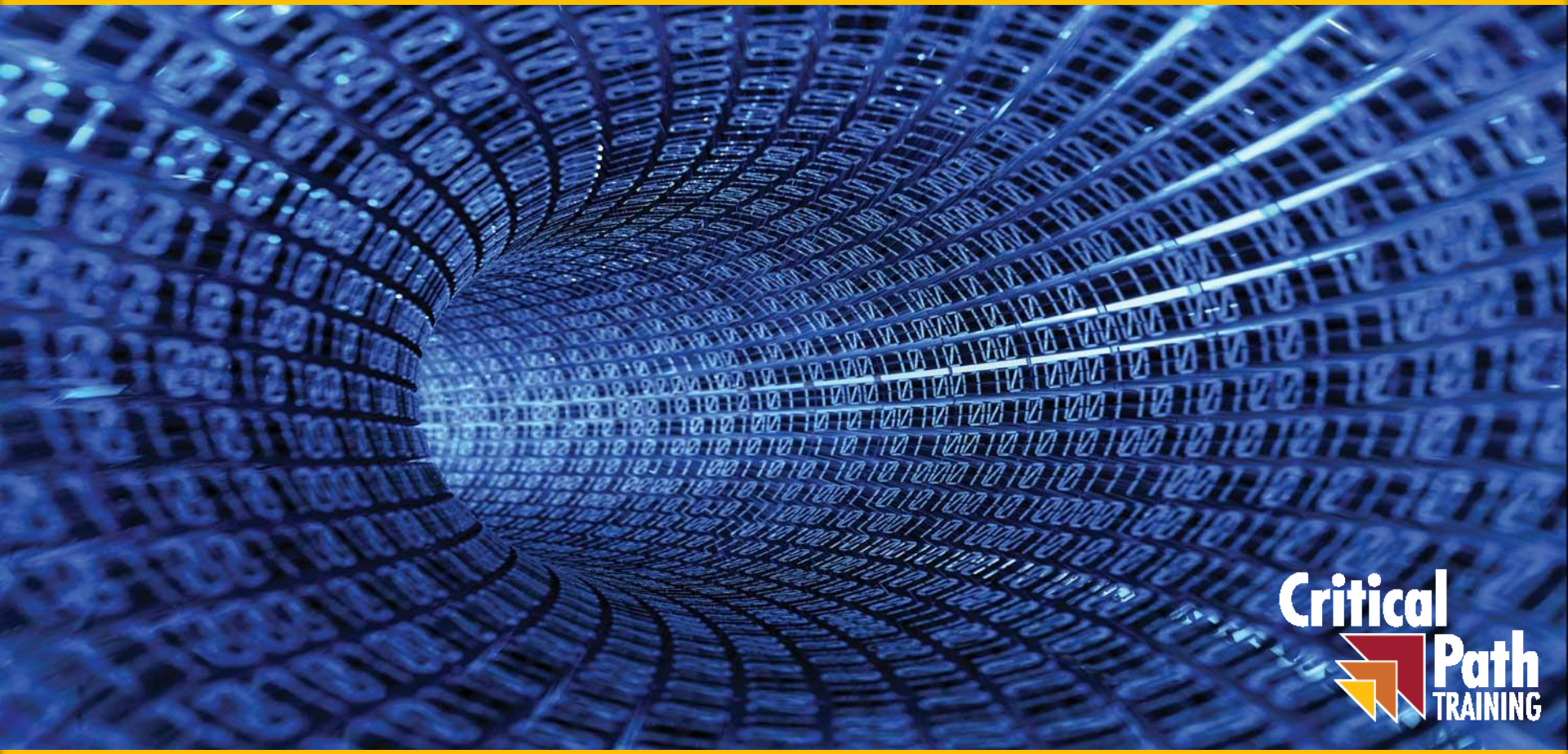
# Writing Advanced DAX Expressions

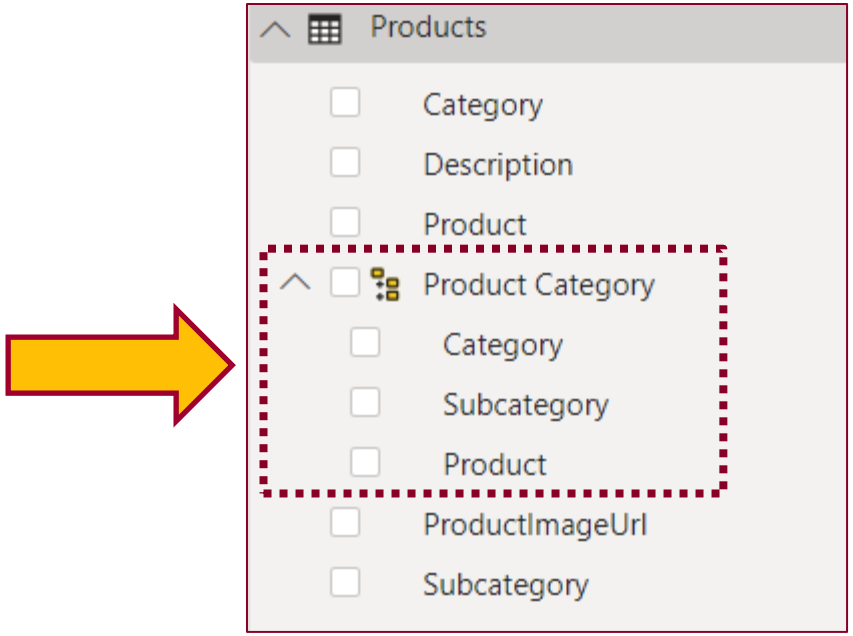


# Agenda

- Creating Dimensional Hierarchies
- Understanding Evaluation Context and Calculate
- Creating a Calendar Table
- Calling DAX Time Intelligence Functions
- Writing Contextually-aware DAX Expressions
- Calculating the Top 5 Products

# 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



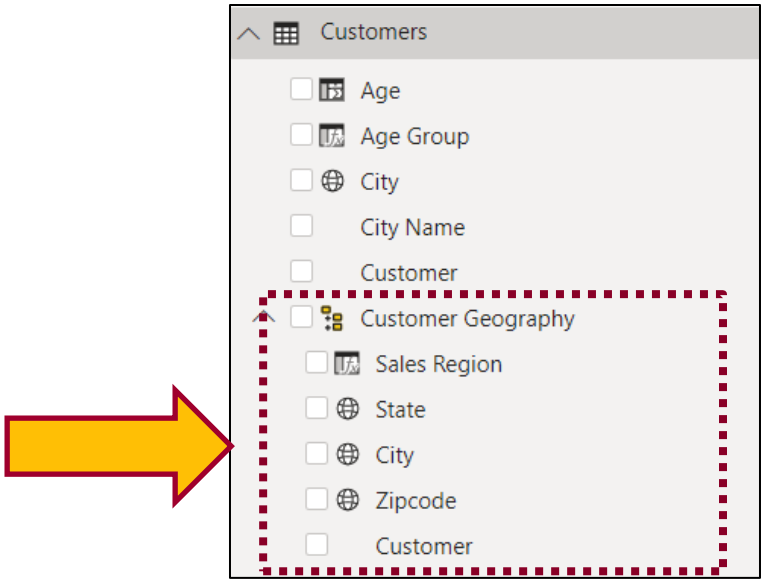
# Pulling Columns for Hierarchy into Single Table

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

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

| CustomerId | City | State | Zipcode | Gender | BirthDate | Customer | Customer Type | Age | Age Group | Sales Region |
|---|---|---|---|---|---|---|---|---|---|---|
| 760 | San Jose | CA | 95133 | Female | 3/16/1968 | Lucile Blake | One-time Customer | 51 | 50 to 64 | Western Region |
| 881 | San Jose | CA | 95133 | Female | 7/19/1942 | Rochelle Owen | One-time Customer | 77 | 65 and over | Western Region |

  - Then create hierarchy in the table with all the columns



# Agenda

- ✓ Creating Dimensional Hierarchies
- ➢ Understanding Evaluation Context and Calculate
- Creating a Calendar Table
- Calling DAX Time Intelligence Functions
- Writing Contextually-aware DAX Expressions
- Calculating the Top 5 Products

# 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

```
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 |

```
Age = Floor( (TODAY()-Customers[BirthDate])/365, 1)
```

| Customer | Customer Type | Age | Age Group | Sales Region | State Name |
|---|---|---|---|---|---|
| Lucile Blake | One-time Customer | 48 | Ages 40 TO 49 | Western Region | California |
| Rochelle Owen | One-time Customer | 74 | Ages 65 and over | Western Region | California |
| Corinne Finch | One-time Customer | 73 | Ages 65 and over | Western Region | California |

# 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 |
|---|---|---|---|---|---|
| Jan | $3,063 | $307,182 | $629,969 | $959,863 | **$1,900,077** |
| Feb | $33,218 | $291,942 | $609,637 | $969,330 | **$1,904,126** |
| Mar | $49,213 | $346,186 | $628,618 | $675,533 | **$1,699,551** |
| Apr | $40,434 | $380,869 | $661,588 | $722,456 | **$1,805,347** |
| May | $83,840 | $377,376 | $748,193 | $698,311 | **$1,907,720** |
| Jun | $136,670 | $353,586 | $814,333 | $785,793 | **$2,090,382** |
| Jul | $144,244 | $391,202 | $788,469 | $921,994 | **$2,245,908** |
| Aug | $197,952 | $476,884 | $869,143 | $1,084,189 | **$2,628,168** |
| Sep | $215,097 | $504,532 | $890,958 | $1,088,863 | **$2,699,449** |
| Oct | $239,513 | $577,439 | $988,789 | $1,211,810 | **$3,017,551** |
| Nov | $376,503 | $579,507 | $999,574 | $1,305,029 | **$3,260,613** |
| Dec | $424,240 | $769,473 | $1,644,980 | $1,732,932 | **$4,571,625** |
| **Total** | **$1,943,986** | **$5,356,177** | **$10,274,251** | **$12,156,103** | **$29,730,517** |

**Filters on this evaluation**

[Year] = 2015

[Month in Year] = "October"

- Filter context also affected by slicers and other filters

Sales Region
- ■ Western Region
- ☐ Central Region
- ☐ Eastern Region

Customer Type
- ☐ One-time Customer
- ■ Repeat Customer

| Month in Year | 2012 | 2013 | 2014 | 2015 | Total |
|---|---|---|---|---|---|
| Jan | | $117,712 | $202,751 | $182,616 | **$503,079** |
| Feb | $8,264 | $126,522 | $181,564 | $184,674 | **$501,024** |
| Mar | $22,434 | $148,668 | $160,857 | $169,933 | **$501,892** |
| Apr | $22,235 | $178,506 | $183,987 | $194,197 | **$578,925** |
| May | $36,719 | $169,582 | $210,150 | $173,661 | **$590,112** |
| Jun | $55,119 | $158,668 | $217,947 | $196,431 | **$628,166** |
| Jul | $72,823 | $187,093 | $233,333 | $193,830 | **$687,079** |
| Aug | $90,917 | $169,789 | $233,101 | $209,895 | **$703,703** |
| Sep | $77,898 | $155,469 | $225,287 | $213,017 | **$671,672** |
| Oct | $84,735 | $208,700 | $197,377 | $207,227 | **$698,039** |
| Nov | $130,678 | $168,821 | $227,856 | $190,144 | **$717,498** |
| Dec | $147,043 | $203,781 | $234,393 | $195,796 | **$781,013** |
| **Total** | **$748,866** | **$1,993,312** | **$2,508,601** | **$2,311,421** | **$7,562,200** |

**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])
    )
)
```

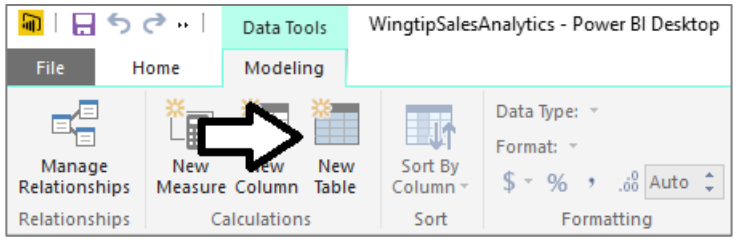
```
Pct of Product Category =
DIVIDE(
    SUM( Sales[SalesAmount] ),
    CALCULATE(
        Sum (Sales[SalesAmount] ),
        ALL( Products[Subcategory], Products[Product] )
    )
)
```

# Agenda

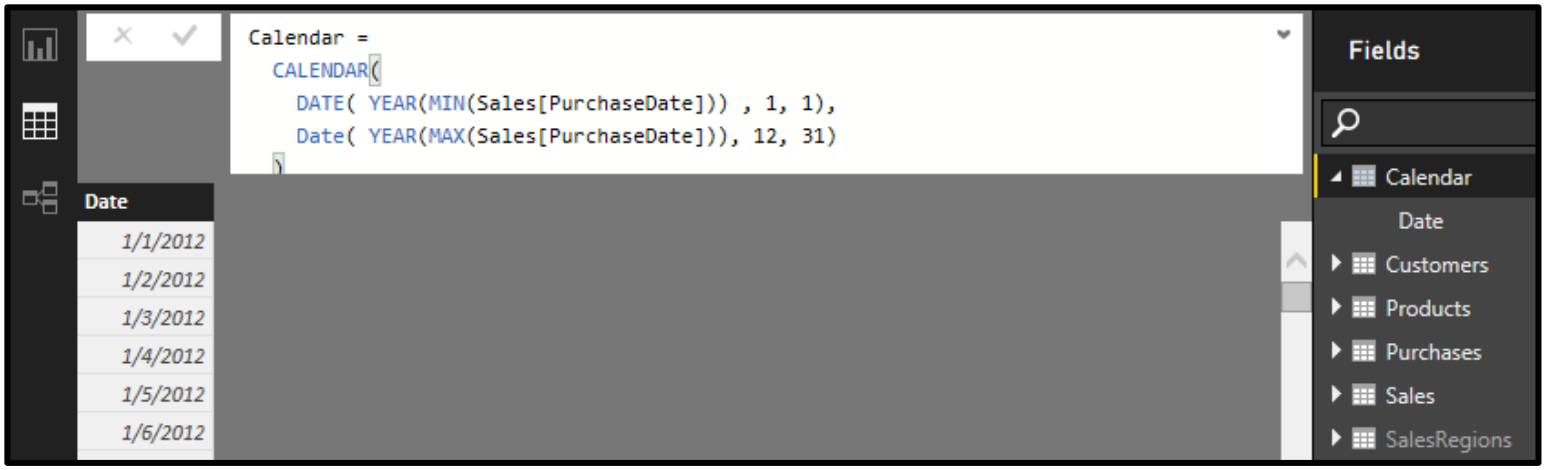
- ✓ Creating Dimensional Hierarchies
- ✓ Understanding Evaluation Context and Calculate
- ➢ Creating a Calendar Table
- Calling DAX Time Intelligence Functions
- Writing Contextually-aware DAX Expressions
- Calculating the Top 5 Products

# 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

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

| Date | Year |
|---|---|
| 1/1/2012 | 2012 |
| 1/2/2012 | 2012 |
| 1/3/2012 | 2012 |

- Creating the **Quarter** column

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

| 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 |

- Creating the **Month** column

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

| 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 |

# Configuring Sort Columns

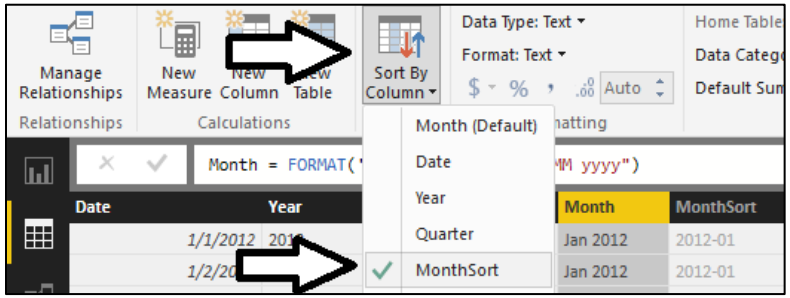
- Month column will not sort in desired fashion by default
  - For example, April will sort before January, February and March

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

```
MonthSort = FORMAT('Calendar'[Date], "yyyy-MM")
```

| Date | Year | Quarter | Month | MonthSort |
|---|---|---|---|---|
| 1/1/2012 | 2012 | 2012-Q1 | Jan 2012 | 2012-01 |
| 1/2/2012 | 2012 | 2012-Q1 | Jan 2012 | 2012-01 |

  - 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 |

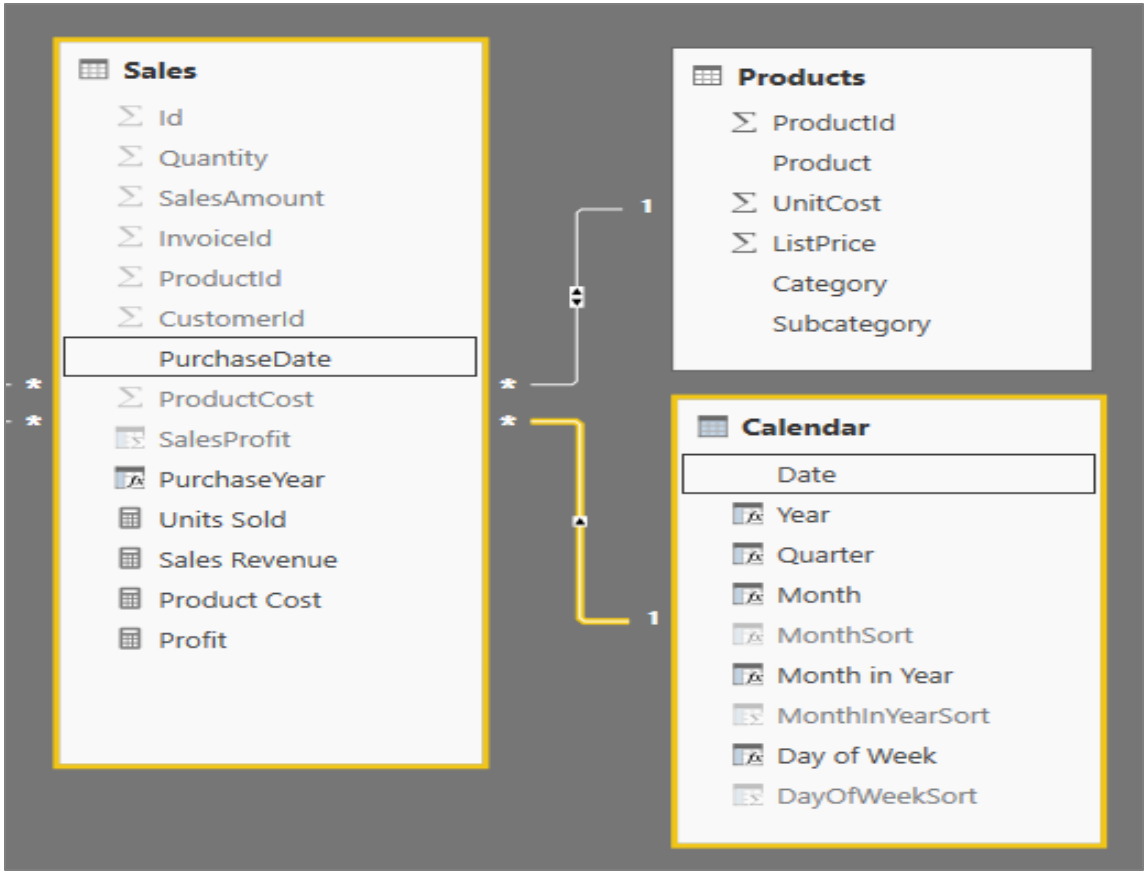
- Creating the **DayOfWeekSort** column

```
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

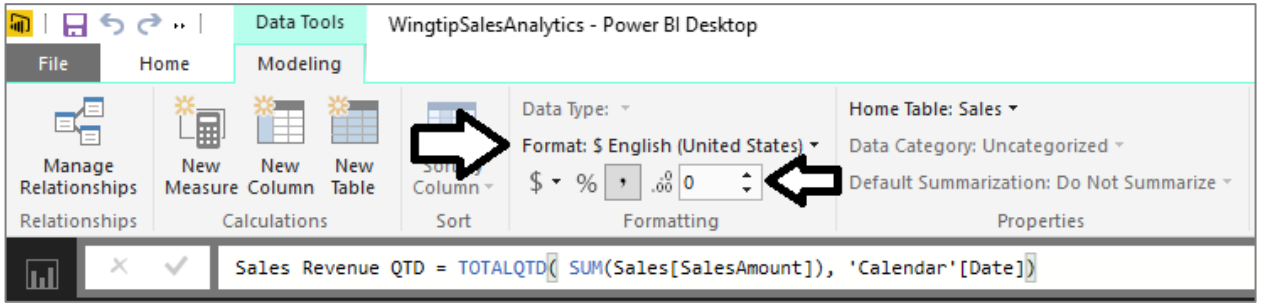


# Agenda

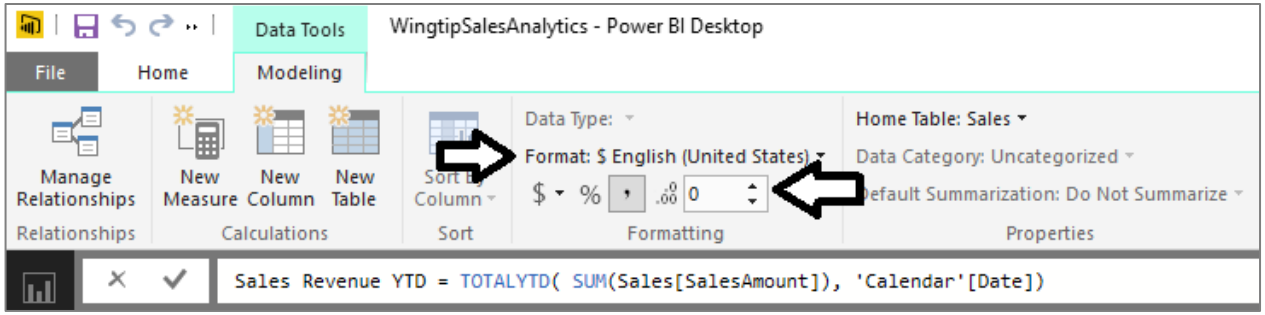
- ✓ Creating Dimensional Hierarchies
- ✓ Understanding Evaluation Context and Calculate
- ✓ Creating a Calendar Table
- ➢ Calling DAX Time Intelligence Functions
- Writing Contextually-aware DAX Expressions
- Calculating the Top 5 Products

# Calculated Fields for QTD and YTD Sales

- TOTALQTD function calculates quarter-to-date totals

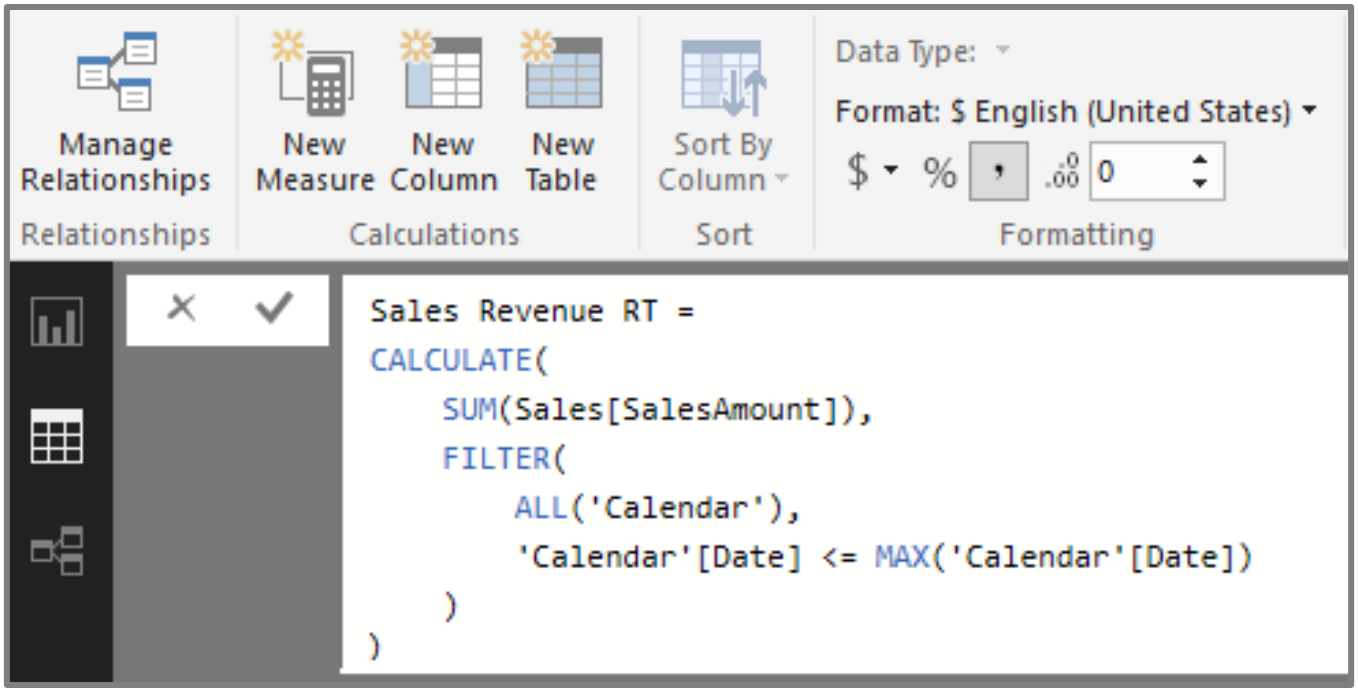



- TOTALYTD function calculates year-to-date totals



# Creating Running Total using CALCULATE

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



# Agenda

- ✓ Creating Dimensional Hierarchies
- ✓ Understanding Evaluation Context and Calculate
- ✓ Creating a Calendar Table
- ✓ Calling DAX Time Intelligence Functions
- ➢ Writing Contextually-aware DAX Expressions
- • Calculating the Top 5 Products

# 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 % |

# Problems with the Filter Context

- RANKX function is affected by filter context
  - Sometimes you get the results you are expecting

Year
- [x] 2012
- [ ] 2013
- [ ] 2014
- [ ] 2015

| Product Rank | Product | Sales Revenue |
|---|---|---|
| 1 | Easel with Supply Trays | $538,161 |
| 2 | Etch A Sketch | $243,512 |
| 3 | Godzilla Action Figure | $194,552 |
| 4 | Red Stomper Bully | $158,825 |
| 5 | Crate o' Crayons | $111,093 |

  - Sometimes you might get unexpected results

Year
- [ ] 2012
- [ ] 2013
- [ ] 2014
- [ ] 2015

Category
- [x] Action Figures
- [ ] Arts and Crafts
- [ ] Remote Control Vehicles

| Product Rank | Product | Sales Revenue |
|---|---|---|
| 2 | Twitter Follower Action Figure | $3,508,806 |
| 3 | Godzilla Action Figure | $2,970,735 |

# Writing Context Aware DAX Code

- When using RANKX…
  - It's recommended to call **HASONEVALUE** function
  - When calling ALL function, pass one or more columns

```
Product Rank =
IF(
  HASONEVALUE(Products[Product]),
  RANKX(
    ALL( Products[Subcategory], Products[Product] ),
    CALCULATE( SUM(Sales[SalesAmount]) )
  )
)
```

  - Ranking function now evaluates product ranking for specific Category

Year
- 2012
- 2013
- 2014
- 2015

Category
- Action Figures
- Arts and Crafts
- Remote Control Vehicles

| Product Rank | Product | Sales Revenue |
|---|---|---|
| 1 | Twitter Follower Action Figure | $3,508,806 |
| 2 | Godzilla Action Figure | $2,970,735 |
| 3 | Captain America Action Figure | $855,607 |
| 4 | Spiderman Action Figure | $698,614 |
| 5 | Perry the Platypus Action Figure | $654,110 |

# Agenda

- ✓ Creating Dimensional Hierarchies
- ✓ Understanding Evaluation Context and Calculate
- ✓ Creating a Calendar Table
- ✓ Calling DAX Time Intelligence Functions
- ✓ Writing Contextually-aware DAX Expressions
- ➢ Calculating the Top 5 Products

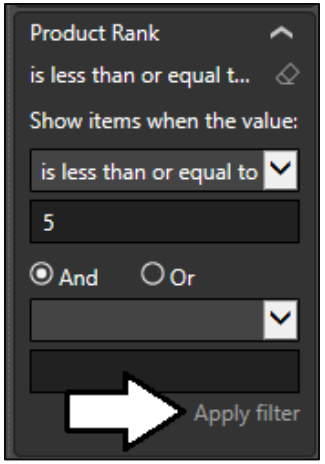
# Ranking Products By Sales using RANKX

- DAX provides RANKX function for ranking
  - Can be used to track top 5 products by sales revenue

```
Product Rank =
RANKX(
    ALL(Products),
    CALCULATE( SUM(Sales[SalesAmount]) )
)
```

  - You can sort and filter on output of RANKX function

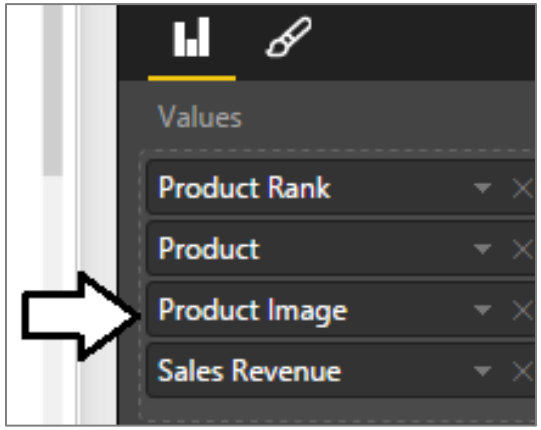
| Product Rank | Product | Sales Revenue |
|---|---|---|
| 1 | Flying Squirrel | $3,828,783 |
| 2 | Twitter Follower Action Figure | $3,508,806 |
| 3 | Godzilla Action Figure | $2,970,735 |
| 4 | Personal Commuter Chopper | $2,613,193 |
| 5 | Red Stomper Bully | $2,538,233 |



# More Ranking Evaluation Problems

- Adding new column to table creates new problem
  - Ranking run separately for each separate Product Image
  - Every product has unique Product Image and is given rank of 1

| Product Rank | Product | Product Image | Sales Revenue |
|---|---|---|---|
| 1 | Batman Action Figure | | $40,395 |
| 1 | Black Power Ranger Action Figure | | $4,223 |
| 1 | Captain America Action Figure | | $125,110 |
| 1 | Crate o' Crayons | | $322,711 |
| 1 | Crayloa Crayon Set | | $12,868 |
| 1 | Easel with Supply Trays | | $928,620 |
| 1 | Etch A Sketch | | $293,175 |



# Getting It Right

- Call to RANKX must be modified again
  - You must specify which columns to factor into ranking

```
Product Rank =
IF(
  HASONEVALUE(Products[Product]),
  RANKX(
    ALL( Products[Subcategory], Products[Product], Products[Product Image] ),
    CALCULATE( SUM(Sales[SalesAmount]) )
  )
)
```

  - Context-aware DAX code corrects problems with visual

| Product Rank | Product | Product Image | Sales Revenue |
|---|---|---|---|
| 1 | Flying Squirrel | | $3,828,783 |
| 2 | Twitter Follower Action Figure | | $3,508,806 |
| 3 | Godzilla Action Figure | | $2,970,735 |
| 4 | Personal Commuter Chopper | | $2,613,193 |
| 5 | Red Stomper Bully | | $2,538,233 |

# Summary

- ✓ Creating Dimensional Hierarchies
- ✓ Understanding Evaluation Context and Calculate
- ✓ Creating a Calendar Table
- ✓ Calling DAX Time Intelligence Functions
- ✓ Writing Contextually-aware DAX Expressions
- ✓ Calculating the Top 5 Products