

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



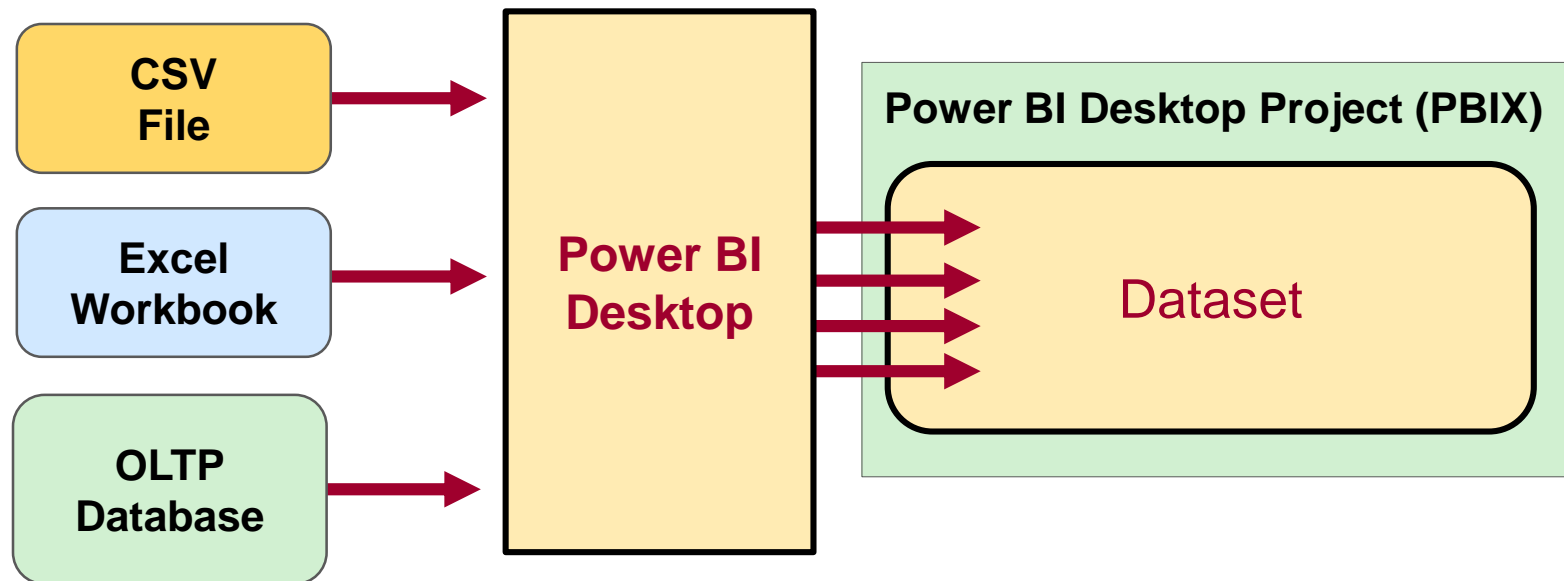
Agenda

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



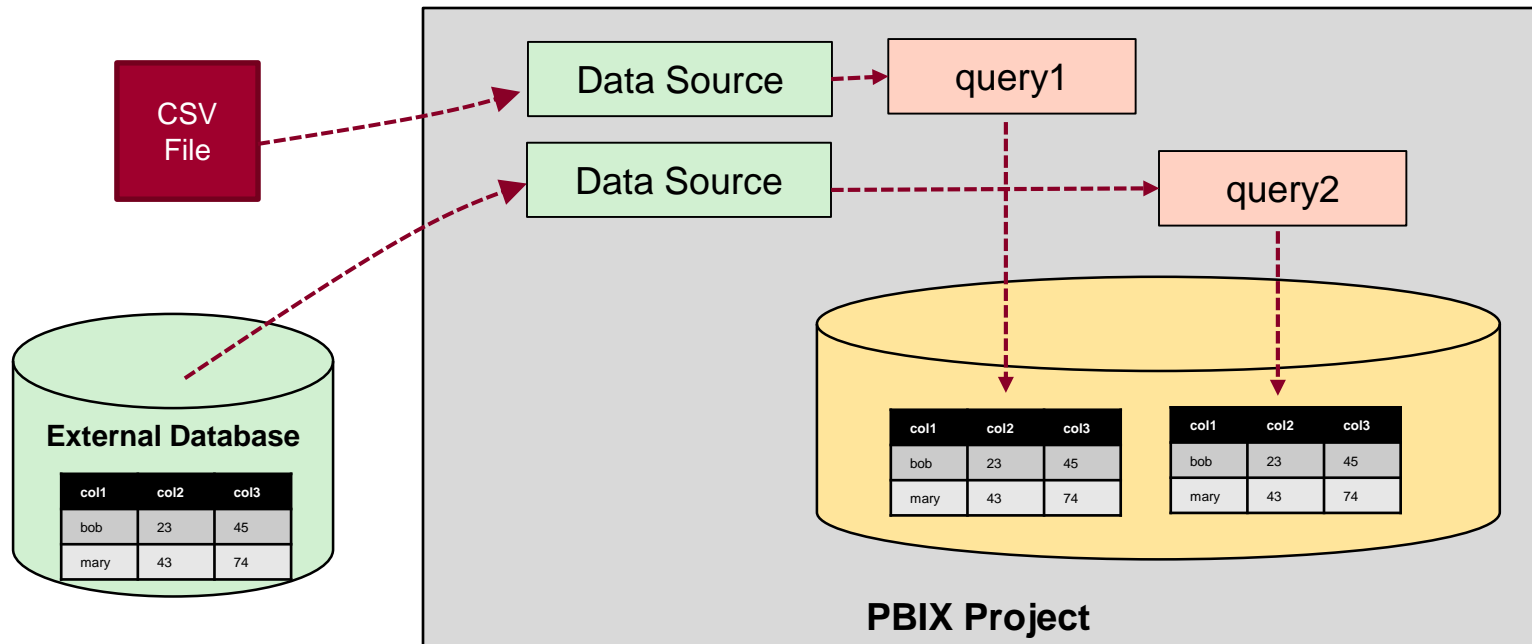
Power BI Desktop is an ETL Tool

- 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



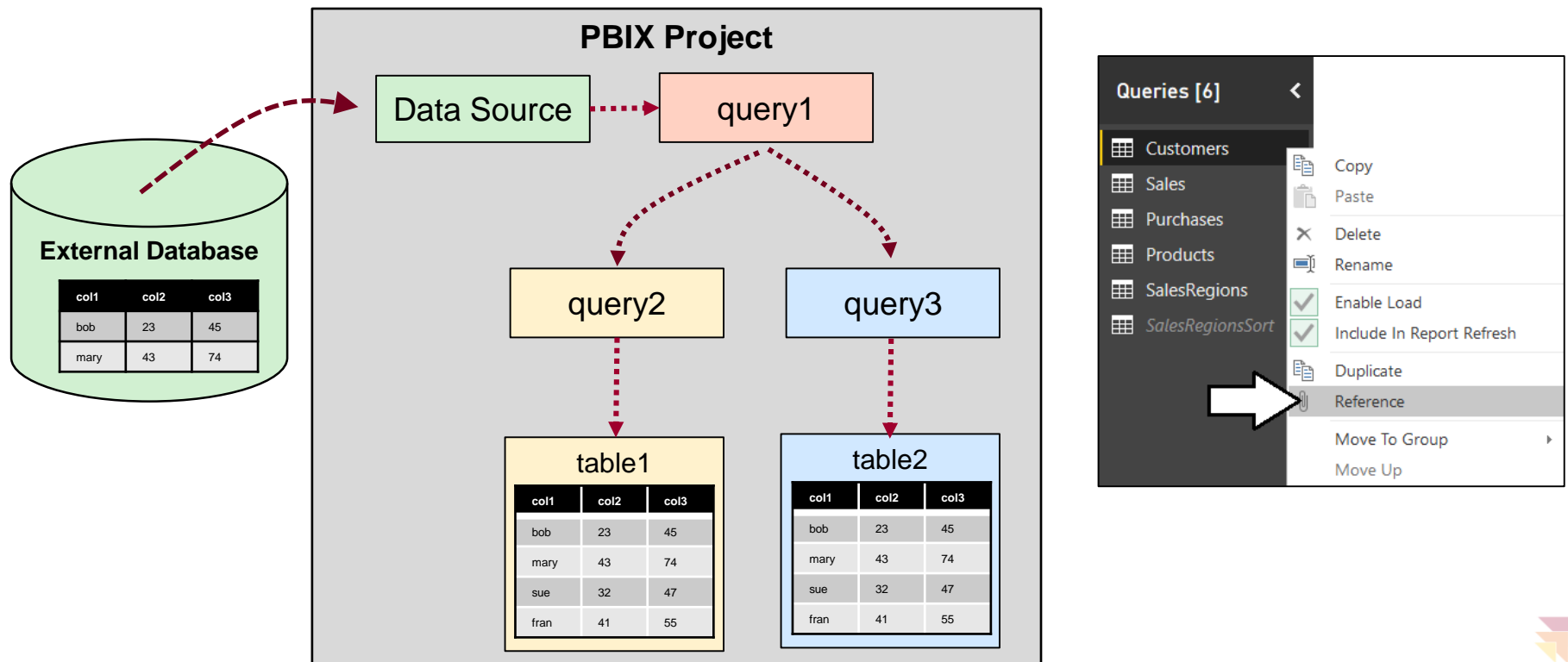
Understanding Query Input and Output

- 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



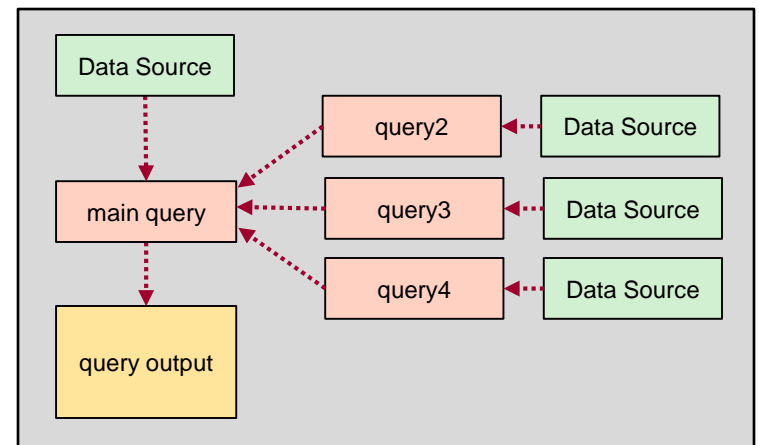
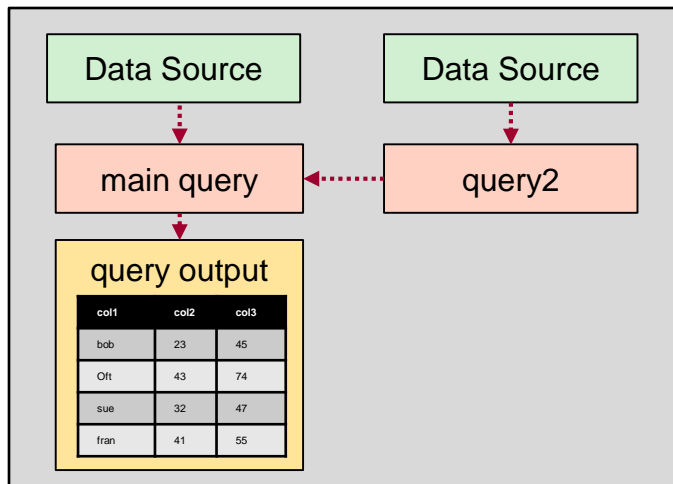
Query Composition

- 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 can be 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



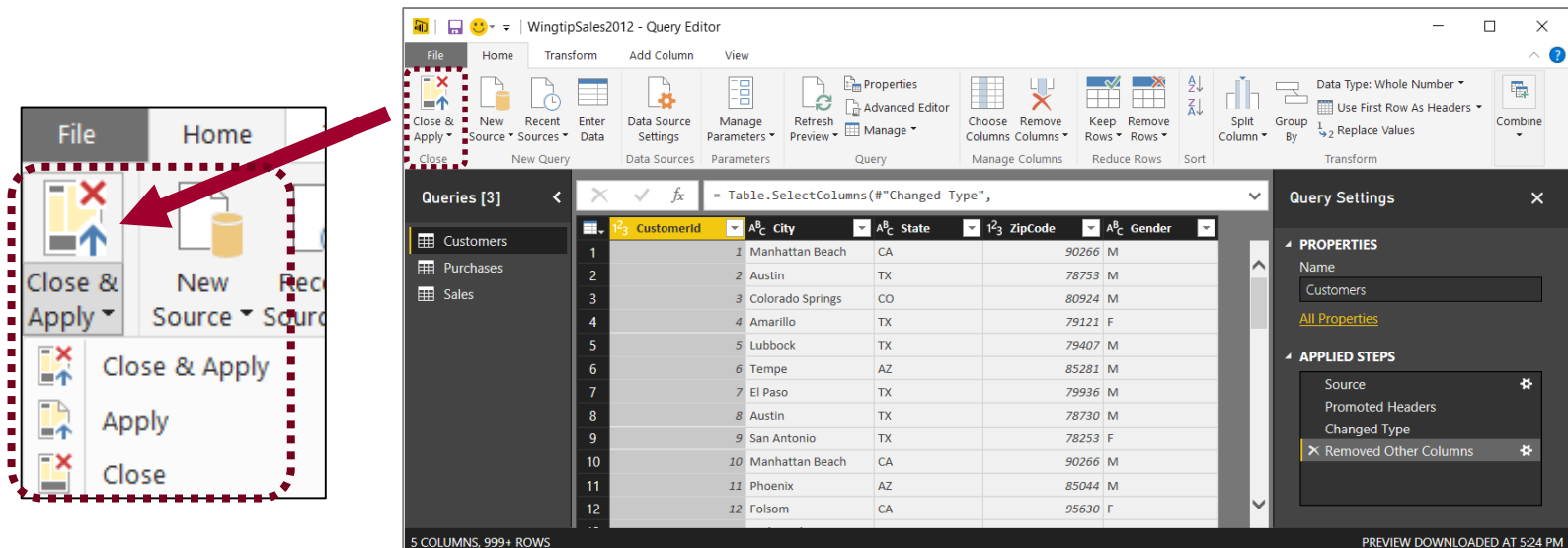
Designing Queries to Build a Star Schema

- 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

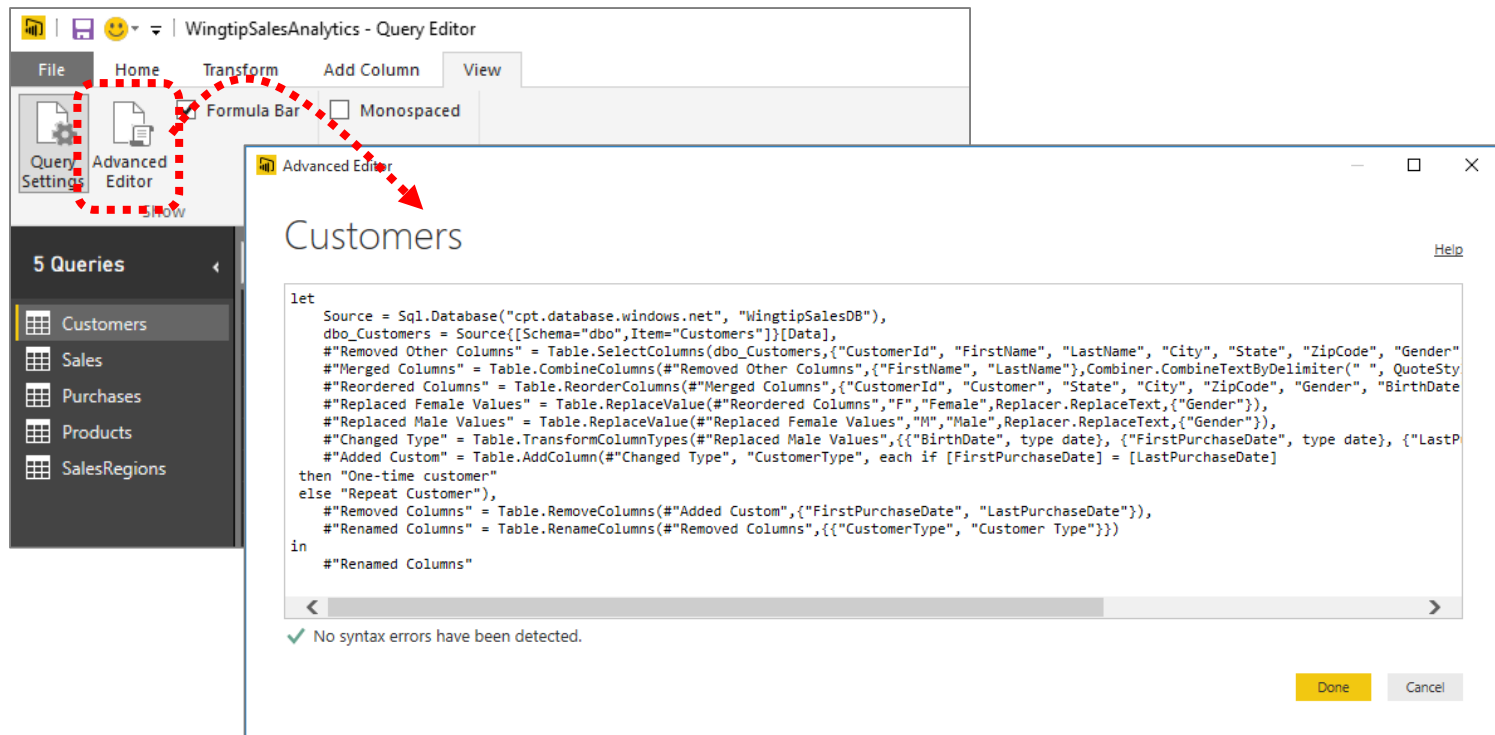
- A query is created as a sequence of steps
 - Each step is a parameterized operation on the data
 - Each step has formula which can be viewed/edited in formula bar
 - Query starts with Source step to extract data from a data source
 - Additional steps added to perform transform operations on data
 -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

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

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

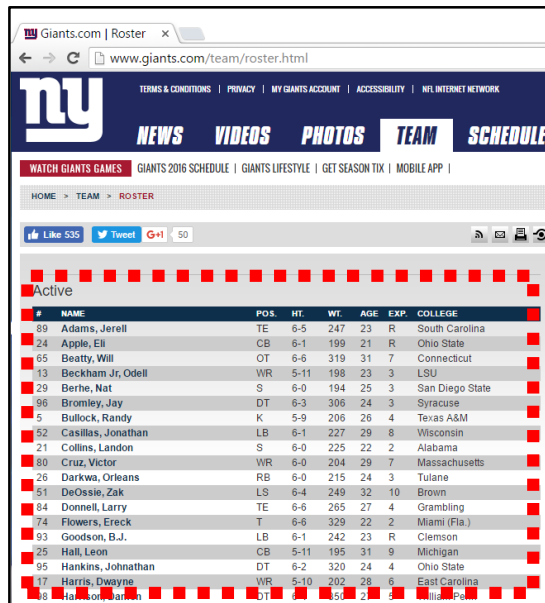
Advanced Editor

- 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

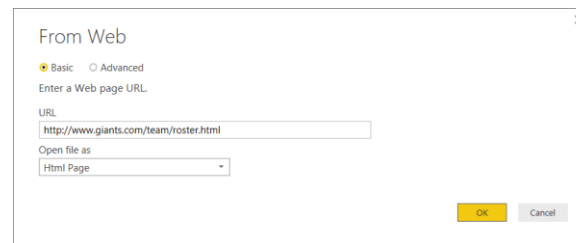


Working with Web Data Sources

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



#	NAME	POS.	HT.	WT.	AGE	EXP.	COLLEGE
89	Adams, Jerrell	TE	6-5	247	23	R	South Carolina
24	Apple, Eli	CB	6-1	199	21	R	Ohio State
65	Beatty, Will	OT	6-6	319	31	7	Connecticut
13	Beckham Jr, Odell	WR	5-11	198	23	3	LSU
29	Berhe, Nat	S	6-0	194	25	3	San Diego State
96	Bromley, Jay	DT	6-3	306	24	3	Syracuse
5	Bullock, Randy	K	5-9	206	26	4	Texas A&M
52	Casillas, Jonathan	LB	6-1	227	29	8	Wisconsin
21	Collins, Landon	S	6-0	225	22	2	Alabama
80	Cruz, Victor	WR	6-0	204	29	7	Massachusetts
26	Dar kwa, Orleans	RB	6-0	215	24	3	Tulane
51	DeOssie, Zak	LS	6-4	249	32	10	Brown
84	Donnell, Larry	TE	6-6	265	27	4	Grambling
74	Flowers, Ereck	T	6-6	329	22	2	Miami (Fla.)
93	Goodson, B.J.	LB	6-1	242	23	R	Clemson
25	Hall, Leon	CB	5-11	195	31	9	Michigan
55	Hankins, Johnathan	DT	6-2	320	24	4	Ohio State
17	Harris, Davyne	WR	5-10	202	28	6	East Carolina
16	Harris, Jamari	DT	6-4	255	22	5	Illinois



From Web

Basic Advanced

Enter a Web page URL.

URL

Open file as

OK Cancel

Query Input

	#	Name	Pos.	HT.	WT.	Age	Exp.	College
1	89	Adams, Jerrell	TE	6-5	247	23	R	South Carolina
2	24	Apple, Eli	CB	6-1	199	21	R	Ohio State
3	65	Beatty, Will	OT	6-6	319	31	7	Connecticut
4	13	Beckham Jr, Odell	WR	5-11	198	23	3	LSU
5	29	Berhe, Nat	S	6-0	194	25	3	San Diego State
6	96	Bromley, Jay	DT	6-3	306	24	3	Syracuse

Query Output

	Number	Last Name	First Name	Weight	Height	Age	Experience	Position	Category	Side	College
1	89	Adams	Jerrell	247	77	23	0	Tight End	Backs and Receivers	Offense	South Carolina
2	84	Donnell	Larry	265	78	27	4	Tight End	Backs and Receivers	Offense	Grambling
3	45	Tye	Will	262	74	24	1	Tight End	Backs and Receivers	Offense	Stony Brook
4	24	Apple	Eli	199	73	21	0	Cornerback	Defensive Backs	Defense	Ohio State
5	25	Hall	Leon	195	71	31	9	Cornerback	Defensive Backs	Defense	Michigan
6	20	Jenkins	Janoris	198	70	27	5	Cornerback	Defensive Backs	Defense	North Alabama
7	41	Rodgers-Cromartie	Dominique	205	74	30	8	Cornerback	Defensive Backs	Defense	Tennessee State
8	65	Beatty	Will	319	78	31	7	Offensive Tackle	Offensive Line	Offense	Connecticut
9	13	Beckham Jr	Odell	198	71	23	3	Wide Receiver	Backs and Receivers	Offense	LSU
10	80	Cruz	Victor	204	72	29	7	Wide Receiver	Backs and Receivers	Offense	Massachusetts

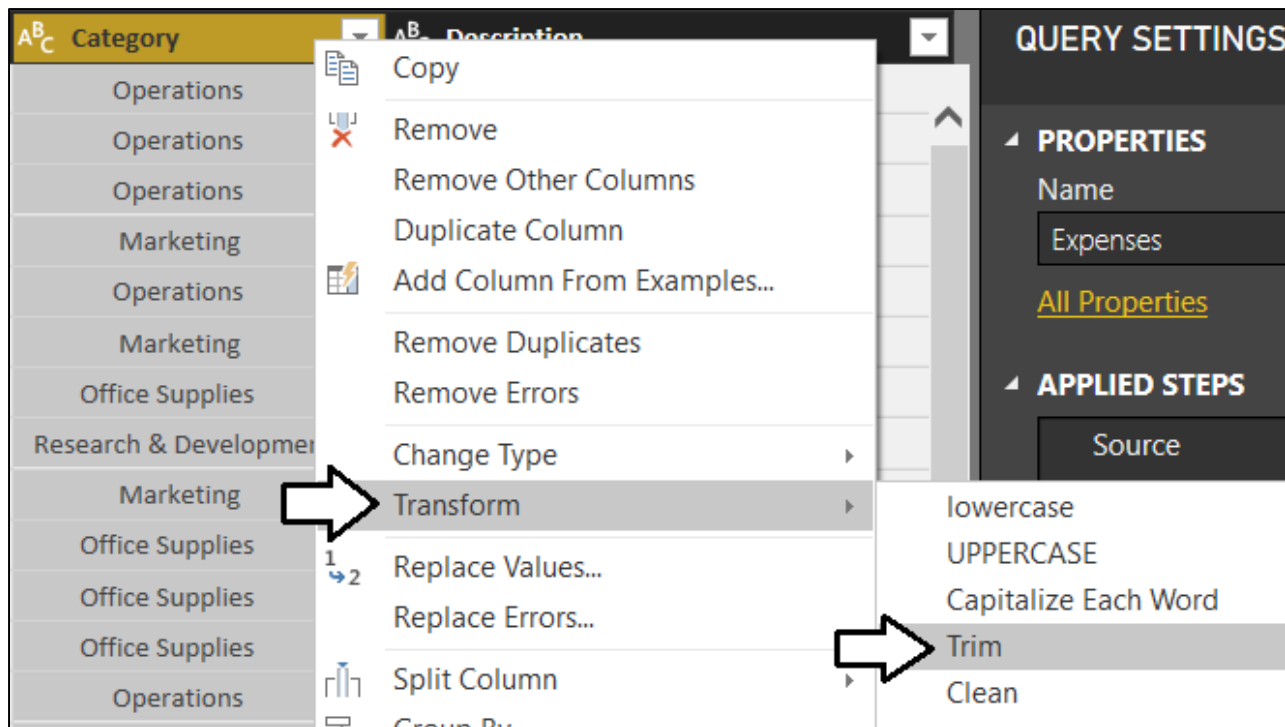
Examples of Basic Power BI Desktop Steps

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



Expanding Related Columns

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

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

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

Search Columns to Expand

(Select All Columns)

☐ InvoiceId

☒ InvoiceDate

☐ InvoiceAmount

☐ InvoiceType

☒ CustomerId

☐ Customers

☐ InvoiceDetails

☐ Use original column name as prefix

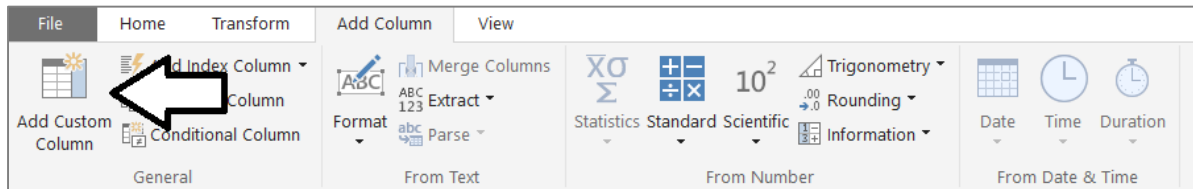
OK Cancel

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



Adding a Custom Column

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



Add Custom Column

New column name

Custom column formula:

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

Available columns:

- CustomerId
- Customer
- State
- City
- ZipCode
- Gender
- BirthDate

[Learn about Power BI Desktop formulas](#)

✓ No syntax errors have been detected.

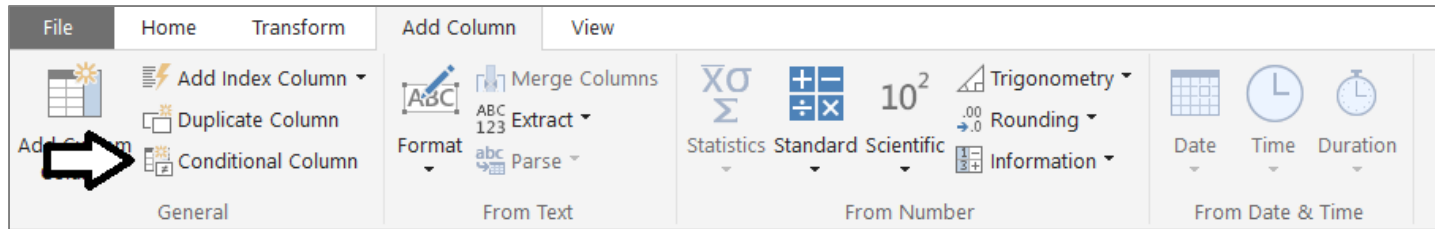
OK Cancel

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



Adding a Conditional Column

- Abstracts away need to write M code



Add Conditional Column [X]

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

New column name
Customer Type

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

+ Add Rule

Otherwise

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

OK Cancel



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



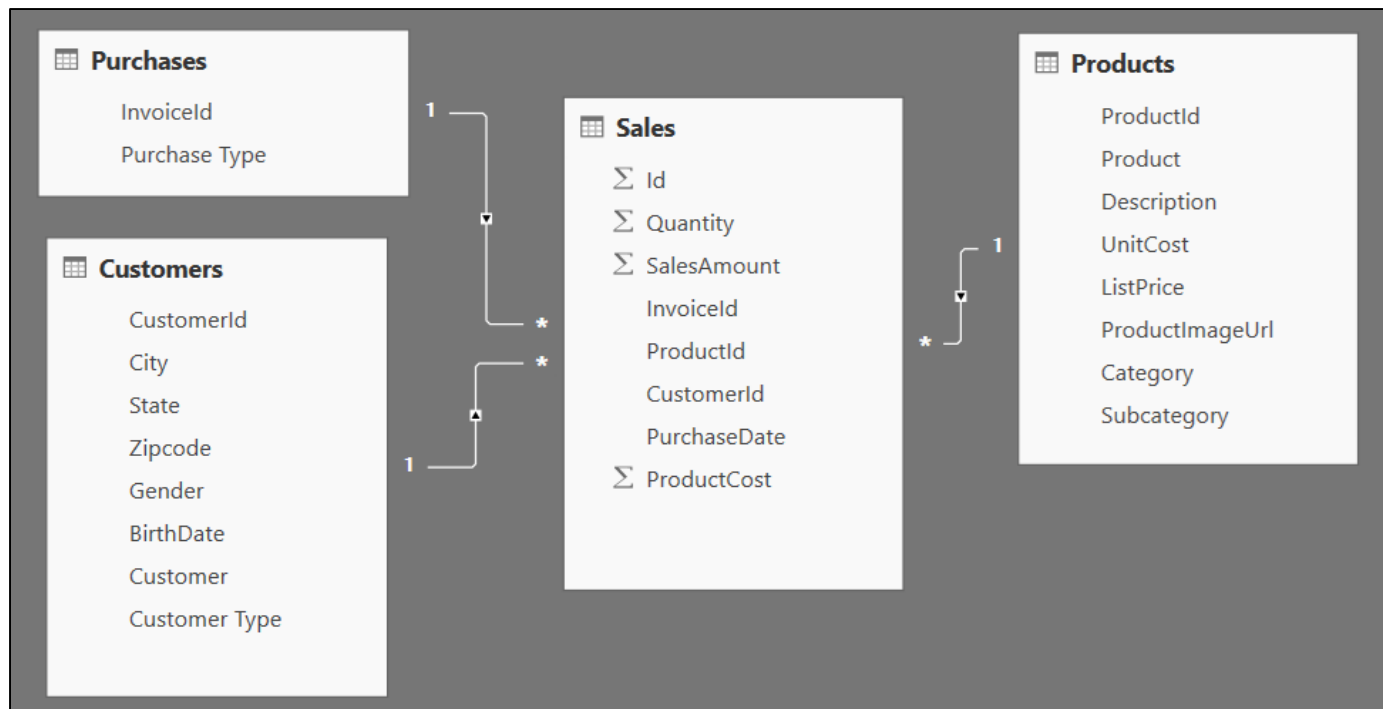
Agenda

- ✓ 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

Cardinality

Many to One (*:1)

Many to One (*:1)

One to One (1:1)

One to Many (1:*)

- Cross filter direction

Cross filter direction

Both

Single

Both

Edit Relationship

Select tables and columns that relate to one another.

Sales

Id	Quantity	SalesAmount	InvoiceId	ProductId	CustomerId	PurchaseDate	ProductCost
2899	100	100	1457	14	888	Thursday, June 21, 2012	\$8
3824	100	100	1901	14	1137	Saturday, July 21, 2012	\$8
3968	100	100	1969	14	1173	Wednesday, July 25, 2012	\$8

Customers

CustomerId	City	State	ZipCode	Gender	BirthDate	Customer	CustomerType
55	San Jose	CA	95110	Female	Thursday, March 10, 1949	Jewell Ryan	Repeat Customer
73	San Jose	CA	95123	Male	Thursday, May 9, 1985	Granville Perry	Repeat Customer
74	San Jose	CA	95122	Female	Tuesday, June 19, 1979	Sheri Mercado	Repeat Customer

Cardinality

Many to One (*:1)

Cross filter direction

Both

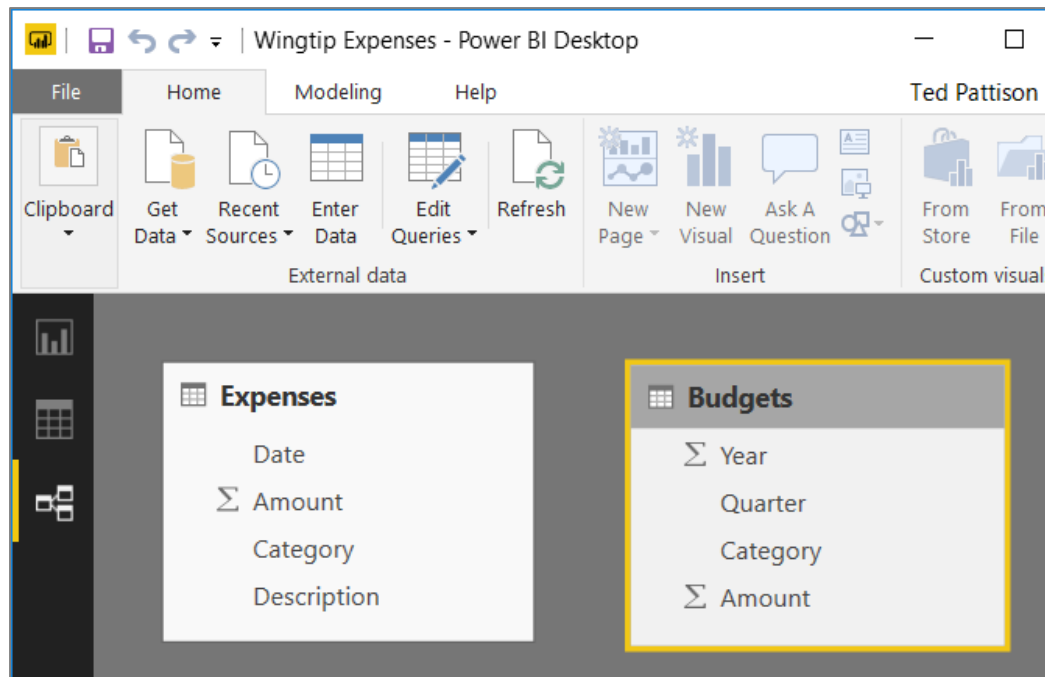
☒ Make this relationship active

OK Cancel



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

The screenshot shows the Power BI interface with a table view of Budgets. The formula bar at the top displays the formula: `Budget Key = [Year] & "-" & [Quarter] & "-" & [Category]`. The table has five columns: Year, Quarter, Category, Amount, and Budget Key. The Budget Key column contains values like "2017-Q1-Marketing". On the right, the FIELDS pane shows the Budgets table with fields Amount, Budget Key (highlighted), and Category.

Year	Quarter	Category	Amount	Budget Key
2017	Q1	Marketing	\$5,000	2017-Q1-Marketing
2017	Q1	Office Supplies	\$8,000	2017-Q1-Office Supplies
2017	Q1	Operations	\$8,000	2017-Q1-Operations
2017	Q1	Research & Development	\$5,000	2017-Q1-Research & Development
2017	Q2	Marketing	\$6,000	2017-Q2-Marketing
2017	Q2	Office Supplies	\$4,000	2017-Q2-Office Supplies
2017	Q2	Operations	\$7,000	2017-Q2-Operations

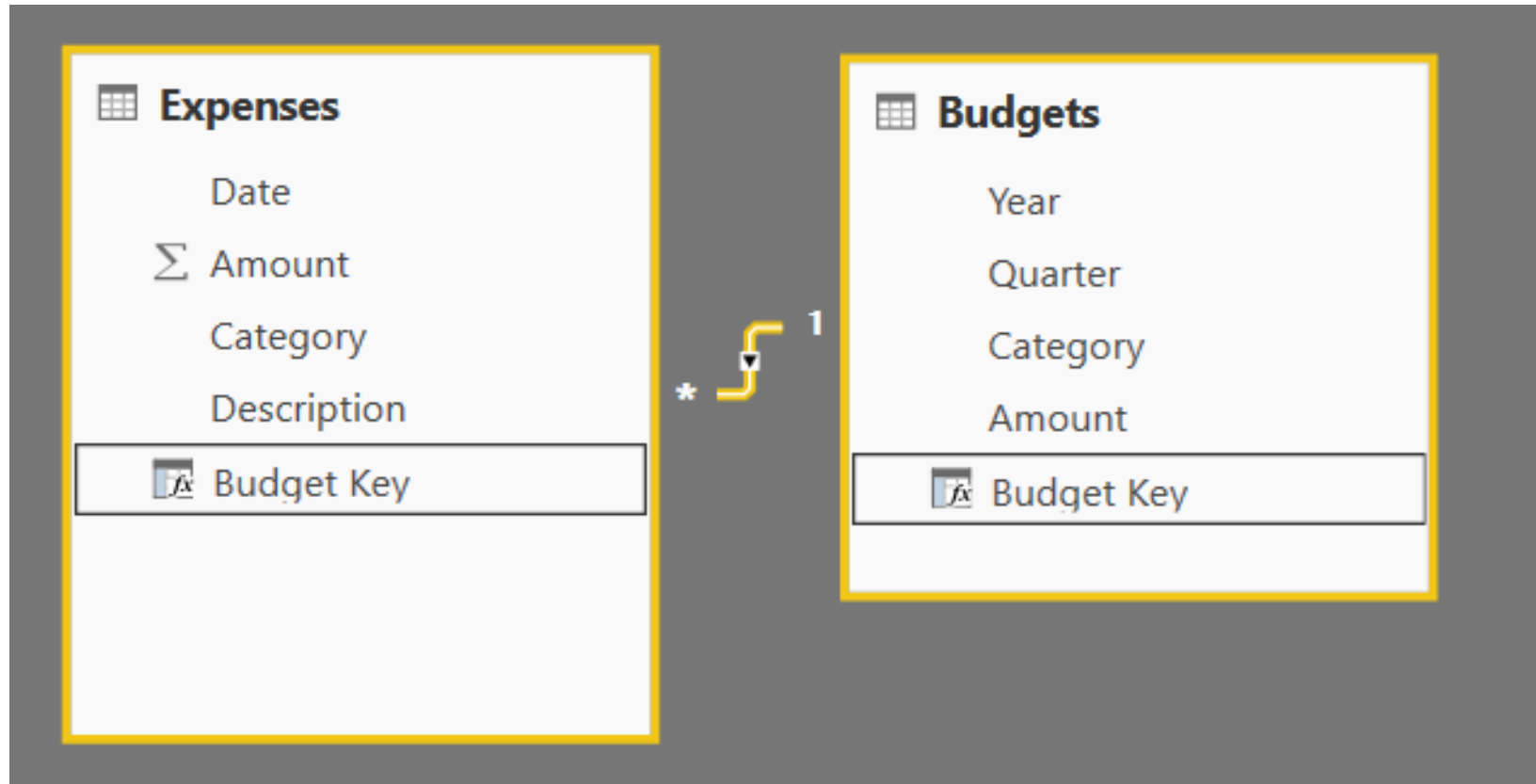
- Create composite key column in Expenses

The screenshot shows the Power BI interface with a table view of Expenses. The formula bar at the top displays the formula: `Budget Key = VAR BudgetYear = YEAR([Date]) VAR BudgetMonth = "Q" & FORMAT([Date], "q") RETURN BudgetYear & "-" & BudgetMonth & "-" & [Category]`. The table has five columns: Date, Amount, Category, Description, and Budget Key. The Budget Key column contains values like "2017-Q2-Operations". On the right, the FIELDS pane shows the Expenses table with fields Amount, Budget Key (highlighted), and Category.

Date	Amount	Category	Description	Budget Key
Sunday, April 2, 2017	\$925	Operations	Verizon - Telephone and Internet	2017-Q2-Operations
Monday, April 3, 2017	\$142	Office Supplies	Postage Stamps	2017-Q2-Office Supplies
Wednesday, April 5, 2017	\$294	Operations	Electricity Bill	2017-Q2-Operations
Wednesday, April 5, 2017	\$120.25	Office Supplies	Coffee Supplies	2017-Q2-Office Supplies
Thursday, April 13, 2017	\$1,200	Operations	Cleaning Service	2017-Q2-Operations



Create Relationship Using Composite Keys



Working with DAX

- DAX is the language used to create data models
 - DAX stands for "Data Analysis Expression 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>`



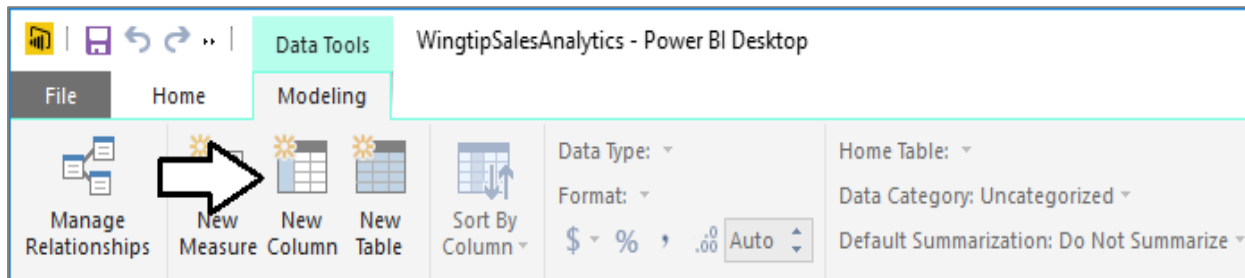
When to Create Calculated Columns

- 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

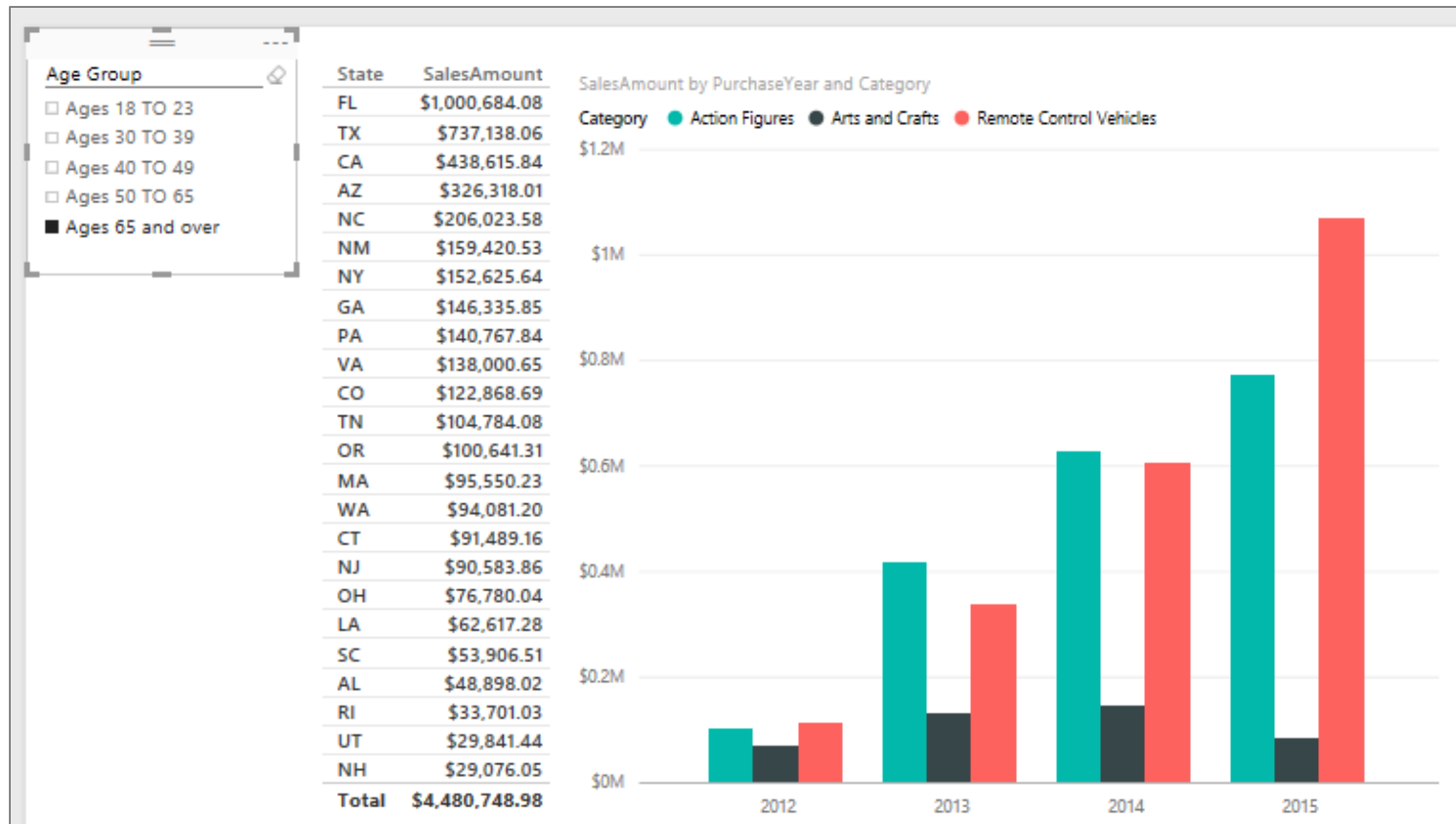


PurchaseYear = YEAR(Sales[PurchaseDate])									
Id	Quantity	SalesAmount	InvoiceId	ProductId	CustomerId	PurchaseDate	ProductCost	SalesProfit	PurchaseYear
2899	100	\$100.00	1457	14	888	6/21/12	\$8.00	\$92.00	2012
3824	100	\$100.00	1901	14	1137	7/21/12	\$8.00	\$92.00	2012
3968	100	\$100.00	1969	14	1173	7/25/12	\$8.00	\$92.00	2012
4008	100	\$100.00	1987	14	1186	7/26/12	\$8.00	\$92.00	2012
4224	100	\$100.00	2096	14	1239	8/3/12	\$8.00	\$92.00	2012
4724	100	\$100.00	2352	14	1390	8/19/12	\$8.00	\$92.00	2012



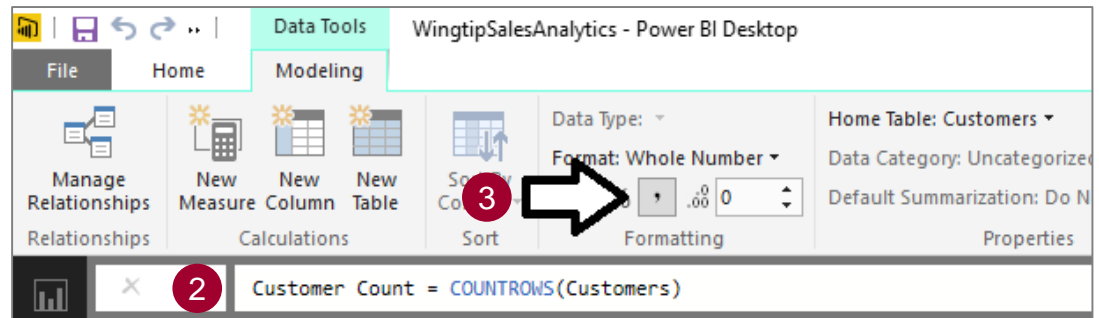
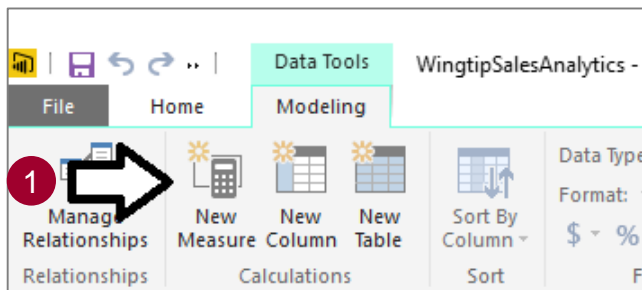
Calculated Column used in a Slicer

- Calculated column can populate slicer values



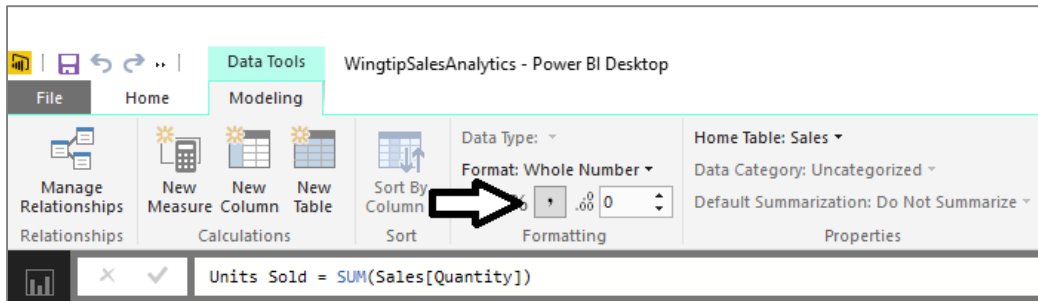
Creating Measures

- 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

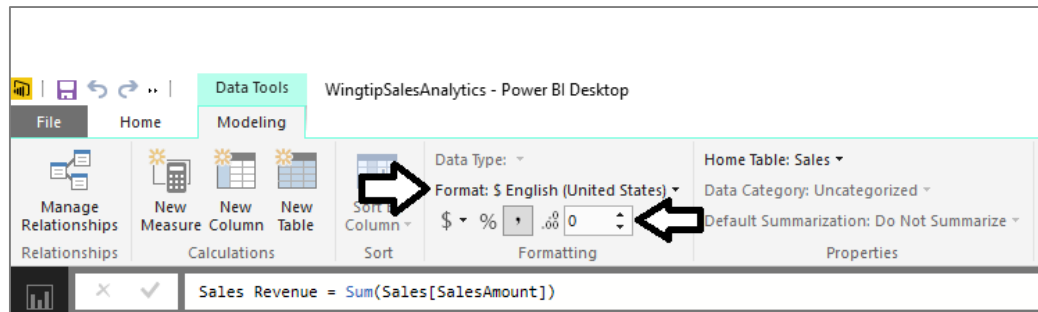


Formatting Measures

- Format as whole number



- Format as currency

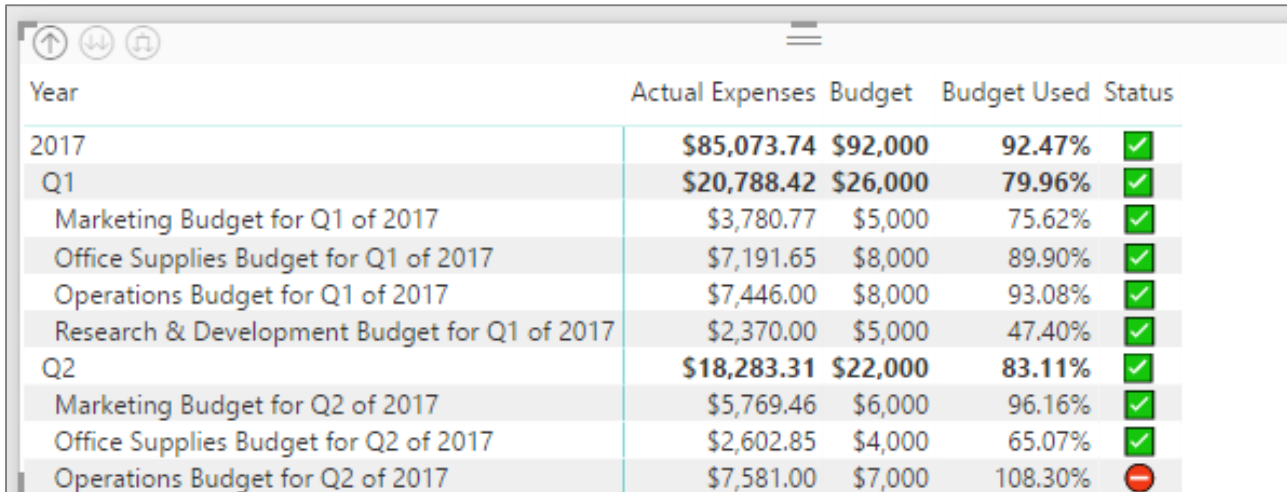


Working with UNICHAR Characters

- 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

Unichar Browser - Power BI Desktop

File Home View Modeling Help

Clipboard: Paste, Cut, Copy, Format Painter

External data: Get Data, Recent Sources, Enter Data, Edit Queries, Refresh

Insert: New Page, New Visual, Ask A Question, Text box, Image, Shapes

Custom visuals: From Store, From File

Themes: Switch Theme

Relationships: Manage Relationships

Calculations: New Measure, New Column, New Quick Measure

Share: Publish

Ted Pattison

8000	8200	8400	8600	8800	9000	9200	9400	9600	9800	10000	10200	10400	10600	10800	11000	11200	11400	11600	11800
8100	8300	8500	8700	8900	9100	9300	9500	9700	9900	10100	10300	10500	10700	10900	11100	11300	11500	11700	11900

9900 9901 9902 9903 9904 9905 9906 9907 9908 9909 9910 9911 9912 9913 9914 9915

9916 9917 9918 9919 9920 9921 9922 9923 9924 9925 9926 9927 9928 9929 9930 9931

9932 9933 9934 9935 9936 9937 9938 9939 9940 9941 9942 9943 9944 9945 9946 9947

9948 9949 9950 9951 9952 9953 9954 9955 9956 9957 9958 9959 9960 9961 9962 9963

9964 9965 9966 9967 9968 9969 9970 9971 9972 9973 9974 9975 9976 9977 9978 9979

9980 9981 9982 9983 9984 9985 9986 9987 9988 9989 9990 9991 9992 9993 9994 9995

9996 9997 9998 9999

UNICHAR Browser

PAGE 1 OF 1

VISUALIZATIONS

Charts

Page

Symbol

Symbol Layout

Value

Values

Drag data fields here

FILTERS

Page level filters

Drag data fields here

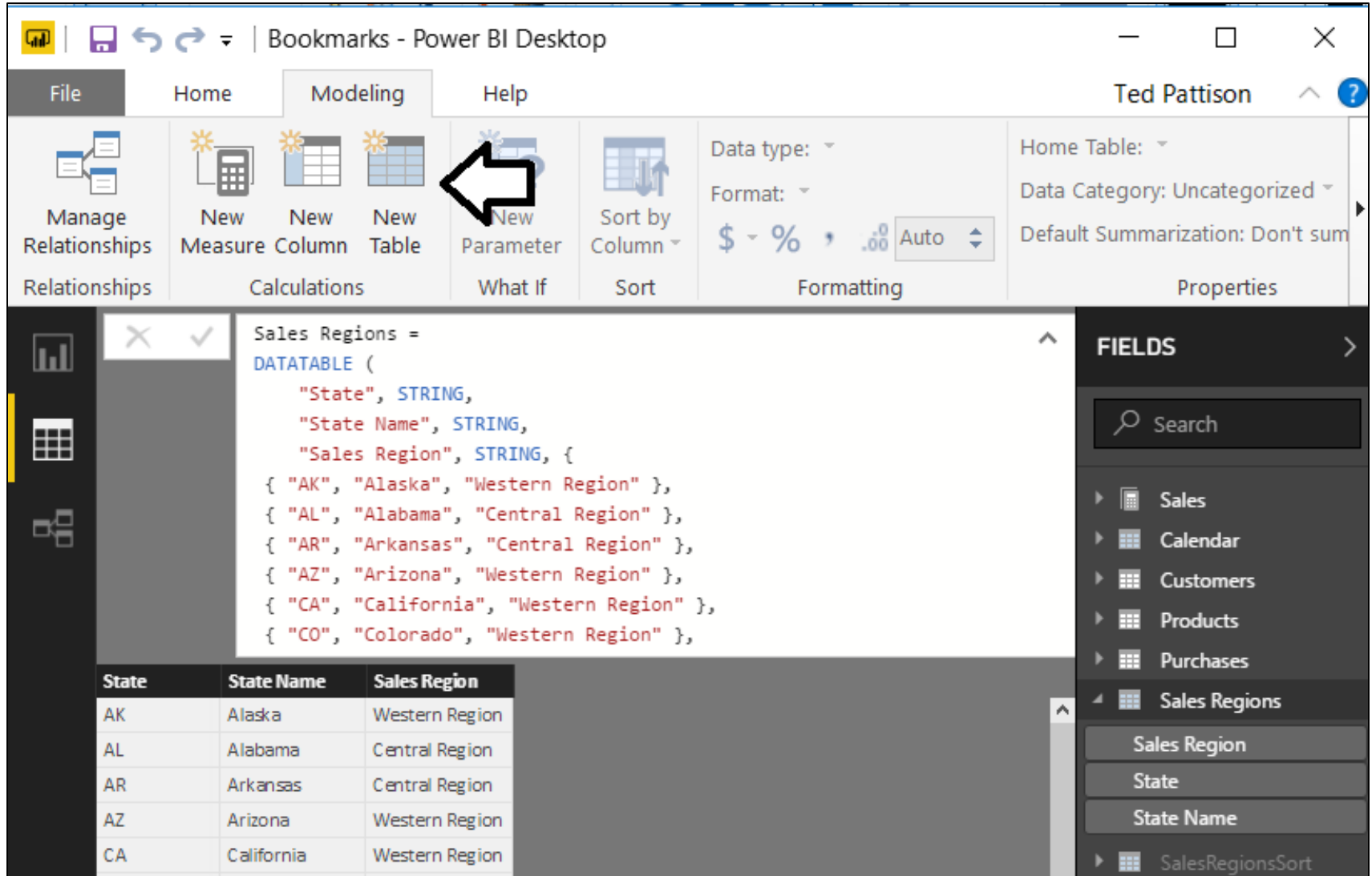
Drillthrough filters

Drag drillthrough fields here

Report level filters

Drag data fields here

Creating Tables Dynamically using DAX



The screenshot shows the Power BI Desktop interface with the 'Modeling' tab selected. A large black arrow points to the 'New Table' button in the ribbon. Below the ribbon, the DAX formula for the 'Sales Regions' table is displayed in the formula bar:

```
Sales Regions =  
DATATABLE (  
    "State", STRING,  
    "State Name", STRING,  
    "Sales Region", STRING, {  
        { "AK", "Alaska", "Western Region" },  
        { "AL", "Alabama", "Central Region" },  
        { "AR", "Arkansas", "Central Region" },  
        { "AZ", "Arizona", "Western Region" },  
        { "CA", "California", "Western Region" },  
        { "CO", "Colorado", "Western Region" },  
    }  
)
```

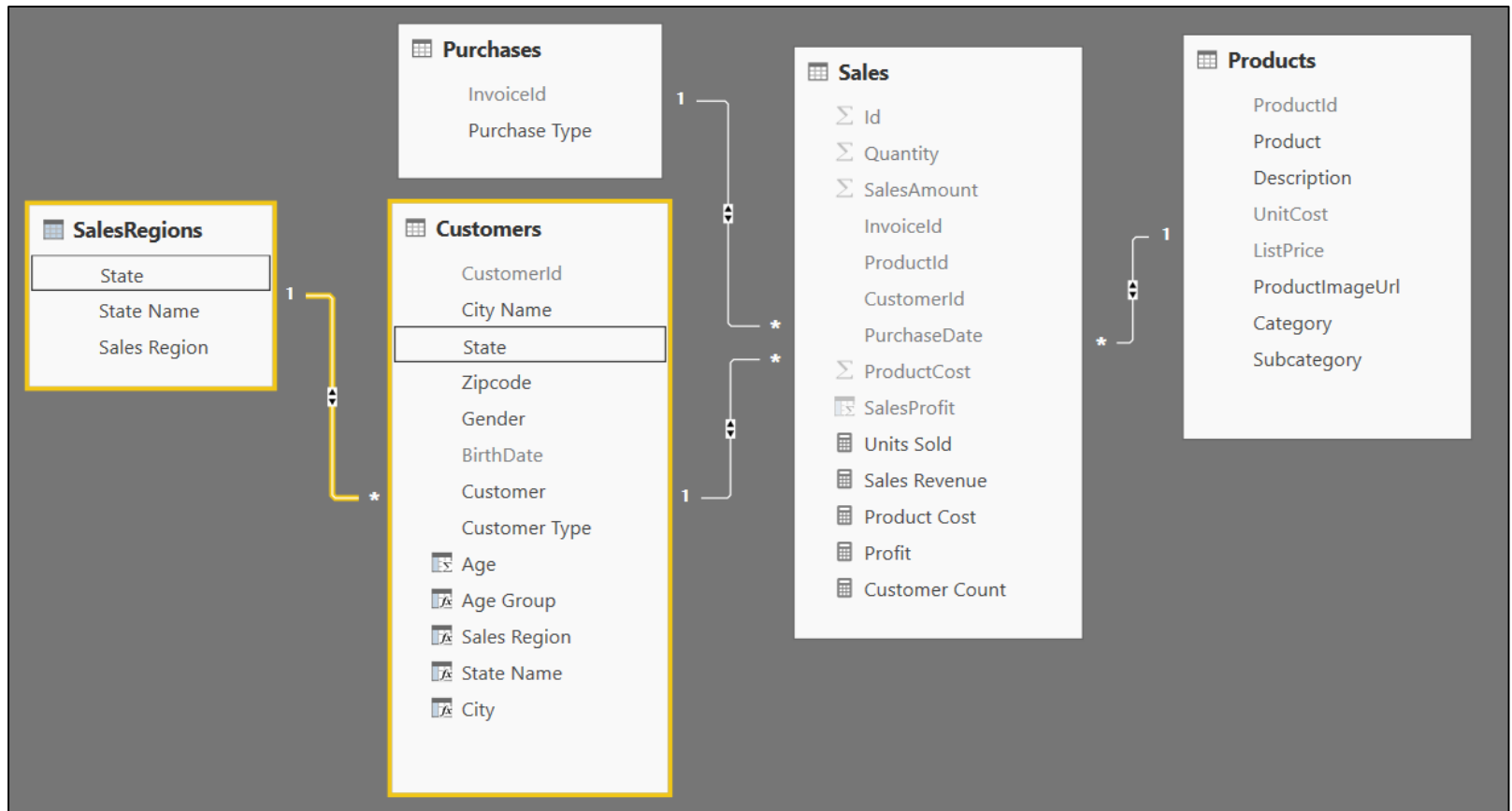
Below the formula bar, a preview of the 'Sales Regions' table is shown:

State	State Name	Sales Region
AK	Alaska	Western Region
AL	Alabama	Central Region
AR	Arkansas	Central Region
AZ	Arizona	Western Region
CA	California	Western Region

On the right side of the interface, the 'FIELDS' pane is visible, showing a list of tables: Sales, Calendar, Customers, Products, Purchases, and Sales Regions. The 'Sales Regions' table is selected, and its columns (Sales Region, State, State Name) are listed below it.

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

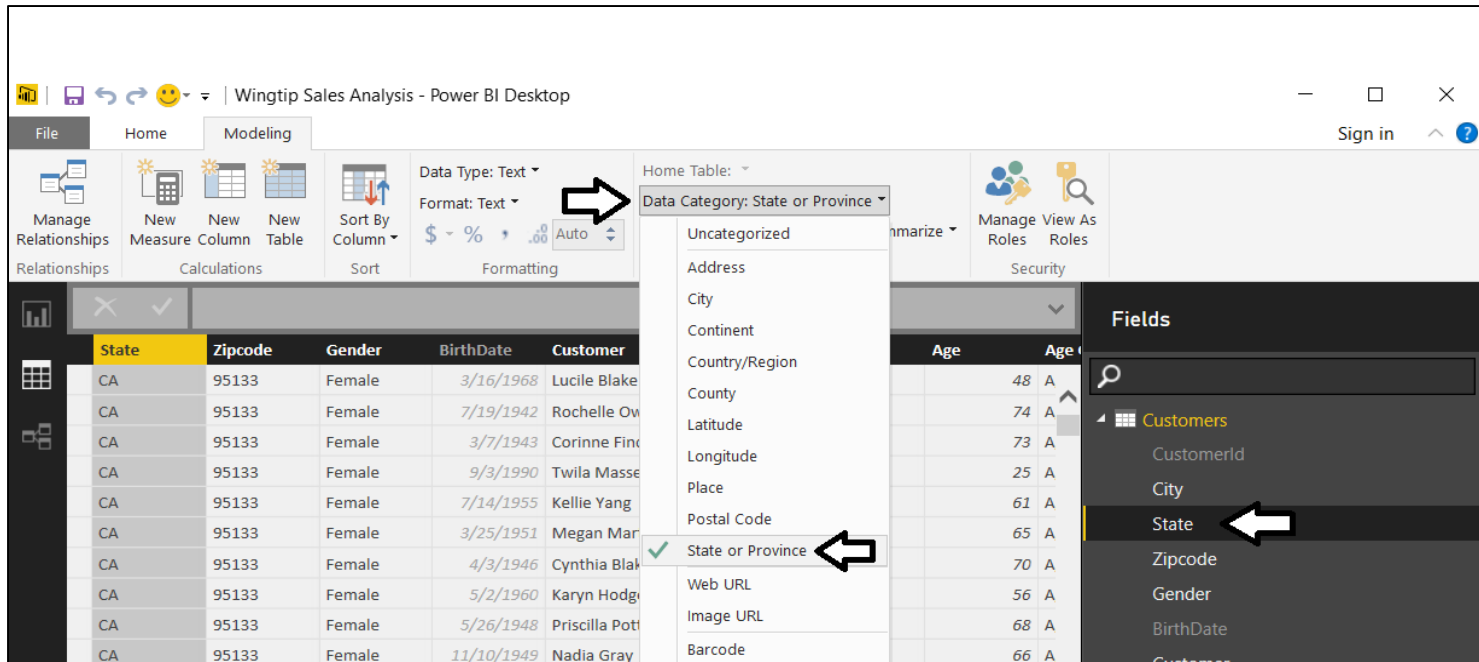
Sales Region = RELATED(SalesRegions[SalesRegion])										
CustomerId	City	State	ZipCode	Gender	BirthDate	Customer	CustomerType	Age	Age Group	Sales Region
55	San Jose	CA	95110	Female	3/10/49	Jewell Ryan	Repeat Customer	66	Ages 65 and over	Western Region
73	San Jose	CA	95123	Male	5/9/85	Granville Perry	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

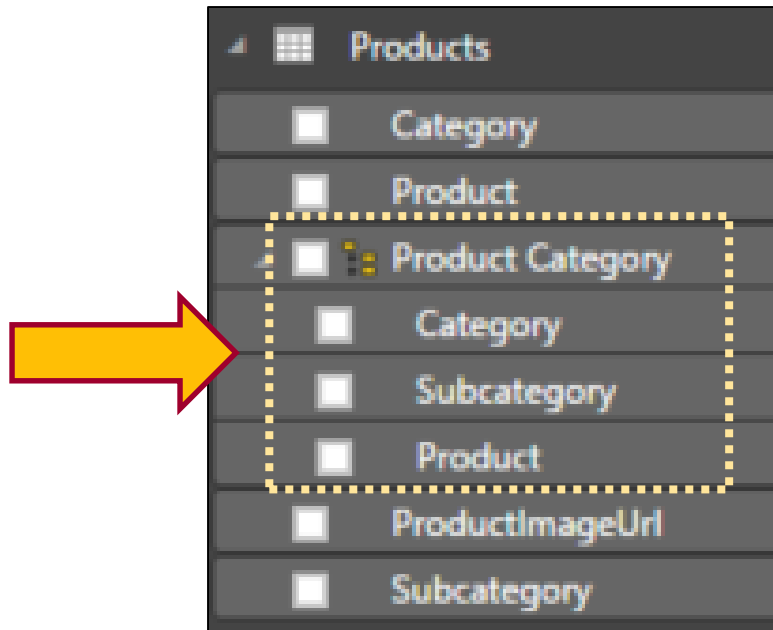
- City name alone is ambiguous
 - "Athens" defaults to Greece not Georgia
 - Concatenate city name with state to disambiguate

 		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

Sales Region = RELATED(SalesRegions[SalesRegion])					
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
Twila Massey	One-time Customer	25	Ages 18 TO 23	Western Region	California

- Then create hierarchy in the table with all the columns

Customer Geography
Sales Region
State
City
Zipcode



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

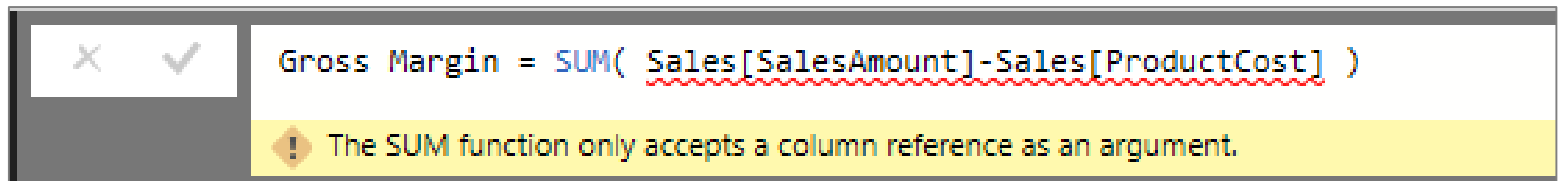
✕	✓	City = [City Name] & ", " & [State]			
	Age Group	Sales Region	State Name	SalesRegionSort	City
48	Ages 40 TO 49	Western Region	California	1	San Jose, CA
74	Ages 65 and over	Western Region	California	1	San Jose, CA
73	Ages 65 and over	Western Region	California	1	San Jose, CA
25	Ages 18 TO 23	Western 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

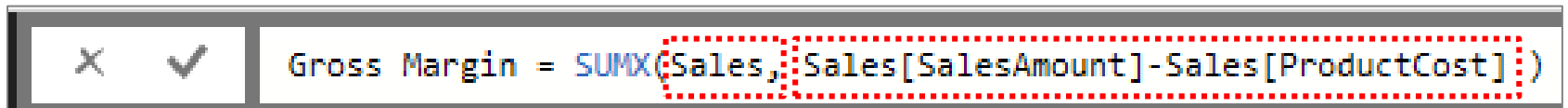


Understanding Iterators Like SUMX

- 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



- Iterator functions (e.g. SUMX) iterate through rows in target table



- 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	\$613,971	\$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

	Month in Year	2012	2013	2014	2015	Total
Sales Region	January	\$425	\$50,169	\$61,295	\$76,614	\$188,503
<input type="checkbox"/> Select All	February	\$13,891	\$40,133	\$63,670	\$101,542	\$219,236
<input type="checkbox"/> Central Region	March	\$19,121	\$58,411	\$73,839	\$84,180	\$235,551
<input type="checkbox"/> Eastern Region	April	\$19,128	\$53,711	\$67,919	\$91,762	\$232,520
<input checked="" type="checkbox"/> Western Region	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
Customer Type	September	\$39,415	\$68,653	\$82,697	\$94,805	\$285,570
<input type="checkbox"/> One-time customer	October	\$51,994	\$69,122	\$99,344	\$84,177	\$304,637
<input checked="" type="checkbox"/> Repeat Customer	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

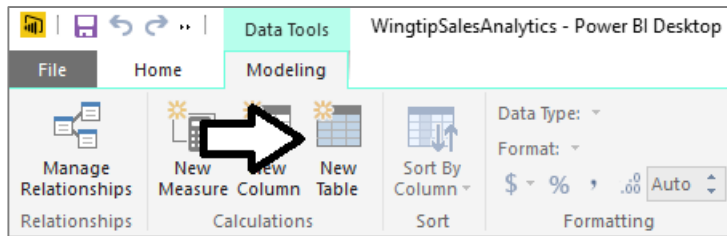
```
Pct of All Products =  
DIVIDE(  
    SUM( Sales[SalesAmount] ),  
    CALCULATE(  
        Sum (Sales[SalesAmount] ),  
        ALL(Products[Category], Products[Subcategory], Products[Product])  
    )  
)
```

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

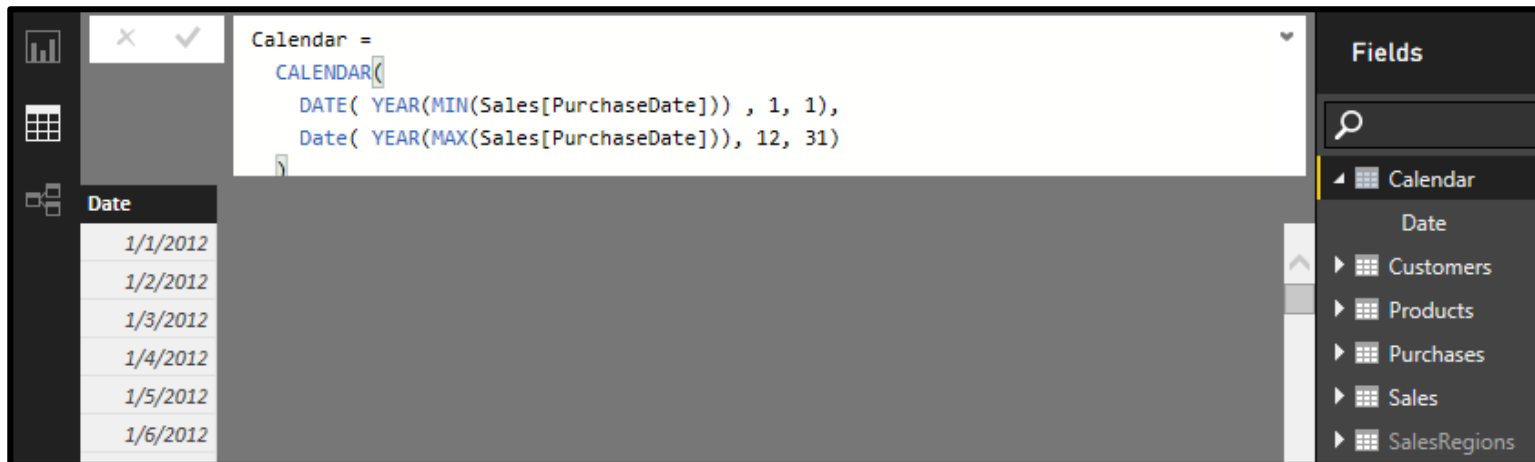


Creating Calendar Table as Calculated Table

- Use **New Table** command in ribbon



- Create calendar table using DAX **CALENDAR** function



Adding Columns to Calendar Table

- Creating the **Year** column

X ✓ Year = YEAR('Calendar'[Date])	
Date	Year
1/1/2012	2012
1/2/2012	2012
1/3/2012	2012

- Creating the **Quarter** column

X ✓ 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

X ✓ 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

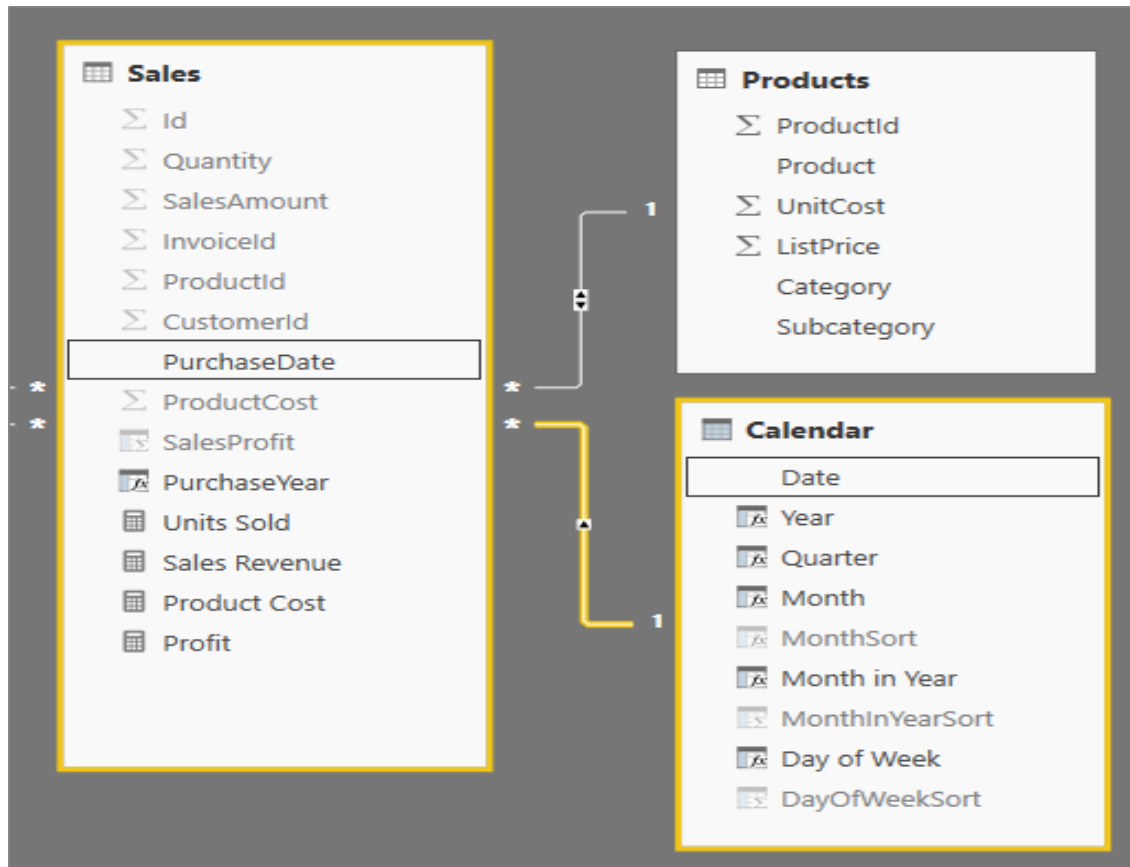
- Configure **Month** column with **MonthSort** as sort column

The screenshot shows the Power BI Desktop interface. On the left, the 'Model View' is visible with columns 'Date', 'Year', 'Quarter', 'Month', and 'MonthSort'. The 'Month' column is highlighted. In the center, the 'Sort By Column' dropdown menu is open, showing 'Month (Default)' and 'MonthSort'. The 'MonthSort' option is selected, indicated by a green checkmark. On the right, the 'Fields View' is visible, showing the 'Month' and 'MonthSort' columns. The 'MonthSort' column is highlighted. The formula bar at the top shows the DAX formula: `MonthSort = FORMAT('Calendar'[Date], "yyyy-MM")`. Arrows indicate the flow from the 'Month' column in the Model View to the 'Sort By Column' dropdown and then to the 'MonthSort' column in the Fields View.



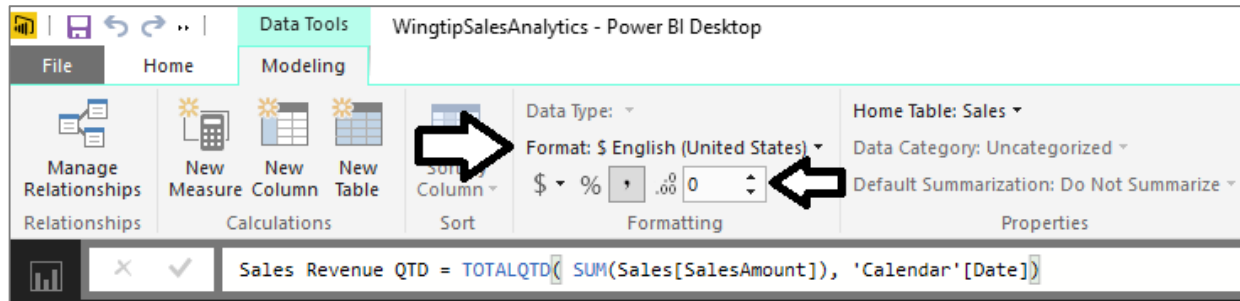
Integrating Calendar Table into Data Model

- Calendar table needs relationship to one or more tables

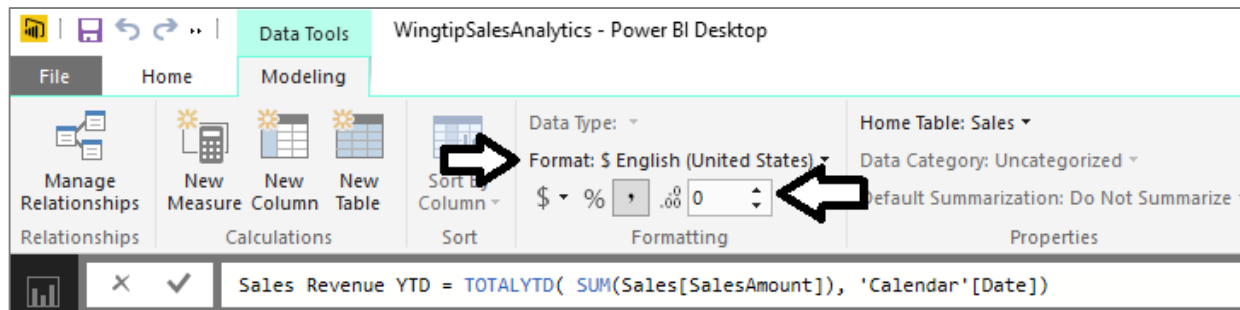


Calculated Fields for QTD and YTD Sales

- TOTALQTD function calculates quarter-to-date totals

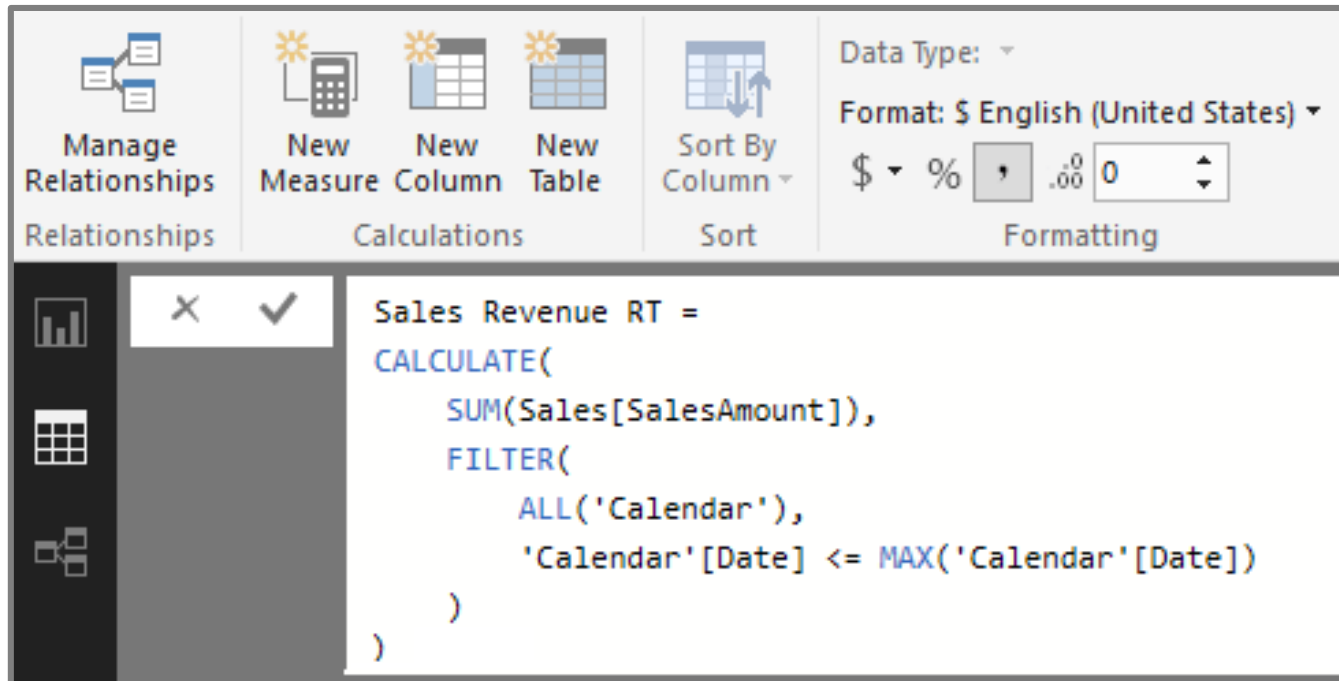


- TOTALYTD function calculates year-to-date totals



Creating Running Total using CALCULATE

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



Sales 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



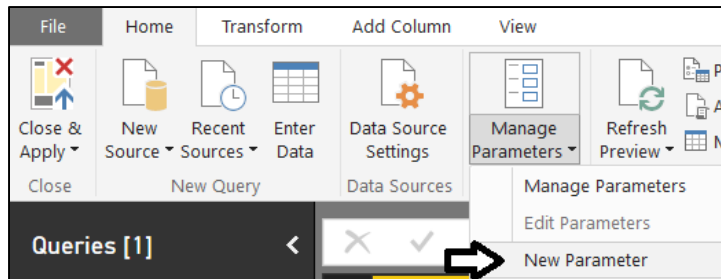
Query Parameters

- 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

Parameters

New

Customer State

Name

Customer State

Description

This parameter is used in the Customers query to filter the customer rows which are loaded into the dataset for the Power BI Desktop project.

☒ Required

Type

Text

Allowed Values

List of values

1	CA
2	OR
3	WA
4	AZ
5	TX
*	

Default Value

CA

Current Value

CA

OK Cancel

Referencing Parameters in a Query

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

The screenshot shows a 'Filter Rows' dialog box with a close button (X) in the top right corner. It has two tabs: 'Basic' (unselected) and 'Advanced' (selected). Below the tabs, it says 'Show rows where:'. There are two columns: 'And/Or' and 'Column'. The 'And/Or' column has a dropdown menu with 'And' selected. The 'Column' column has a dropdown menu with 'State' selected. The 'Operator' column has a dropdown menu with 'equals' selected. The 'Value' column has a dropdown menu with 'Customer State' selected. There is a small icon of a table with a red row highlighted next to the 'Value' dropdown. Below the filter rule, there is a button labeled 'Add Clause'. At the bottom right, there are two buttons: 'OK' (yellow) and 'Cancel' (gray).

And/Or	Column	Operator	Value
	State	equals	Customer State
And	State		ABC

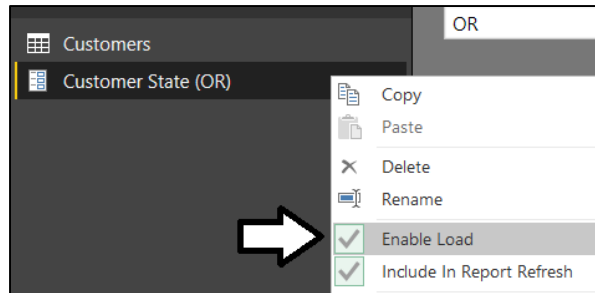
Add Clause

OK Cancel

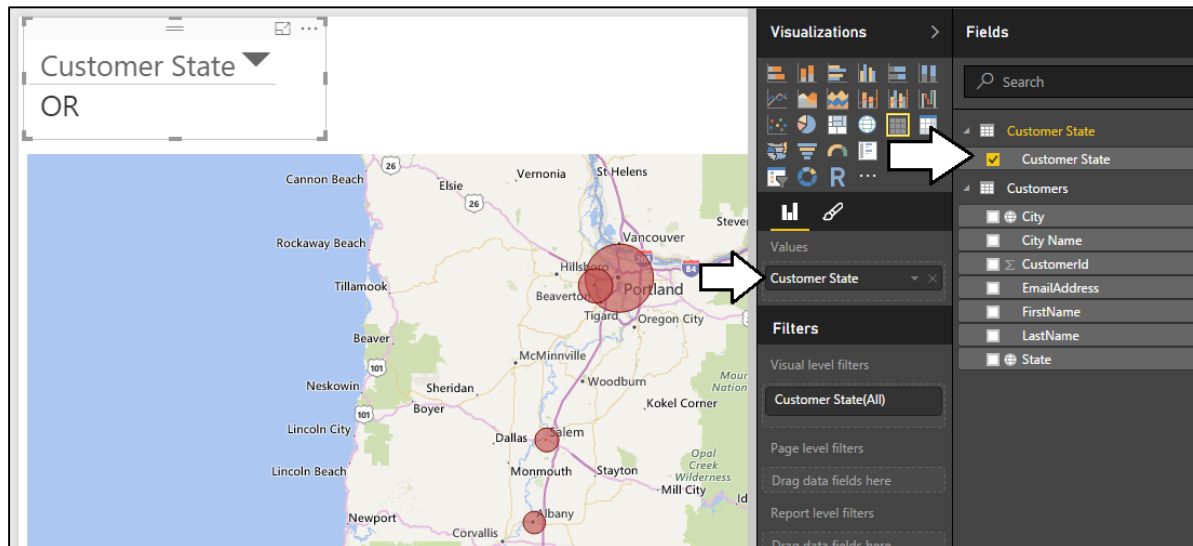


Making Parameters Available to Data Model

- Configure parameter's Enable Load setting



- Parameter becomes visible within fields list in report view



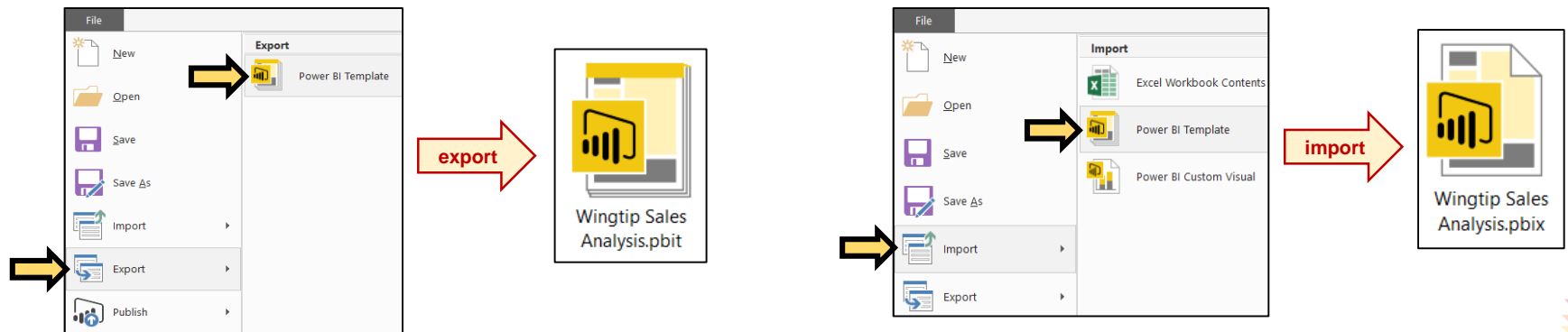
Agenda

- ✓ 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

