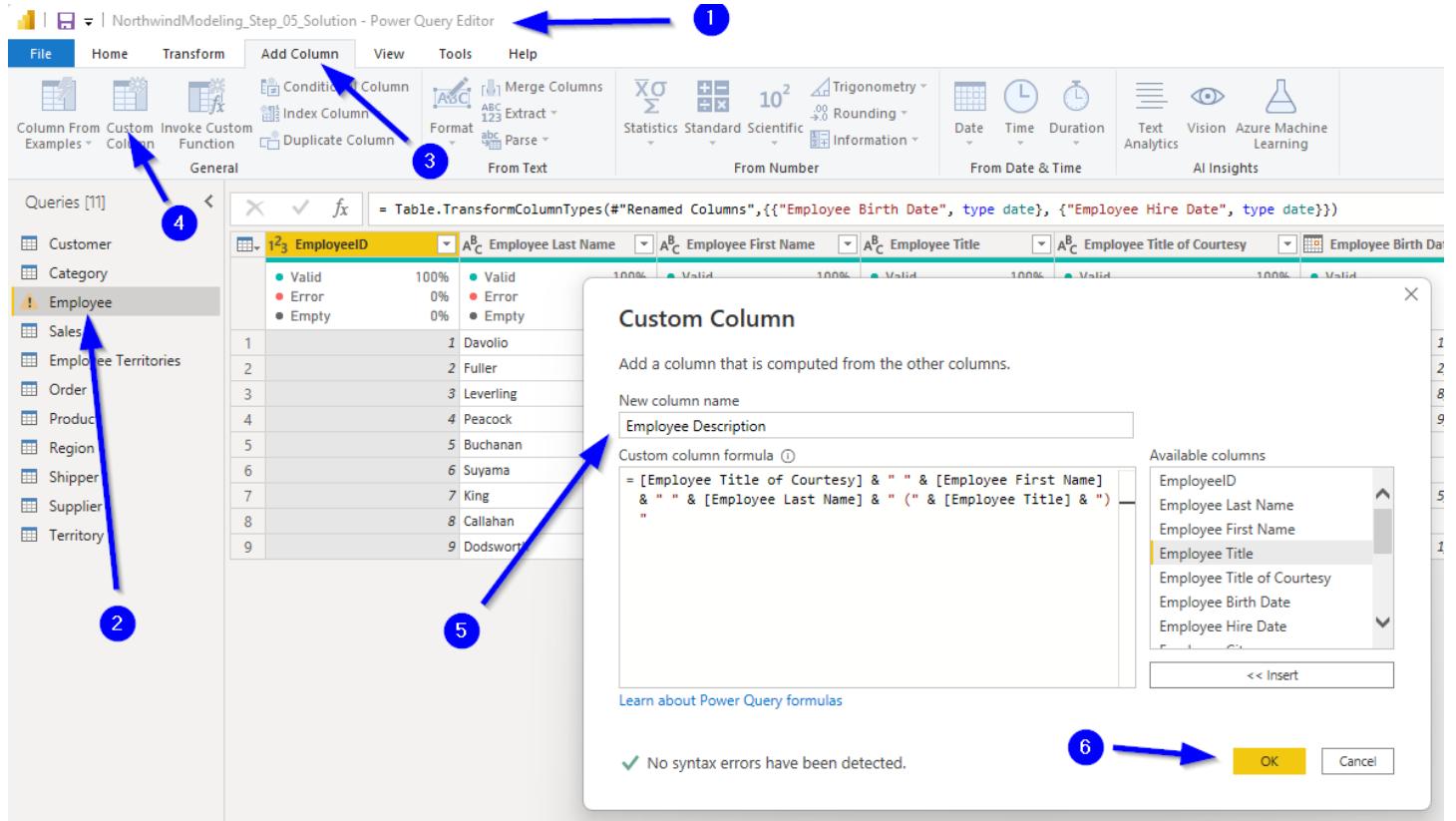
- Selecting the appropriate Data Types for your columns is important!
- Data Types will determine what Transformations will work on values
  - ◊ Numeric Aggregations like Sum or Average will only work on Numeric Types
  - ◊ Transformations like Split Column will only work on Text Columns
  - ◊ Sometimes the User Interface will hide or deactivate transformations that are not relevant to the selected columns current data type
- Some Data Sources, such as SQL Server, carry information over about the correct data type
  - ◊ There will be little work to do here
  - ◊ It is still worth reviewing, as some databases may surprisingly use unexpected types
- Data Sources such as Text Files are just guessing data types
  - ◊ These should be reviewed column by column to confirm the type is correct

## 4.3.5 Changing a Columns Data Type: Illustrated



5. Select the Query
6. Select the Column
7. Select the Home Ribbon
8. In the Transformation Region, select the Data Type dropdown
9. Select the desired Data Type

## 4.3.6 Adding Calculated Fields



## **Section 4.4**

## **Exercises**

# Exercise 4.1 Northwind Modeling – Importing and Transforming our Data with Power Query

---

## Introduction

In this exercise, we will be importing data from the Northwind Database into our Power BI Model, using Power Query.

You will notice that we won't need to be writing any code, the Graphical User Interface enables us to import and perform most common transformations.

We will be using the Spreadsheet that you created earlier. We had to put a lot of thinking and creative work into making that spreadsheet, but now that those decisions are made, we have a clear list of technical needs that we can apply using Power BI.

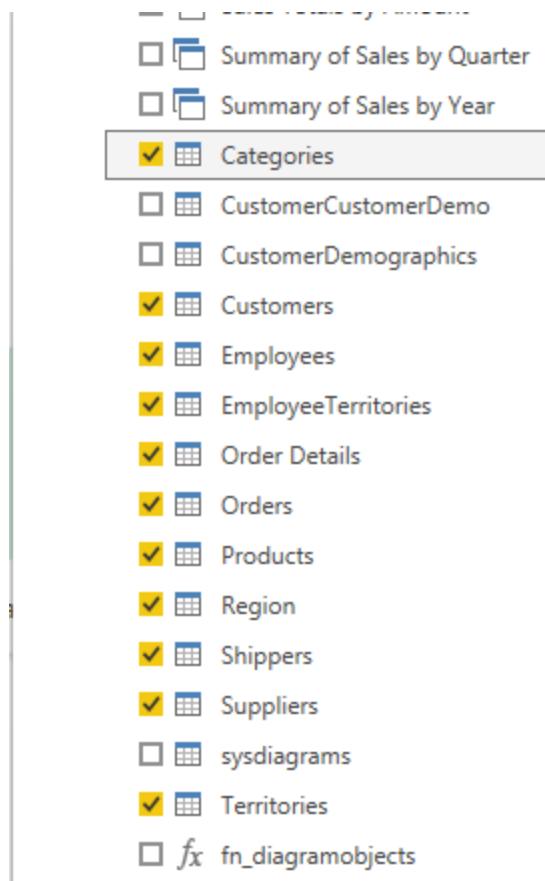
## Creating our Project

1. Open up a new Power BI Project
2. Save the new Empty Project at the following location:

{LabFiles}\MyWork\NorthwindModeling\_Step\_00\_YourName.pbix

## Importing our Raw Data

3. In the Home Tab of the Ribbon, in the Data Region, Choose Get data -> SQL Server
4. In the SQL Server database dialog, type the name of the SQL Server that contains the Northwind Database (if a port is required, follow the name with a colon and the port number), and select OK (Note: Leave the Database blank, and leave the Import option selected)
5. If necessary, provide security information to connect to the server.
6. In the Navigator dialog, Expand the Northwind Database and select each of the selected tables in the image below:



7. Choose Transform Data to open Power Query Editor.

8. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_01\_YourName.pbix

## Changing Table Names

9. Rename each of the Queries as follows, note the plurality of each new name. We're giving Data Tables plural names the describe the event, and Lookup tables non-plural names that describe the dimension / attributes they represent except where representing a many-to-many relationship:

Old Name	New Name
Customers	Customer
Categories	Category
Employees	Employee
Order Details	Sales
EmployeeTerritories	Employee Territories
Orders	Order
Products	Product
Region	Region

Shippers	Shipper
Suppliers	Supplier
Territories	Territory

10. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_02\_YourName.pbix

## Removing Unnecessary Columns

11. Open the following Excel file:

{LabFiles}\Solutions\NorthwindModeling\NorthwindTablesAndColumns\_Step\_03\_Solution.xlsx

12. Go through each Query and remove all the columns marked in this spreadsheet as columns we should not keep.

Note: When looking at the column lists you may notice columns at the end of each query that represent database relationships, you can ignore those for now, we'll look at them later.

13. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_03\_YourName.pbix

## Renaming Columns

14. Go through each Query and Rename each of the Columns that are marked in the spreadsheet with new names.

15. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_04\_YourName.pbix

## Assigning Column Types

16. Systematically go through each Query and Column and change the Data Type

- \* Remember to change Date/Time columns to Date instead of DateTime
- \* Discount Percent should be a Percentage
- \* Sales Quantity should be a Whole Number
- \* Any Currencies should be Fixed Decimal Point

17. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_05\_YourName.pbix

## Adding Calculated Columns

18. Add a Calculated Column to the Employees Table named **Employee Description**, use the following expression:

```
[Employee Title of Courtesy] & " " & [Employee First Name] & " " &  
[Employee Last Name] & " (" & [Employee Title] & ")"
```

19. Add a Calculated Column to the Employees Table named **Employee**, use the following expression:

```
[Employee Last Name] & ", " & [Employee First Name]
```

20. Add a Calculated Column to the Sales Table named **Discount Amount**, use the following expression:

```
[Sales Quantity] * [Unit Price] * [Discount Percent]
```

21. Set the Discount Amount column to be a Fixed Decimal Number

22. Add a Calculated Column to the Sales Table named **Sales Amount**, use the following expression:

```
[Sales Quantity] * [Unit Price] * ( 1 - [Discount Percent] )
```

23. Save your work as:

{LabFiles}\MyWork\NorthwindModeling\NorthwindModeling\_Step\_06\_YourName.pbix

## Exercise 4.2 Northwind Modeling – Modeling and Visualizing our new data

---

(Instructor demonstrates while students follow along)

- \* Use the Model view to position tables to reflect a snowflake like model
- \* Hide any Key Columns from the report view
- \* Use the Data view to review data and select column Formatting options
- \* Hide any tables that do not contain measures or dimensions from the report view
- \* Create some basic visuals, demonstrating the Table, Matrix, Slicer, and Column Graphs visualizations

## Exercise 4.3 Northwind Modeling – A first glance at Relationships and Collapsing Categories into Products

---

(Instructor demonstrates while students follow along)

- \* Review current Relationships
- \* Collapse Categories into Products with Power Query (use Foreign Key table field)
- \* Create a Products Navigational Hierarchy
- \* Demonstrate visuals using the Hierarchy

## Module 5

# Power Query Formula Language (M)

## Section 5.1

### Introducing M

## 5.1.1 M Language

---

- Learning M Language will help you to perform more advanced tasks that the GUI does not handle well
- After you learn M Language, you'll find that some work that can be done using the GUI can be done faster by writing M code
- The ability to Cut and Paste M can save a lot of time

## 5.1.2 Programming for non-programmers

---

- M will feel different than other query languages because it was designed with Excel power users in mind, not for programmers
- M's structure allows for expressions to be broken down into individual steps
  - ◊ This makes it easier to conceptualize a long list of tasks that will need to be performed
- M's evaluation engine determines each expression's dependencies and computes the order that expressions should be evaluated in
  - ◊ This makes it easier in the sense that the author does not need to worry as much about when to do things, but rather what needs to be done
  - ◊ This is similar to a spreadsheet

## 5.1.3 Power Query Imports Data

---

- Power Query controls what is and is not brought into the model
- The end result of Power Query is the starting point of the Power BI Data Model
- After data is imported, it can be further modeled and extended using Power BI Desktop and DAX
- Data brought into the model using Power Query remains stale until a refresh is executed

## 5.1.4 Query Folding

---

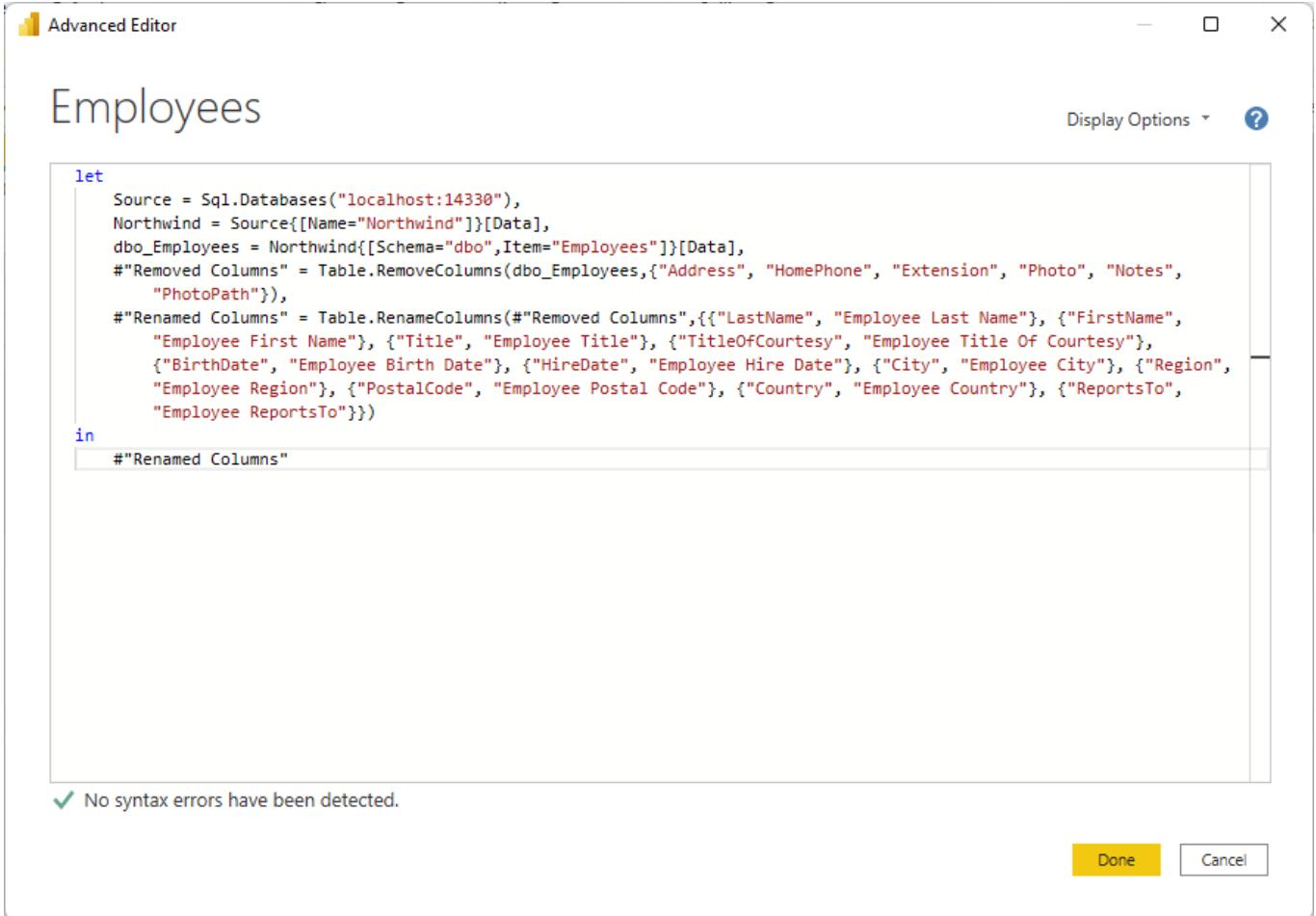
- Power Query utilizes a technique known as query folding, which pushes a lot of the transformation logic downstream for Get Data adapters that support it
- Data Sources like Microsoft SQL Server have work like filtering, exclusion of columns, joining, and grouping pushed into the SQL Query
- This means that much less data gets transferred into Power BI
- This allows data engines like SQL Server to utilize indexes to perform expensive operations like filtering or ordering data more effectively

## Section 5.2

### Dissecting a Let Expression

## 5.2.1 Let Expressions

- A let expression is the first thing that most people will notice when beginning to learn M
- The following query was written by Power Query Editor during the exercise where we prepared data from Northwind
  - ◊ Tip: Drop down display options in the advanced editor to enable word-wrap



The screenshot shows the Power Query Advanced Editor window titled "Employees". The code pane contains the following M code:

```
let
    Source = Sql.Databases("localhost:14330"),
    Northwind = Source[[Name="Northwind"]][Data],
    dbo_Employees = Northwind{[Schema="dbo",Item="Employees"]}[Data],
    #"Removed Columns" = Table.RemoveColumns(dbo_Employees, {"Address", "HomePhone", "Extension", "Photo", "Notes", "PhotoPath"}),
    #"Renamed Columns" = Table.RenameColumns(#"Removed Columns", {{"LastName", "Employee Last Name"}, {"FirstName", "Employee First Name"}, {"Title", "Employee Title"}, {"TitleOfCourtesy", "Employee Title Of Courtesy"}, {"BirthDate", "Employee Birth Date"}, {"HireDate", "Employee Hire Date"}, {"City", "Employee City"}, {"Region", "Employee Region"}, {"PostalCode", "Employee Postal Code"}, {"Country", "Employee Country"}, {"ReportsTo", "Employee ReportsTo"}})
in
    #"Renamed Columns"
```

The status bar at the bottom left indicates "No syntax errors have been detected." The bottom right features "Done" and "Cancel" buttons.

## 5.2.2 Expression naming

---

- Because named expressions are visible in the designer as steps, they will commonly contain spaces for readability
- When a variable name has spaces in it, place it in quotes and prefix it with a #
- Example

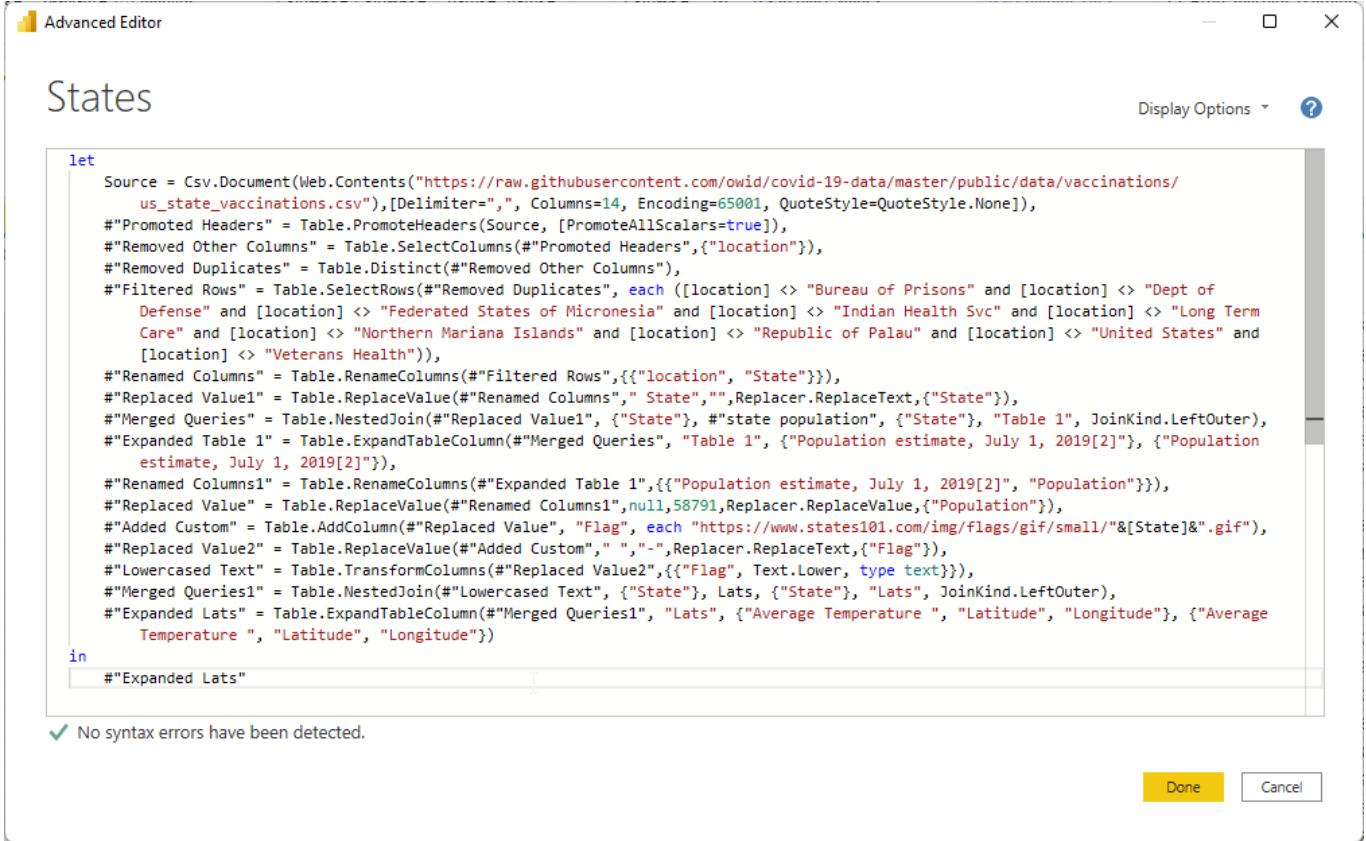
```
let
    NameOne = "This expression is bound to the name 'NameOne', which does
not contain spaces",
#"Name Two" = "This expression is bound to the name 'Another Name', which
does contain a space."
in
#"Name Two"
```

## 5.2.3 Formatting Expressions

---

- White space (space, tabs, carriage returns) can be added to your query text
- This can make your query significantly easier to read and write
- There is a fantastic resource at <https://www.powerqueryformatter.com/> that will format your queries considerably nicer than Power Query Editor does for you.

## 5.2.4 Formatting Before:



The screenshot shows the 'Advanced Editor' window in Power BI. The title bar says 'Advanced Editor'. The main area contains M code for transforming a CSV file into a Power BI table. The code includes steps like Csv.Document, Table.PromoteHeaders, Table.SelectColumns, Table.Distinct, Table.SelectRows, Table.NestedJoin, Table.ExpandTableColumn, Table.RenameColumns, Table.AddColumn, Table.ReplaceValue, and Table.TransformColumns. A note at the bottom says 'No syntax errors have been detected.' At the bottom right are 'Done' and 'Cancel' buttons.

```

let
    Source = Csv.Document(Web.Contents("https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/us_state_vaccinations.csv"),[Delimiter=",", Columns=14, Encoding=65001, QuoteStyle=QuoteStyle.None]),
    #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),
    #"Removed Other Columns" = Table.SelectColumns(#"Promoted Headers",{"location"}),
    #"Removed Duplicates" = Table.Distinct(#"Removed Other Columns"),
    #"Filtered Rows" = Table.SelectRows(#"Removed Duplicates", each ([location] <> "Bureau of Prisons" and [location] <> "Dept of Defense" and [location] <> "Federated States of Micronesia" and [location] <> "Indian Health Svc" and [location] <> "Long Term Care" and [location] <> "Northern Mariana Islands" and [location] <> "Republic of Palau" and [location] <> "United States" and [location] <> "Veterans Health")),
    #"Renamed Columns" = Table.RenameColumns(#"Filtered Rows",{{"location", "State"}}),
    #"Replaced Value1" = Table.ReplaceValue(#"Renamed Columns", " State", "", Replacer.ReplaceText,{"State"}),
    #"Merged Queries" = Table.NestedJoin(#"Replaced Value1", {"State"}, "#state population", {"State"}, "Table 1", JoinKind.LeftOuter),
    #"Expanded Table 1" = Table.ExpandTableColumn(#"Merged Queries", "Table 1", {"Population estimate, July 1, 2019[2]"}, {"Population estimate, July 1, 2019[2]}),
    #"Renamed Columns1" = Table.RenameColumns(#"Expanded Table 1",{{"Population estimate, July 1, 2019[2]", "Population"}}),
    #"Replaced Value" = Table.ReplaceValue(#"Renamed Columns1",null,58791,Replacer.ReplaceValue,{"Population"}),
    #"Added Custom" = Table.AddColumn(#"Replaced Value", "Flag", each "https://www.states101.com/img/flags/gif/small/"&[State]&.gif"),
    #"Replaced Value2" = Table.ReplaceValue(#"Added Custom", " ", "-", Replacer.ReplaceText,{"Flag"}),
    #"Lowercased Text" = Table.TransformColumns(#"Replaced Value2",{{"Flag", Text.Lower, type text}}),
    #"Merged Queries1" = Table.NestedJoin(#"Lowercased Text", {"State"}, Lats, {"State"}, "Lats", JoinKind.LeftOuter),
    #"Expanded Lats" = Table.ExpandTableColumn(#"Merged Queries1", "Lats", {"Average Temperature ", "Latitude", "Longitude"}, {"Average Temperature ", "Latitude", "Longitude"})
in
    #"Expanded Lats"

```

✓ No syntax errors have been detected.

Done Cancel

## 5.2.5 Formatting After

---

```
1 let
2     Source = Csv.Document(
3         Web.Contents(
4             "https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/us_state_vaccinations.csv"
5         ),
6         [Delimiter = ",", Columns = 14, Encoding = 65001, QuoteStyle = QuoteStyle.None]
7     ),
8     #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars = true]),
9     #"Removed Other Columns" = Table.SelectColumns(#"Promoted Headers", {"location"}),
10    #"Removed Duplicates" = Table.Distinct(#"Removed Other Columns"),
11    #"Filtered Rows" = Table.SelectRows(
12        #"Removed Duplicates",
13        each (
14            [location]
15            <> "Bureau of Prisons" and [location]
16            <> "Dept of Defense" and [location]
17            <> "Federated States of Micronesia" and [location]
18            <> "Indian Health Svc" and [location]
19            <> "Long Term Care" and [location]
20            <> "Northern Mariana Islands" and [location]
21            <> "Republic of Palau" and [location]
22            <> "United States" and [location]
23            <> "Veterans Health"
24        )
25    ),
26    #"Renamed Columns" = Table.RenameColumns(#"Filtered Rows", {"location", "State"}),
27    #"Replaced Value1" = Table.ReplaceValue(
28        #"Renamed Columns",
29        "State",
30        "",
31        Replacer.ReplaceText,
32        {"State"}
33    ),
34    #"Merged Queries" = Table.NestedJoin(
35        #"Replaced Value1",
36        {"State"},
37        {"state population",
38        {"State"},
39        "Table 1",
40        JoinKind.LeftOuter
41    ),
42    #"Expanded Table 1" = Table.ExpandTableColumn(
43        #"Merged Queries",
44        "Table 1",
45        {"Population estimate, July 1, 2019[2]"},
46        {"Population estimate, July 1, 2019[2]"}
47    ),
48    #"Renamed Columns1" = Table.RenameColumns(
49        #"Expanded Table 1",
50        {"Population estimate, July 1, 2019[2]"}))
```

## 5.2.6 Let expression sets

- A let expression allows a set of expressions each to be assigned a name
- Each expression can utilize the result of other expressions
- A final in statement finishes the set, and declares a final result for the expression
- Usually this is as simple as the name of an above expression
- Example:

```
1 let
2   Source = Sql.Databases("localhost:14330")
3   ,
4   Northwind = Source{[Name = "Northwind"]}[Data]
5   ,
6   dbo_Customers = Northwind{[Schema = "dbo", Item = "Customers"]}[Data]
7   .
8   #"Removed Columns" = Table.RemoveColumns(dbo_Customers, {"Address", "Phone", "Fax"})
9   ,
10  #"Renamed Columns" = Table.RenameColumns(
11    #"Removed Columns"
12    ,
13    {
14      {"CompanyName", "Customer"},
15      {"ContactName", "Customer Contact Name"},
16      {"ContactTitle", "Customer Contact Title"},
17      {"City", "Customer City"},
18      {"Region", "Customer Region"},
19      {"PostalCode", "Customer Postal Code"},
20      {"Country", "Customer Country"}
21    }
22  )
23 in
24  #"Renamed Columns"
```

## 5.2.7 Expressions in Let

- Each expression in a Let statement can reference the result of other expressions

```
1 let
2   Source = Sql.Databases("localhost:14330")
3   ,
4   Northwind = Source[Name = "Northwind"]{Data}
5   ,
6   dbo_Customers = Northwind{Schema = "dbo", Item = "Customers"}{Data}
7   ,
8   #"Removed Columns" = Table.RemoveColumns(dbo_Customers, {"Address", "Phone", "Fax"})
9   ,
10  #"Renamed Columns" = Table.RenameColumns(
11    "Removed Columns",
12    {
13      {"CompanyName", "Customer"},
14      {"ContactName", "Customer Contact Name"},
15      {"ContactTitle", "Customer Contact Title"},
16      {"City", "Customer City"},
17      {"Region", "Customer Region"},
18      {"PostalCode", "Customer Postal Code"},
19      {"Country", "Customer Country"}
20    }
21  )
22 in
23  #"Renamed Columns"
```

The diagram illustrates the execution flow of the Power BI Let statement. It shows the dependencies between different parts of the code. The 'Source' connection (6) feeds into the 'Northwind' database (5), which in turn feeds into the 'dbo\_Customers' table (4). The 'Removed Columns' step (3) depends on the 'dbo\_Customers' table. The 'Renamed Columns' step (2) depends on the 'Removed Columns' step. Finally, the entire process results in the 'Renamed Columns' table (1).

## 5.2.8 Let Expression Flow

- In this example each step depends only on the steps defined before it:

```
let
    step1 = "!stnedutS",
    step2 = "olleH",
    step3 = step1 & " " & step2,
    step4 = Text.Reverse(step3)
in
    step4
```

The screenshot shows the Power Query Editor interface with the following details:

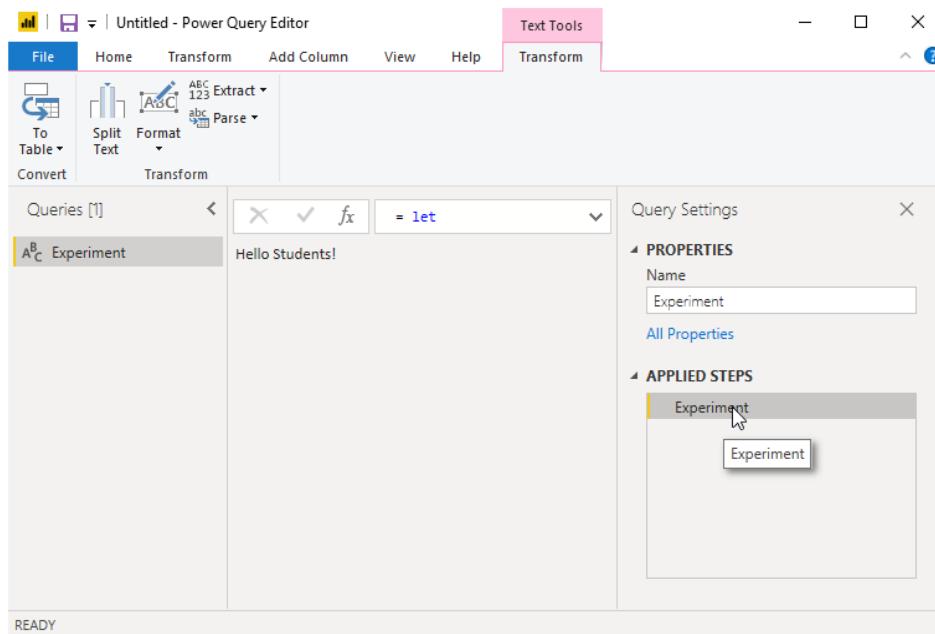
- File Bar:** Untitled - Power Query Editor, File, Home, Transform, Add Column, View, Help, Transform (highlighted).
- Transform ribbon:** Text Tools, To Table, Split Text, Format, Extract ABC 123, Parse abc.
- Queries [1]:** A single query named "Experiment" containing the text "Hello Students!".
- Text pane:** Formula bar shows the formula: `= Text.Reverse(step3)`.
- Query Settings pane:** Properties section shows Name: Experiment, Applied Steps section lists step1, step2, step3, and step4 (with step4 highlighted).
- Status bar:** READY.

## 5.2.9 Let Expression Flow: Out of Order

---

- In this example we have reversed the order of the steps of the previous example
- Take note that this DOES work!
- However, notice that the applied steps section in the GUI breaks!
- Always try to order steps in the let expression so that each step depends only on previous steps

```
let
    step4 = Text.Reverse(step3),
    step3 = step1 & " " & step2,
    step2 = "olleH",
    step1 = "!stneduts"
in
    step4
```



## Section 5.3

### M Language Lexicon

## 5.3.1 M Expressions and Values

---

- The central construct in M is the expression
- An expression can be evaluated (computed) yielding (resulting in) a single value
- Many values can be written literally as an expression, but a value is not an expression

### Example

In this example the expression is 42, and it will evaluate to the value of 42:

```
SomeValue = 42
```

### Example

In this example the expression is  $40 + 2$ , and it will evaluate to the value of 42:

```
AnotherValue = 40 + 2
```

- An expression is a recipe for evaluation
- Values are the result of evaluation

## 5.3.2 Types of Values

---

- There are different types of values
  - ◊ Some values are Primitive and consist of a single value:
    - \* Null
    - \* Logical (Boolean)
    - \* Number
    - \* Time
    - \* Date
    - \* DateTime
    - \* DateTimeZone
    - \* Duration
    - \* Text
    - \* Binary
  - ◊ Others are structured, and consist of a combination of one or more primitives or other structured values:
    - \* List
    - \* Field
    - \* Record
    - \* Table
    - \* Function
    - \* Type

## 5.3.3 Primitive Values

---

- A Primitive Value is a single part value, such as a number, date, text, or null
  - ◊ A null value can be used to indicate the absence of any data

### Example

```
123          // A number
true         // A logical (or Boolean)
"hello student" // A text
null         // null value
```

- Time based values are stored as a single value but are expressed through special functions

### Example

```
#time(09,15,00)           // a time only
#date(2011,01,26)         // a date only
#datetime(2011,01,06,09,15,00) // a date and a time stored together
#datetimezone(2011,01,06,09,15,00,6,0) // a datetime and offset from UTC
#duration(0,1,30,0)
```

- Numbers allow for whole numbers and fractions, and can be expressed in scientific notation or hexadecimal

### Example

```
3.14    // Fractional number
-1.5    // Fractional number
1.0e3   // Fractional number in scientific notation
123     // Whole number
1e3     // Whole number in scientific notation
0xff    // Whole number in hex (255)
```

## 5.3.4 List Values

---

- A List Value is an ordered sequence of values
  - ◊ M supports infinite lists, but when written as a literal a list must have a fixed length
  - ◊ Curly brace characters { and } denote the beginning and end of a list

### Example

A list containing a number, a logical, and a text:

```
{ 42, true, "orange" }
```

A list of text values:

```
{"red", "orange", "yellow", "green", "blue", "indigo", "violet" }
```

## 5.3.5 Fields and Record Values

---

- A Field is a name/value pair where the name is a text value that is unique within the field's Record
  - ◊ A Field must belong to a Record
- A Record is a set of Fields
  - ◊ The syntax for record values allows the field names to be written without quotes, a form referred to as identifiers

### Example

The following example shows a record containing three fields named “A”, “B”, “C”, “D” and “E”, which have values 2, 3, 5, 7, and 11:

```
[  
    A = 2,  
    B = 3,  
    C = 5,  
    D = 7,  
    E = 11  
]
```

## 5.3.6 Table Values

---

- A Table is a set of values organized into columns and rows
  - ◊ Columns are identified by rows
  - ◊ There is no literal syntax for creating a table
    - \* There are several standard functions that can be used to create tables from lists or records

### Example

This example uses the Table function to create a table:

```
#table( {"Product", "Description",
{
    {"Super Soaker", "A water gun powered by compressed air"},  

    {"Pog", "A collectable cardboard or plastic disk"},  

    {"Gameboy", "An 8-bit handheld game console"}  

})
```

The resulting table would look like this:

Products	Descriptions
Super Soaker	A water gun powered by compressed air
Pog	A collectable cardboard or plastic disk
Gameboy	An 8-bit handheld game console

## 5.3.7 Function Values

---

- A function is a value which, when invoked with arguments, produces a new value
- A function is written by listing the function's:
  - ◊ Parameters in parentheses
  - ◊ Followed by the such-that symbol =>
  - ◊ Followed by the expression defining the function
    - \* The expression typically refers to the parameters by name

### Example

The following expression takes the values X, and Y, and returns the average of the two numbers:

`(x, y) => (x + y) / 2`

## 5.3.8 Functions are Values

---

- A function is a value just like a number or text value
- When a function is invoked, a set of values are specified which are logically substituted for the required set of input values within the function body expression

### Example

This example shows a record with three fields. The first field is a function that is invoked by the expressions of the second and third fields:

```
[  
    Add = ( x , y ) => x + y,  
    OnePlusOne = Add ( 1, 1 ),  
    TwoPlusOne = Add ( 2 , 1)  
]
```

- M includes a common set of function definitions available for use from an expression
  - ◊ This set is called the Standard Library
  - ◊ Functions defined in the library are available for use without having been explicitly defined by the expression

### Example

```
Number.E                      // Euler's number e (2.7182...)  
Text.PositionOf("Hello", "llo") // 2
```

## 5.3.9 Expression Evaluation

- M Language has an evaluation model that is similar to how a spreadsheet evaluates its expressions
- Parts of an expression can reference other parts of the expression by name
  - ◊ The evaluation process will automatically determine the order in which referenced expressions are calculated
    - \* Behind the scenes, it does this by calculating a dependency tree

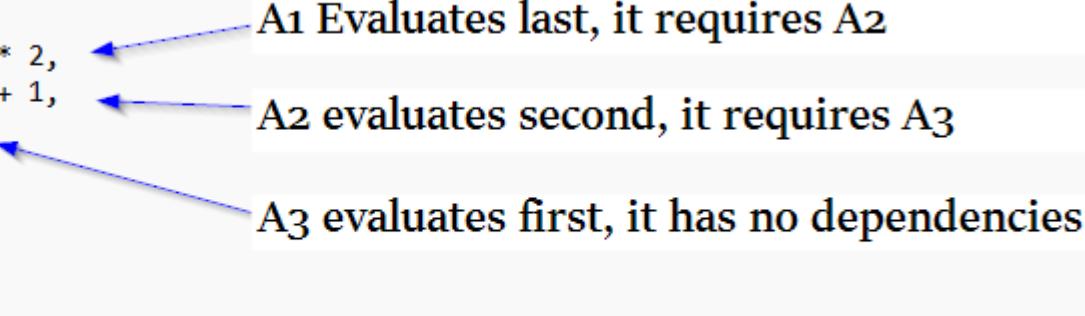
### Example

```
[  
  A1 = A2 * 2,  
  A2 = A3 + 1,  
  A3 = 1  
]  
[  
  A1 = 4,  
  A2 = 2,  
  A3 = 2  
]
```

A1 Evaluates last, it requires A2

A2 evaluates second, it requires A3

A3 evaluates first, it has no dependencies



## 5.3.10 Expression Evaluation Caution

---

- Dependency based Expression Evaluation is a powerful execution model, but it has a few side effects to be careful with
  - ◊ If during the evaluation of an expression external state is being modified, then the order that expressions execute would become important to you
    - \* If each time a function was executed it returned a different value, and you need those in a predictable order, you would be forced to reverse engineer the dependencies of your expression to understand what is happening
  - ◊ For this reason, try to favor functions that do not change external state
- Whenever possible try to design deterministic functions
  - ◊ This means that when executed given a set of input values, the output is always the same
    - \* A function that removes a column is deterministic
    - \* A function that adds two values is deterministic
    - \* A function that retrieves data from a database is non-deterministic
    - \* A function that subtracts a customer's birthdate from the current time is non-deterministic

## 5.3.11 Nesting and the Lookup Operator

---

- Records can be nested within other records
- Use the lookup operator [ ] to access fields of a record by name

### Example

The following record has a field named Sales containing a record, and a field named Total that accesses the FirstHalf and SecondHalf of the Sales record:

```
[  
  Sales = [ FirstHalf = 1000, Second Half = 1100 ],  
  Total = Sales[FirstHalf] + Sales[SecondHalf]  
]
```

## 5.3.12 Positional Index Operator

---

- Records can be contained within lists
- Use the positional index operator { } to access an item in a list by its numeric index
  - ◊ Values within a list are referred to using a zero based index from the beginning of the list

### Example

```
[  
    Sales =  
        {  
            [  
                Year = 2017,  
                FirstHalf = 2000,  
                SecondHalf = 1100,  
                Total = FirstHalf + SecondHalf // 3100  
            ],  
            [  
                Year = 2018,  
                FirstHalf = 2200,  
                SecondHalf = 1300,  
                Total = FirstHalf + SecondHalf // 3500  
            ]  
        },  
        TotalSales = Sales{0}[Total] + Sales{1}[Total] // 6600  
]
```

## 5.3.13 Let expressions compared with Lists

---

- Let expressions might be used to eliminate the need for the positional operator used in the previous example

### Example

Compare this example with the previous one, both return the same result using different techniques:

```
let
    Sales2007 =
        [
            Year = 2017,
            FirstHalf = 2000,
            SecondHalf = 1100,
            Total = FirstHalf + SecondHalf // 3100
        ],
    Sales2008 =
        [
            Year = 2018,
            FirstHalf = 2200,
            SecondHalf = 1300,
            Total = FirstHalf + SecondHalf // 3500
        ]
in
    Sales2017[Total] + Sales2018[Total] // 6600
```

- Although it is not common, it is perfectly acceptable to nest let expressions

## 5.3.14 Lazy versus Eager Evaluation

---

- Lists, Tables, Record Members, and Let expressions are evaluated using lazy evaluation
  - ◊ Only as you access a List Item, Record, or Field are those expressions evaluated
  - ◊ This means it is possible that they are never evaluated at all
- All other expressions are evaluated using eager evaluation
- They are evaluated immediately as encountered during the evaluation process

## 5.3.15 If Expression

---

- If can be used to conditionally use one of two expressions, depending on the result of evaluating an initial expression

### Example

In the following example we're testing to see if the customer's age is less than 18, if it is, we'll use the string RED, but if it isn't we'll use the string GREEN

```
if CustomerAge < 18 then  
    "RED"  
  
else  
    "GREEN"
```

## 5.3.16 Each Keyword

---

- The Each keyword is used in queries when an expression should be performed to create output for each provided input
  - ◊ Within the function following each, you use the underscore character to represent the current record
  - ◊ If all you need is a fieldname for the current record, the underscore is implied

### Example

```
let
    Source = #table( {"col1", "col2"}, { {1, 2}, {10,20} } ),
    AddColumn =
        Table.AddColumn(
            Source,
            "RowTotal",
            each _[col1] + _[col2]
        )
in
    AddColumn
```

## 5.3.17 Comments

---

- Comments are used to insert text explaining the query
- Comments are not interpreted as code

<https://docs.microsoft.com/en-us/powerquery-m/comments>

## 5.3.18 Operator Order of Evaluation

---

- Similar to algebra, expressions are evaluated in an order based on the operators, not from left to right

<https://docs.microsoft.com/en-us/powerquery-m/m-spec-operators>

## Section 5.4

# M Language Reference

## 5.4.1 Function Library Reference

- There are over 700 out of the box functions provided by the M Language Function Library
- Microsoft provides numerous updates each year, and each update brings more functions into availability
- The best reference for this material is Microsoft's online reference found here:

<https://docs.microsoft.com/en-us/powerquery-m/power-query-m-function-reference>

The screenshot shows a Microsoft Docs page titled "Power Query M function reference". The URL in the address bar is <https://docs.microsoft.com/en-us/powerquery-m/power-query-m-function-reference>. The page includes a navigation bar with links to Microsoft, Docs, Documentation, Learn, Certifications, Q&A, Code Samples, Shows, and Events. A sidebar on the left contains a "Filter by title" search bar and a list of topics: "Power Query M formula language", "Quick tour of the Power Query M formula language", "Power Query M language specification", "Power Query M type system", "Expressions, values, and let expression", "Comments", and "Evaluation model". The main content area features the title "Power Query M function reference", a timestamp of "Article • 05/20/2022 • 2 minutes to read • 4 contributors", and a summary text about the reference articles for over 700 functions.

## 5.4.2 Function Categories

---

### Functions by category

- [Accessing data functions](#)
- [Binary functions](#)
- [Combiner functions](#)
- [Comparer functions](#)
- [Date functions](#)
- [DateTime functions](#)
- [DateTimeZone functions](#)
- [Duration functions](#)
- [Error handling](#)
- [Expression functions](#)
- [Function values](#)
- [List functions](#)
- [Lines functions](#)
- [Logical functions](#)
- [Number functions](#)
- [Record functions](#)
- [Replacer functions](#)
- [Splitter functions](#)
- [Table functions](#)
- [Text functions](#)
- [Time functions](#)
- [Type functions](#)
- [Uri functions](#)
- [Value functions](#)

## Exercise 5.1

# Contoso M – Part 1 – Shaping flat file data with M

---

### Introduction

In this project we'll be creating a report that allows the viewer to compare Online sales verse Retail sales. Unfortunately for us, we can't access the underlying database. We're limited to two different data sources provided:

- An Excel Spreadsheet containing summaries of Online Sales
- A tab delimited text file export of Retail Sales, over 160MBs of text! Who made this?!

Ultimately, we would like to see a column chart that lets us see sales side-by-side by date, filtered by product category

We'll be faced with a few modelling challenges along the way. Whenever possible we will create Power Queries using the Graphical User Interface - but you'll see that we'll have to resort to writing some custom M code along the way.

Let's get started

1. Open up a new instance of Power BI Desktop. This should begin a new project.
2. Give the project a name by saving it. Select Save As from the file menu and save the file to the following location:

{LabFiles}\MyWork\ContosoM\ContosoM\_Step\_00\_YourName.pbix

3. Click on Get Data in the Home Ribbon to launch the Get Data dialog.

4. Choose Excel and click connect.

5. Navigate to and select the following file:

{LabFiles}\StarterFiles\ContosoM\ContosoOnlineSalesReport.xlsx

6. Select the Excel sheet named ContosoOnlineSales and click load.

7. Click on Get Data again, this time choose Text/CSV and click connect.

8. Drop down the file type and choose All Files (\*.\*).

9. Navigate to and select the following file:

10. Click Load.

11. Navigate to the Data View and take a moment to explore the two tables of data that we have to work with.

- Notice that in ContosoOnlineSales, the time granularity of an entry is per month.
- Notice that in ContosoRetailSales the time granularity of an entry is per day.
- Notice that in ContosoOnlineSales, the location granularity of an entry is per State.
- Notice that in ContosoRetailSales the time granularity of an entry is per City.

12. Open Power Query Editor by clicking on Edit Queries in the Home Ribbon.

13. Change the data types of each existing column based on the following table, if prompted replace the current step instead of creating a new step:

<b>Table</b>	<b>Column</b>	<b>Datatype</b>
ContosoOnlineSales	Total Sales	Fixed Decimal Number
	State	Text
	Product	Text
	Category	Text
	CalendarMonth	Text
ContosoRetailSales	Location	Text
	ProductName	Text
	ProductCategoryName	Text
	FullDateLabel	Date
	TotalCost	Fixed Decimal Number

The Location column of the RetailSalesOrigin table contains both a City and State, we want to split this into two columns:

14. Select the ContosoRetailSales table.

15. Right click on the Location column and select Split Column -> By Delimiter.

16. On the Split Column by Delimiter dialog, confirm that Comma is selected as the delimiter and click OK.

17. It's hard to tell but the Location.2 column has a space at the beginning of each value. Right Click on the Location.2 column and choose Transform -> Trim.

**18. Rename the following columns:**

	<b>Old Name</b>	<b>New Name</b>
ContosoRetailSales	Location.1	City
ContosoRetailSales	Location.2	State
ContosoRetailSales	ProductName	Product
ContosoRetailSales	ProductCategoryName	Category
ContosoRetailSales	TotalCost	Retail Sales
ContosoOnlineSales	Total Sales	Online Sales

**19.** Close the Power Query Editor window. When prompted, select Yes to save your changes.

**20.** At this point Power BI will take a moment as the queries run against the data sources.

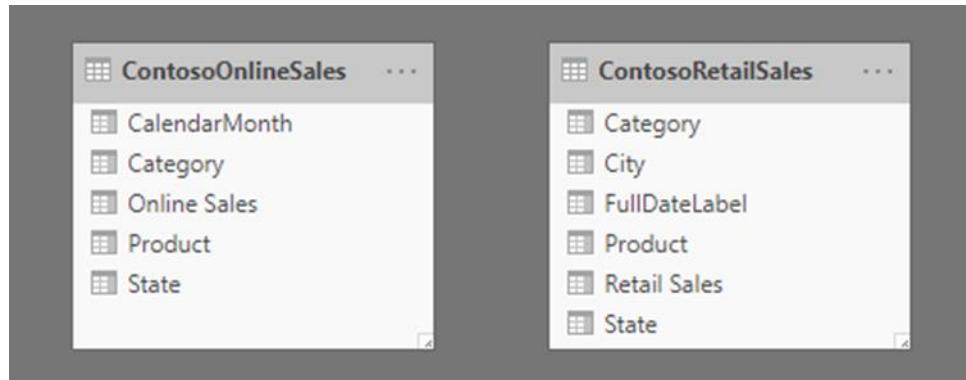
**21.** Save the project using the Save As option. Use the following file location:

{LabFiles}\MyWork\ContosoM\ContosoM\_Step\_01\_YourName.pbix

## Exercise 5.2

### Contoso M – Part 2 – Building Dimensions

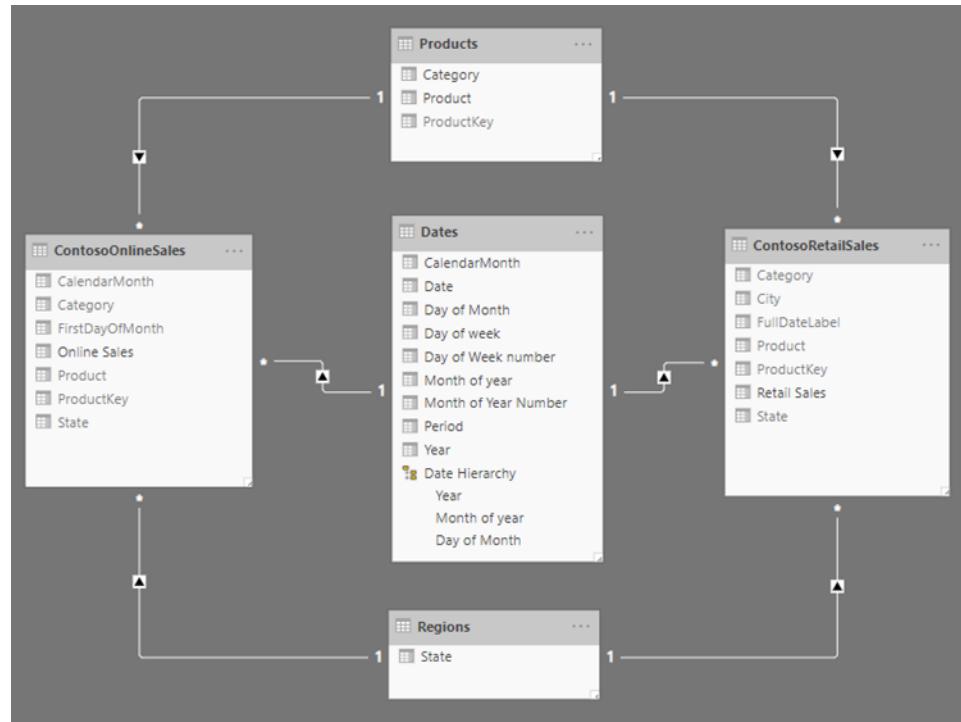
If you look at the current data model, you'll see that it looks like this:



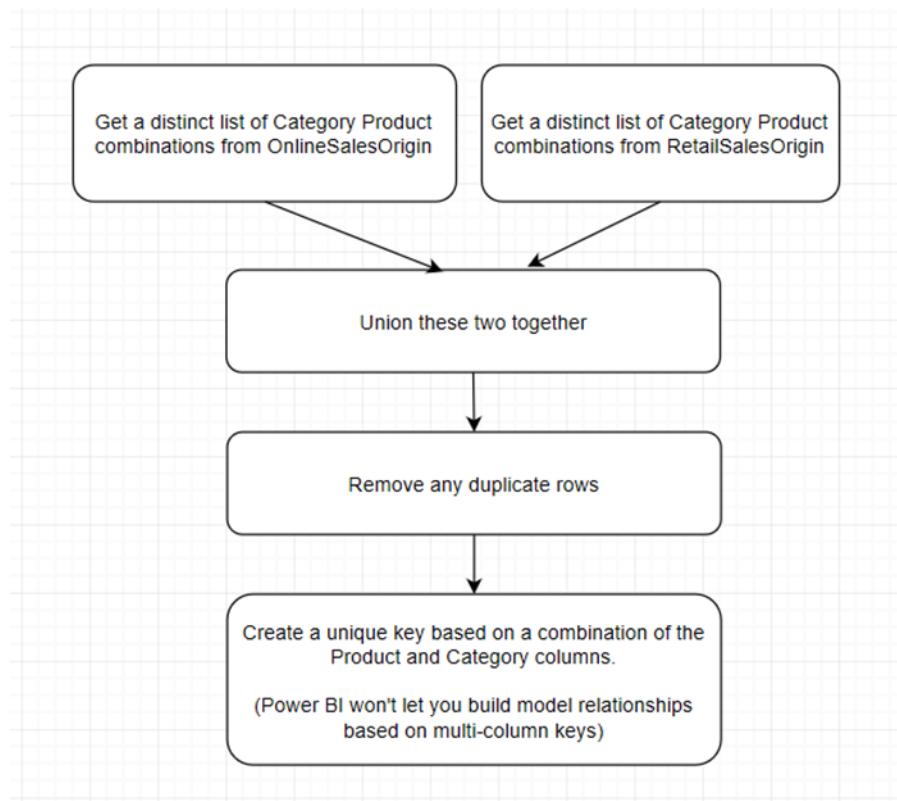
There are some problems with this model:

- Online sales are granular to the Month, but Retail sales are granular to the Day.
- Nothing tells the model that a Product in OnlineSalesOrigin is the same as a Product in RetailSalesOrigin. The same is true for Category and State columns.

Here is the ideal model that we would rather report against, this represents a star schema with two fact tables and three dimension tables:



Getting there is going to take a little work. As you're learning a new language like Power Query M, it's worth a little bit of planning ahead of time. Sketching out the steps involved on some graph paper (or a draw.io diagram) is a great first step:



1. Continue working with the previous file.
2. Open the Power Query Editor.
3. Create a new source based on a blank query.
4. Rename the query from Query1 to Products
5. Right-Click on the Products query and choose Advanced Editor.

6. Take a moment to read through the query to get a sense of what it is doing! If you need to, google “Table.Combine”, “Table.Distinct”, and “Table.AddColumn” and pull up the Power Query Reference pages on these functions! When you are finished understanding what it will do, type it out into the editor.

```

let
    UnionedProductData = Table.Combine(
        (
            {ContosoOnlineSales, ContosoRetailSales},
            {"Category", "Product"}
        ),
    DistinctRows = Table.Distinct(UnionedProductData),
    AddKeyColumn = Table.AddColumn(
        (
            DistinctRows,
            "ProductKey",
            each [Category] & "|" & [Product]
        )
    in
        AddKeyColumn

```

7. Add a ProductKey column to the ContosoRetailSales query by adding to the let statement:

- \* Make sure the circled statements match!

Advanced Editor

## ContosoRetailSales

Display Options ?

```

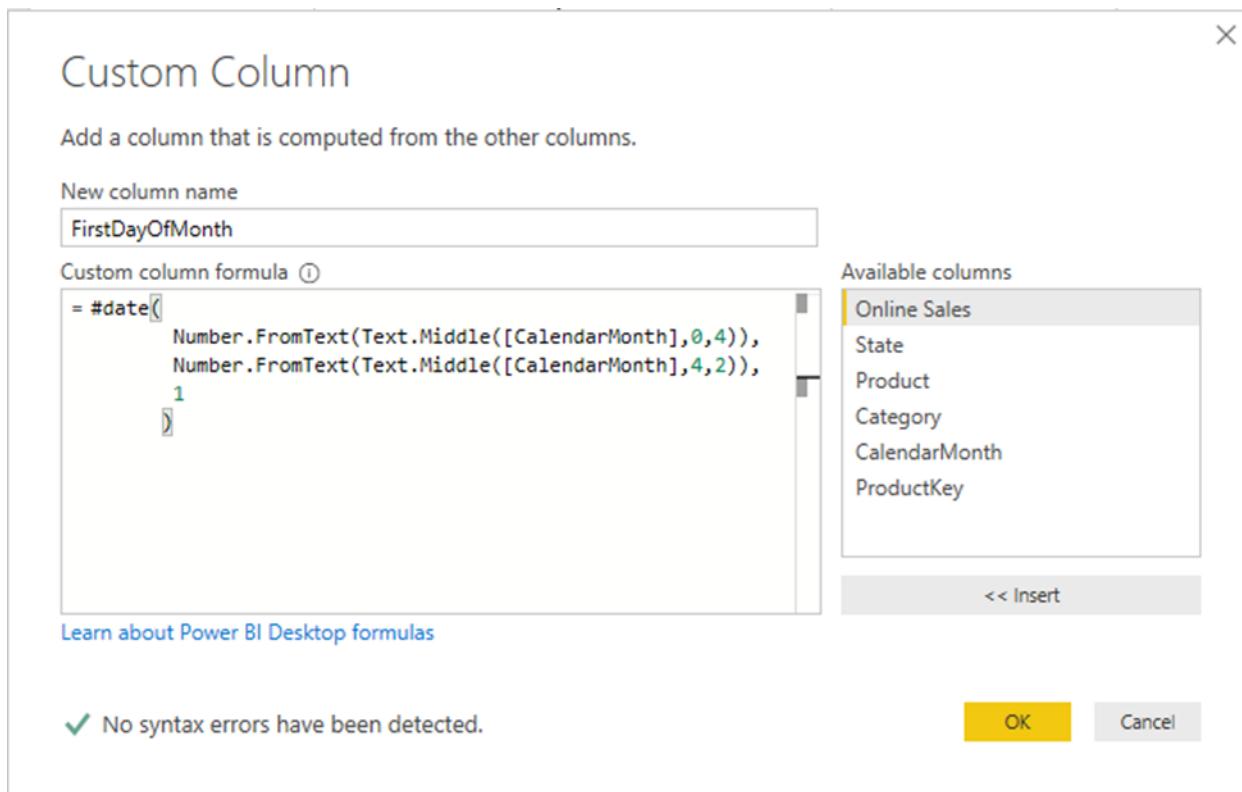
let
    Source = Csv.Document(File.Contents("C:\HOTT-DAX-M\StarterFiles\ContosoRetailSalesForHOTT.rpt"),[Delimiter="",
        "Columns=5, Encoding=65001, QuoteStyle=QuoteStyle.None]),
    #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),
    #"Changed Type" = Table.TransformColumnTypes(#"Promoted Headers",{{"Location", type text}, {"ProductName", type text}, {"ProductCategoryName", type text}, {"FullDateLabel", type date}, {"TotalCost", Currency.Type}}),
    #"Split Column by Delimiter" = Table.SplitColumn(#"Changed Type", "Location", Splitter.SplitTextByDelimiter(",",
        QuoteStyle.Csv), {"Location.1", "Location.2"}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Split Column by Delimiter",{{"Location.1", type text},
        {"Location.2", type text}}),
    #"Renamed Columns" = Table.RenameColumns(#"Changed Type1",{{"Location.1", "City"}, {"Location.2", "State"}, {"ProductName", "Product"}, {"ProductCategoryName", "Category"}, {"TotalCost", "Retail Sales"}})
    ,
    #"AddProductKey" = Table.AddColumn(#"Renamed Columns", "ProductKey", each [Category] & "|" & [Product])
in
    #"AddProductKey"

```

✓ No syntax errors have been detected.

Done Cancel

8. Add a ProductKey column to the ContosoOnlineSales query by using the same pattern that you applied on the above step.
9. Add another column to the ContosoOnlineSales table, but this time utilized the dialog found at Power Query Editor -> Add Column Ribbon -> Custom Column button. Use the following values:



10. Change the datatype of the FirstDayOfMonth column to Date.

11. Create another new query and name it Regions.

12. Using the advanced editor, enter the following query:

```
let  
    CombinedRegions = Table.Combine  
    (  
        {ContosoOnlineSales, ContosoRetailSales},  
        {"State"}  
    ),  
    DistinctStates = Table.Distinct(CombinedRegions)  
in  
    DistinctStates
```

13. Create another new query and name it Dates.

14. Using the advanced editor, enter the following query. NOTE: You can cut and paste this from:

{LabFiles}\StarterFiles\ContosoM\M\_DateTableExpression.txt.

15. Close the Power Query Editor, applying changes if it asks.

16. Back in Power BI Desktop, navigate to the Data view.

17. Configure the Month of Year column to order by the Month of Year Number column.

18. Configure the Day of Week column to order by the Day of Week number column.

19. Configure the Period column to order by the CalendarMonth column.

- \* Create a Hierarchy in the Date table named Date Hierarchy with the following columns: Year, Month of year, Day of Month.
- \* Hint: Do this from the Modeling View.

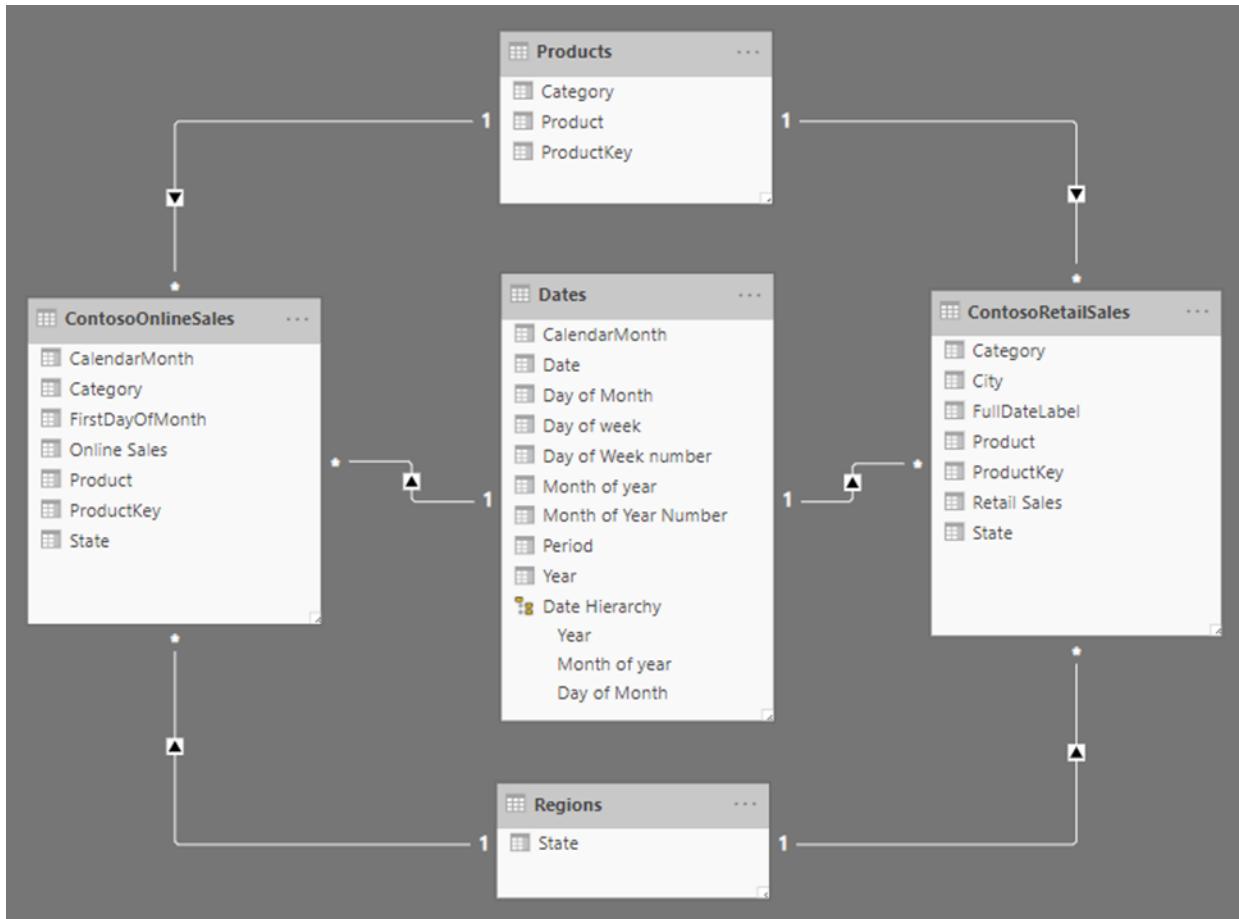
20. Save the project using the Save As option. Use the following file location:

{LabFiles}\MyWork\ContosoM\ContosoM-Step02.pbix

## Exercise 5.3

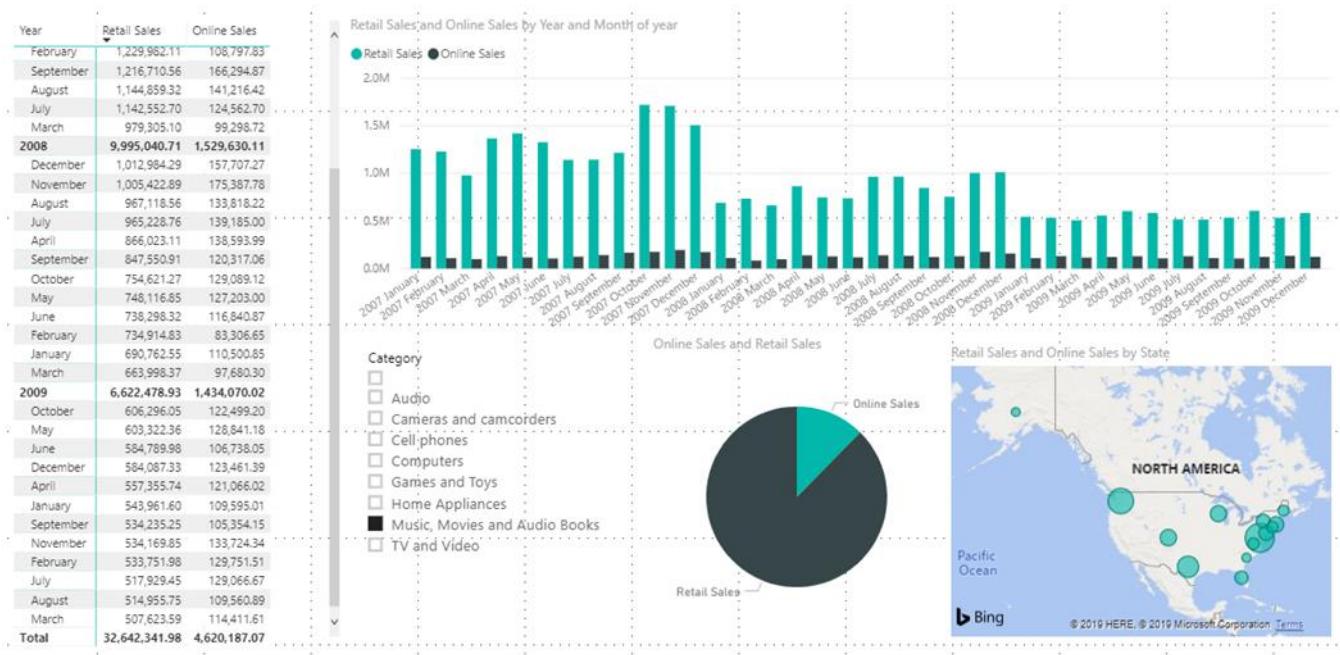
### Contoso M – Part 3 – Relationships and reporting

1. Continue working with the previous file.
2. Using the Modeling view, build out the following relationships:
  - \* Hint: The ContosoOnlineSales table does not have a specific date, so you'll need to build the relationship on the FirstDayOfMonth date column that we had calculated out



3. Hide every column in the ContosoOnlineSales table except for Online Sales
4. Hide every column in the ContosoRetailSales table except for Retail Sales
5. Hide the ProductKey column in the Products table.

6. Using the skills you've developed in this class, create a report that looks like this:



7. Save the project using the Save As option. Use the following file location:

{LabFiles}\MyWork\ContosoM\ContosoM-Step03.pbix

# Module 6

## Introducing DAX

## Section 6.1

### Defining DAX

## 6.1.1 Data Analysis eXpression Language

---

- DAX stands for Data Analytics eXpression Language
- DAX was developed by Microsoft as the expression language native to the in-memory columnar database used for information analysis
- DAX is used to add meaning to the data that you are analyzing through formulas, or expressions
- DAX is used everywhere xVelocity is used
- Learning how to write expressions with DAX is essential to getting the most out of Power BI

## 6.1.2 DAX: Data

---

- How the data that DAX works with is shaped will drastically affect how useful our analytics can be
- DAX is used both to help model data, and to analyze the modeled data
- DAX helps you create new information from data already in your data model
  - ◊ For example
    - \* Percentages
    - \* Summaries
    - \* Key Performance Indicators
    - \* Trends
- DAX Expressions become a part of your model
- DAX helps you to manage the granularity of your data
  - ◊ It might turn an order level discount into a line item discount for reports that work on the granularity of an order detail row
  - ◊ It helps you define calculations that will summarize data with totals or percentages after that data has been filtered to a subset of itself

## 6.1.3 DAX: Analysis

---

- DAX is used to analyze data
- It is most useful for two types of analysis
  - ◊ Aggregation
  - ◊ Filtering
- When you are dealing with millions of detail records, you need a way to summarize their meaning at various levels or granularities
  - ◊ You want your eyes to be able to see the data in graphical reports without having to add your own meaning to each number
- There are many built in functions that help with various forms of analysis
  - ◊ Statistical Analysis
  - ◊ Time-based Analysis
  - ◊ Dimension slicing and dicing through user defined hierarchies

## 6.1.4 DAX: Expression Language

---

- DAX is an expression language
  - ◊ A collection of functions, operators, and constants that can be used in a formula or expression to calculate and return one or more values
- Writing DAX feels a little like writing spreadsheet expressions
  - ◊ Excel might add a column by using other values on the same row by referencing data co-ordinates
  - ◊ Excel might add a summary by using values referenced by a range of data coordinates
  - ◊ DAX does not use co-ordinates to get to its data like excel
  - ◊ DAX uses a combination of Expression Context and Active Relationships within a data model
- DAX understands how data is related because of its data model
  - ◊ This makes DAX a much more effective way to analyze data than using Excel Expressions

## 6.1.5 DAX verses Power Query: Similarities

---

- Both Power Query and DAX have some similarities
  - ◊ Both can be used to calculate column values
  - ◊ Both can be used to calculate entire tables
  - ◊ Both can be used to change a column's name
  - ◊ Both can be used to change a table's name
  - ◊ Either can feel more intuitive than the other for certain situations

## 6.1.6 DAX verses Power Query: Power Query

---

- Power Query
  - ◊ Used to bring data into the model
  - ◊ Made up of multiple “steps” that each alter the shape of the result, one step at a time.
  - ◊ Power Query may end up performing better because of query folding
    - \* Query Folding pushes work into the underlying data source, such as Microsoft SQL Server
    - \* This could mean that less data gets transferred into Power BI
    - \* This could allow engines like SQL Server to utilize indexes to perform expensive operations like filtering for data that meets certain criteria, or putting data in a specific order
  - ◊ Power Queries make up multiple steps containing many expressions, where DAX does not
  - ◊ Power Queries cannot be used to calculate Measures
    - \* This is because DAX computations are made within a filter context, or for many different contexts, which are not known until the user incorporates the measure into a report

## 6.1.7 DAX verses Power Query: DAX

---

- DAX

- ◊ Limited to working with data that has already been brought into the model
- ◊ Does not have steps
  - \* An expression is a single series of nested operations
- ◊ Feels like writing spreadsheet expressions
  - \* Only with Tables and Columns instead of Cell Coordinates
- ◊ Can be used to calculate measures
  - \* This allows advanced summary level analysis

## 6.1.8 DAX verse Power Query: Which?

---

- Which would I use?
  - ◊ Use Power Query if you can avoid bringing unnecessary data into the model
    - \* This will perform better and keep the memory footprint smaller
  - ◊ Use Power Query if you want to re-use queries in multiple projects
    - \* Queries can be cut-and-pasted or moved to Power BI Service
  - ◊ Only DAX can be used to calculate Measures
    - \* These powerful calculations need to be recalculated for different filter contexts and values are not stored with the row

## 6.1.9 Calculations not suitable for storage in relational databases

---

- DAX can be used to define calculations that are not suitable for storage
  - ◊ These are values where if data in another row changes, the result of the calculation changes
  - ◊ These are values where the calculation may be performed in many different contexts, where the available contexts are driven by the data itself
- Examples include:
  - ◊ For a given line item of an order, what percentage of that order does the single line item represent?
  - ◊ What was the average age of the customer at the time a sale was made?
  - ◊ Correctly calculated percentages at different grouping levels
  - ◊ Aggregate summaries
- DAX allows us to reshape our data and add expressions so that reporting against our data becomes as easy as checking off boxes

## 6.1.10 DAX Family of Products

---

- Power Pivot for Microsoft Excel
  - ◊ Became available as an add-on in 2010
  - ◊ Distributed as a built-in feature in Office 2016
- Microsoft Power BI
  - ◊ Locally in Power BI Desktop
  - ◊ In the Cloud with Power BI Service
  - ◊ On-Premise Server with Power BI Report Server
  - ◊ Supports Personal, Team, and Organizational BI Needs
- Microsoft SQL Server Analysis Services - Tabular Mode
  - ◊ Tabular mode is the newer of the two modes supported by SSAS
  - ◊ Became available starting in 2012
- DAX Studio
  - ◊ 3rd party free utility to work with DAX
  - ◊ Learn more at <https://daxstudio.org>

## Section 6.2

# DAX Syntax Fundamentals

## 6.2.1 Calculations

---

- DAX is used to add values to your existing data model
- There are four types of calculations that can be added to your model with DAX
  - ◊ Calculated Columns
    - \* Used to add value at the row level
    - \* Can be calculated on a row by row basis as soon as it's added
  - ◊ Measures
    - \* Used to add value at a summary level
    - \* Requires context to be calculated
  - ◊ Calculated Tables
    - \* Can be used to enhance the model with tables derived from other data found in the model
  - ◊ Row Filters
    - \* Can be used to provide row-level security
- A calculation is expressed through a formula

## 6.2.2 Formulas

---

- A DAX formula always starts with an equal sign (=)
  - ◊ There may be a Name on the left side of the equal sign, to store the result of the formula
- A DAX formula cannot modify values in the model, they only add new values
- After the equal sign, you can provide any expression that evaluates to a scalar, or an expression that can be converted to a scalar. These include the following:
  - ◊ A scalar constant, or expression that uses a scalar operator ( +, - , \* , / , >= , .....)
  - ◊ References to columns or tables
    - \* The Dax Language always uses table and columns as inputs to functions, never an array or arbitrary set of values
  - ◊ Operators, constants, and values provided as part of an expression
  - ◊ The result of a function and it's required arguments
    - \* Some DAX functions return a table instead of a scalar, and must be wrapped in a function that evaluates the table and returns a scalar
    - \* If the table is a single column, single row table, then it is treated as a scalar value

## ◊ Expressions

- \* An expression can contain any or all of the following
- \* Operators
- \* Constants
- \* References to columns

## 6.2.3 Formula Evaluation

---

- Formulas need to be evaluated
  - ◊ Evaluation of a formula will produce a result
- Formulas behave differently depending on whether they are used in a Calculated Column, or in a Measure
- Formulas are evaluated with an Evaluation Context
- Evaluation Context consists of
  - ◊ Row Context
    - \* This is relevant only for Calculated Columns and inside of an iterator
    - \* Allows an expression to reference a column value outside of an aggregate
  - ◊ Filter Context
    - \* This is most relevant for measures
    - \* Both implicit and explicit filters will be applied, which limits the data the formula will see

## 6.2.4 Formula Examples

---

### Example

= 8

The result is the number 8

### Example

= "Sales"

The result is the string *Sales*

### Example

= 'Sales'[Amount]

When used in a formula defining a value in the Sales table, you will get the value of the [Amount] column for the current *Row Context*

### Example

= ( 0.03 \* [Amount] )

The result is three percent of the value in the amount column of the current *Row Context* of the current table

### Example

= 0.03 \* [Amount]

Although this formula can be used to calculate a percentage, the results is not shown as a percentage unless you apply formatting in the table

### Example

=PI()

The value of the constant  $\pi$

## 6.2.5 Naming Requirements

---

- Each table must have a unique name
- Within a table, each column must have a unique name
- Object names are case-insensitive
  - ◊ SALES is the same as Sales
- Each column or measure you add to a data model must belong to a specific table

## 6.2.6 Fully Qualified Names

---

- When you use a table or column as an input to a function you must Qualify the name
- To accommodate spaces, reserved keywords, or disallowed characters, table names are specified inside single quote marks: ‘European Sales’, ‘Номер телефона’
  - ◊ This is not required for simple one-word names
- Column names are specified inside square brackets: [Amount]
- This is always required
- A fully qualified column name includes both the table and the column: ‘Online Sales’[Sales Amount]
- A fully qualified name is required when you reference a column in the following contexts
  - ◊ As an argument to the function VALUES()
  - ◊ As an argument to the functions ALL() or ALLEXCEPT()
  - ◊ As an argument to the function RELATEDTABLE()

- ◊ As an argument to any time intelligence function
- ◊ In a filter argument for the functions CALCULATE() or CALCULATETABLE()

## 6.2.7 Naming Examples

---

Table Name	Sales
Table Name	‘European Sales’
Fully Qualified Column Name	Sales[Amount]
Fully Qualified Measure Name	Sales[Profit]
Unqualified column name	[Amount]
Fully qualified column name	‘Canada Sales’[Qty]

## 6.2.8 Operators

---

- Operators perform operations on one (unary operators) or two (binary operators) inputs, without the explicit use of a function

### Example

= 2 + 9

The ‘+’ symbol above is considered an operator. 2 and 9 are inputs to this operator. Because the ‘+’ operator is working on two different values, this is considered a binary operator.

- ◊ There are different categories of operators, used in different scenarios
  - \* Arithmetic Operators
  - \* Comparison Operators
  - \* Text Concatenation Operator
  - \* Logical Operators
- Many operators are often combined within a single formula

## 6.2.9 Operator Reference

---

<https://docs.microsoft.com/en-us/dax/dax-operator-reference>

## 6.2.10 Operator Precedence

---

- Most formulas contain multiple calculations
- The order in which calculations are performed can affect the return value
- Operator evaluation takes place in order from left to right
- The order is:
  1. Anything in parenthesis ‘( )’
  2. Exponent operator ‘^’
  3. Multiplication and Division ‘\*’, ‘-’
  4. NOT Operator ‘!’
  5. Addition and Subtraction ‘+’, ‘-’
  6. Concatenation ‘&’
  7. Comparison ‘=’, ‘==’, ‘<’, ‘>’, ‘<=’, ‘>=’, ‘<>’
- Make good use of parenthesis to control the order of evaluation
- This is the same principal that applies to algebra

## 6.2.11 Operator Precedence

---

### Example

In this example we're computing just a single operator in each step to illustrate the order of operations

```
= 3 + 3 / 1 * 4 - 2  
= 3 + 3 * 4 - 2  
= 3 + 12 - 2  
= 15 - 2  
= 13
```

### Example

In this example we've added parenthesis to the expression above which changes the order of evaluation and results in a different answer

```
= (3 + 3) / 1 * (4 - 2)  
= 6 / 1 * (4 - 2)  
= 6 / 1 * 2  
= 6 * 2  
= 12
```

### Example

In this example, notice that the minus '-' sign is an operator, and participates in order of operations! This kind of behavior may not be expected as it differs from algebra

```
= -2^2
```

```
// this results to -4
```

```
= (-2)^2
```

```
// this results to 4
```

## 6.2.12 Scalar Data Types

---

- Each value has a type, the type depends on how the source column was defined
- A data type restricts the data that can be stored in a table and helps the operators to know how to interpret the value
- When writing a Calculated Column or Measure the formula must return a scalar value
- A scalar value is a single value
- DAX works with the following scalar data types
  - ◊ Integer
    - \* Numbers that have no decimal places
    - \* Can be positive or negative
    - \* Whole numbers between -9,223,372,036,854,775,808 ( $-2^{63}$ ) and 9,223,372,036,854,775,807 ( $2^{63-1}$ )
  - ◊ Real Number
    - \* Numbers that can have decimal places
    - \* Negative values from  $-1.79E +308$  through  $-2.23E -308$ ; zero; and positive values from  $2.23E -308$  through  $1.79E + 308$
    - \* However the number of significant digits is limited to 15 decimal digits

◊ Boolean

- \* A true or a false

◊ String

- \* Unicode Text

◊ Date/Time

- \* Dates and Times in an accepted date-time representation
- \* Valid dates are all dates after December 30th 1899
- \* Stored as a floating point value
- \* Integer part is number of days after December 30th 1899
- \* Decimal part one second is equal to  $1 / ( 24 * 60 * 60 )$
- \* You can Add 1 to a date to increase the day by 1
- \* You can Subtract 1 from a date to decrease the day by 1

◊ Currency

- \* Values between -922,337,203,685,477.5808 to 922,337,203,685,477.5807 with four decimal digits of fixed precision.

◊ Blank

- \* A blank is a data type in DAX that represents and replaces SQL nulls. You can create a blank by using the BLANK function, and test for blanks by using the logical function, ISBLANK.

## 6.2.13 The Table Data Type

---

- In addition to scalar data types, DAX uses a table data type
  - ◊ Many functions require a reference to a table
  - ◊ Other functions return a table that can then be used as input to other functions.
  - ◊ In some functions that require a table as input, you can specify an expression that evaluates to a table
  - ◊ For some functions a reference to a base table is required, an evaluated table will not work
- When writing a Calculated Table the formula must return a table value

## 6.2.14 Data Type Conversions

---

- Each DAX function has specific requirements as to the types of data that are used as inputs and outputs.
  - ◊ A function may require an integer for one argument and a date for another
  - ◊ Other functions require text or tables
- DAX will attempt to automatically convert data into the correct type
  - ◊ You can type a date as a string and DAX will parse the string and attempt to cast it as one of the windows date and time formats
    - \* The following expression returns a date: = "7/4/2019"
    - \* This following expression returns the same date: "July 4th 2019"
  - ◊ TRUE is implicitly converted to the number 1
    - \* The following expression returns 3: = TRUE + 2
  - ◊ If you add values in two columns where one is a number represented as text "17" and the other as a number 17, DAX implicitly converts the string to a number
  - ◊ The following expression returns 24: = "12" + 12
- This is called implicit conversion

- If the implicit conversion fails, DAX will return an error

## 6.2.15 Implicit Conversions with Operators

---

- Each operator has its own set of rules regarding how it deals with operating on data of different types
- In most cases these rules are intuitive
  - ◊ For example a REAL \* CURRENCY becomes a CURRENCY
  - ◊ This makes sense, as a 2.5 of a \$20 item might cost \$50
- In other cases the rules may be unexpected
  - ◊ For example a REAL + CURRENCY becomes a REAL
    - \* But then, when would you add a currency to a number that isn't a currency?
- Some situations are very unusual
  - ◊ For example a DateTime / DateTime becomes a REAL
    - \* But you would usually use date functions to do this kind of work
- It's best to check the Microsoft documentation where you have questions, they provide a full set of tables that define the rules for each operator

## 6.2.16 Function Fundamentals

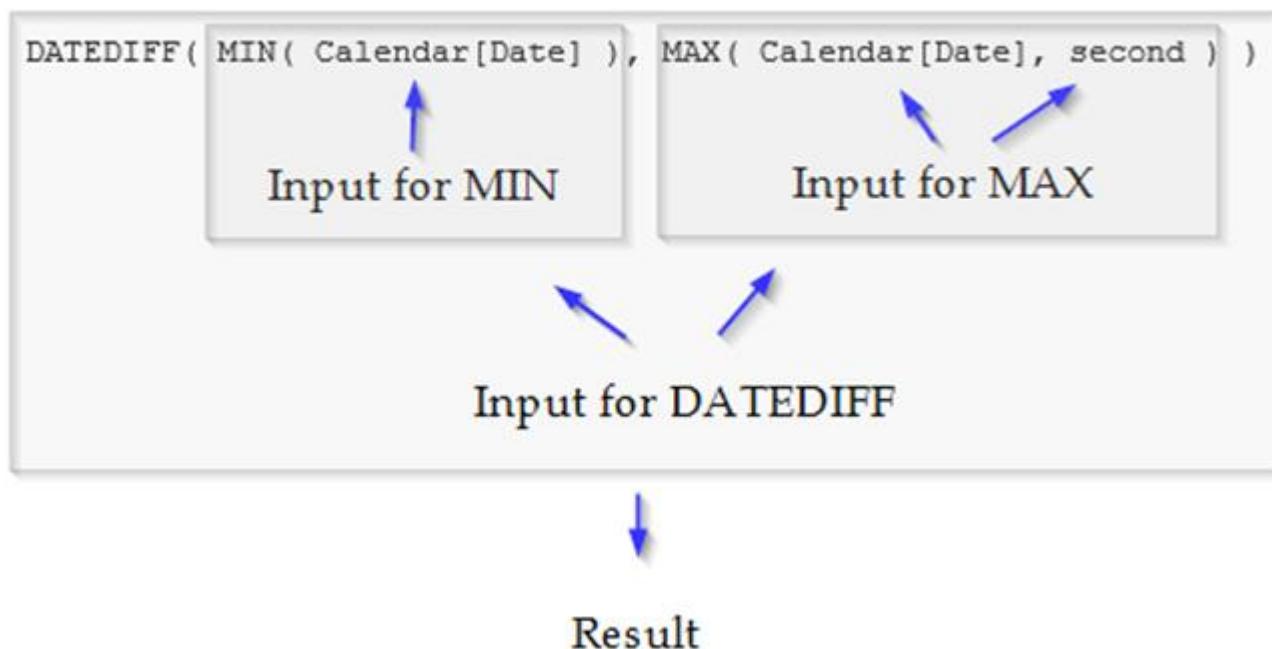
---

- DAX is a functional language, the execution flows with function calls
- Each Function consists of
  - ◊ Its name
  - ◊ A list of input parameters
  - ◊ A return type (implicit)
- We'll be covering many functions as the course continues
- Microsoft's reference can be found here:  
<https://docs.microsoft.com/en-us/dax/dax-function-reference>
- There are over 250 functions available in numerous categories

## 6.2.17 Function Syntax

- Your first step to learning a new function should be to reference the Microsoft documentation provided online for that function
- When calling a function, you will pass arguments into the input parameters, inside parenthesis
- Example

```
DATEDIFF(MIN(Calendar[Date]) ,MAX(Calendar[Date],second))
```



## 6.2.18 Order of Evaluation

- Functions are evaluated from the inside out
- Example

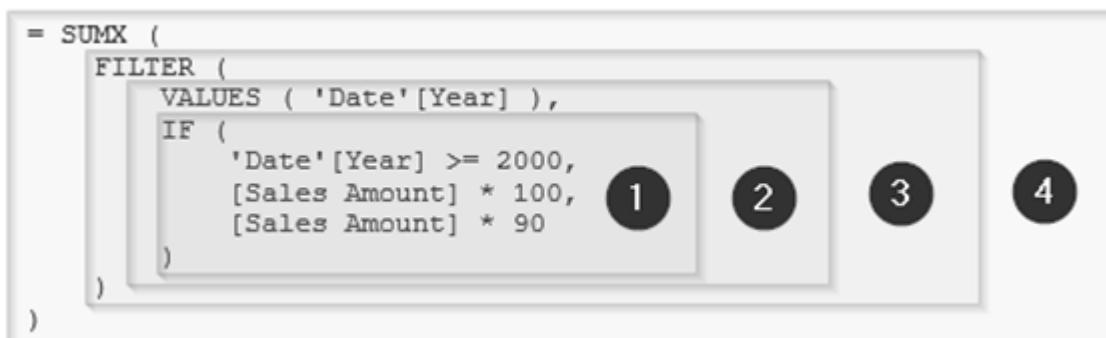
This formula

```
= SUMX(FILTER(VALUES('Date'[Year])), IF('Date'[Year] >= 2000, [Sales Amount] * 100, [Sales Amount] * 90))
```

- Can be expanded into a more readable format:

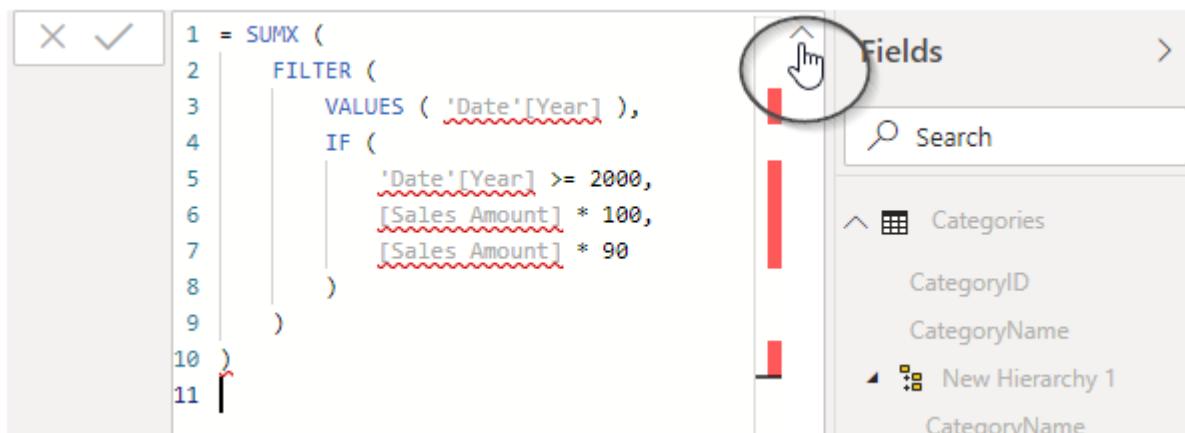
```
= SUMX (
    FILTER (
        VALUES ( 'Date'[Year] ),
        IF (
            'Date'[Year] >= 2000,
            [Sales Amount] * 100,
            [Sales Amount] * 90
        )
    )
)
```

- This gets executed from the inside out:



## 6.2.19 Formatting Formulas

- Power BI allows you to expand the formula bar to type multiple lines at once



A screenshot of the Power BI formula bar. The formula is:

```
1 = SUMX (
2   FILTER (
3     VALUES ( 'Date'[Year] ),
4     IF (
5       'Date'[Year] >= 2000,
6       [Sales Amount] * 100,
7       [Sales Amount] * 90
8     )
9   )
10 )
11
```

The formula bar has a circled up arrow icon at the top right, indicating it can be expanded. To the right of the formula bar is the Fields pane, which shows a hierarchy under the 'Categories' node: CategoryID, CategoryName, New Hierarchy 1, and CategoryName.

- You will find yourself thinking about writing formulas from the outside in or sometimes the inside out, instead of from left to right, or top to bottom

## 6.2.20 Using Online Reference Materials

---

- A quick web search to bring you to the official Microsoft reference for a specific function will save you time
- With over 250 functions and new ones added multiple times a year, it wouldn't hurt to spend a little time browsing through the Microsoft directory
- <https://docs.microsoft.com/en-us/dax/dax-function-reference>

## 6.2.21 Common Functions by Category

---

### ◊ Aggregation Functions

- \* SUM()
- \* AVERAGE()
- \* MIN()
- \* MAX()
- \* SUMX()
- \* ... and other X functions

### ◊ Counting Functions

- \* COUNT()
- \* COUNTA()
- \* COUNTBLANK()
- \* COUNTROWS()
- \* DISTINCTCOUNT()

### ◊ Logical Functions

- \* AND()
- \* OR()
- \* NOT()
- \* IF()
- \* IFERROR()

## ◊ Information Functions

- \* ISBLANK()
- \* ISNUMBER()
- \* ISTEXT()
- \* ISNONTEXT()
- \* ISERROR()

## ● Text Functions

- \* CONCATENATE()
- \* REPLACE()
- \* SEARCH()
- \* UPPER()
- \* FIXED()

## ● Date Functions

- \* DATE()
- \* HOUR()
- \* NOW()
- \* EOMONTH()
- \* WEEKDAY()

## Section 6.3

### Calculated Columns

## 6.3.1 Calculated Columns

---

- Calculated columns can be used to add a new field to a table within your model
  - ◊ A Calculated Column is computed for each row of data within the table it belongs
  - ◊ A Calculated Column's value can be used in other calculations as if it was a regular column
- As soon as you create a calculated column in the user interface, values are immediately calculated for each row and displayed within the table
  - ◊ Data is recalculated when you process (refresh) your data
  - ◊ Data is lost when the model is unloaded from memory, and then recalculated when the data is re-loaded
    - \* This happens when you close and open Power BI Desktop as the data does not remain in memory after the program closes

- Use of Calculated Columns when using DirectQuery is limited
  - ◊ This is because Power BI has no way of knowing when external records change, that may cause this value to need to be recalculated
  - ◊ Only data in the current row can be used in an expression
  - ◊ DirectQuery provides the advantage of real-time data, but this is one of many disadvantages

## 6.3.2 Calculated Column Examples

---

### Example

You might combine [First Name] and [Last Name] columns into a [Full Name] column

```
[Full Name] = Customer[Last Name] & ", " & Customer[First Name]
```

### Example

You might add a new field that identifies with a boolean if an order shipped late

```
[Shipped Late] = [ShipDate] > [RequiredDate]
```

### Example

You might combine the year and quarter into a single field for display purposes

```
[Quarter Name] = [Year] & " Q" & [Quarter]
```

### Example

You might calculate a line item gross sales

```
[Sales] = [Price] * [Quantity] * ( 1 - [DiscountPercent] )
```

## 6.3.3 Walkthrough: Adding Calculated Columns

---

- Lets add a few calculated columns together:
- Northwind Modeling
  - ◊ Employee Age
  - ◊ Employee Years Employed

## 6.3.4 Row Context

---

- When a calculated column is being evaluated, it is evaluated once for each row, within the context of that row
- This is called Row Context
- To consider values outside of the context of the row that is being processed, you need to use a special function to walk the relationships of the model
  - ◊ RELATED()
  - ◊ RELATEDTABLE()
  - ◊ USERELATIONSHIP()

## 6.3.5 RELATED()

---

- RELATED() is used to navigate a predefined active relationship to grab columns from another table
- RELATED() only works to follow a relationship where only a single row would be returned
  - ◊ This means the Row Context table would need to be the many side of a one-to-many relationship
  - ◊ It would also work with one-to-one relationships
- Example

With Calculated Columns, it is common to use RELATED() to flatten normalized data. The AdventureWorksDW2017 database has three tables: DimProduct, DimProductSubcategory, and DimProductCategory that we want to flatten into DimProduct. If all we are interested is the Category and Subcategory names, we can add the following Calculated Columns to Products:

```
[Subcategory Name] =  
RELATED(DimProductSubcategory[EnglishProductSubcategoryName])  
  
[Category Name] = RELATED(DimProductCategory[EnglishProductCategoryName])
```

## 6.3.6 RELATEDTABLE()

---

- RELATEDTABLE() is used to navigate a predefined active relationship to grab multiple rows from a related table
  - ◊ It returns a table value
- The result of RELATEDTABLE() needs to be turned into a scalar value before it can be useful to a Calculated Column, which must return a scalar value
- Iterator Functions can be used to turn a table value into a scalar result
- Example

In this example, the table value result of RELATEDTABLE() is passed to the SUMX() Function, which is an iterator. The SUMX() function will add up all of the values in the LineTotal column of the SalesOrderDetail Table, bot only for rows that relate to the current Row Context

```
TotalSales =  
SUMX(RELATEDTABLE(SalesOrderDetail), SalesOrderDetail[LineTotal])
```

SUMX(), MINX(), MAXX(), COUNTX() are all Iterators and might be used with the result of RELATEDTABLE()

## 6.3.7 USERELATIONSHIP()

---

- USERELATIONSHIP() works like RELATED() except that you can define an inactive relationship to use
- This is handy because xVelocity data models only allow a single active relationship between two tables
  - ◊ If more than one relationship is added, all but one will be disable
  - ◊ This is to solve an issue with ambiguity
    - \* This frequently comes up when working with Date tables
  - ◊ For example an order might have the following dates:
    - \* Order Date
    - \* Arrive By Date
    - \* Ship Date
  - ◊ While this solution is fast and easy, another solution might be to utilize role playing tables (covered later)

## Section 6.4

### Measures

## 6.4.1 Measures

---

- Measures are the second of four types of calculations that can be added to a model with DAX
- Measures are possibly the most important feature of DAX
  - ◊ This is because they are providing summaries which can concisely tell someone what all the detail below it means
  - ◊ This kind of calculation cannot be done in Power Query or SQL without severe performance implications
    - \* This is because to provide a value data from multiple rows are required
    - \* Languages like DAX or MDX are needed
- These are added at the table level but they do not provide one value per row
- A measure is an expression that can summarize the data in the table that it belongs to
- Measures are computed in real-time when a report queries the data model, they are not computed when you create them

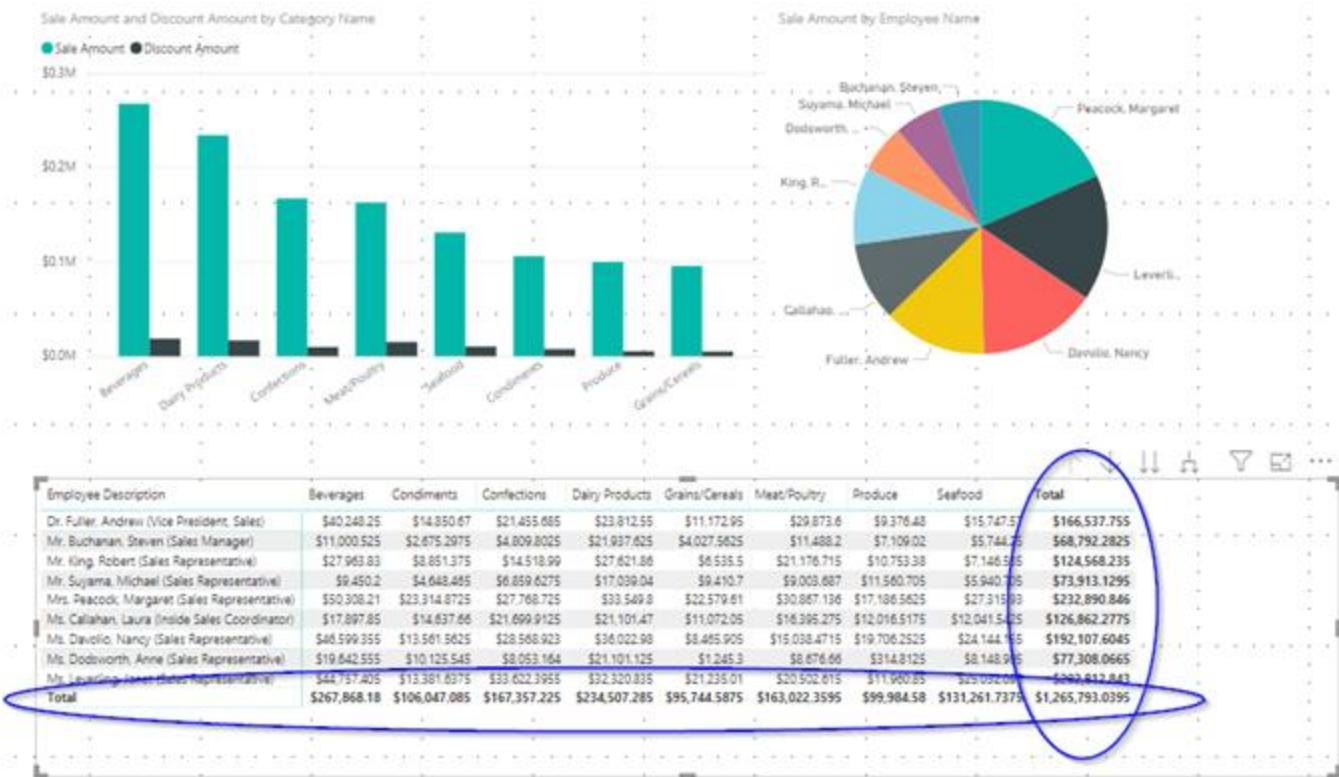
## 6.4.2 Implicit Measures

- Any time you add a column to a visual and what you see is an aggregate of many values instead of a single value from a single row, you are using a measure
- Notice below that Sale Amount has been added to a visualization. Implicitly, this is actually the formula: Sum(Sales[Sale Amount])

The screenshot shows the Power BI Fields pane on the right side of the interface. In the 'Values' section under the 'Sales' category, the 'Sale Amount' field is highlighted with a blue oval. This indicates that it is the current selected measure. The 'Fields' pane also shows other measures like 'Quantity' and 'Unit Price' in the Sales category.

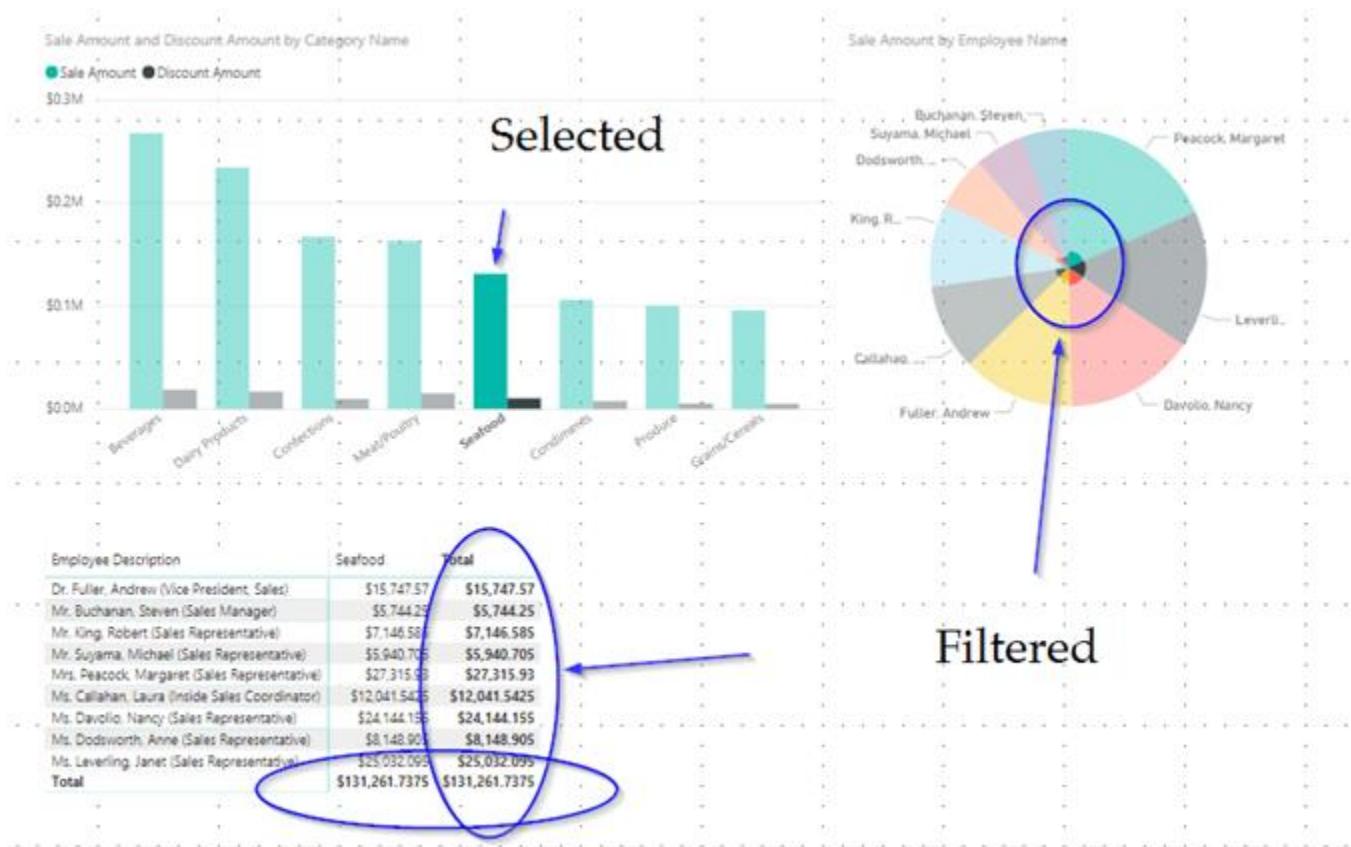
## 6.4.3 Filter Context

- How data gets summarized is affected by the Filter Context
  - ◊ In the image below you see many numbers, but every single number is the result of the same expression:
  - \* `SUM(Sales [Sales Amount])`
  - ◊ The result of the expression is different at each location on the grid because the filter context is different, so a different subset of rows are available to the calculation



## 6.4.4 Interactive Filter Context

- When a user interacts with report items, selecting an area changes the filter context
- All other report items will limit themselves to display data based on this new filter context



## 6.4.5 Example Measures

---

- **Example**

This creates a "Total quantity sold" measure. Although such a simple measure can be created implicitly by selecting Sales[Quantity] in a report, defining it like this gives you the opportunity to establish a better name for the calculation

```
[Total quantity sold] = SUM(Sales[Quantity])
```

- **Example**

Here we compute the actual sales values for the given filter context

```
[Actual sales values] =
SUMX(Sales, [Price] * [Quantity])
```

- **Example**

Here we compute the list price values for the given filter context

```
[List price values] =
SUMX(
    Sales,
    RELATED('Product'[ListPrice]) * [Quantity]
)
```

- **Example**

Here we establish the percentage the customer paid of the list price

```
[Pct of full value] =
DIVIDE(
    SUMX(Sales, [Price] * [Quantity]),
    SUMX(Sales, RELATED('Product'[ListPrice]) * [Quantity])
)
```

## 6.4.6 Demonstration: Adding Custom Measures

---

- Lets add a few measures to a Power BI Project

## 6.4.7 Measures verses Calculated Columns

---

- Use A Calculated Column when
  - ◊ You will later want to filter on the resulting value
  - ◊ Need to use the value to support another calculated column
- Use a Measure when
  - ◊ You want to use Percentages in a report
  - ◊ You want to use Ratios in a report
  - ◊ You need complex aggregations
  - ◊ You are describing how you'll present a value that will look different in different filter contexts

## Section 6.5

### Exercises

## Exercise 6.1 Online review of Function Reference

---

### Introduction

Getting a sense of what some options may be will help you later when you are brainstorming for useful calculations or working through a specific problem.

1. Spend 10 minutes reviewing the list of 250 DAX Functions listed Online. Review a few in each category and then move on

<https://docs.microsoft.com/en-us/dax/dax-function-reference>

## Exercise 6.2 Northwind Modeling – Simple Calculated Columns

---

Adding Employee Age, Employee Years Employed – this may have already been completed during lecture demonstration.

## Exercise 6.3 Northwind Modeling – Working with Percentages

---

### Introduction

In this exercise we will demonstrate how percentages can be a dangerous value when expressed as a column instead of a measure.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. In the Data view, select the Discount Percent column of the Sales table.
3. Using the Modeling Ribbon, set the format of Discount Percent to Percentage.
4. Observe the data in the Discount Percent column. You may need to scroll down a while in the data before you see numbers other than 0.
5. In the Report view, create a new report tab named Percentage Problems.
6. Add a table with the following fields:
  - \* Category
  - \* Sales Amount
  - \* Discount Amount
  - \* Discount Percent.

If you look at the numbers in the Discount Percent column you will notice that the Discount Percent for the Beverages category is 2,500%. It should be obvious that this number is not useful to us. By default, all of the percentages have been added together. An implicit measure was created that uses the expression `SUM([Discount Percent])`.

7. Using the steps outlined below, change the implicit measure of the Discount Percent column to use an average instead of a sum:

The screenshot shows the Power BI Desktop interface with a table visual titled "Northwind-Step10 - Power BI Desktop". The table contains data for various product categories with columns for Category, Sales Amount, Discount Amount, and Discount Percent. The "Discount Percent" column shows values like 1500.00% for Beverages and 1155.00% for Total. A blue arrow points from step 1 to the table. Another blue arrow points from step 2 to the "Discount Percent" item in the "Values" section of the Visualizations pane.

Category	Sales Amount	Discount Amount	Discount Percent
Beverages	\$267,868.18	\$18,658.77	1500.00%
Seafood	\$131,281.7375	\$10,361.3525	768.00%
Dairy Products	\$234,507.265	\$16,823.215	7056.00%
Confections	\$167,357.225	\$9,741.875	1102.00%
Condiments	\$106,047.065	\$7,647.665	1137.00%
Meat/Poultry	\$163,022.3995	\$15,166.4405	1155.00%
Grains/Cereals	\$95,744.5875	\$4,902.2125	51.00%
Produce	\$99,584.58	\$5,284.02	43.00%
Total	\$1,265,793.0395	\$88,665.5505	12109.00%

- \* 1: Select the Table
- \* 2: Click on the chevron next to the Discount Percent column in the Values section of the Visualizations pane
- \* 3: In the context menu that appears, change the selected item from Sum to Average

8. Observe the values in the Discount Percent column. You'll find them to be in the 4-6% range, and at a first glance this looks like it could be correct. However, this figure represents an average of the values at the detail level, and that is not the Discount Percent when summarizing detail at the Category level.

- \* This number is not useful and is dangerous because it could be easily misinterpreted!

Looking at the Beverages row, using a calculator we have found that  $\$18,658 / \$267,868 = 0.06965$ . The value listed here is 6.19%.

To solve this problem, we need to use a measure. Measures are calculated separately at each grouping level as needed by the report where it is utilized. They also take into account implicit filters based on rows the user may have selected within the report.

9. In the Data view, right-click on the Sales table of the Fields pane and choose New Measure.

10. Enter the following in the formula bar:

```
Discount Percent Measure = SUM(Sales[Discount Amount]) / SUM(Sales[Sales Amount])
```

You will notice that after adding this, a column does not appear in the table. This isn't calculated until it's used in a report.

11. To protect against the [Discount Percent] column from being used in reports, mark it as hidden from the report view. This will coach users to use the [Discount Percent Measure] instead.

12. Navigate to the Report pane and add the [Discount Percent Measure] to the table.

13. Set the [Discount Percent Measure] column to be a percentage.

14. Observe the difference between the Discount Percent Measure and Discount Percent columns.

15. Remove the [Discount Percent] column.

16. Save your work using a new Step Number.

## Exercise 6.4 Northwind Modeling – Context based age calculations

---

### Introduction

Previously we added a column to the Employees table that calculates the age of the employee. What if we want to look at the ages of employees at the time they made specific sales? This age would belong to the Sale record, instead of the Employee record.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. In the Reports View, create a new Report Tab named Ages
3. Create a clustered column chart with Employee's Age as the Axis and Sales Quantity as Values
4. Notice that all employees are over the age of 50, this is an older dataset and it is showing us the current age of the employees who sold products! That isn't useful.
5. Notice that the number of distinct ages found is 8 (count the bars)
6. Create a new Calculated Column on the Sales table using the following expression:  
`Emp Age At Order = FLOOR( (RELATED(Orders[Order Date]) - RELATED(Employees[Employee Birth Date])) / 365,1)`
7. Back on the Ages report, replace the Employee's Age with the Emp Age At Order as the axis field on the chart.
8. Notice that we have over 20 distinct ages now, as employees changed age over the three-year period that sales were recorded.
9. Add a table to the report.
10. Add the following columns to the table:

- Sales[Quantity]
- Orders[OrderDate].[Date Hierarchy].[Year]
- Sales[Emp Age At Order]
- Employee[Employee's Age]

11. Change the implicit measure that was created for both Sales[Emp Age At Order] and Employee[Employee's Age] to display average instead of sum
12. Notice that the Average of Emp Age at Order values are different for each year, while the Average of Employee's Age are the same. This is because Emp Age at Order belongs to the Orders table so it's averaging one age per order instead of one age per employee.
13. Save your work using a new Step Number.

## Exercise 6.5 Northwind Modeling – Flattening many-to-many relationships

---

### Introduction

One thing that Power BI is not great with is dealing with many to many relationships between dimensions. In the Northwind database we have Employees that can belong to multiple territories, and multiple Territories can be handled by the same Employee. To make reporting simpler, We're going to collapse this descriptive list into a single column of text.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. Add a Calculated Column to the EmployeeTerritories table named Employee Territory, set it equal to the Territory column from the related Territories row.
3. Add a Calculated Column to the EmployeeTerritories table named Employee Region, set it equal to the Region column from the related Region row.
4. Hide the Territories and Region tables from the report view.
5. Add a Calculated Column to the EmployeeTerritories table named [Employee Territory Description], combine the Territory and Region fields together to get an output that looks like this (include the braces):

[Wilton (Eastern) ]  
[Atlanta (Southern) ]

6. Add a Calculated Column to the Employees table named [Employee Territories] that uses the CONCATENATEX() iterator to create a string the contains all of the territories that an employee covers, delimited with a comma and a space. You should get values that look like this:

[Phoenix (Western)] , [Scottsdale (Western)] , [Bellevue (Western)] ,  
[Redmond (Western)] , [Seattle (Western)]

\* Hint: You will need to use both CONCATENATEX() and RELATEDTABLE().

7. Create a new report tab named Employees that lists employees and the territories they belong to as well as the total sales amount for each employee.
8. Save your work using a new Step Number.

# Module 7

## Advanced DAX Concepts

## Section 7.1

### Filtering

## 7.1.1 Filtering

---

- Filters are used to filter a subset of values from the data that are used in calculation
  - ◊ You might filter values based on a dimension's attribute
    - \* For example filtering Sales Amount by Product Category
  - ◊ You might "drill down" to see how a large summary is split up
    - \* Such as expanding Category and looking at the Subcategories or Products below it
  - ◊ You might perform Period-Based comparisons
    - \* To compute how this year is doing compared to last year
    - \* To determine year-to-date or month-to-date sales
- You can apply basic filters with the CALCULATE() function
- You can apply advanced filters with the FILTER() function
- You can undo implicit filters with the ALL() function()

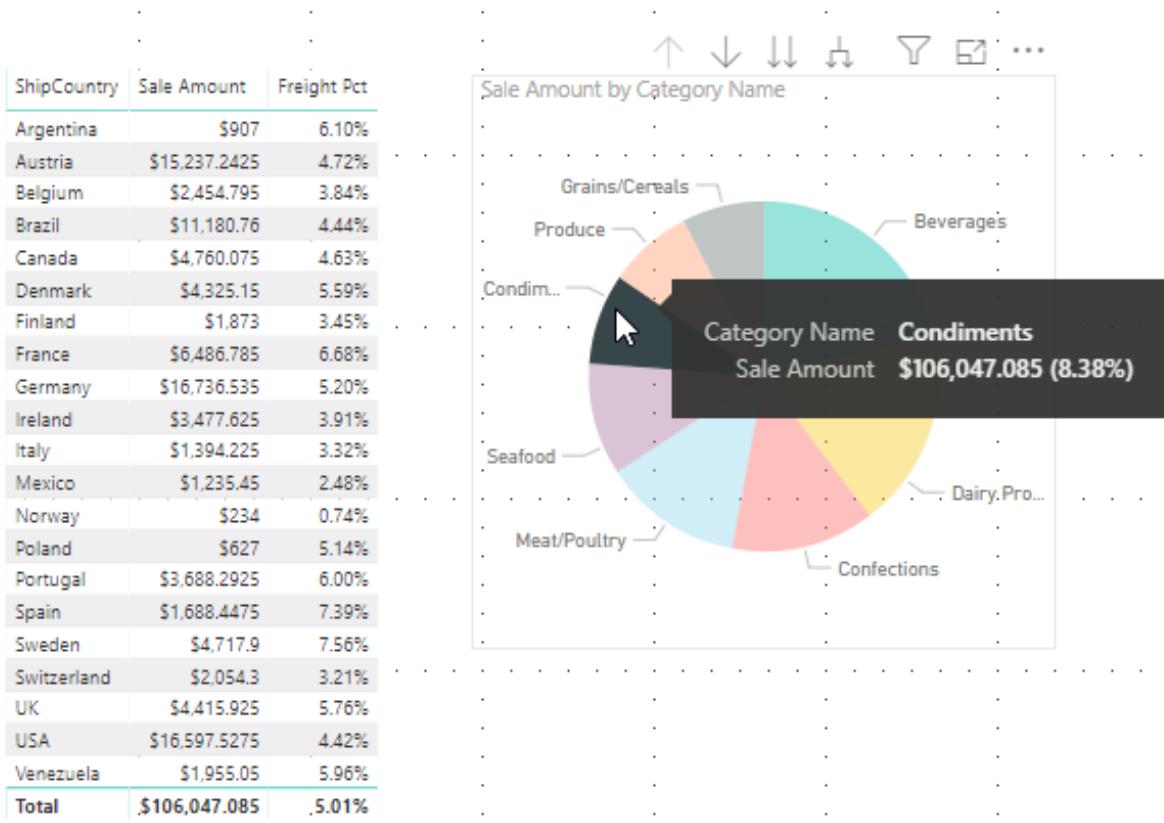
## 7.1.2 Implicit Filtering

---

- Implicit Filtering is filtering applied by the user or the layout of the report
  - ◊ When you add a field to a bar chart, the DAX engine is implicitly filtering by that field for each bar
    - \* This means that each bar has its own implicit filter context
  - ◊ When you click on a pie slice within a pie chart, the DAX engine changes the filter context so that all of the other visualizations on the page only see the data represented by that slice of the pie

## 7.1.3 Implicit Filtering Example

- We can see implicit filtering at two places in the below visual
  - At the intersection of Argentina and Sale Amount, we see that SUM(Sale Amount) has been filtered to show only the sales that happened in Argentina
    - The value of \$907 represents Argentina, no other country.
  - Because the Condiments slice has been selected in the pie chart, all of the visuals are only showing Condiment values
    - This means that the \$907 of Sales Amount in Argentina is only the sales in Argentina for Condiments



## 7.1.4 Explicit Filtering

---

- Explicit filtering is filtering applied by the DAX formula itself
- Explicit filtering will override implicit filtering on a column by column basis
- You can specify filters in DAX expressions with these functions:
  - ◊ CALCULATE()
  - ◊ FILTER()
  - ◊ ALL()

## 7.1.5 CALCULATE() function

---

- The CALCULATE() function is the fastest way to explicitly filter data in DAX
- Use CALCULATE() to override filters applied by the user

### Example

```
Sum Green Products Sold  
= CALCULATE ( SUM ( Sales[Total] ), [Color] = "Green")
```

- Using the CALCULATE() function by itself is very fast, but is limited to filtering on a single column's value
  - ◊ This is because it uses the fact that the columnar database has a single column's data stored together to its advantage

### Example

The following will work:

```
Average Freight for Products Greater than Ten =  
CALCULATE(AVERAGE(Sales[Freight Amount]), Sales[Sale Amount] > 10)
```

### Example

The following will not work:

```
Average Freight for Products Greater than Ten =  
CALCULATE(AVERAGE(Sales[Freight Amount]), Sales[Unit Price] * ( 1 -  
[Discount Percent]) > 10)
```

## 7.1.6 FILTER() function

---

- The filter function can be used when you need to consider multiple columns in your filter criteria
- The FILTER() function takes two parameters
  - ◊ A table value that represents the input data
  - ◊ An expression that will be evaluated for each row in the input data
  - ◊ We will learn later that the filter is an iterator
- The FILTER() function returns a table value
  - ◊ The return table value will contain each row from the input where the expression evaluated to true
- The FILTER() function can be used anywhere that expects a table
- The result of the FILTER() function is passed to another function for operations such as custom aggregations
- FILTER() can be used inside of a CALCULATE()
  - ◊ But when you do this, it significantly slows down the calculation

## 7.1.7 FILTER() function example

---

### Example

The following example is a simple example of the FILTER() function.

```
Products = FILTER(
    'Products',
    'Products'[BrandName] = "GoodForYou"
)
```

- Example

In this example, we're demonstrating that FILTER() can utilize multiple columns in it's filter criteria.

```
HighMarginProducts =
FILTER
(
    'Products',
    AND(
        'Products'[BrandName] = "GoodForYou",
        'Products'[UnitPrice] > 'Products'[UnitCost] * 2
    )
)
```

## 7.1.8 ALL() Function

---

- The ALL() function is a way to clear any implicit filters
  - ◊ Can be applied to a specific column
  - ◊ Can be applied to an entire table
- This is useful in situations where you want to perform ratio between the filtered value and the whole value

### Example

```
SalesPercent =  
DIVIDE(  
    SUMX(  
        Sales,  
        Sales[Quantity] * Sales[NetPrice]  
    ),  
    SUMX(  
        ALL(Sales),  
        Sales[Quantity] * Sales[NetPrice]  
    )  
)
```

## Section 7.2

### Calculated Tables

## 7.2.1 Calculated Tables

---

- Calculated Tables are the third of four types of calculations that can be added to a model with DAX
- A Calculated Table must be a formula that returns a Table data type
  - ◊ Such as the FILTER() function
- Using Calculated Tables is sometimes easier than repeating the same FILTER() expression in different calculations
- Can be used to create "Role Playing" tables
  - ◊ Used to overcome the limitation of having only a single active relationship between tables
  - ◊ Will be discussed in detail when we cover Time

## 7.2.2 Calculated Table Example

---

### Example

The following example creates a table named Global Sales that represents the combined data of the NorthAmericanSales and EuropeanSales tables.

```
Global Sales = UNION(NorthAmericanSales, EuropeanSales)
```

### Example

The following example creates a table named Product Categories that combines the Categories and Subcategories tables through an inner join.

This could be used as an alternative way to flatten data from multiple tables into a single table

This might be preferred when more than 1 or 2 columns are being brought over.

```
Product Categories = NATURALINNERJOIN(Categories, Subcategories)
```

### Example

The following example creates a table named Order Status that "hard codes" text descriptions of numeric OrderStatusId values.

```
Order Status = DATATABLE
(
    "OrderStatusId", Integer, "Status", String,
    {
        {1, "In Cart"}, {2, "Placed"}, {3, "Acknowledged"}, {4, "Shipped"}, {5, "Delivered"}, {6, "Cancelled"}
    }
)
```

## Section 7.3

### Iterators

## 7.3.1 Iterators

- An Iterator is a special class of Functions
- Many of the built-in iterator functions end with the letter X
- An iterator accepts a table value and an expression
  - ◊ FILTER() is an example of an iterator that returns a table value
  - ◊ SUMX() is an example of an iterator that returns a scalar value
- Example

In this example, Sales[Net Price] \* Sales[Quantity] is going to be evaluated for each row in Sales, then each one of the return values will be added together to produce a sum

```
Total Sales = SUMX( Sales, Sales[Net Price] * Sales[Quantity] )
```

The diagram illustrates the components of the SUMX function. A box highlights the 'Sales' parameter. Three blue arrows point from the text 'Table Value', 'Expression', and 'Iterator' to the corresponding parts of the highlighted code.

- Table Value: Points to the word "Sales" in the highlighted code.
- Expression: Points to the multiplication operator "\*" in the highlighted code.
- Iterator: Points to the opening parenthesis "(" in the highlighted code.

## 7.3.2 Evaluation Context

---

- Evaluation Context a combination of Row and Filter contexts in which a formula is evaluated
- Row Context
  - ◊ Limits the current row in an iterator
  - ◊ Used in Calculated Columns
  - ◊ Used in Iterators
  - ◊ Does not propagate filters
  - ◊ Uses RELATED() or RELATEDTABLE() to access data in other tables
- Filter Context
  - ◊ Combination of visual filters, user filters, and DAX
  - ◊ Can be changed with CALCULATE()
  - ◊ Can be overridden with ALL()
  - ◊ Propagates filters to other tables

## 7.3.3 Common Iterator Functions

---

- There are many built in iterator functions, below is a small partial list of common iterators
  - ◊ SUMX()
  - ◊ MINX()
  - ◊ MAXX()
  - ◊ COUNTX()
  - ◊ COUNTAX()
  - ◊ RANKX()
  - ◊ CONCATENATEX()
  - ◊ AVERAGEX()
  - ◊ PRODUCTX()
  - ◊ CONTAINSROW()

## 7.3.4 Nesting Iterators

---

- It is a common practice to nest iterators
  - ◊ Think of this as defining multiple steps
  - ◊ Write these from the inside out
- Be aware of possible performance implications
  - ◊ Sometimes nesting iterators is the only way to solve the problem but take some time to think of other possible strategies first

```
[RanksTable] =  
RANKX (  
    ALL( Products ),  
    SUMX (  
        RELATEDTABLE( InternetSales ),  
        'InternetSales'[SalesAmount]  
    )  
)
```

## Section 7.4

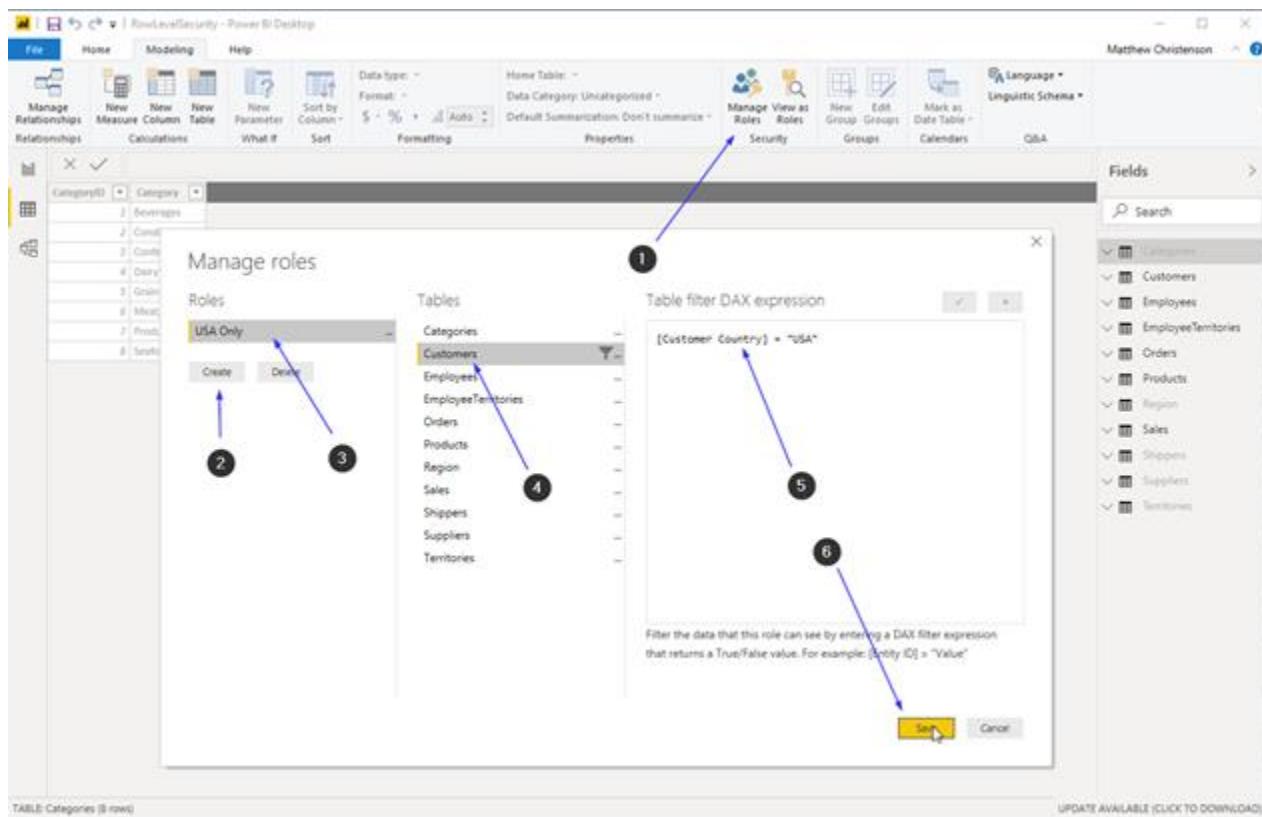
### Row Level Security

## 7.4.1 Row Level Security

---

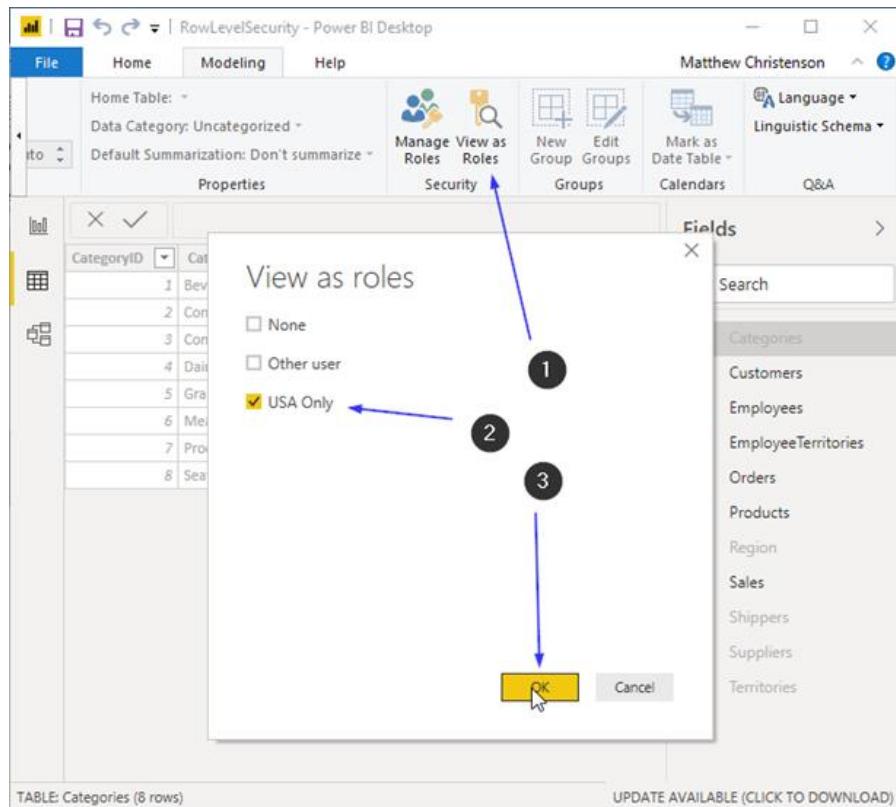
- Row Level Security can be used to restrict data access for given users
- Filters are used to restrict data access at the row level
- Filters are defined within roles
- This course speaks only to how DAX is involved in restricting a user's access to data
  - ◊ If using the Power BI Service some additional research should be done into access rules on the underlying datasource

## 7.4.2 Defining RLS

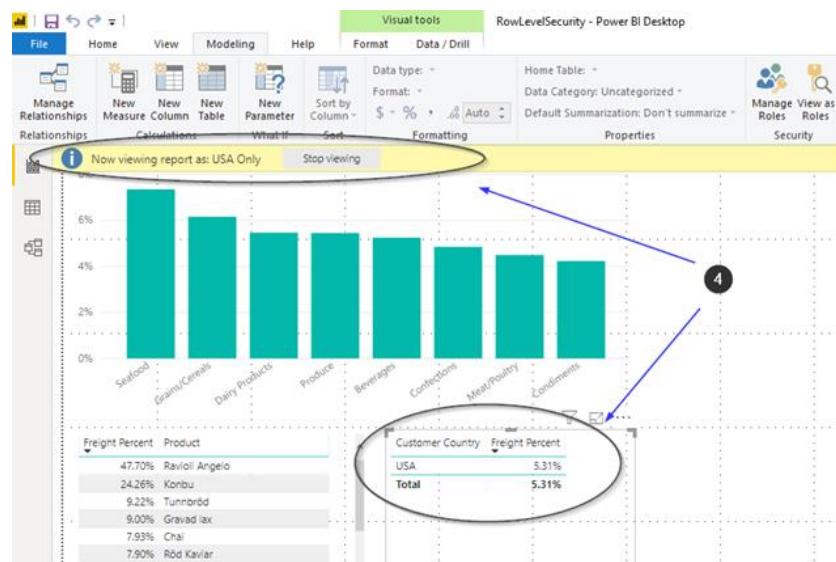


1. On the Modeling Tab, select Manage Roles
2. In the Manage Roles Dialog, Click Create
3. Provide a name for the role
4. Select a table with data you want to filter on
5. Provide a DAX expression that restricts records
6. Click Save

## 7.4.3 Testing RLS



1. Select View as Roles
2. Chose the Roles
3. Click Okay
4. View a report and see that data has been filtered



## **Section 7.5**

## **Exercises**

# Exercise 7.1 Northwind Modeling – Resolving Granularity Issues

---

## Introduction

In this exercise we are going to consider a problem with our current Northwind Model. Each table in a Power BI Model should either be a Data (fact) Table or a Lookup (dimension) Table.

Data Tables should never be related to each other, instead Data Tables should be related to Lookup Tables, it is only through Lookup tables that Data Tables should relate - representing the dimensions that they have in common.

We have two related Data Tables, the Orders table, and the Sales (Order Detail) table, and these both contain measures at different grains.

The Orders table has the following measure:

- Order Freight Amount

The Sales table has many measures and is more granular than the Orders table. If we create a report that puts the Sum(Sales[Sales Amount]) side by side with the Sum(Orders[Order Freight Amount]), we will find that the Orders record repeats for each product in the order, which incorrectly inflates the Sum of Order Freight Amount.

We want to move Order Freight Amount from the Orders table to the Sales table..

Our goal is that if we total these two side by side, that totals related to Freight become accurate. We may also be able to gain some insight around how individual products influence freight costs at the order level.

There's a business rule that the Freight Amount belongs to the order. It isn't the products that directly drive the cost of Freight, because we're putting multiple products in a box and shipping an order as a unit. So there is not a perfect fix to this problem.

A compromise might be to split the cost of shipping across each Sales line based on the percentage of the total quantity of that order that the sales line represents. This isn't a perfect solution, but it may provide some insight into our shipping costs so that we can start to see trends around what items are costing to ship.

## Lets begin!

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. Our first objective is to understand the total number of items being shipped in an order. Create a Calculated Column on the Orders table named [Order Quantity]. Set it equal to the Sum of the Sales[Sales Quantity] column for each of the rows in the Sales table that relate to this row in the Orders table.
  - \* Hint: You will need to use the RELATEDTABLE() and SUMX() functions.
3. Hide both the [Order Quantity] and [Order Freight Amount] columns in the Orders table, so that they will not be directly used in reports. Both are not the correct granularity to be used in this model.
4. Add a Calculated Column to the Sales table named [Percent of Order Quantity], set it equal to the percentage of the order quantity that this specific line item represents. Pull the order quantity from the column you created in the order table.
  - \* Hint: Divide the Sales[Sales Quantity] by RELATED(Orders[Order Quantity])
5. Format [Percent of Order Quantity] as a percentage, and hide it from the report view. This is a supporting calculation and should not be used directly in reports.
6. Add a Calculated Column to the Sales Table named Sales Freight Amount. Set it equal to the order level freight amount multiplied by the Percentage of Order Quantity.
  - \* Hint: No hint on this one, but if you're stuck you might reflect on previous hints!
7. Add a Calculated Column to the Sales table named Sales Amount /w Freight. Set it equal to the Sales Amount plus the Sales Freight Amount.
8. Add a Measure to the Sales table named Sales Freight Percent. Use the following expression:  

```
Sales Freight Percent = Sum(Sales[Sales Freight Amount]) /  
Sum(Sales[Sales Amount /w Freight])
```
9. Set the Sales Freight Percent column to be a percentage.
10. Create a new Report tab I the Report view and name it Freight Reports

11. Using the values that we added to our data model above, create a report that identifies the:

- a) Product Category with the highest Freight Percent.?
- b) Product with the highest Freight Percent?
- c) Country with the highest Freight Percent?
- d) The Category with the highest freight percent within the United States?
- e) What Country has the highest freight percent when shipping products in the Beverages Category?

12. The answers are: Seafood, Konbu, Argentina, Seafood, and Italy

These numbers are prone to error because of the way that we took an order level value and spread it out among its children.

We could have selected a few different ways to have spread the freight costs out, if this was an important metric we might take some time to experiment with what effect various calculation methods may have:

- \* We could have given each line an equal amount. Four lines would each get 25% of the cost of the freight, regardless of Quantity per line.
- \* We could have utilized the percentage that a single line represents of the Sales Amount for the total order, instead of the percentage of the quantity.
- \* We could have studied the business details around what causes a product's freight to increase - in the real world we may have product weight and size available, for example..
- \* ... there are certainly other considerations.

Perhaps we are considering offering Free Shipping promotions, and we want to understand what the cost of such a promotion would be, and raise prices for product groups that are most effected?

13. Save your work using a new Step Number.

## Exercise 7.2 Northwind Modeling – Row Level Security

---

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. On the Modeling Tab of the Ribbon select Manage Roles.
3. Click Create to create a new role, and name it US ONLY.
4. Click on the Customers table and enter the expression: [Customer Country] = "USA"
5. Click Save in the Create a Role dialog.
6. Navigate to the Freight Reports tab on the Reports view. Notice we have data for many countries.
7. On the Modeling Tab of the Ribbon select View As Roles.
8. Check US ONLY and click Okay. Observe that the report now limits data to a single country.
9. Navigate through the other reports and notice that every one of them now returns a smaller subset of data.
10. Save your work using a new Step Number.

# Module 8

## Date and Time Intelligence

## Section 8.1

### Dates and Date Tables

## 8.1.1 Date Dimensions

---

- One of the most common dimensions that you will want to report on is a date
- A date field is not adequate without a date table that provides supporting attributes
- Power BI Desktop generates automatic hidden date tables for each date column
- You will frequently want to create your own, more powerful, date tables

## 8.1.2 Auto date/time in Power BI Desktop

---

- Power BI Desktop offers an option called Auto date/time
- Auto date/time allows for convenient time intelligence reporting based on date columns loaded into a model
  - ◊ It allows report authors to filter, group, and drill down by using calendar date periods (years, quarters, months, and days)
- Auto date/time works automatically when the following conditions are met
  - ◊ The table storage mode is import
  - ◊ The column data type is date or date/time
  - ◊ The column isn't the many side of a model relationship
- Behind the scenes, the DAX CALENDAR() function is used to create a date table with the following columns:
  - ◊ Day, MonthNo, Month, QuarterNo, Quarter, Year
- The Model Language is followed generating month names

- The table will contain a full calendar year encompassing all date values stored in the model date column
- The auto date/time tables are permanently hidden, they cannot be seen
  - ◊ Date columns expand into hierarchies for selection in visualizations

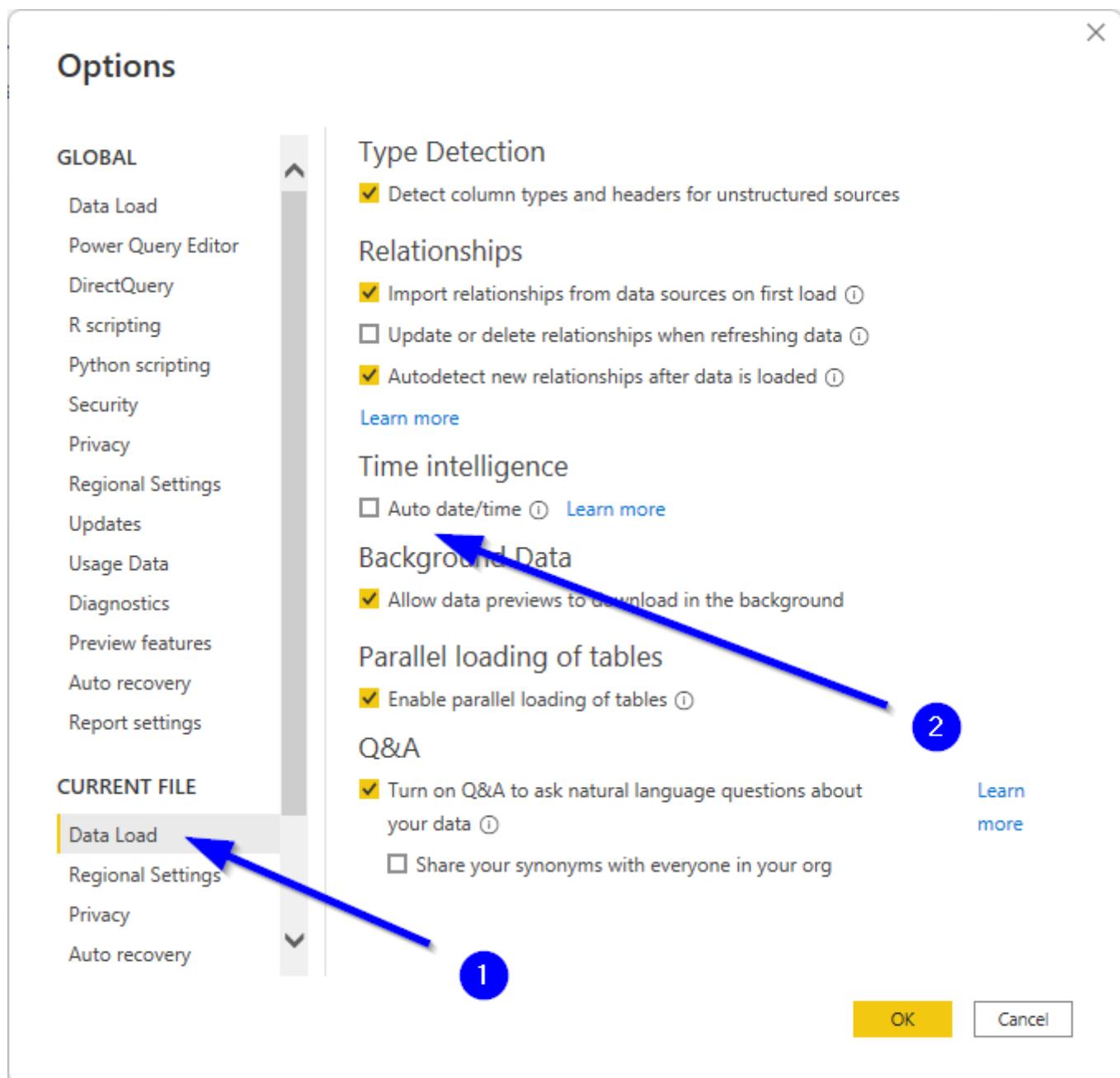
## 8.1.3 Auto date/time limitations

---

- These are useful for quick single-table graphs and very basic needs
- These have limitations
  - ◊ You cannot add fields to these tables
  - ◊ They do not accommodate fiscal calendars
  - ◊ They do not include weeks
  - ◊ Two Data Tables cannot share a dimension for shared filtering capabilities
- In all but very simple scenarios, you will benefit from creating explicit date tables instead of this built-in behavior

## 8.1.4 Disabling Auto date/time

- If you are not going to use the Automatic Date/Time feature it is best to turn it off for your project
  - ◊ This will improve performance
- You can turn it off in the Data Load section in Options



## 8.1.5 Date Tables

---

- A Date Table is a table with one row for each day
- A Date Table is broken down into many columns that describe attributes of that day
  - ◊ Common Examples
    - \* Year
    - \* Quarter
    - \* Month Number
    - \* Month Name
    - \* Day of Year
    - \* Day of Month
    - \* Day of Week
    - \* Day Name
    - \* Week Number
    - \* Is Weekday
    - \* Is Weekend
- Frequently you'll see a second group of columns representing Fiscal Years verse Calendar years

- Each Organization may have different needs within a date table
  - ◊ An organization with fiscal years that do not align with calendar years might need a second grouping of columns
  - ◊ Some organizations may need multi-lingual descriptive columns

## 8.1.6 Holiday Fields

---

- If you do business in a single country you might add an IsHoliday field, or names of holidays
- If you do business in multiple countries, you may end up with one row for each Day/Country combination

## 8.1.7 Time Tables

---

- Time is not usually considered in a date table
- If time of Day is required, this is usually a separate Table
- For some organizations, a time table will have exactly 1440 records, one for each minute in a day
- For some organizations, a time table will have exactly 24 records, one for each hour in a day
- The following columns may be included
  - ◊ TimeKey
  - ◊ Hour
  - ◊ Minute
- In more complex scenarios, geography is considered, and a Daytime field is calculated to determine if there was sunlight for the period

## 8.1.8 Demonstration: Exploring Date Tables

---

- Lets take a look at the Date tables in AdventureWorksDW and WideWorldImportsDW

## 8.1.9 Creating Date Tables

---

- If your data source is an OLAP database, you may already have a date table that you can import
- If you are using an OLTP database or text files, you will likely need to generate your own
- You have a lot of different options when it comes to creating a Date Table
  - ◊ You could add a Table to a SQL Database, and import it
  - ◊ You could use a Power Query M Expression to generate a Date Table
  - ◊ You could use a DAX Expression to generate a Date Table
  - ◊ You could build data in Excel, and then import it

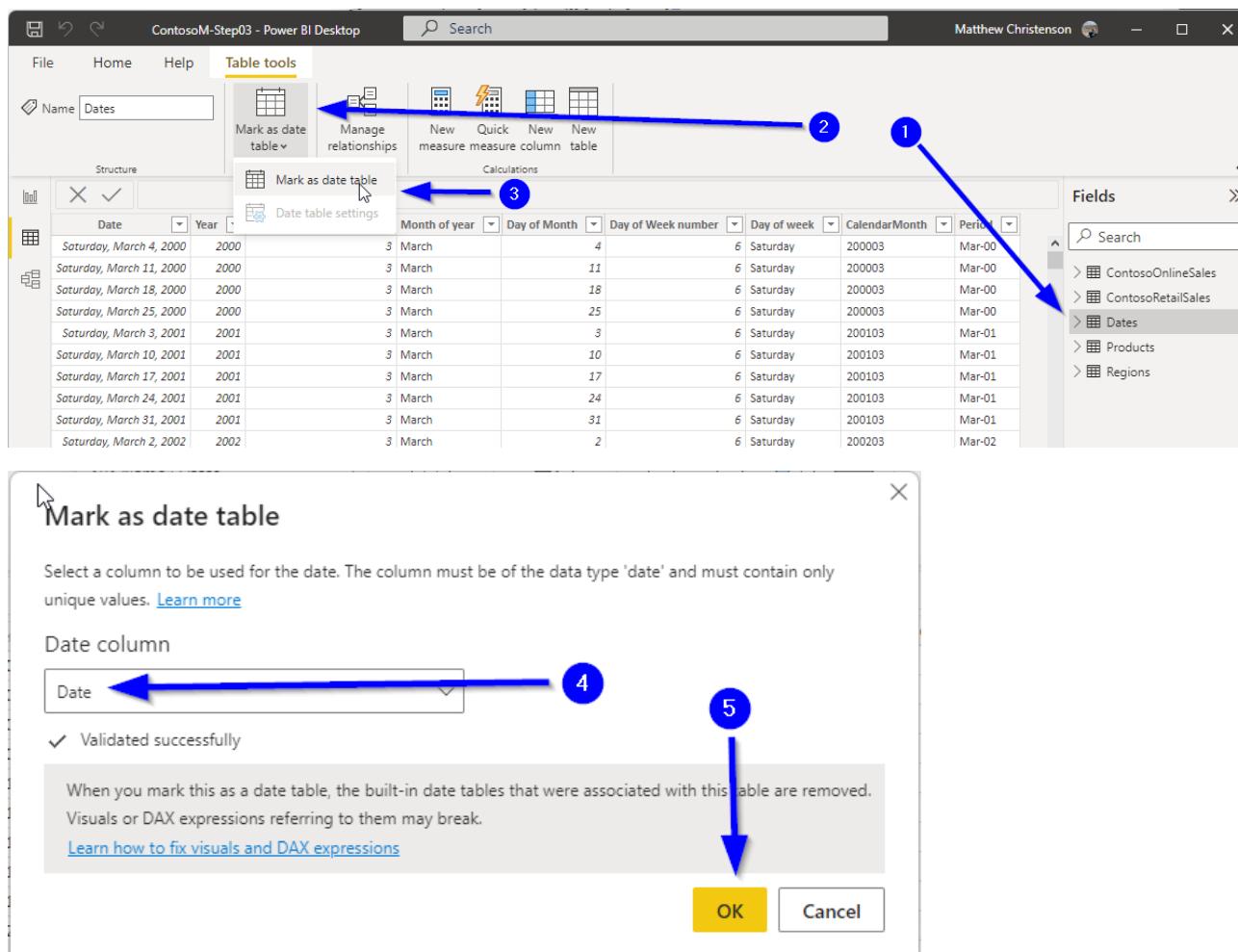
## 8.1.10 Tips for creating Date Tables

---

- Date Tables should have an entry for every date within the range that data might exist, even if no data exists for that date
  - ◊ Some of the date functions will not function properly without this
  - ◊ Report Visualizations would skip dates where an aggregate should show 0
- Date Tables should include one column of type Date with unique values
  - ◊ DateTime type is a poor choice because the time part of the field is not guaranteed to be empty
  - ◊ It is not necessary that the relationship between this table and your fact tables use the Date column
- Avoid unnecessarily creating large date ranges such as 1900-2100, this will create performance implications with some calculations
- The Date Table should be marked as Date in Power BI

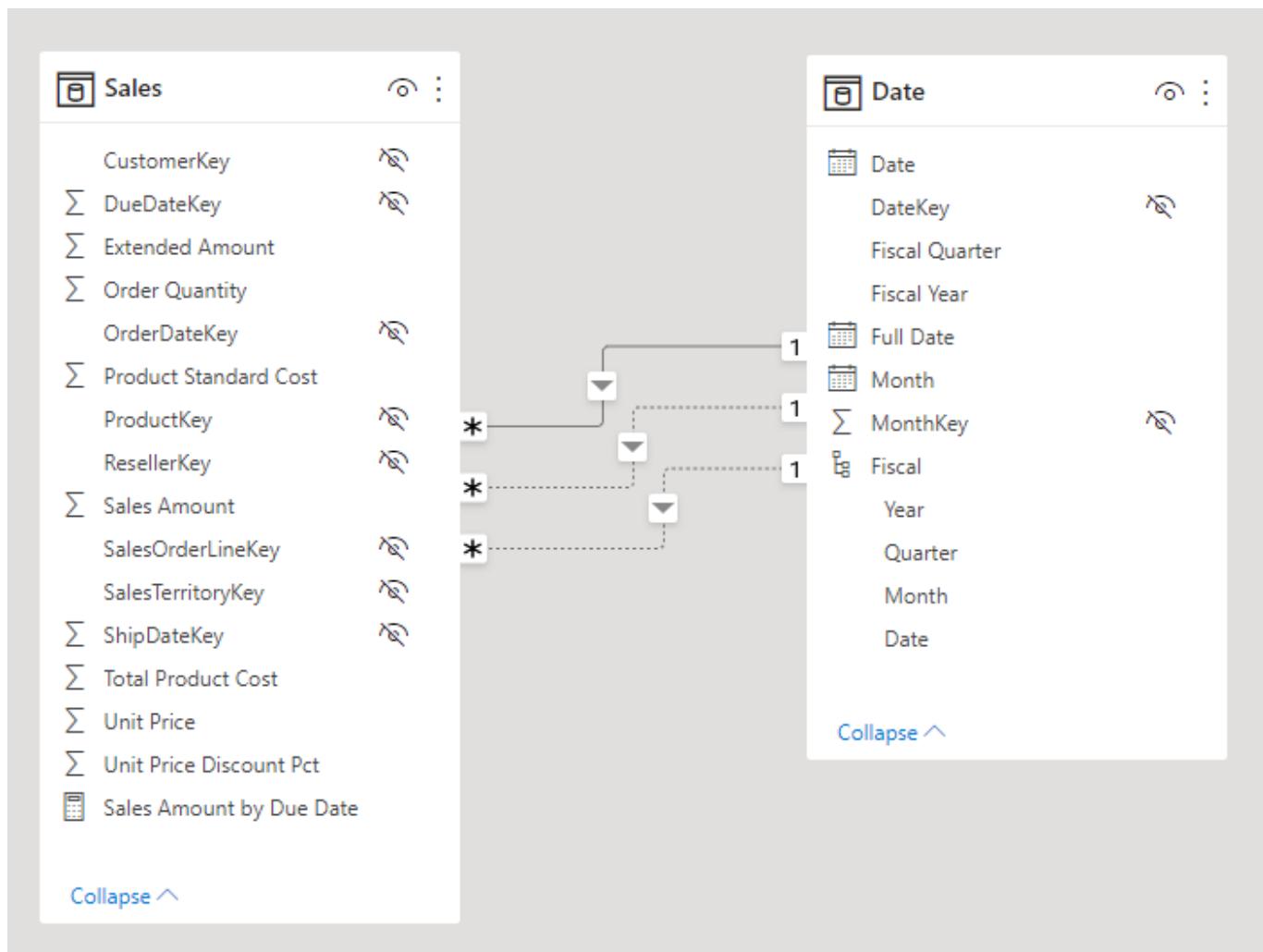
## 8.1.11 Making a Date Table

- It is important to mark any Date Tables as a Date Table in your data model
- This allows time intelligence functions to work properly
  - ◊ Implicitly applies ALL() to your date table
  - ◊ Only needed for non-date keys
    - \* If your Key is a date, this will be inferred



## 8.1.12 Multiple Relationships and Date Tables

- Frequently in OLAP Databases a single date table will exist and there will be multiple relationships between it and a fact table
- Although Power BI will let you model these relationships, only one of them will be considered Active
- This is to avoid ambiguity



## 8.1.13 Avoiding Ambiguous Relationships

---

- Ambiguity occurs when there are multiple ways to start from one table and reach another table
- DAX expressions are implicitly aware of active relationships
  - ◊ When using RELATED() or RELATEDTABLE(), there is no need to explicitly identify how the two tables relate
  - ◊ When bringing Measures and Attributes into visualizations, there is no need to explicitly explain how the two are related
- You can deal with multiple relationships in one of two ways
  - ◊ You can create DAX formulas that explicitly use relationships with the USERELATIONSHIP() function
  - ◊ You can use Role Playing tables to avoid multiple relationships

## 8.1.14 Role Playing Tables

---

- Role Playing tables are new tables for each date you'll need to represent
- To make your reports easier to read, you can change the names of the columns in each Date Table to represent the type of date that is being represented
- For Example
  - ◊ Order Date
    - \* Order Date
    - \* Order Year
    - \* Order Month
  - ◊ Ship Date
    - \* Ship Date
    - \* Ship Year
    - \* Ship Month
- You can use DAX or PowerQuery to duplicate a base reference table

## Section 8.2

# DAX Time Intelligence Functions

## 8.2.1 Time Intelligence: The Hard Way

---

- You don't need any special time intelligence functions to perform time-based analysis
- You already have the knowledge to write powerful expressions
  - ◊ ... but they can be complicated!

### Example

The following Example shows how we can calculate the Year-To-Date without using any special functions:

```
Sales YTD =
CALCULATE (
    SUM ( Sales[Sales Amount] ),
    FILTER(
        ALL ( Order[OrderDate] ),
        Order[OrderDate] <= MAX ( Order[OrderDate] )
        &&
        YEAR ( Order[OrderDate] ) = YEAR ( MAX ( Order[OrderDate] ) )
    )
)
```

Pseudocode:

Calculate the total Sales amount from the following set:

```
(  
    Starting with every order ever (ignoring any filter context),  
    consider only the rows that meet the following conditions:  
        (  
            The specific row's order date is less than the greatest  
            date found in the current filter context  
            Further filter that down to only the rows with an order  
            date that has the same year as the greatest date found in the current  
            filter context  
        )  
)
```

## 8.2.2 Time Intelligence: The Easy Way

---

- Time Intelligence Functions make life easier
- There are requirements
  - ◊ You need to have a date table in your Data Model
  - ◊ The Date Table must include a column considered to be the date column by marking the table as a Date Table in the modeling tools
    - \* Every date must exist once and only once
    - \* You cannot skip dates, such as skipping days with no data, or weekends
  - ◊ DAX time intelligence functions work only on a standard calendar with a start date of January 1st
    - \* For Fiscal Years there are often workarounds
- There are three categories of Time Intelligence Functions
  - ◊ Functions that return a single date
  - ◊ Functions that return a table of dates
  - ◊ Functions that evaluate expressions over a time period

## 8.2.3 Date / Time Function Reference

---

- Microsoft's reference is the best place to see what options are there to perform date based calculations
- Functions tend to fall into three categories
  - \* Functions that return a single date
  - \* Functions that return a table of dates
  - \* Functions that perform an evaluation over a calculated date range
- Basic Date / Time Function Reference
  - ◊ <https://docs.microsoft.com/en-us/dax/date-and-time-functions-dax>
- Time Intelligence Function Reference
  - ◊ <https://docs.microsoft.com/en-us/dax/time-intelligence-functions-dax>

## 8.2.4 Time Intelligence Examples

---

- Example

In this example we are calculating a Year-To-Date Sales Amount using the DATESYTD() function. Note this returns the same results as the Hard Way example that we provided earlier in this module.

```
Sales YTD =  
    CALCULATE(  
        [Sales Amount] , DATESYTD(Orders[OrderDate])  
)
```

A lot less work!

- Example

The following example does the same thing!

```
Sales YTD = TOTALYTD([Sales Amount], Orders[Order Date])
```

## Section 8.3

### Exercises

# Exercise 8.1 Northwind Modeling – Computing a Date Table

## Introduction

In this exercise, we will create a Date Table for our model using DAX.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. In the Main Menu navigate to File -> Options and Settings -> Options -> Current File -> Data Load.
3. Unselect the Auto date/time option under Time intelligence.
4. Click Okay to close the Options dialog.
5. With the Report view Active, select the Modeling tab within the Ribbon.
6. Select New Table in the Ribbon.
7. Enter the following query into the expression bar. Save yourself some time by cut-and-pasting it from

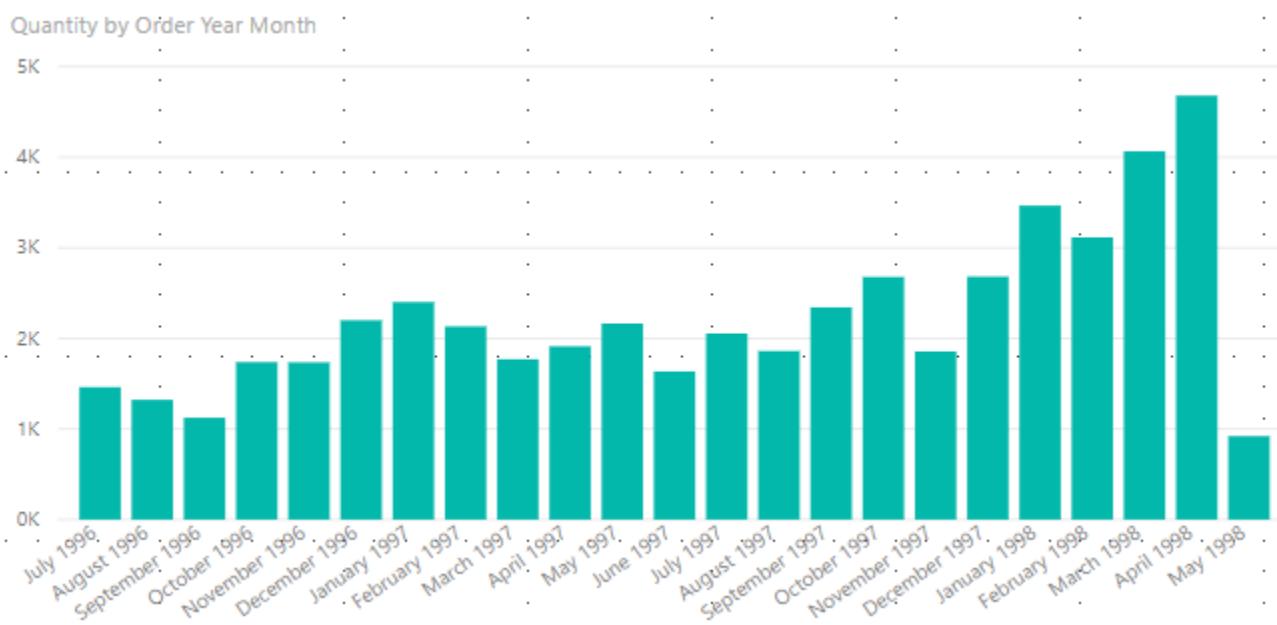
{LabFiles}\StarterFiles\DateTableExpression.txt.

```
1 Order Date =
2 VAR MinYear = Year ( MIN ( Orders[Order Date] ) )
3 VAR MaxYear = Year ( MAX ( Orders[Order Date] ) )
4 RETURN
5 ADDCOLUMNS (
6   FILTER (
7     CALENDARAUTO ( ),
8     YEAR ( [Date] ) >= MinYear &&
9     YEAR ( [Date] ) <= MaxYear
10   ),
11   "Order Year", YEAR ( [Date] ),
12   "Order Quarter Number", INT ( FORMAT ( [Date] , "q" ) ),
13   "Order Quarter", "Q" & INT ( FORMAT ( [Date] , "q" ) ),
14   "Order Month Number", MONTH ( [Date] ),
15   "Order Month", FORMAT ( [Date], "mmmm" ),
16   "Order Week Day Number", WEEKDAY ( [Date] ),
17   "Order Week Day", FORMAT ( [Date], "dddd" ),
18   "Order Year Month Number", YEAR ( [Date] ) * 100 + MONTH ( [Date] ),
19   "Order Year Month", FORMAT ( [Date], "mmmm" ) & " " & YEAR ( [Date] ),
20   "Order Year Quarter Number", Year ( [Date] ) * 100 + INT ( FORMAT ( [Date] , "q" ) ) ,
21   "Order Year Quarter", "Q" & FORMAT ( [Date], "q" ) & "-" & YEAR ([Date] )
22 )
```

8. Click the check mark to save the query.
9. Navigate to the Data view and confirm you have data in the table.
10. Rename the Date column to Order Date.
11. Change the Order Date column type from Date/Time to Date to remove the time component of the value.
12. In the Orders table change the Order Date, Order Required Date, and Order Shipped Date columns to the Date datatype to remove the time component of the value.
13. In the Model view, in the Modeling tab in the Ribbon, click Manage Relationships.
14. Click New.
15. In the New Relationship dialog, create a relationship between the Orders and Order Date tables. Be sure to select the Order Date column from the Orders table and the Order Date column from the Order Date table.

Cardinality:	Many to one (*:1)
Cross filter direction:	Single
Make this relationship active:	<input checked="" type="checkbox"/>
16. With the Order Date table selected in the Data view, select the Mark as Date Table icon in the Modeling tab of the Ribbon. If Prompted, use the Order Date column as the date.
17. Configure the Order Year Month column to sort by the Order Year Month Number column.
18. Configure the Order Month column to sort by the Order Month Number column.
19. Apply a similar sort pattern to the Order Year Quarter, Order Week Day, and Order Quarter columns.

20. Create a new Report tab named Dates 1 and create a report that looks like this:

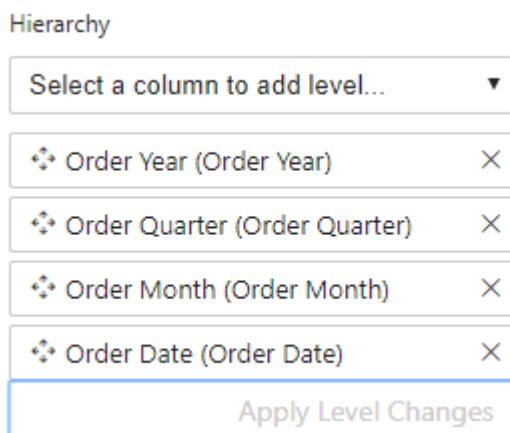


21. Save your work using a new Step Number.

## Exercise 8.2 Northwind Modeling – Useful Date Hierarchy

In this exercise we will add a Hierarchy to the Date table to be used in reports.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. Using the Model view, Create a Hierarchy off the Order Date column of the Order Date table
3. Name the Hierarchy Order Date Hierarchy and add the Order Year, Order Quarter, and Order Month columns.
4. Move the Order Date column below the others in the hierarchy and then select Apply Level Changes. When you're done the Hierarchy should look like this:



5. Create a new Report tab named Dates 2 and add a Matrix.
6. With the Matrix selected, check [Sales].[Sales Amount] and [Order Date].[Order Date Hierarchy].

7. With the Matrix Selected, click the "upside down pitchfork" twice to expand down two levels within the hierarchy, your report should look similar to this:

Order Year	Sales Amount
1996	\$208,083.97
Q3	\$79,728.57
July	\$27,861.895
August	\$25,485.275
September	\$26,381.4
Q4	\$128,355.4
October	\$37,515.725
November	\$45,600.045
December	\$45,239.63
1997	\$617,085.2035
Q1	\$138,288.925
January	\$61,258.07
February	\$38,483.635
March	\$38,547.22
Q2	\$143,177.045
April	\$53,032.9525
May	\$53,781.29
June	\$36,362.8025
Q3	\$153,937.77
July	\$51,020.8575
August	\$47,287.67
September	\$55,629.2425
Total	\$1,265,793.0395

Expand all down one level



8. Save your work using a new Step Number.

## Exercise 8.3 Northwind Modeling – Running Totals

In this exercise, we'll create a measure that will compute running totals.

1. Using Power BI Desktop, Open up the Northwind Modeling project that we have been building throughout the course.
2. Add the following measures to the Sales table:

```
Sales Amount YTD = CALCULATE( sumx(sales, Sales[Sales Amount]), DATESYTD('Order Date'[Order Date]))
```

```
Sales Amount QTD = CALCULATE( sumx(sales, Sales[Sales Amount]), DATESQTD('Order Date'[Order Date]))
```

3. Set the data type for the new measures to currency.
4. Experiment with adding your new measures to the Dates 2 report.

Order Year	Sales Amount	Sales Amount YTD	Sales Amount QTD
1998	\$440,623.866	\$440,623.866	
Q1	\$298,491.553	\$298,491.553	\$298,491.553
January	\$94,222.1105	\$94,222.1105	\$94,222.1105
February	\$99,415.2875	\$193,637.398	\$193,637.398
March	\$104,854.155	\$298,491.553	\$298,491.553
Q2	\$142,132.313	\$440,623.866	\$142,132.313
April	\$123,798.6825	\$422,290.2355	\$123,798.6825
May	\$18,333.6305	\$440,623.866	\$142,132.313
June		\$440,623.866	\$142,132.313
Q3		\$440,623.866	
July		\$440,623.866	
August		\$440,623.866	
September		\$440,623.866	
Q4		\$440,623.866	
October		\$440,623.866	
November		\$440,623.866	
December		\$440,623.866	
Total	\$440,623.866	\$440,623.866	

5. Save your work using a new Step Number.

## Exercise 8.4 Northwind Modeling – Parallel Period Comparison

---

1. Using Power BI Desktop, open up the Northwind Modeling project that we have been building throughout the course.
2. Add the following measures to the sales table:

```
Previous Sales Amount YTD = Calculate ( sum(Sales[Sales Amount]), DATESYTD(SAMEPERIODLASTYEAR('Order Date'[Order Date])))
```

```
Change in YTD from previous = Sales[Sales Amount YTD] / Sales[Previous Sales Amount YTD]
```

3. Set Previous Sales Amount YTD to use the datatype.
4. Set Change in YTD from previous to be formatted as a percentage.
5. Add these fields to your matrix on report Dates 2.
6. Spend a few moments exploring your data.
7. Save the project using a new Step Number.