Power BI Desktop Primer

Agenda

➢ Power BI Desktop Overview

- Building Queries

- Designing Data Models

- Designing Reports

- Query Parameters

- Power BI Desktop Template Files

Power BI Desktop Projects

- Design queries to create a dataset
  - Most datasets created using import model
  - Datasets can be created with DirectQuery and Live Connect

- Enrich the dataset using data modeling features
  - Add calculated columns, measures, hierarchies, etc

- Visualize insights from your dataset by creating reports
  - Build reports using visuals

- Publish projects to the Power BI Service
 - Your datasets & reports can be used in Power BI embedding

Agenda

- ✓ Power BI Desktop Overview
- ➢ Building Queries
- • Designing Data Models
- • Designing Reports
- • Query Parameters
- • Power BI Desktop Template Files

Power BI Desktop is an ETL Tool

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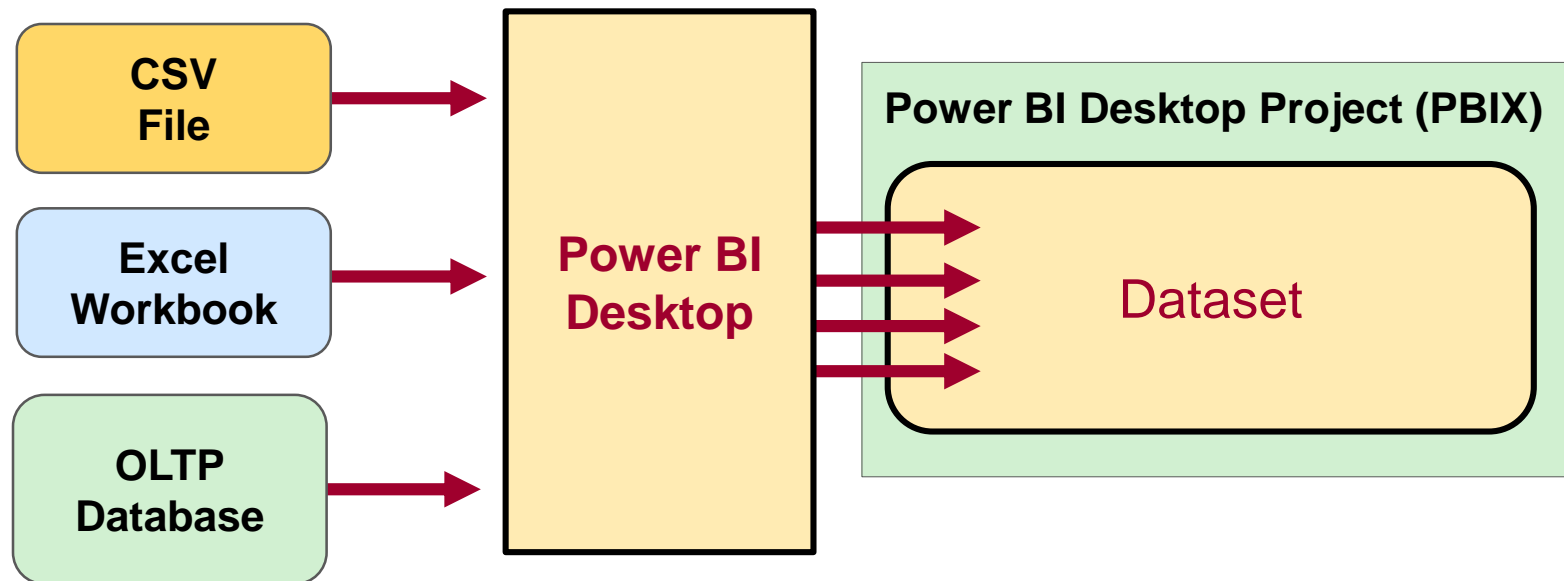
Understanding Query Input and Output

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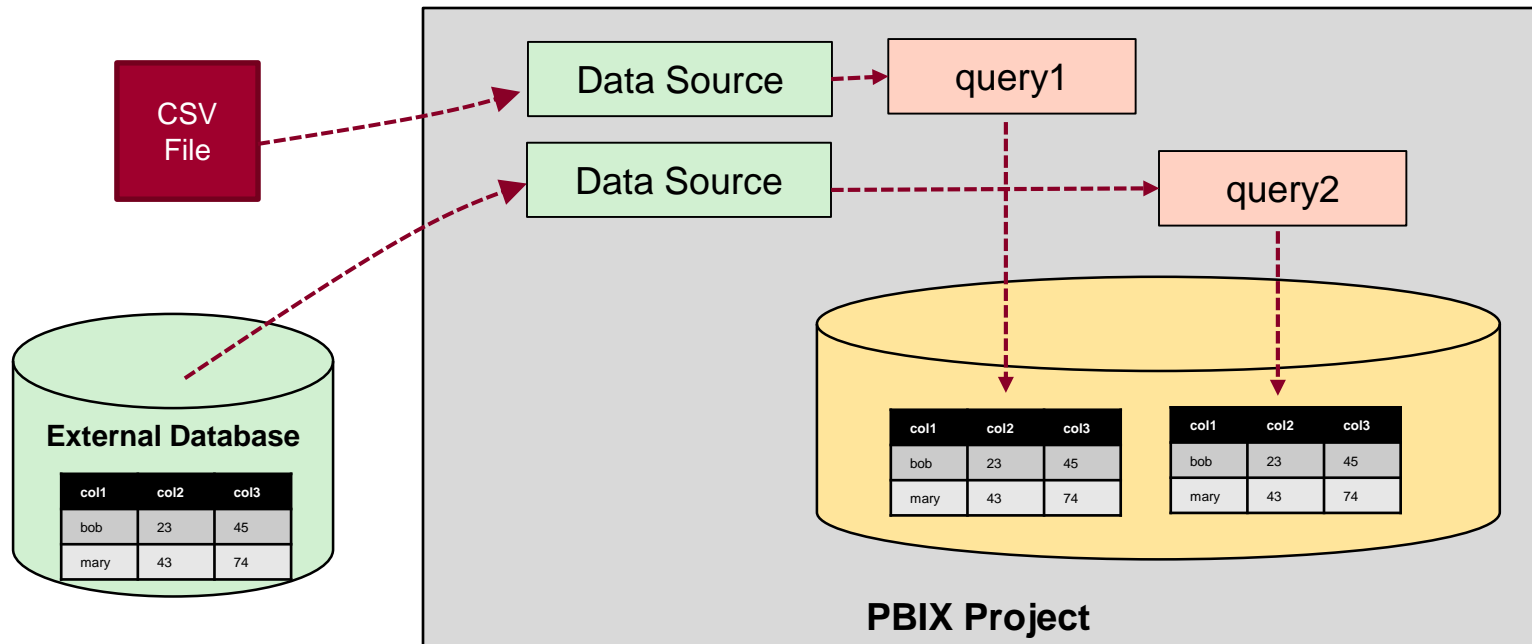
Query Composition

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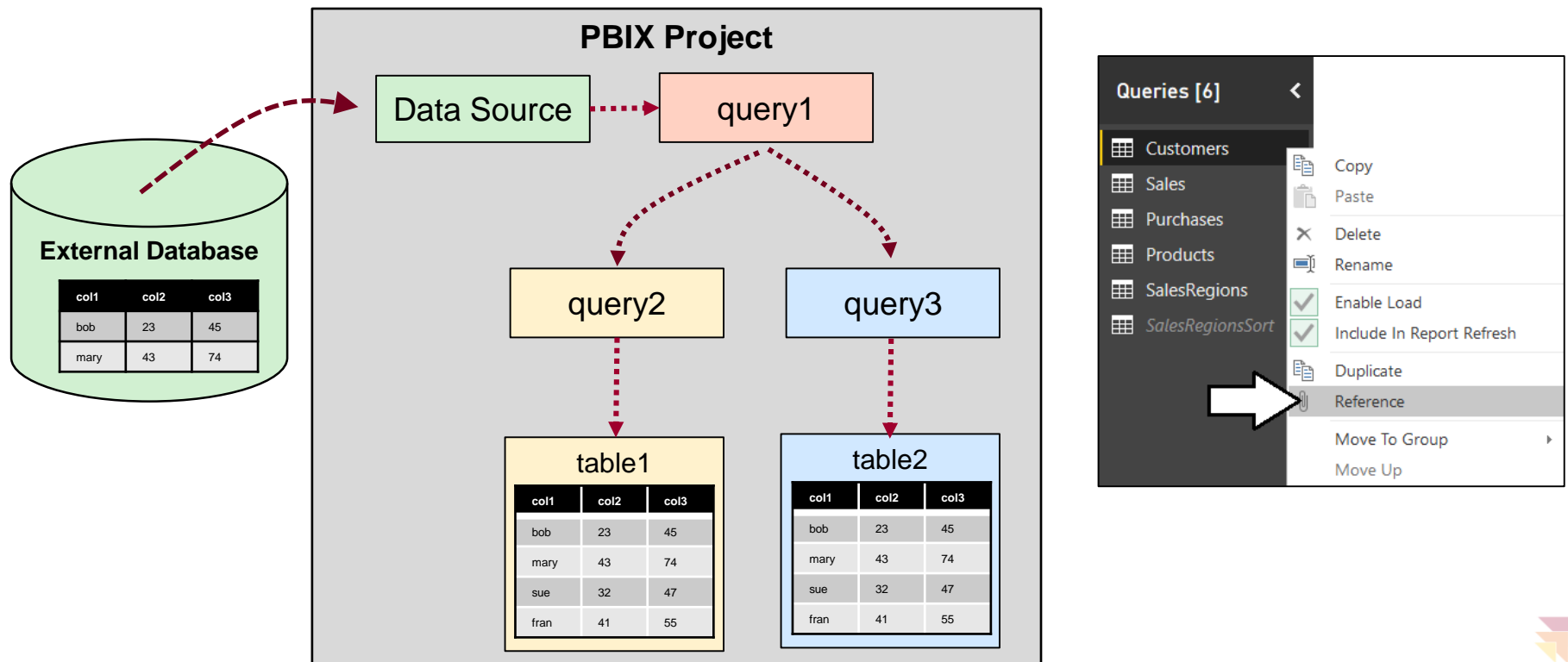
Combining Queries

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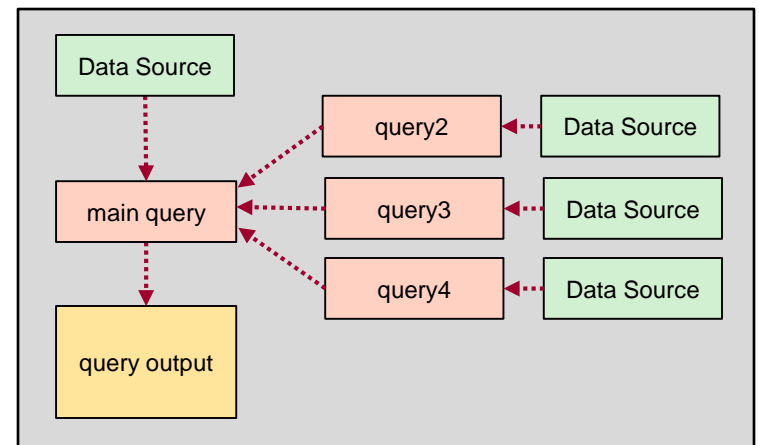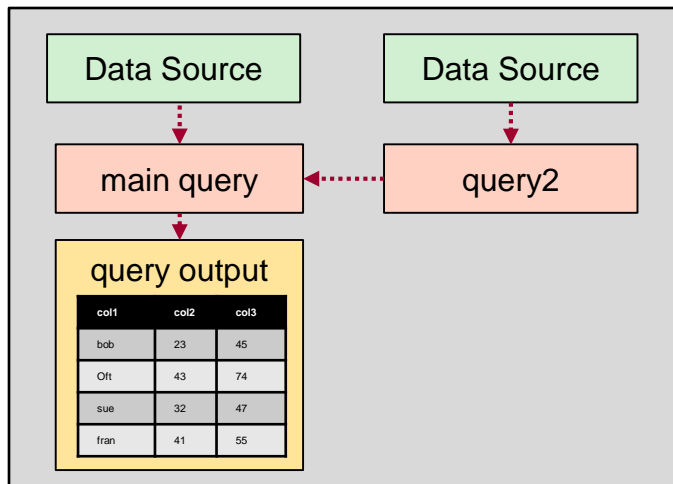
Sample OLTP Database: WingtipSalesDB

- Online Transaction Processing (OLTP) System
  - Used for real-time data access and transaction-based data entry
  - Optimized for faster transactions (e.g. inserts, updates & deletes)
  - Tables normalized to reduce/eliminate redundancies
  - Table schemas can be hard for business users to understand

Data Modeling using a Star Schema

- OLAP Modeling often based on Star Schema
  - Tables defined as fact tables or dimension tables
  - Fact tables related to dimension table using 1-to-many relationships

Designing Queries to Build a Star Schema

- Converts OLTP Data Model to OLAP Data Model
  - Sales table is modeled as a OLAP Fact Table
  - Other tables are modeled as OLAP Dimension tables
  - Requires pulling CustomerId column into Sales table
  - All dimension tables should be directly related to fact table

Query Editor Window

- Power BI Desktop provides separate Query Editor window
  - Provides powerful features for designing queries
  - Displays list of all queries in project on the left
  - Displays **Properties** and **Applied Steps** for selected query on right
  - Preview of table generated by query output shown in the middle
  - Query can be executed using **Apply** or **Close & Apply** command
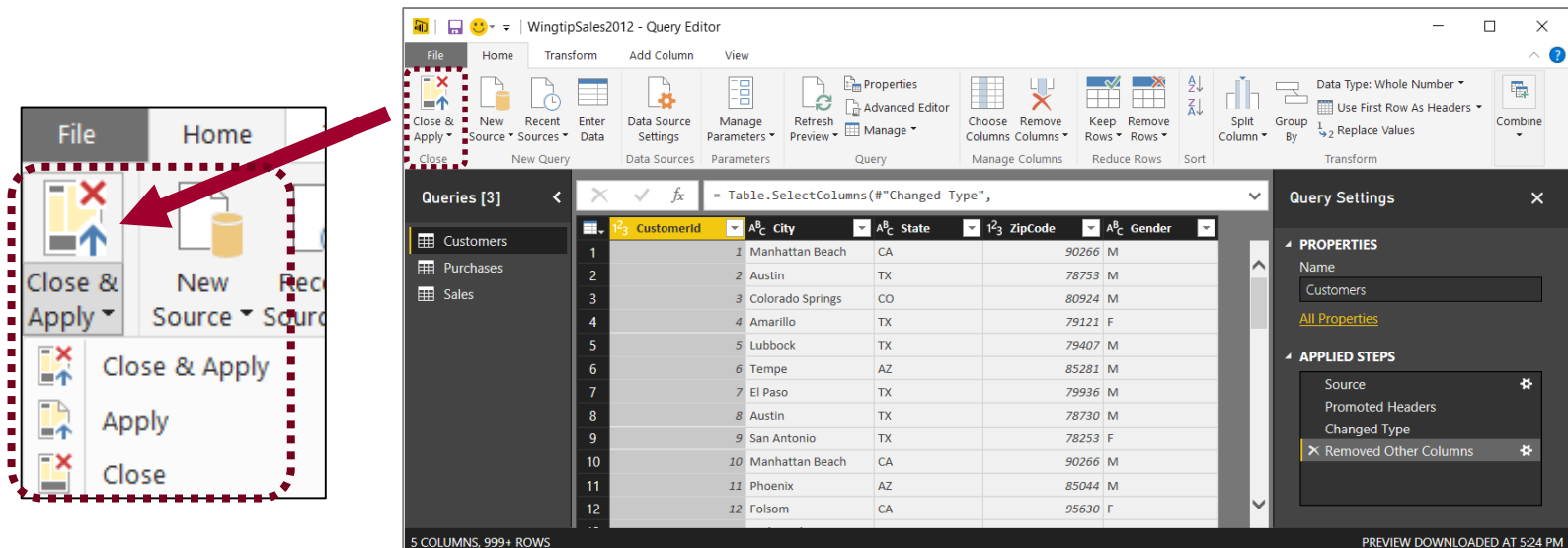
Query Steps

- A query is created as a sequence of steps
  - Each step is a parameterized operation on the data
  - Each step has formula which can be viewed/edited in formula bar
  - Query starts with Source step to extract data from a data source
  - Additional steps added to perform transform operations on data
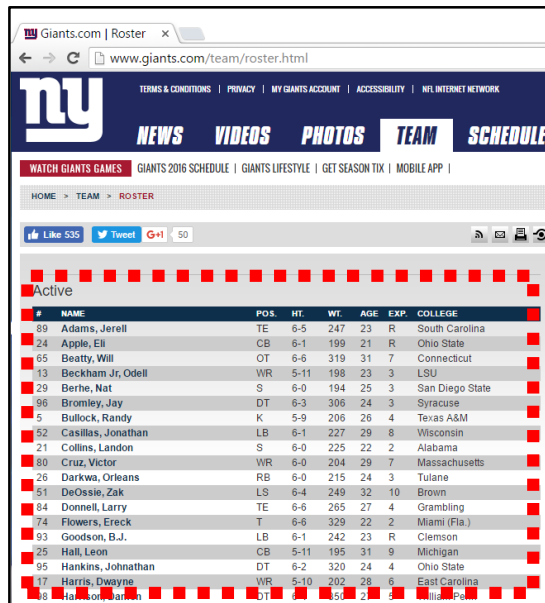 - You can replay query operations one by one by clicking on steps

Advanced Editor

- ## Power BI Desktop based on "M" functional language

  - Query in Power BI Desktop saved as set of M statements in code

  - Query Editor generates code in M behind the scenes

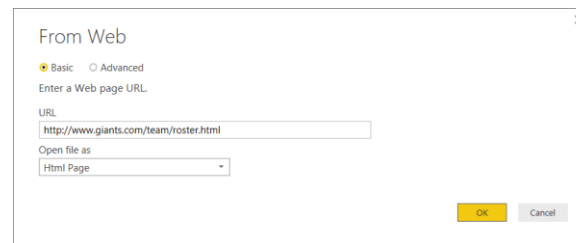 - Advanced users can view & modify query code in Advanced Editor
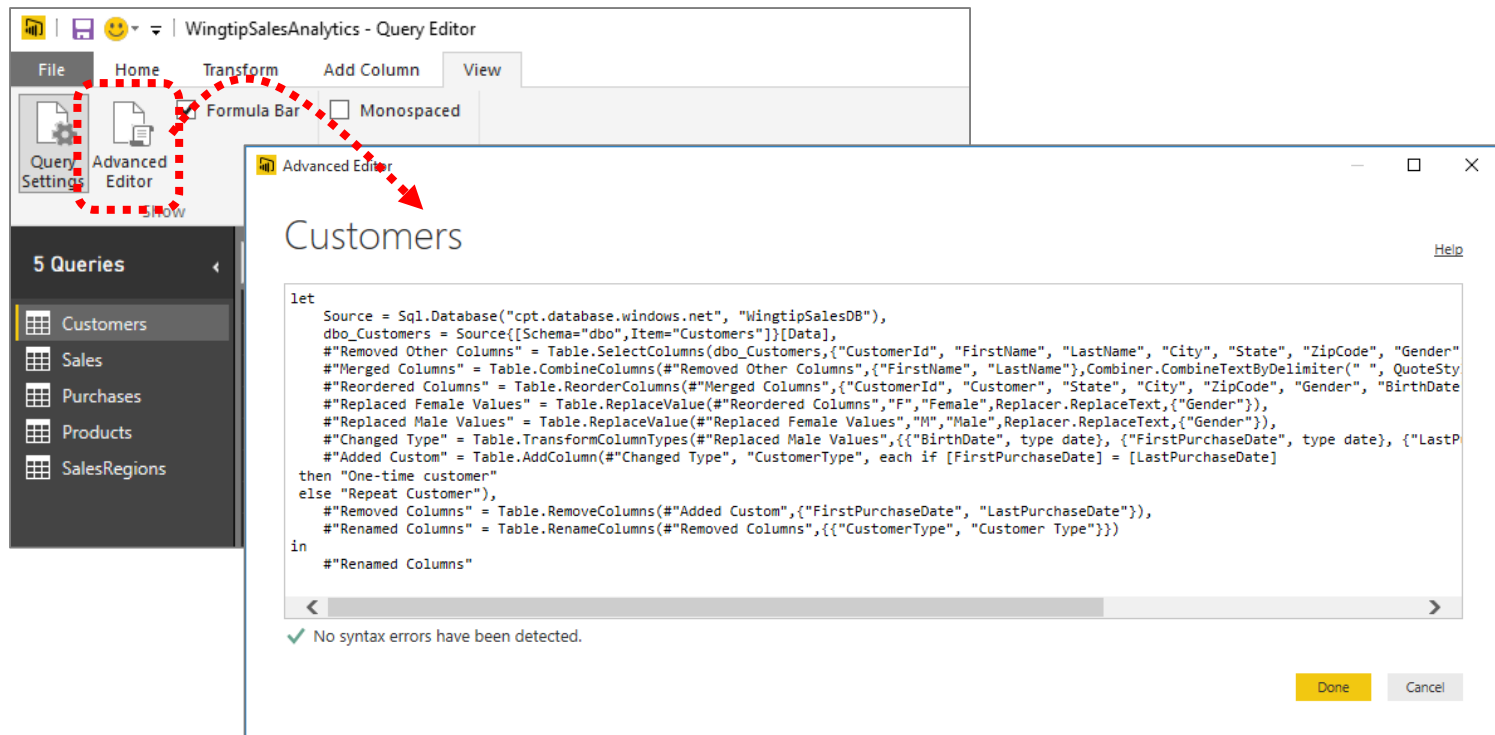
Working with Web Data Sources

- Many public websites publish data using HTML tables
  - Power BI desktop can scrape data from tables in HTML pages

Examples of Basic Power BI Desktop Steps

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

Cleaning Data

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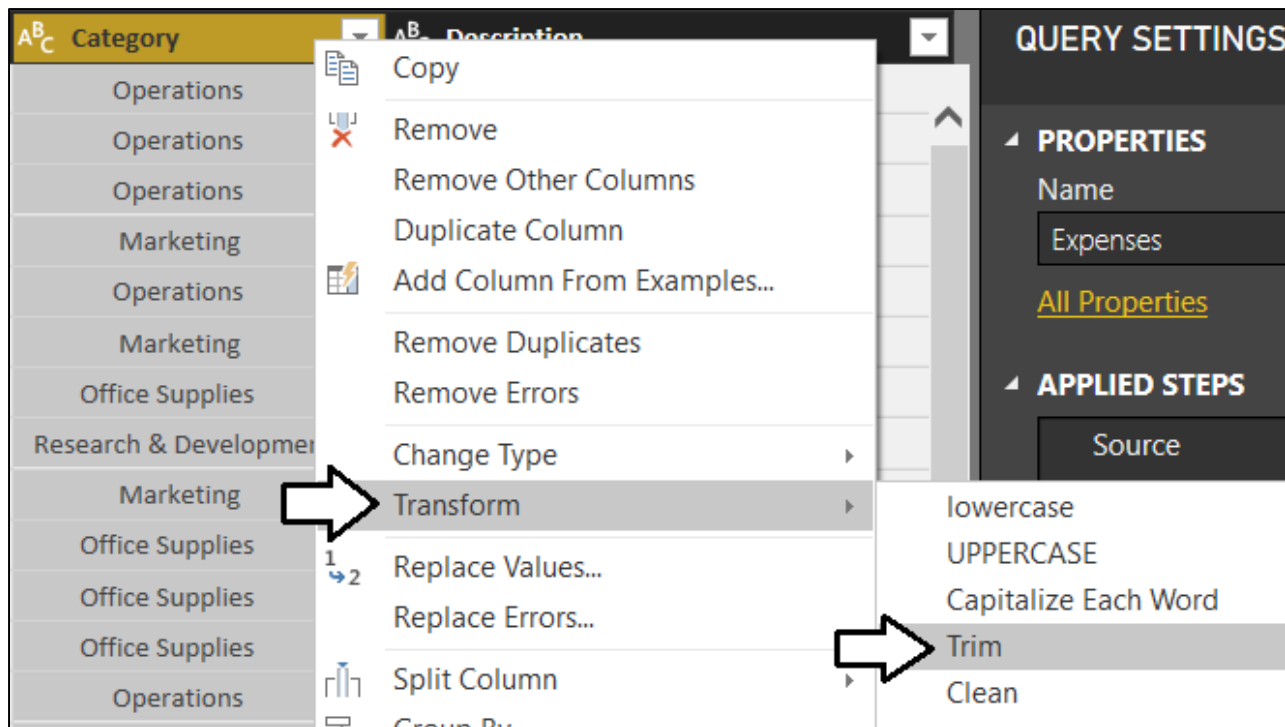
Converting Column Types

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

PurchaseDate	1²3 Quantity	$ SalesAmount	$ ProductCost	
1/28/2012	1	2.95	1.2 Decimal Number	
1/28/2012	6		$ Fixed Decimal Number	
1/28/2012	1	19.95	1²3 Whole Number	
1/28/2012	5	249.75	Date/Time	
1/28/2012	1	2.95	Date	

Expanding Related Columns

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

Adding a Custom Column

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

Adding a Conditional Column

- Abstracts away need to write M code

Understanding Function Queries

- Query can be converted into reusable function
  - Requires editing query M code in Advanced Editor
  - Function query can be defined to accept parameters

```
GetExpensesFromFile

(FilePath as text) =>

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

  - Function query can't be edited with visual designer

Agenda

- ✓ Power BI Desktop Overview
- ✓ Building Queries
- ➤ Designing Data Models
- • Designing Reports
- • Query Parameters
- • Power BI Desktop Template Files

Table Relationships

- Tables in data model associated with relationships
  - Relationships based on single columns
  - Tabular model supports [1-to-1] and [1-to-many] relationships
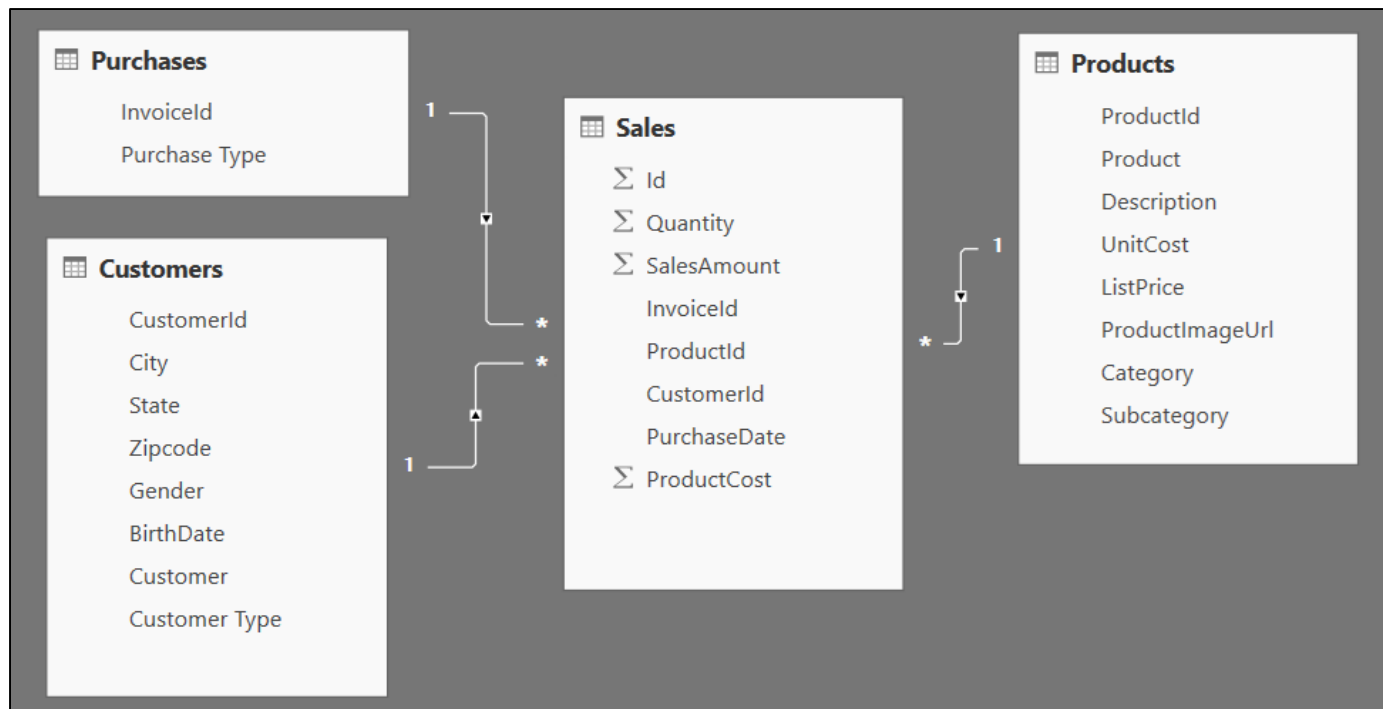
Relationship Properties

- Cardinality

- Cross filter direction

How Do You Create a Relationship Here?

- Two tables don't have fields to create relationship
  - The solution is to create two new calculated columns
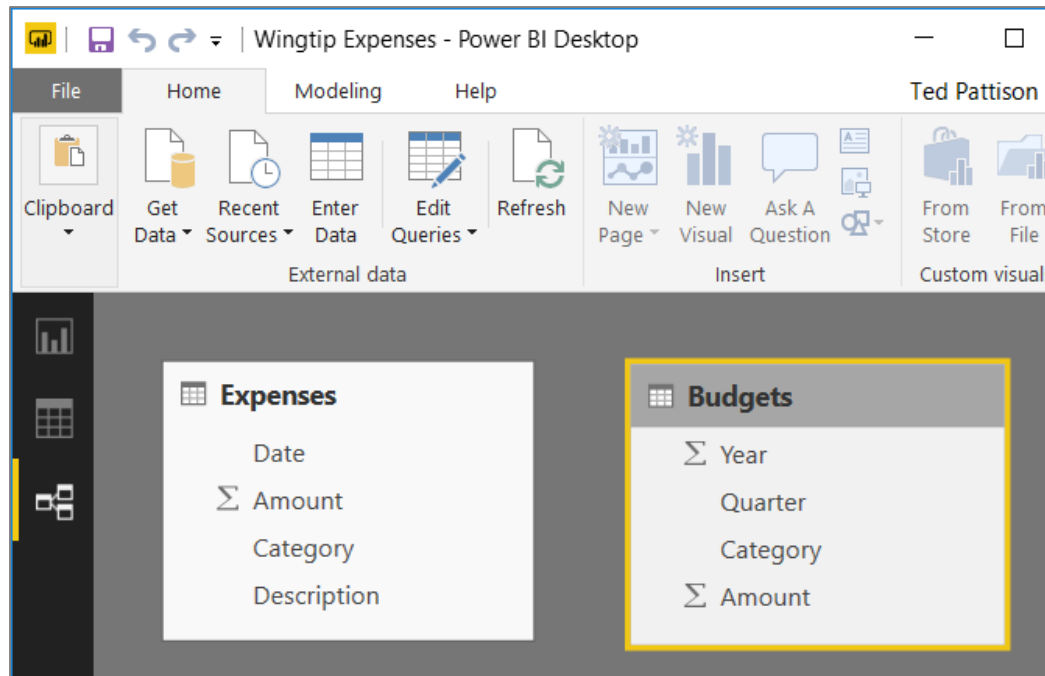
Creating Composite Key Fields

- ## Create composite key column in Budgets

- ## Create composite key column in Expenses

Create Relationship Using Composite Keys
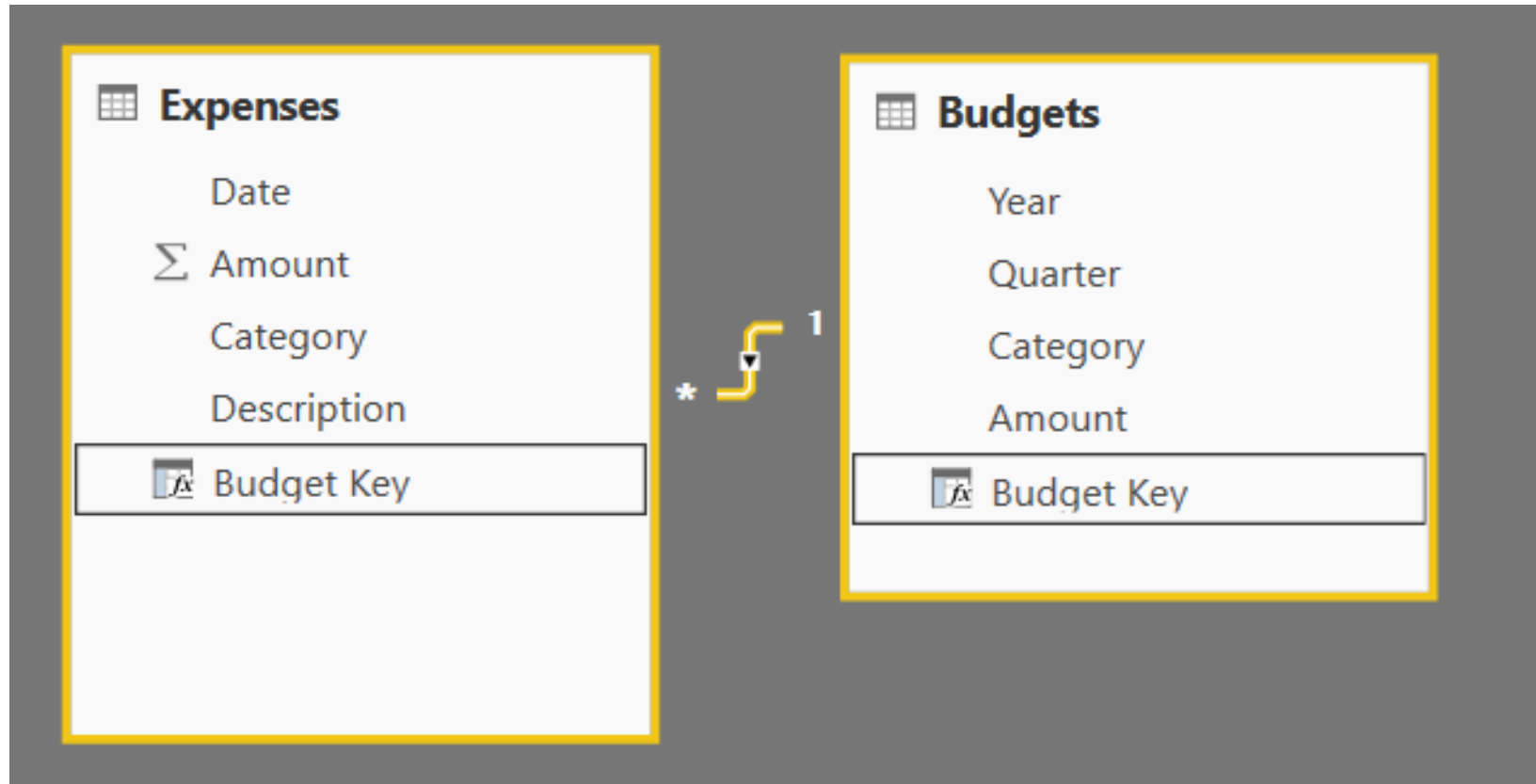
Working with DAX

- DAX is the language used to create data models
  - DAX stands for "**D**ata **A**nalysis E**x**pression Language"

- DAX expressions are similar to Excel formulas
  - They always start with an equal sign (=)
  - DAX provides many built-in functions similar to Excel

- DAX Expressions are unlike Excel formulas…
  - DAX expressions cannot reference cells (e.g. A1 or C4)
  - Instead DAX expressions reference columns and tables

```
=SUM('Sales'[SalesAmount])
```

Writing DAX Expressions

- Some DAX expressions are simple

```
Sales Revenue = Sum(Sales[SalesAmount])
```

- Some DAX expressions are far more complex

```
Sales Growth PM = IF(
   ( ISFILTERED(Calendar[Month]) && ISFILTERED(Calendar[Date]) = FALSE() ),
   DIVIDE(
     SUM(Sales[SalesAmount]) -
     CALCULATE(
       SUM(Sales[SalesAmount]),
       PREVIOUSMONTH(Calendar[Date])
     ),
     CALCULATE(
       SUM(Sales[SalesAmount]),
       PREVIOUSMONTH(Calendar[Date])
     )
   ),
   BLANK()
)
```

Creating Variables in DAX Expressions

- Variables can be added at start of expression
  - Use **VAR** keyword once for each variable
  - Use **RETURN** keyword to return expression value

```
Budget Key =
  VAR BudgetYear = YEAR([Date])
  VAR BudgetMonth = "Q" & FORMAT([Date],"q")
RETURN
  BudgetYear & "-" & BudgetMonth & "-" & [Category]
```

Calculated Columns vs Measures

- Calculated Columns (aka Columns)
  - Evaluated based on context of a single row
  - Evaluated when data is loaded into memory

  ```
  Column1 = <DAX expression>
  ```

- Measures
  - Evaluated at query time based on current filter context
  - Commonly used for aggregations (e.g. SUM, AVG, etc.)
  - Used more frequently than calculated columns

  ```
  Measure1 = <DAX expression>
  ```

When to Create Calculated Columns

- Measures often better choice than calculate columns
  - Don't create calculated column when you need a measure
  - Prefer to create calculated columns only in specific scenarios

- When should you create calculated columns?
  - To create headers for row labels or column labels
  - To place calculated results in a slicer for filtering
  - Define an expression strictly bound to current row
  - Categories text or numbers (e.g. customer age groups)

Creating Calculated Columns

- Edited in formula bar of Power Pivot data view
  - Start with name and then equals (=) sign
  - Enter a valid DAX expression
  - Clicking on column adds it into expression

Calculated Column for Customer Age Group

1. Calculate customer age from birthdate

2. Calculate age groups using calculated column

Calculated Column used in a Slicer

- Calculated column can populate slicer values
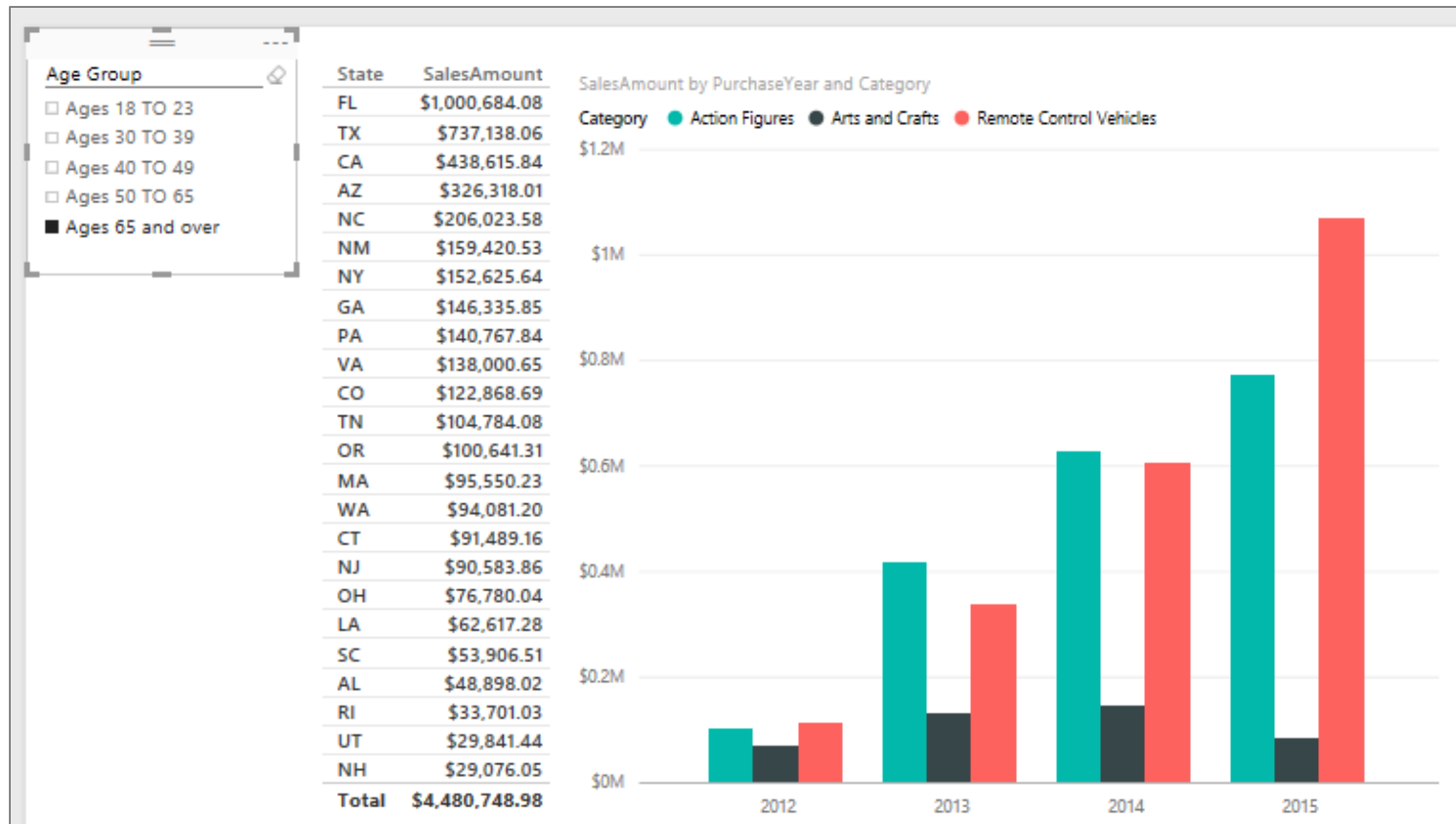
Creating Measures

- Measures have advantage over calculated columns
  - They are evaluated based on the current evaluation context

- Creating a measure with Power BI Desktop
  1. Click New Measure button
  2. Give measure a name and write DAX expressions
  3. Configure formatting
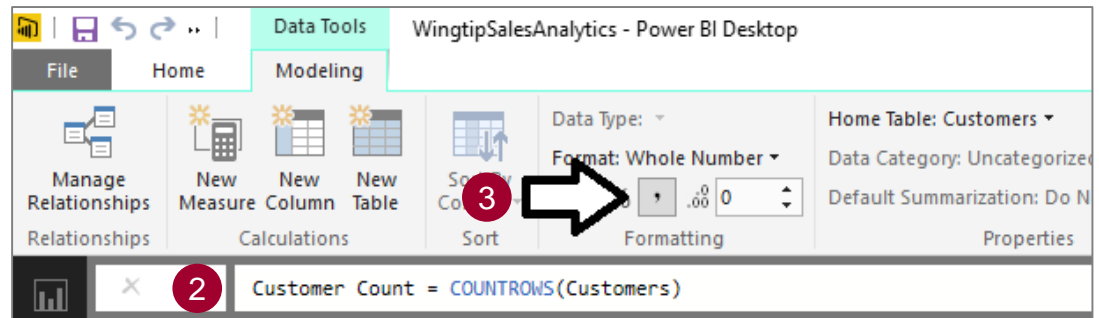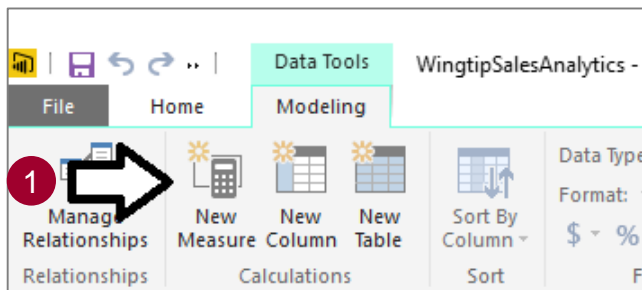
Formatting Measures

- ## Format as whole number
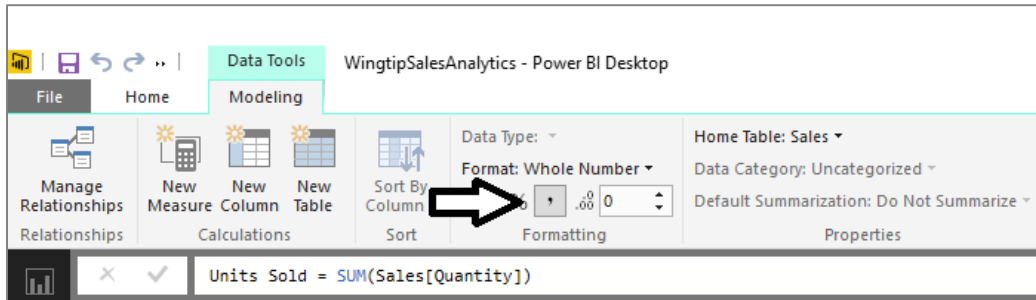
- ## Format as currency
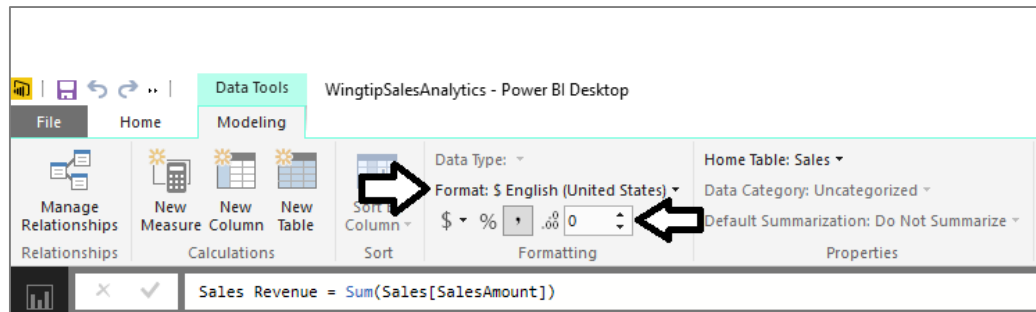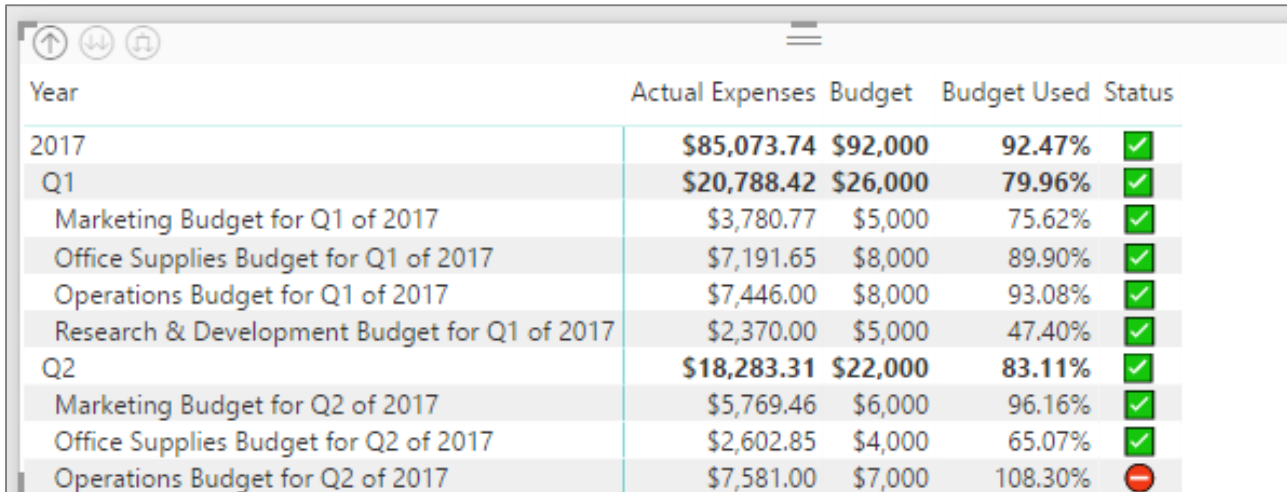
Working with UNICHAR Characters

- Create a measure to return a single UNICHAR character
  - There are many different UNICHAR characters with symbols

```
Status =
IF(
    [Budget Used] > 1,
    UNICHAR(9940),
    UNICHAR(9989)
)
```

  - UNICHAR character symbols can be displayed in table or matrix

Year	Actual Expenses	Budget	Budget Used	Status
2017	$85,073.74	$92,000	92.47%	✅
Q1	$20,788.42	$26,000	79.96%	✅
Marketing Budget for Q1 of 2017	$3,780.77	$5,000	75.62%	✅
Office Supplies Budget for Q1 of 2017	$7,191.65	$8,000	89.90%	✅
Operations Budget for Q1 of 2017	$7,446.00	$8,000	93.08%	✅
Research & Development Budget for Q1 of 2017	$2,370.00	$5,000	47.40%	✅
Q2	$18,283.31	$22,000	83.11%	✅
Marketing Budget for Q2 of 2017	$5,769.46	$6,000	96.16%	✅
Office Supplies Budget for Q2 of 2017	$2,602.85	$4,000	65.07%	✅
Operations Budget for Q2 of 2017	$7,581.00	$7,000	108.30%	⛔

The UNICHAR Browser Demo

Creating Tables Dynamically using DAX

Integrating the Lookup Table into the Data Model

- Lookup table must be integrated into data model
    - Accomplished by creating relationship to one or more tables
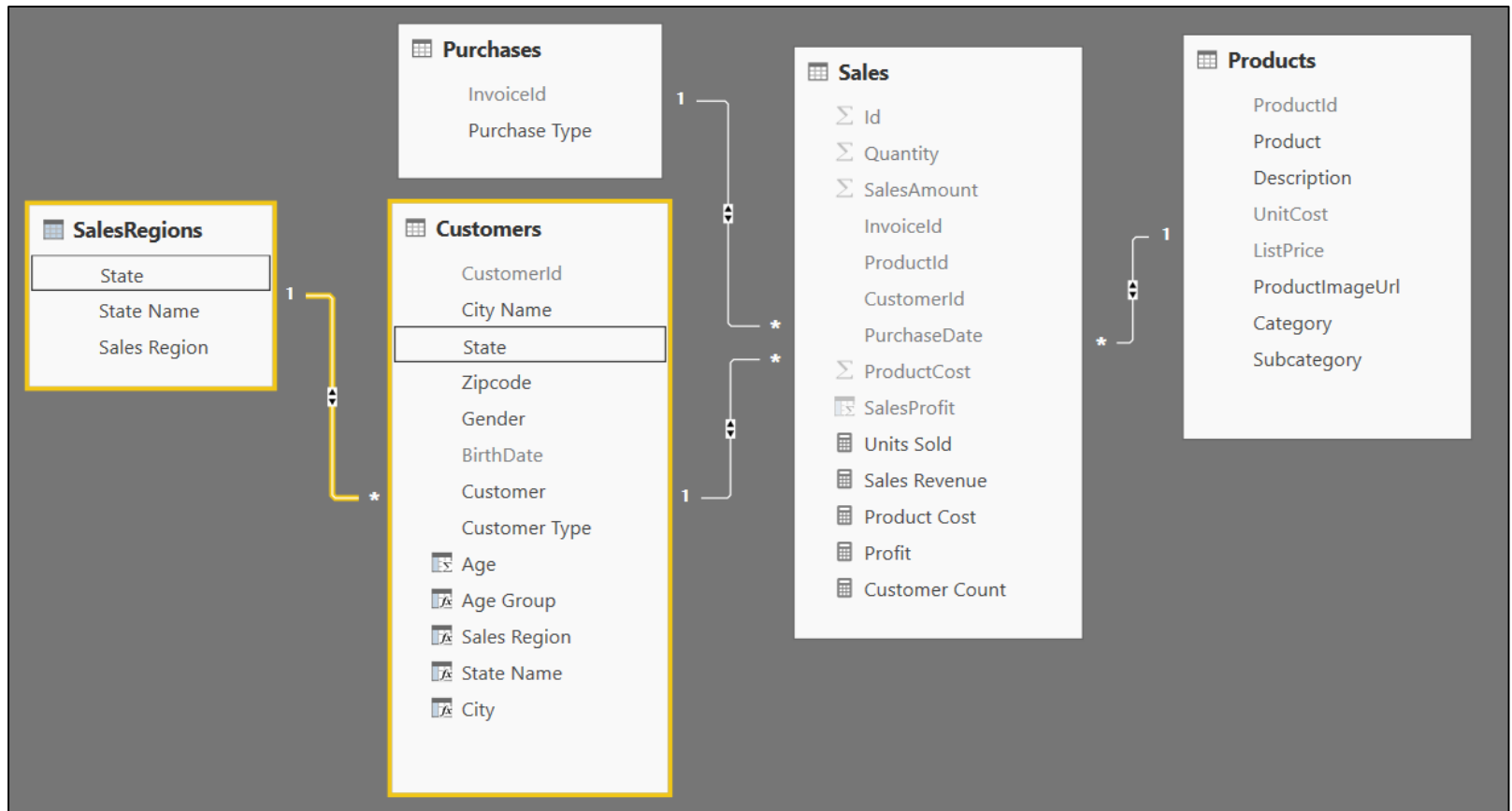
The RELATED Function

- RELATED function performs cross-table lookup
  - Effectively replaces older VLOOKUP function
  - Used in many-side table to look up value from one-side
  - Used to pull data from lookup table into primary table

`Sales Region = RELATED(SalesRegions[SalesRegion])`

CustomerId	City	State	ZipCode	Gender	BirthDate	Customer	CustomerType	Age	Age Group	Sales Region
55	San Jose	CA	95110	Female	3/10/49	Jewell Ryan	Repeat Customer	66	Ages 65 and over	Western Region
73	San Jose	CA	95123	Male	5/9/85	Granville Perry	Repeat Customer	30	Ages 30 TO 39	Western Region
74	San Jose	CA	95122	Female	6/19/79	Sheri Mercado	Repeat Customer	36	Ages 30 TO 39	Western Region
78	San Jose	CA	95110	Male	6/16/78	Raleigh Olson	Repeat Customer	37	Ages 30 TO 39	Western Region
136	San Jose	CA	95124	Female	1/2/45	Carrie Foreman	Repeat Customer	70	Ages 65 and over	Western Region
150	San Jose	CA	95134	Female	8/11/84	Renee McMillan	Repeat Customer	31	Ages 30 TO 39	Western Region

`State Name = RELATED(SalesRegions[StateFullName])`

State	ZipCode	Gender	BirthDate	Customer	CustomerType	Age	Age Group	Sales Region	State Name
CA	95110	Female	3/10/49	Jewell Ryan	Repeat Customer	66	Ages 65 and over	Western Region	California
CA	95123	Male	5/9/85	Granville Perry	Repeat Customer	30	Ages 30 TO 39	Western Region	California
CA	95122	Female	6/19/79	Sheri Mercado	Repeat Customer	36	Ages 30 TO 39	Western Region	California
CA	95110	Male	6/16/78	Raleigh Olson	Repeat Customer	37	Ages 30 TO 39	Western Region	California
CA	95124	Female	1/2/45	Carrie Foreman	Repeat Customer	70	Ages 65 and over	Western Region	California
CA	95134	Female	8/11/84	Renee McMillan	Repeat Customer	31	Ages 30 TO 39	Western Region	California

Geographic Field Metadata

- Fields in data model have metadata properties
  - Metadata used by visuals and reporting tools
  - Used as hints to Bing Mapping service
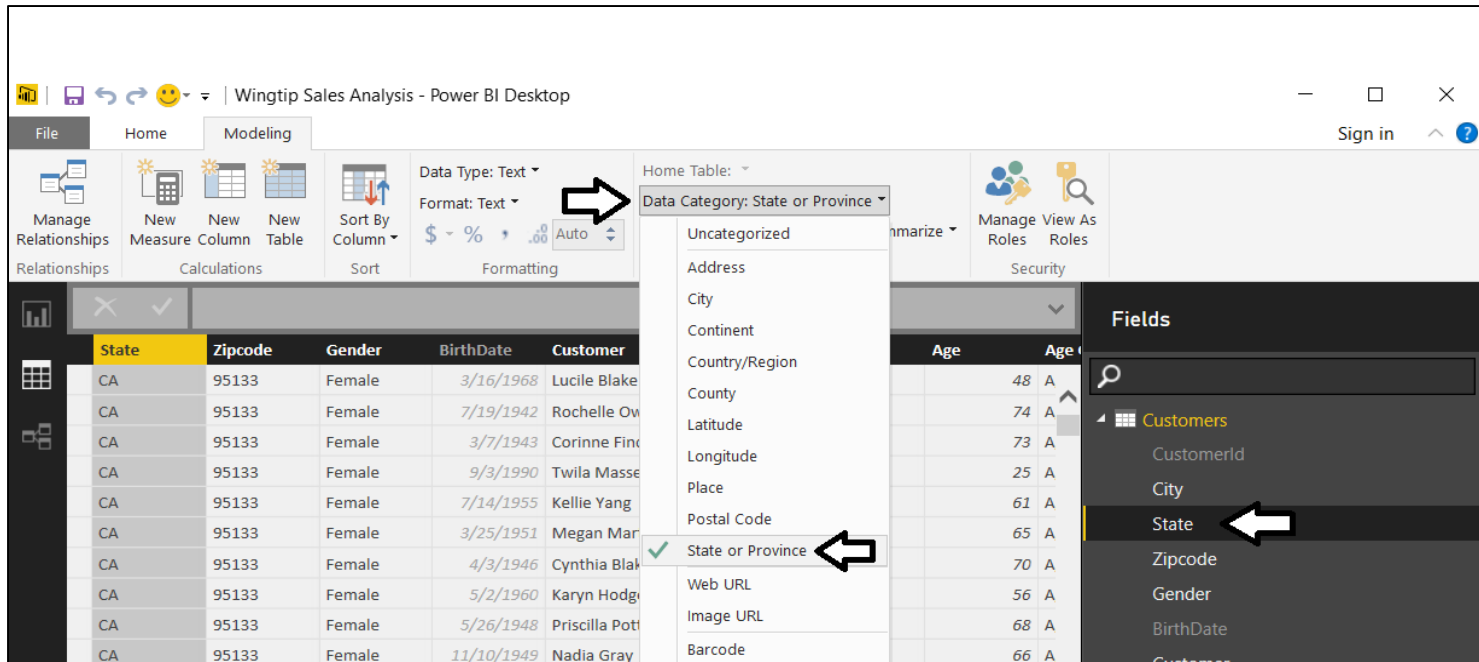
Eliminate Geographic Ambiguity

- ## City name alone is ambiguous
  - ### "Athens" defaults to Greece not Georgia
  - ### Concatenate city name with state to disambiguate

		City = [City Name] & ", " & [State]

	Age Group	Sales Region	State Name	SalesRegionSort	City
48	Ages 40 TO 49	Western Region	California	1	San Jose, CA
74	Ages 65 and over	Western Region	California	1	San Jose, CA
73	Ages 65 and over	Western Region	California	1	San Jose, CA
25	Ages 18 TO 23	Western Region	California	1	San Jose, CA
61	Ages 50 TO 65	Western Region	California	1	San Jose, CA
65	Ages 65 and over	Western Region	California	1	San Jose, CA

Dimensional Hierarchies

- Hierarchy created from two or more columns
  - All columns in hierarchy must be from the same table
  - Defines parent-child relationship between columns
  - Provides path to navigate through data
  - Provides path to drill down into greater level of detail
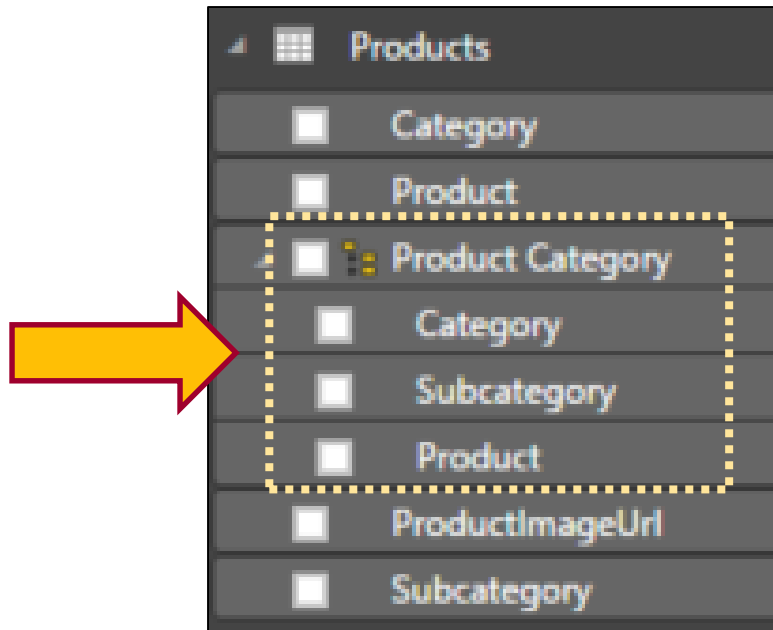
Pulling Columns for Hierarchy into Single Table

- Sometimes hierarchy columns are spread across tables
  - Use RELATED function from DAX to pull columns into single table

  - Then create hierarchy in the table with all the columns

A Tale of Two Evaluation Contexts

- Row Context
  - Context includes all columns in iteration of current row
  - Used to evaluate DAX expression in calculated column
  - Only available in measures with iterator function (e.g. SUMX)

- Filter Context
  - Context includes filter(s) defining current set of rows
  - Used by default to evaluate DAX expressions in measures
  - Can be fully ignored or partially ignored using DAX code
  - Not used to evaluate DAX in calculated columns

Understanding Row Context

- Row context used to evaluate calculated columns

Understanding Iterators Like SUMX

- Standard aggregation functions (e.g. SUM) have no row context
  - You can use SUM to sum values of a single column
  - You cannot use SUM to sum results of an expressions

```
× ✓   Gross Margin = SUM( Sales[SalesAmount]-Sales[ProductCost] )
      ⚠ The SUM function only accepts a column reference as an argument.
```
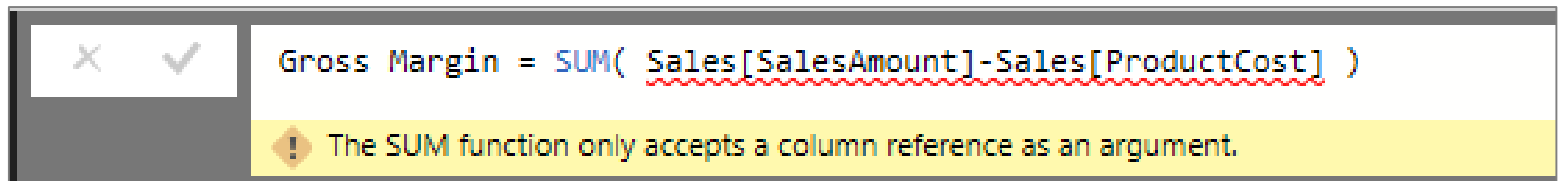
- Iterator functions (e.g. SUMX) iterate through rows in target table

```
× ✓   Gross Margin = SUMX(Sales, Sales[SalesAmount]-Sales[ProductCost] )
```
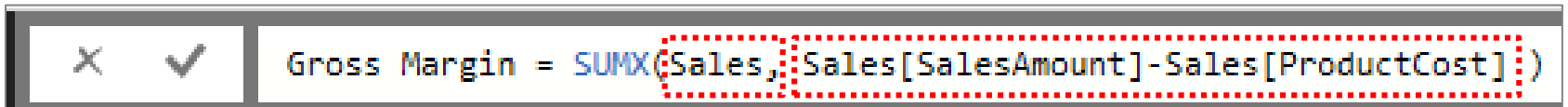
  - First argument accepts expressions that evaluates to table of rows
  - Second argument accepts expression that is evaluated for each row

Understanding Filter Context

- Visuals apply various filters in different evaluation contexts

Month in Year	2012	2013	2014	2015	Total
January	$6,306	$164,334	$385,275	$512,822	$1,068,737
February	$48,815	$126,501	$358,244	$597,684	$1,131,244
March	$53,958	$243,676	$381,309	$532,123	$1,211,067
April	$52,601	$300,872	$381,157	$602,751	$1,337,381
May	$61,756	$334,948	$438,261	$647,276	$1,482,241
June	$76,756	$321,715	$378,749	$608,448	$1,385,668
July	$104,408	$287,800	$359,744	$620,316	$1,372,268
August	$111,167	$298,483	$457,312	$678,499	$1,545,461
September	$110,716	$376,207	$505,332	$613,974	$1,606,229
October	$145,999	$362,943	$602,448	$620,735	$1,732,125
November	$156,751	$340,228	$545,572	$590,220	$1,632,770
December	$147,593	$331,526	$581,977	$686,814	$1,747,910
Total	$1,076,826	$3,489,234	$5,375,379	$7,311,660	$17,253,100

Filters on this evaluation

[Year] = 2015

[Month in Year] = "October"

- Filter context also affected by slicers and other filters

Sales Region
- Select All
- Central Region
- Eastern Region
- ■ Western Region

Customer Type
- One-time customer
- ■ Repeat Customer

Month in Year	2012	2013	2014	2015	Total
January	$425	$50,169	$61,295	$76,614	$188,503
February	$13,891	$40,133	$63,670	$101,542	$219,236
March	$19,121	$58,411	$73,839	$84,180	$235,551
April	$19,128	$53,711	$67,919	$91,762	$232,520
May	$22,939	$64,259	$78,668	$109,689	$275,555
June	$29,082	$50,564	$73,504	$88,047	$241,197
July	$34,809	$62,971	$69,053	$80,749	$247,582
August	$36,096	$61,217	$76,009	$94,719	$268,041
September	$39,415	$68,653	$82,697	$94,805	$285,570
October	$51,994	$69,122	$99,344	$84,177	$304,637
November	$47,020	$52,548	$85,924	$74,611	$260,102
December	$50,580	$66,260	$102,088	$94,877	$313,804
Total	$364,500	$698,018	$934,009	$1,075,771	$3,072,298

Filters on this evaluation

[Year] = 2015

[Month in Year] = "October"

[Sales Region] = "Western Region"

[Customer Type] = "Repeat Customer"

Using the CALCULATE Function

- CALCULATE function provides greatest amount of control
  - First argument defines expression to evaluate
  - Second argument defines table on which to evaluate expression
  - You can evaluate expressions with or without current filter context

```
Pct of All Products =
DIVIDE(
    SUM( Sales[SalesAmount] ),
    CALCULATE(
        Sum (Sales[SalesAmount] ),
        ALL(Products[Category], Products[Subcategory], Products[Product])
    )
)
```

```
Pct of Product Category =
DIVIDE(
    SUM( Sales[SalesAmount] ),
    CALCULATE(
        Sum (Sales[SalesAmount] ),
        ALL( Products[Subcategory], Products[Product] )
    )
)
```

Creating Calendar Table as Calculated Table

- Use **New Table** command in ribbon
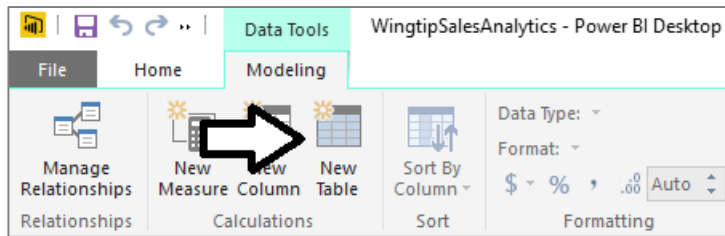
- Create calendar table using DAX **CALENDAR** function
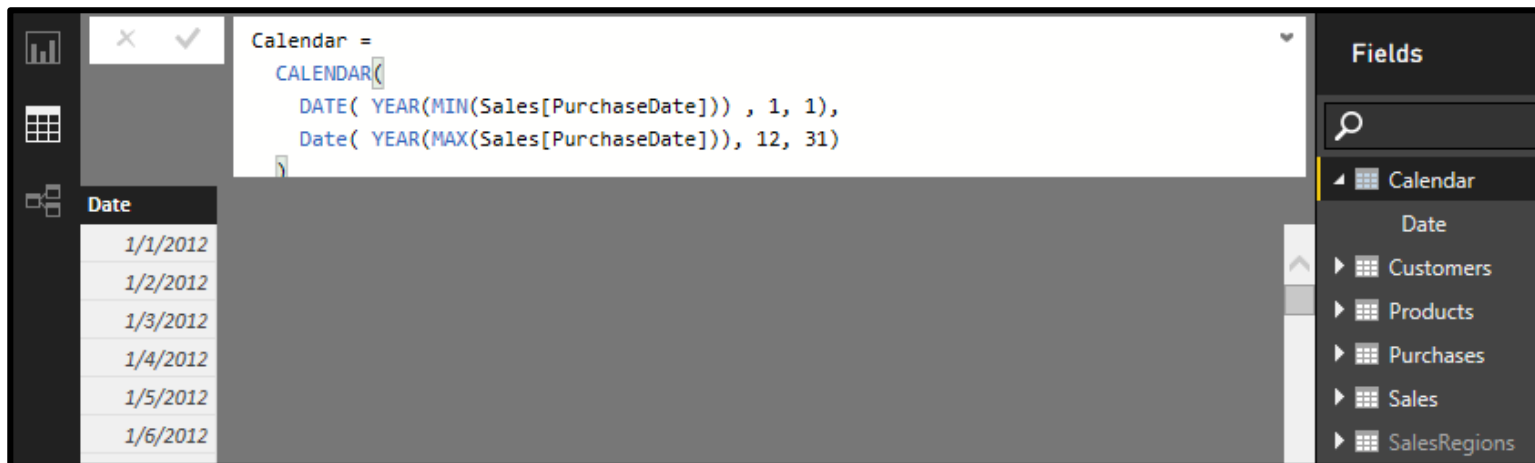
Adding Columns to Calendar Table

- Creating the **Year** column

Date	Year
1/1/2012	2012
1/2/2012	2012
1/3/2012	2012

`Year = YEAR('Calendar'[Date])`

- Creating the **Quarter** column

Date	Year	Quarter
01/01/2012	2012	2012-Q1
01/02/2012	2012	2012-Q1
01/03/2012	2012	2012-Q1
01/04/2012	2012	2012-Q1
01/05/2012	2012	2012-Q1

`Quarter = YEAR('Calendar'[Date]) & "-Q" & FORMAT('Calendar'[Date], "q")`

- Creating the **Month** column

Date	Year	Quarter	Month
1/1/2012	2012	2012-Q1	Jan 2012
1/2/2012	2012	2012-Q1	Jan 2012
1/3/2012	2012	2012-Q1	Jan 2012

`Month = FORMAT('Calendar'[Date], "MMM yyyy")`

Configuring Sort Columns

- Month column will not sort in desired fashion by default
  - For example, April will sort before January, February and March

- Creating a sort column for the **Month** column
  - **MonthSort** sorts alphabetically & chronologically at same time

  - Configure **Month** column with **MonthSort** as sort column

Columns for Month in Year and Day in week

- Creating the **Month in Year** column

		Month in Year = FORMAT('Calendar'[Date], "MMMM")			
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January

- Creating the **MonthInYearSort** column

		MonthInYearSort = MONTH('Calendar'[Date])				
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1

- Creating the **Day of Week** column

		Day of Week = FORMAT('Calendar'[Date], "dddd")					
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**	**Day of Week**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Sunday
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Monday

- Creating the **DayOfWeekSort** column

		DayOfWeekSort = WEEKDAY('Calendar'[Date], 2)						
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**	**Day of Week**	**DayOfWeekSort**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Sunday	7
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Monday	1
1/3/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Tuesday	2
1/4/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Wednesday	3

Integrating Calendar Table into Data Model

- Calendar table needs relationship to one or more tables
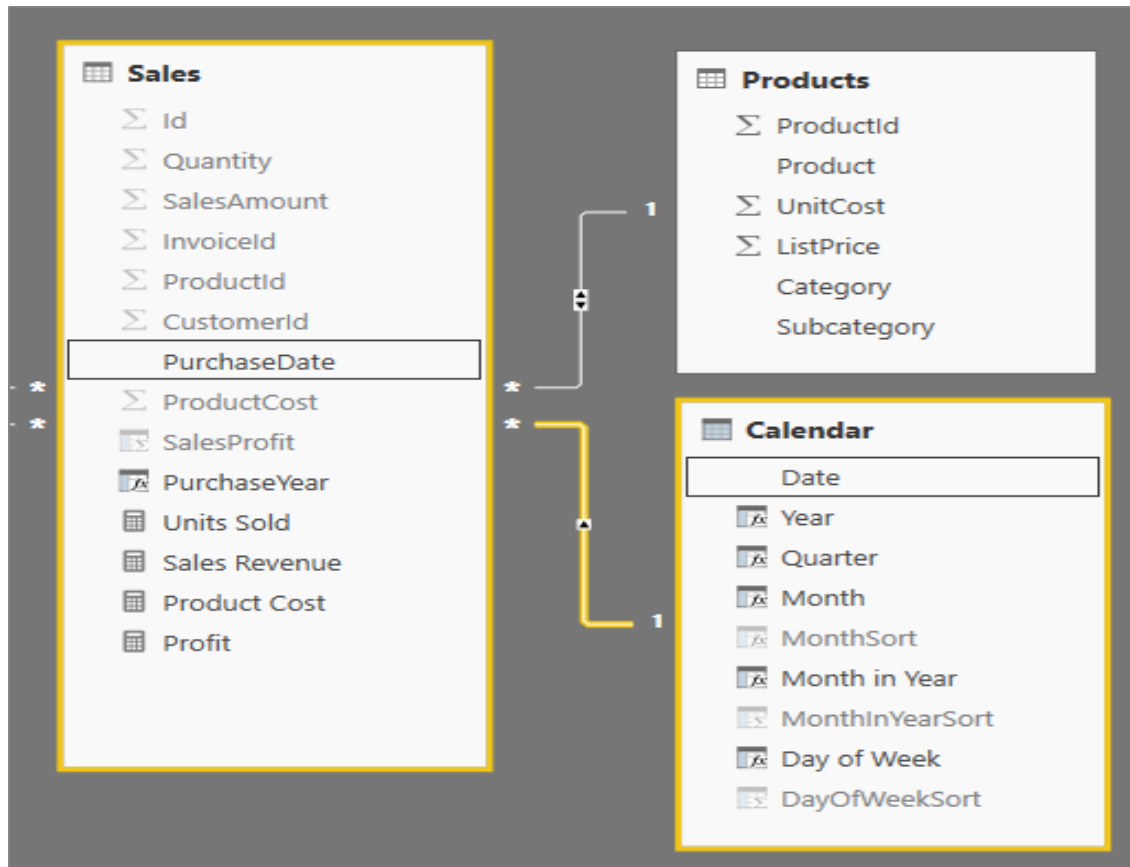
Calculated Fields for QTD and YTD Sales

- TOTALQTD function calculates quarter-to-date totals
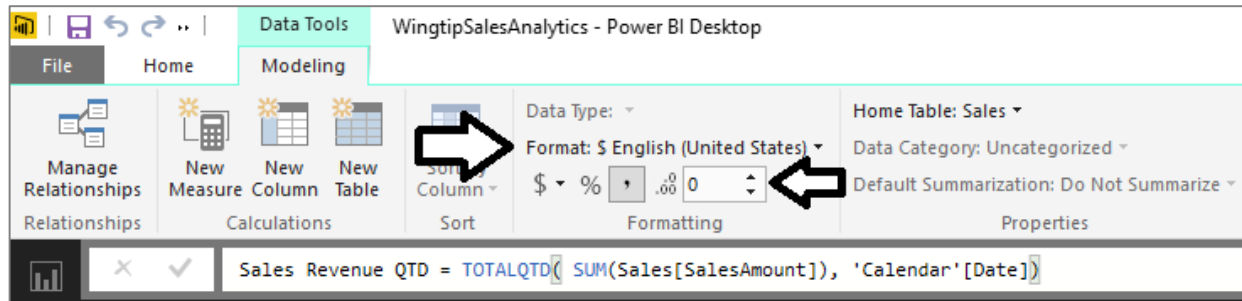
- TOTALYTD function calculates year-to-date totals
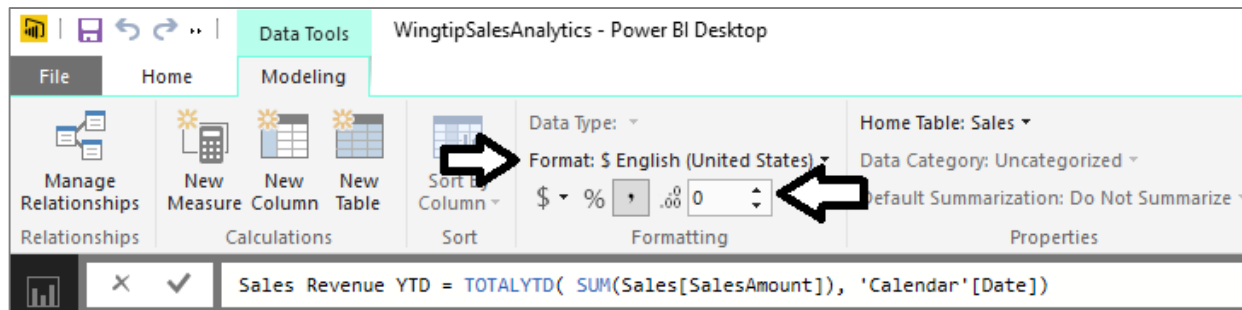
Creating Running Total using CALCULATE

- Calculate a running total of sales revenue across years
  - This must be done using **CALCULATE** function


```
Sales Revenue RT =
CALCULATE(
    SUM(Sales[SalesAmount]),
    FILTER(
        ALL('Calendar'),
        'Calendar'[Date] <= MAX('Calendar'[Date])
    )
)
```

Sales Growth PM Measure - First Attempt

- Create a measure named Sales Growth PM

```
Sales Growth PM =
DIVIDE(
  SUM(Sales[SalesAmount]) -
  CALCULATE(
    SUM(Sales[SalesAmount]),
    PREVIOUSMONTH(Calendar[Date])
  ),
  CALCULATE(
    SUM(Sales[SalesAmount]),
    PREVIOUSMONTH(Calendar[Date])
  )
)
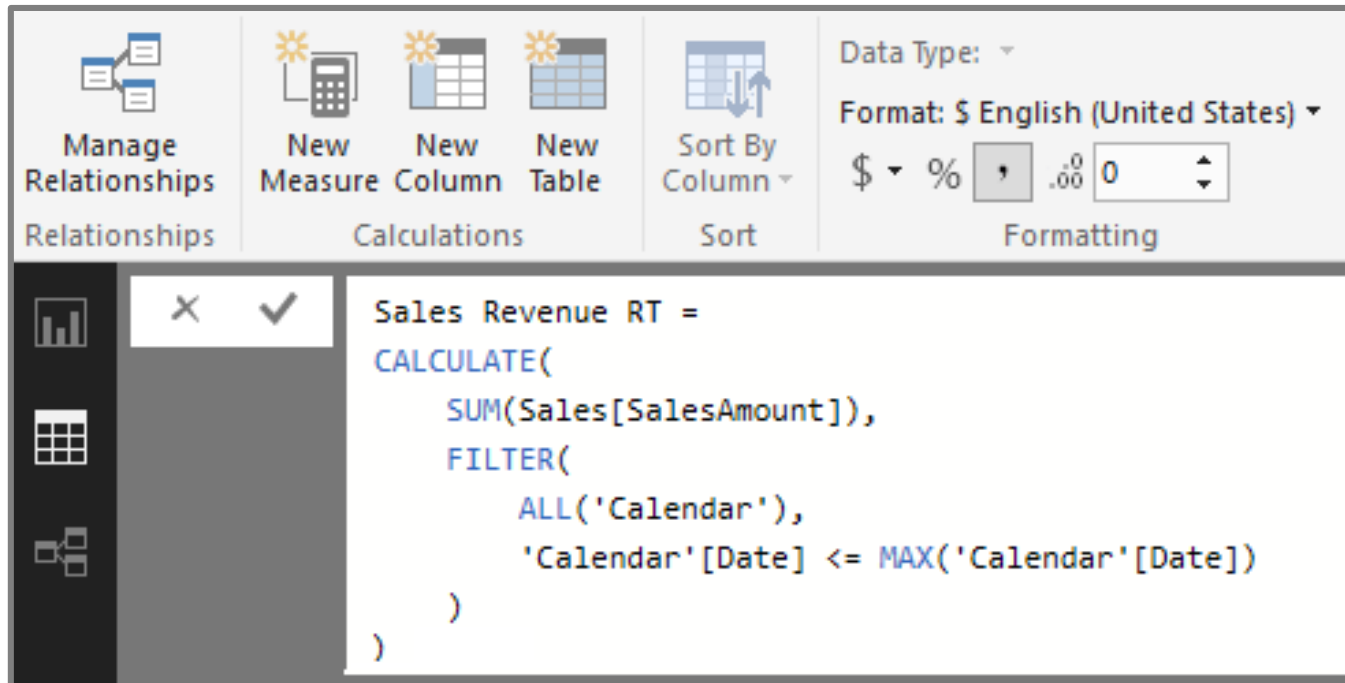```

- Use measure in matrix evaluating month and quarter
  - Measure returns correct value when filtered by Month
  - Measure returns large, erroneous value when filtered by Quarter

| Year | Quarter | Month | Sales Revenue | Sales Growth PM |
|------|---------|-------|---------------|-----------------|
| 2014 | 2014-Q1 | Jan 2014 | $629,969 | -18.13 % |
| | | Feb 2014 | $609,637 | -3.23 % |
| | | Mar 2014 | $628,618 | 3.11 % |
| | | **Total** | **$1,868,225** | **142.79 %** |
| | 2014-Q2 | Apr 2014 | $661,588 | 5.24 % |
| | | May 2014 | $748,193 | 13.09 % |
| | | Jun 2014 | $814,333 | 8.84 % |
| | | **Total** | **$2,224,114** | **253.81 %** |
| | 2014-Q3 | Jul 2014 | $788,469 | -3.18 % |

# Using the ISFILTERED Function

- ISFILTERED function used to determine when perform evaluation

```
Sales Growth PM =
IF(
  ( ISFILTERED(Calendar[Month]) && NOT(ISFILTERED(Calendar[Date])) ),
  DIVIDE(
    SUM(Sales[SalesAmount]) -
    CALCULATE(
      SUM(Sales[SalesAmount]),
      PREVIOUSMONTH(Calendar[Date])
    ),
    CALCULATE(
      SUM(Sales[SalesAmount]),
      PREVIOUSMONTH(Calendar[Date])
    )
  ),
  BLANK()
)
```

- Expression returns Blank value when evaluation context is invalid

Year	Quarter	Month	Sales Revenue	Sales Growth PM
2014	2014-Q1	Jan 2014	$629,969	-18.13 %
		Feb 2014	$609,637	-3.23 %
		Mar 2014	$628,618	3.11 %
		Total	**$1,868,225**	
	2014-Q2	Apr 2014	$661,588	5.24 %
		May 2014	$748,193	13.09 %
		Jun 2014	$814,333	8.84 %
		Total	**$2,224,114**	
	2014-Q3	Jul 2014	$788,469	-3.18 %
		Aug 2014	$869,143	10.23 %

Agenda

- ✓ Power BI Desktop Overview
- ✓ Building Queries
- ✓ Designing Data Models
- ➢ Designing Reports
- • Query Parameters
- • Power BI Desktop Template Files

Agenda

- ✓ Power BI Desktop Overview
- ✓ Building Queries
- ✓ Designing Data Models
- ✓ Designing Reports
- ➢ Query Parameters
- • Power BI Desktop Template Files

Query Parameters

- What is a Query Parameter?
  - Configurable setting with project scope
  - Strongly-typed value to which you can apply restrictions
  - Can be referenced from a query
  - Can be referenced from DAX code in data model

- Where are Parameters commonly used
  - To parameterize data source connection details
  - To filter rows when importing data

Creating Query Parameters

- Parameters can be created using **Manager Parameters** menu
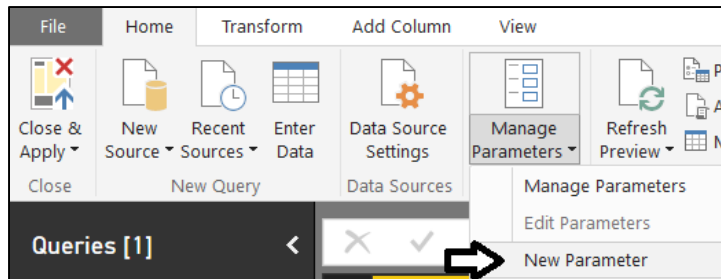
- Parameter properties
  - Name
  - Description
  - Required
  - Allowed Values
  - Default Value
  - Current Value

Referencing Parameters in a Query

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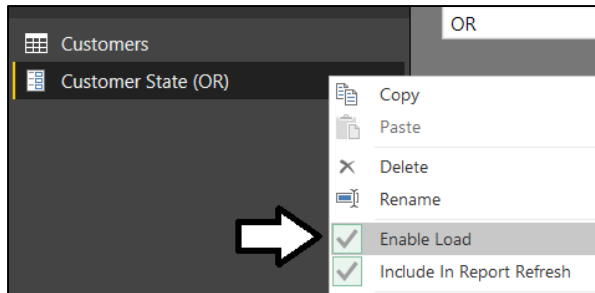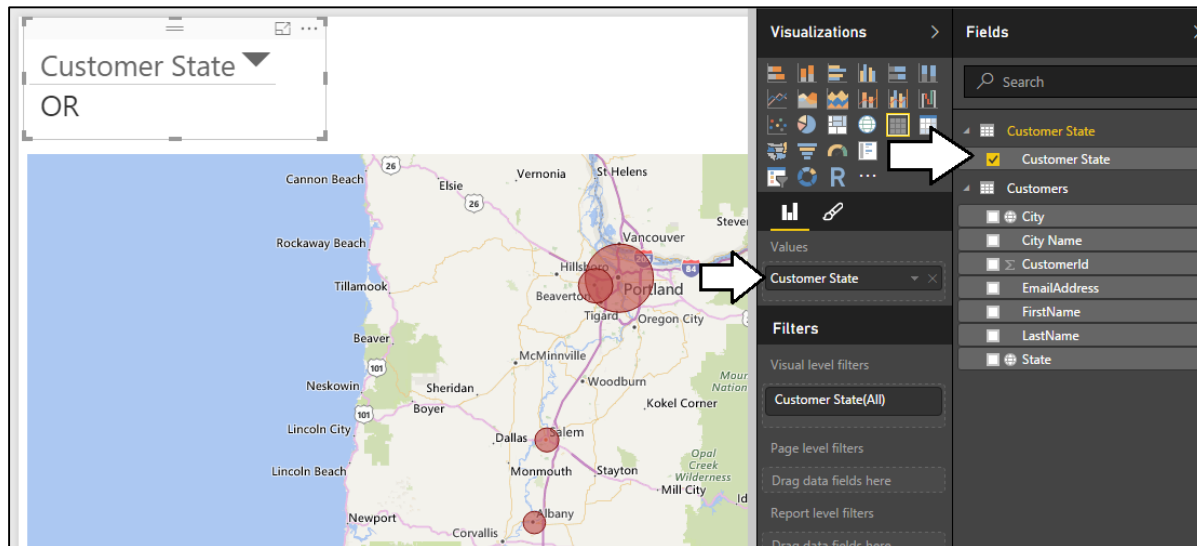 - ### Next query execution uses current parameter value

Making Parameters Available to Data Model

- Configure parameter's Enable Load setting

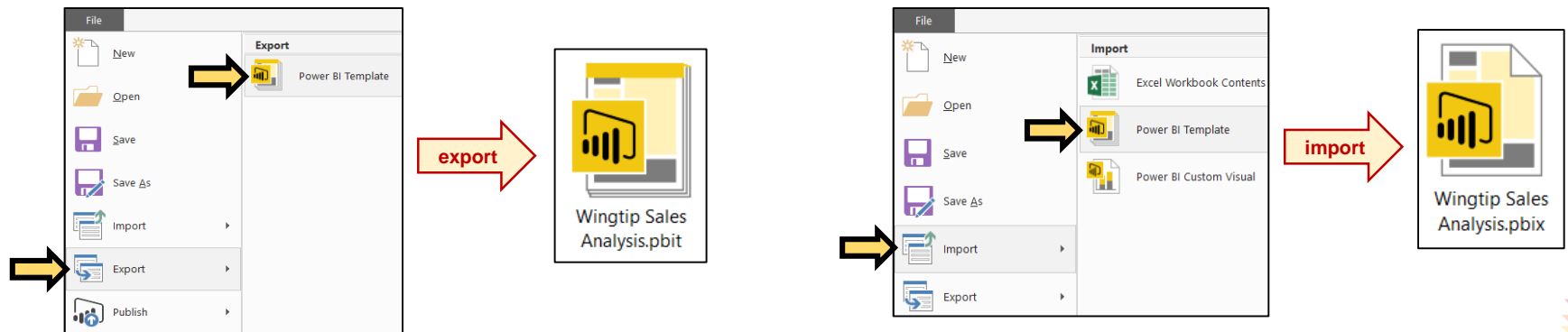- Parameter becomes visible within fields list in report view

Agenda

- ✓ Power BI Desktop Overview
- ✓ Building Queries
- ✓ Designing Data Models
- ✓ Designing Reports
- ✓ Query Parameters
- ➢ Power BI Desktop Template Files

Power BI Project Template Files

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

Summary

- ✓ Power BI Desktop Overview
- ✓ Building Queries
- ✓ Designing Data Models
- ✓ Designing Reports
- ✓ Query Parameters
- ✓ Power BI Desktop Template Files