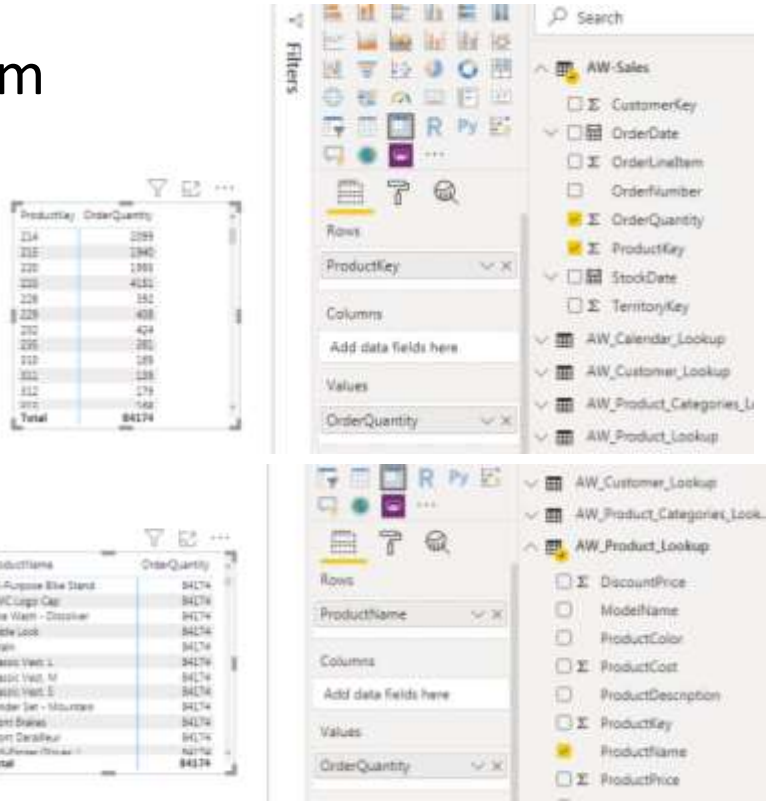
Demo

- Go to view → choose Matrix
- Choose sales table and drag (Order Quantity) to the values field and (Product Key) to the rows fields.
- Replace the (Product Keys) rows with (Product Name) from

(AW_product_lookup) table.



Note: We notice that we are limited to the scope of the sales table as there is no relationship with any Other table.

DATABASE NORMALIZATION

Normalization is the process of organizing the tables and columns in a relational database to reduce redundancy and preserve data integrity. It's commonly used to:

- **Eliminate redundant data** to decrease table sizes and improve processing speed & efficiency
- **Minimize errors and anomalies** from data modifications (inserting, updating or deleting records)
- **Simplify queries** and structure the database for meaningful analysis

TIP: In a normalized database, each table should serve a ***distinct*** and ***specific*** purpose (*i.e. product information, dates, transaction records, customer attributes, etc.*)

date	product_id	quantity	product_brand	product_name	product_sku	product_weight
1/1/1997	869	5	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/7/1997	869	2	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/3/1997	1	4	Washington	Washington Berry Juice	90748583674	8.39
1/1/1997	1472	3	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/6/1997	1472	2	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/5/1997	2	4	Washington	Washington Mango Drink	96516502499	7.42
1/1/1997	76	4	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/1/1997	76	2	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/5/1997	3	2	Washington	Washington Strawberry Drink	58427771925	13.1
1/7/1997	3	2	Washington	Washington Strawberry Drink	58427771925	13.1
1/1/1997	320	3	Excellent	Excellent Cranberry Juice	36570182442	16.4

*When you **don't** normalize, you end up with tables like this; all of the rows with duplicate product info could be eliminated with a lookup table based on **product_id***

This may not seem critical now, but minor inefficiencies can become major problems as databases scale in size!

DATA TABLES VS. LOOKUP TABLES

Models generally contain two types of tables: **data** (or "*fact*") tables, and **lookup** (or "*dimension*") tables

- **Data tables** contain *numbers* or *values*, typically at a granular level, with ID or "*key*" columns that can be used to create table relationships
- **Lookup tables** provide descriptive, often text-based *attributes* about each dimension in a table

date	product_id	quantity
1/1/1997	869	5
1/1/1997	1472	3
1/1/1997	76	4
1/1/1997	320	3
1/1/1997	4	4
1/1/1997	952	4
1/1/1997	1222	4
1/1/1997	517	4
1/1/1997	1359	4
1/1/1997	357	4
1/1/1997	1426	5
1/1/1997	190	4
1/1/1997	367	4
1/1/1997	250	5
1/1/1997	600	4
1/1/1997	702	5

*This **Data Table** contains "**quantity**" values, and connects to lookup tables via the "**date**" and "**product_id**" columns*

date	day_of_month	month	year	weekday	week_of_year	week_ending	month_name	quarter
1/1/1997	1	1	1997	Wednesday	1	1/5/1997	January	Q1
1/2/1997	2	1	1997	Thursday	1	1/5/1997	January	Q1
1/3/1997	3	1	1997	Friday	1	1/5/1997	January	Q1
1/4/1997	4	1	1997	Saturday	1	1/5/1997	January	Q1
1/5/1997	5	1	1997	Sunday	2	1/5/1997	January	Q1
1/6/1997	6	1	1997	Monday	2	1/12/1997	January	Q1

*This **Calendar Lookup** table provides additional attributes about each **date** (month, year, weekday, quarter, etc.)*

product_id	product_brand	product_name	product_sku	product_retail_price	product_cost	product_weight
1	Washington	Washington Berry Juice	90748583674	2.85	0.94	8.39
2	Washington	Washington Mango Drink	96516502499	0.74	0.26	7.42
3	Washington	Washington Strawberry Drink	58427771925	0.83	0.4	13.1
4	Washington	Washington Cream Soda	64412155747	3.64	1.64	10.6
5	Washington	Washington Diet Soda	85561191439	2.19	0.77	6.66
6	Washington	Washington Cola	29804642796	1.15	0.37	15.8
7	Washington	Washington Diet Cola	20191444754	2.61	0.91	18
8	Washington	Washington Orange Juice	89770532250	2.59	0.8	8.97

*This **Product Lookup** table provides additional attributes about each **product** (brand, product name, sku, price, etc.)*

PRIMARY VS. FOREIGN KEYS

date	product_id	quantity
1/1/1997	869	5
1/1/1997	1472	3
1/1/1997	76	4
1/1/1997	320	3
1/1/1997	4	4
1/1/1997	952	4
1/1/1997	1222	4
1/1/1997	517	4
1/1/1997	1359	4
1/1/1997	357	4
1/1/1997	1426	5
1/1/1997	190	4
1/1/1997	367	4
1/1/1997	250	5
1/1/1997	600	4
1/1/1997	702	5

These columns are **foreign keys**; they contain *multiple* instances of each value, and are used to match the **primary keys** in related lookup tables

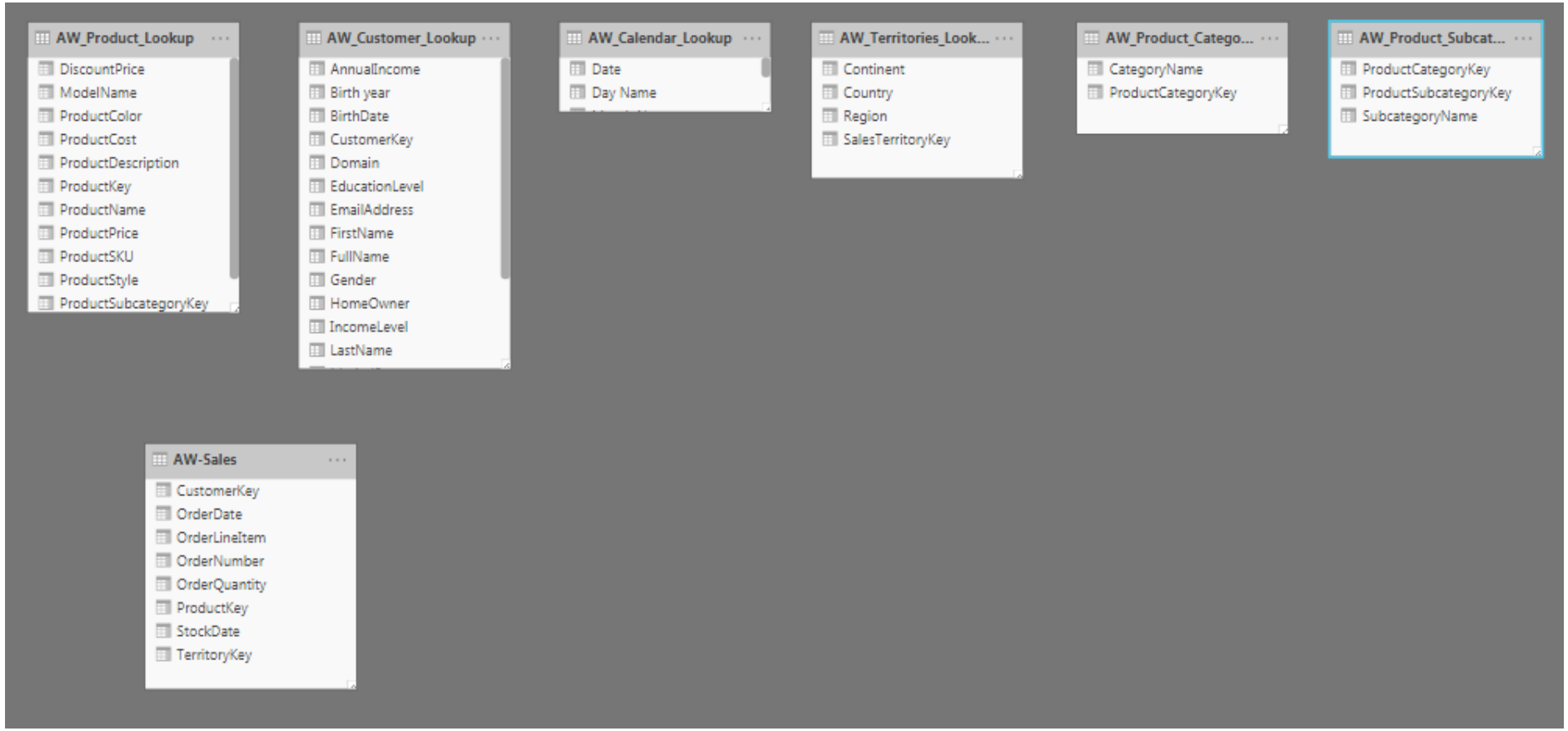
date	day_of_month	month	year	weekday	week_of_year	week_ending	month_name	quarter
1/1/1997	1	1	1997	Wednesday	1	1/5/1997	January	Q1
1/2/1997	2	1	1997	Thursday	1	1/5/1997	January	Q1
1/3/1997	3	1	1997	Friday	1	1/5/1997	January	Q1
1/4/1997	4	1	1997	Saturday	1	1/5/1997	January	Q1
1/5/1997	5	1	1997	Sunday	2	1/5/1997	January	Q1
1/6/1997	6	1	1997	Monday	2	1/12/1997	January	Q1

product_id	product_brand	product_name	product_sku	product_retail_price	product_cost	product_weight
1	Washington	Washington Berry Juice	90748583674	2.85	0.94	8.39
2	Washington	Washington Mango Drink	96516502499	0.74	0.26	7.42
3	Washington	Washington Strawberry Drink	58427771925	0.83	0.4	13.1
4	Washington	Washington Cream Soda	64412155747	3.64	1.64	10.6
5	Washington	Washington Diet Soda	85561191439	2.19	0.77	6.66
6	Washington	Washington Cola	29804642796	1.15	0.37	15.8
7	Washington	Washington Diet Cola	20191444754	2.61	0.91	18
8	Washington	Washington Orange Juice	89770532250	2.59	0.8	8.97

These columns are **primary keys**; they *uniquely* identify each row of a table, and match the **foreign keys** in related data tables

Demo

- Let’s divide our tables in power BI model tab to lookup and data table.



RELATIONSHIPS VS. MERGED TABLES

*Can't I just **merge queries** or use **LOOKUP** or **RELATED** functions to pull those attributes into the fact table itself, so that I have everything in one place??*

-Anonymous confused man

*Original **Fact Table** fields* | *Attributes from **Calendar Lookup** table* | *Attributes from **Product Lookup** table*

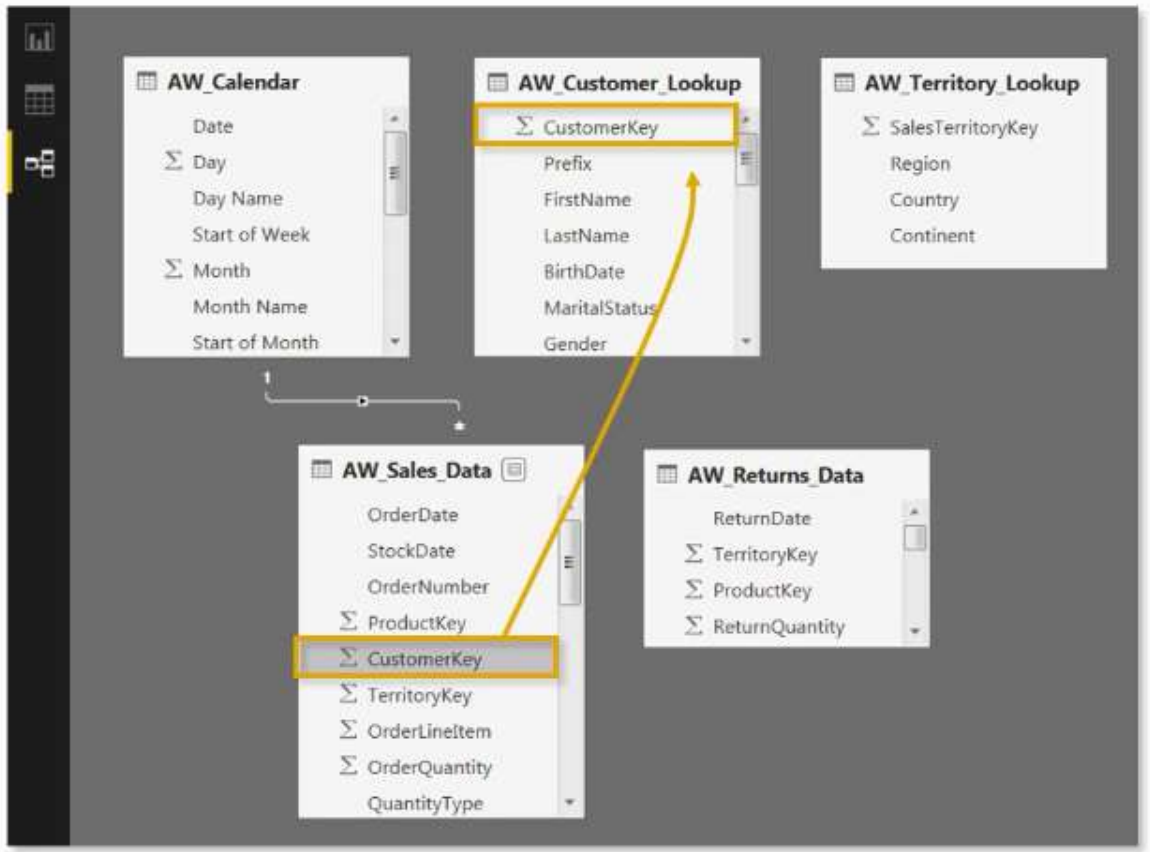
date	product_id	quantity	day_of_month	month	year	weekday	month_name	quarter	product_brand	product_name	product_sku	product_weight
1/1/1997	869	5	1	1	1997	Wednesday	January	Q1	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/7/1997	869	2	7	1	1997	Tuesday	January	Q1	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/3/1997	1	4	3	1	1997	Friday	January	Q1	Washington	Washington Berry Juice	90748583674	8.39
1/1/1997	1472	3	1	1	1997	Wednesday	January	Q1	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/6/1997	1472	2	6	1	1997	Monday	January	Q1	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/5/1997	2	4	5	1	1997	Sunday	January	Q1	Washington	Washington Mango Drink	96516502499	7.42
1/1/1997	76	4	1	1	1997	Wednesday	January	Q1	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/1/1997	76	2	1	1	1997	Wednesday	January	Q1	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/5/1997	3	2	5	1	1997	Sunday	January	Q1	Washington	Washington Strawberry Drink	58427771925	13.1
1/7/1997	3	2	7	1	1997	Tuesday	January	Q1	Washington	Washington Strawberry Drink	58427771925	13.1
1/1/1997	320	3	1	1	1997	Wednesday	January	Q1	Excellent	Excellent Cranberry Juice	36570182442	16.4

Sure you can, ***but it's inefficient!***

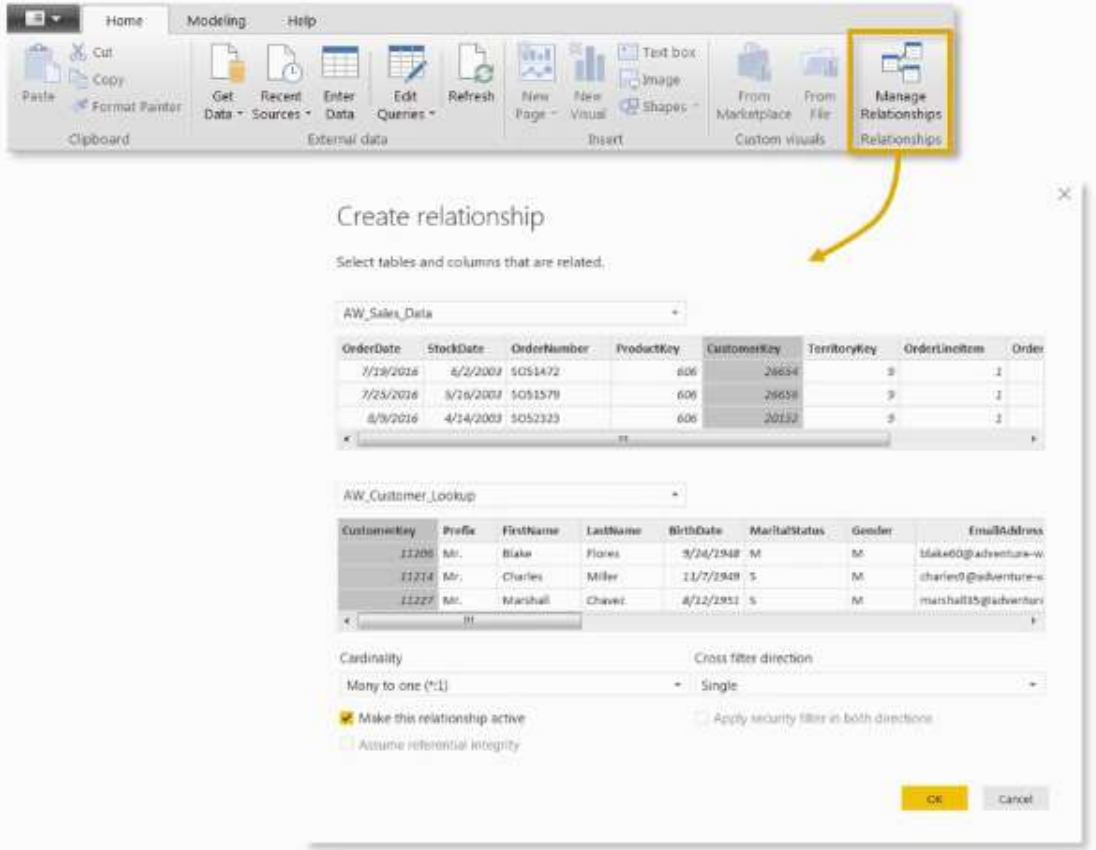
- Merging data in this way creates **redundant data** and utilizes **significantly more memory and processing power** than creating relationships between multiple small tables

CREATING TABLE RELATIONSHIPS

Option 1: Click and drag to connect primary and foreign keys within the **Relationships** pane



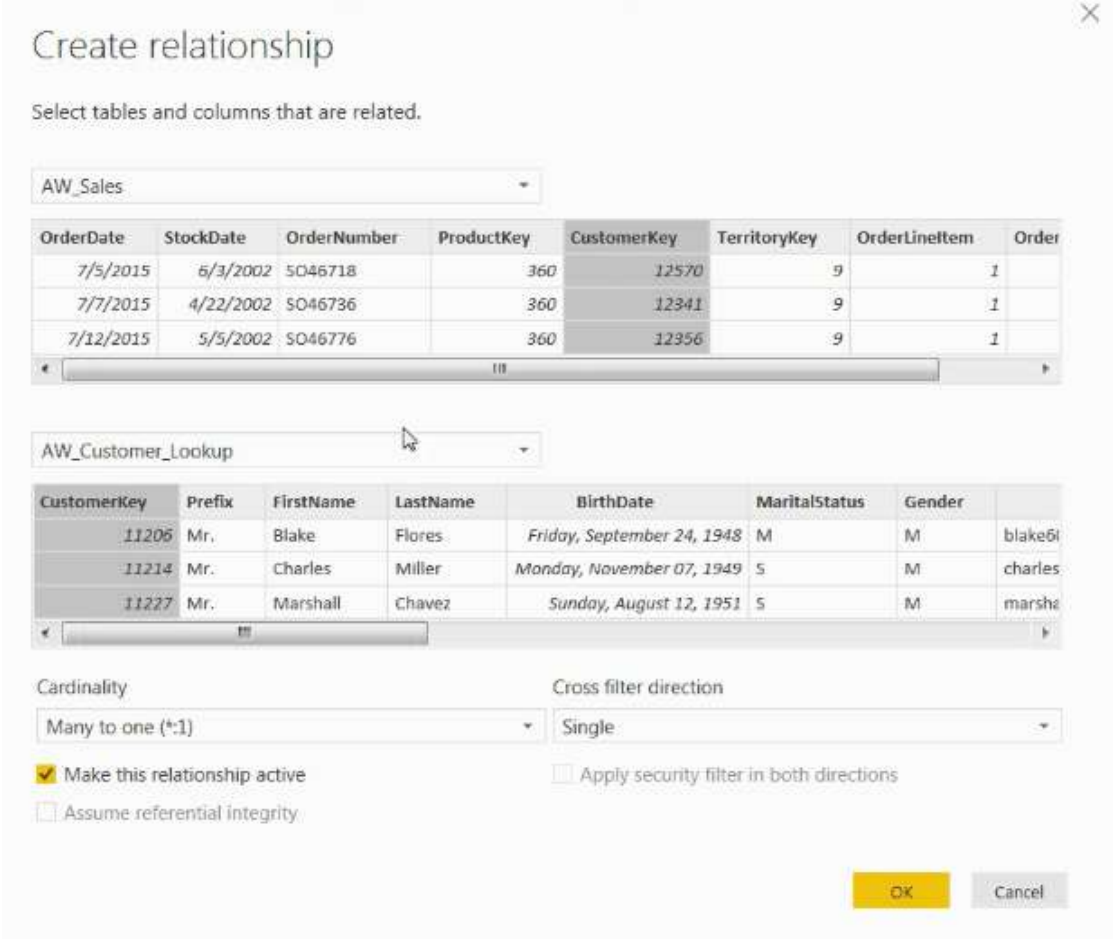
Option 2: Add or detect relationships using the "**Manage Relationships**" dialog box



Demo : Create a Relationship using Manage Relation ships

- Go to model tab →click (Manage Relationship) from the ribbon.
- Click new
- Choose your columns to make a Relationship.

Note: The relationship was already Created as both columns names are same.



Demo : Create a Relationship by drag and drop

- Got to model tab
- Drag (orderdate) from sales table to date from (calender_lookup) to create a manual relationship.
- Drag (Territorieskey) from sales table to (SalesTerritoriesKey) from (Territories_lookup) to create a manual relationship.
- Drag (Productkey) from sales table to (Productkey) from (Product_lookup) to create a manual relationship.
- Now check from the view/report tab, you will find that

The (Order quantity values) are divided based on

(product name)

Join Data - Join Multiple Datasets based on common keys

Microsoft Azure

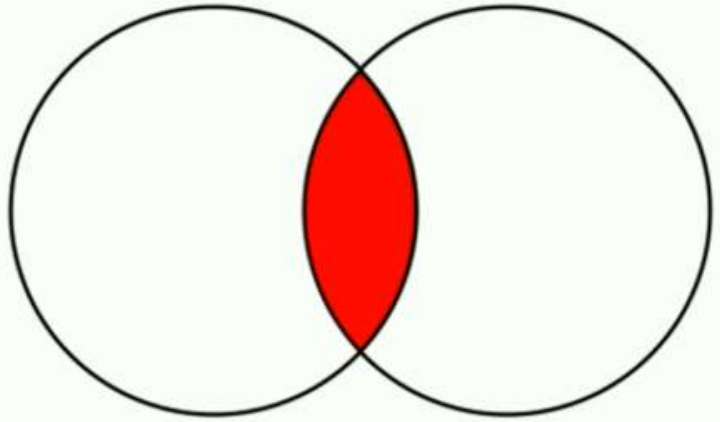
- Information is provided in two or more datasets
  - Different sources
  - Created at different times
- Datasets are related by key columns
- Different types of Join supported by AzureML
  - Inner Join
  - Left Outer Join
  - Full Outer Join
  - Left Semi-join

Inner Join

Microsoft Azure

EmpID	Salary
EMP001	$ 5,000
EMP002	$ 5,500
EMP003	$ 5,200
EMP004	$ 6,000
EMP007	$ 5,800
EMP008	$ 6,700

EmpID	Department
EMP001	IT
EMP003	IT
EMP004	Marketing
EMP007	Finance
EMP009	Marketing
EMP010	Finance



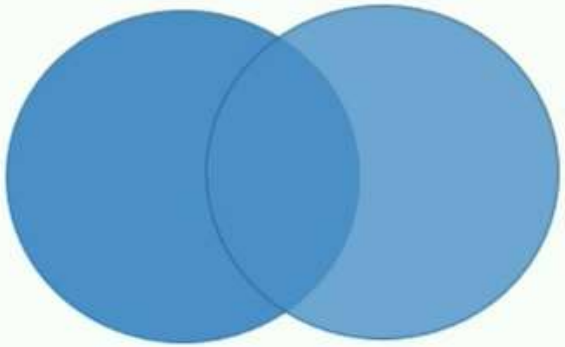
EmpID	Salary	Department
EMP001	$ 5,000	IT
EMP003	$ 5,200	IT
EMP004	$ 6,000	Marketing
EMP007	$ 5,800	Finance

Full Outer Join

Microsoft Azure

EmpID	Salary
EMP001	$ 5,000
EMP002	$ 5,500
EMP003	$ 5,200
EMP004	$ 6,000
EMP007	$ 5,800
EMP008	$ 6,700

EmpID	Department
EMP001	IT
EMP003	IT
EMP004	Marketing
EMP007	Finance
EMP009	Marketing
EMP010	Finance



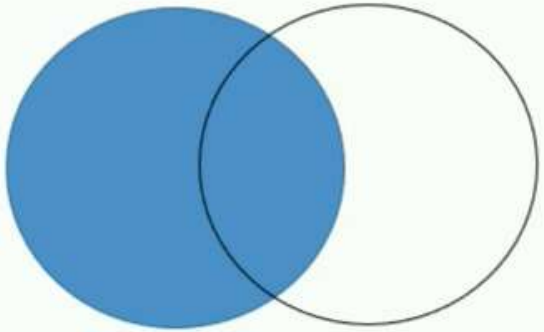
EmpID	Salary	Department
EMP001	$ 5,000	IT
EMP002	$ 5,500	
EMP003	$ 5,200	IT
EMP004	$ 6,000	Marketing
EMP007	$ 5,800	Finance
EMP008	$ 6,700	
EMP009		Marketing
EMP010		Finance

Left Outer Join

Microsoft Azure

EmpID	Salary
EMP001	$ 5,000
EMP002	$ 5,500
EMP003	$ 5,200
EMP004	$ 6,000
EMP007	$ 5,800
EMP008	$ 6,700

EmpID	Department
EMP001	IT
EMP003	IT
EMP004	Marketing
EMP007	Finance
EMP009	Marketing
EMP010	Finance



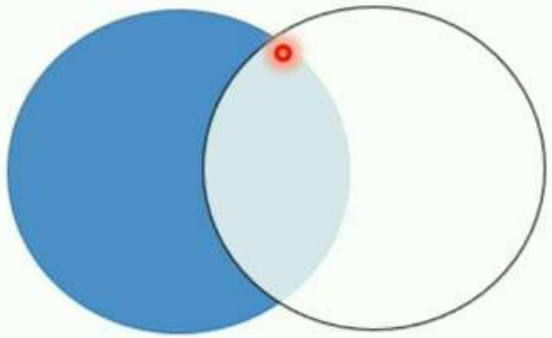
EmpID	Salary	Department
EMP001	$ 5,000	IT
EMP002	$ 5,500	
EMP003	$ 5,200	IT
EMP004	$ 6,000	Marketing
EMP007	$ 5,800	Finance
EMP008	$ 6,700	

Left Semi Join

Microsoft Azure

EmpID	Salary
EMP001	$ 5,000
EMP002	$ 5,500
EMP003	$ 5,200
EMP004	$ 6,000
EMP007	$ 5,800
EMP008	$ 6,700

EmpID	Department
EMP001	IT
EMP003	IT
EMP004	Marketing
EMP007	Finance
EMP009	Marketing
EMP010	Finance

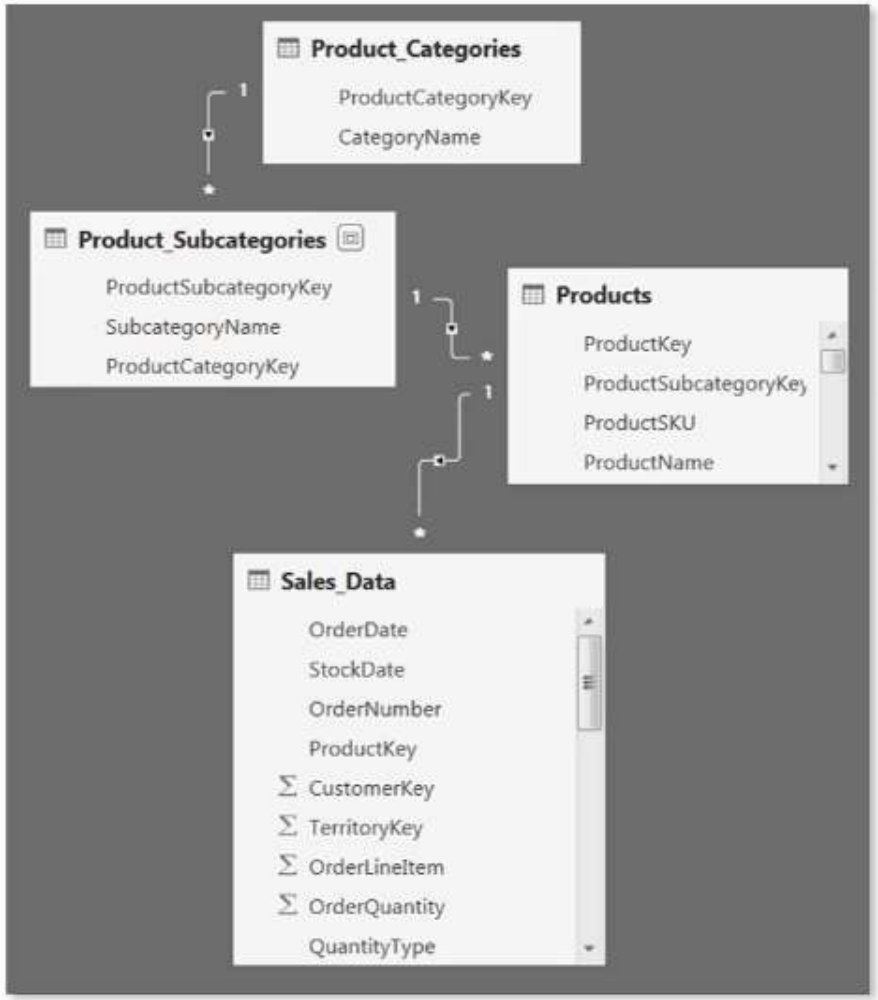


EmpID	Salary
EMP001	$ 5,000
EMP003	$ 5,200
EMP004	$ 6,000
EMP007	$ 5,800

EID	empname	Salary	DateOfJoin	JoiningBonus	DID
00004001	Sone	500000	1/1/2020 12:00:00 AM	50000	1
0001001	Kumar	500000	1/1/2020 12:00:00 AM	50000	2
0002001	Tom	500000	1/1/2020 12:00:00 AM	50000	4
0003001	Hary	500000	1/1/2020 12:00:00 AM	50000	null
01920	Imran	200000	1/3/2020 12:00:00 AM	30000	2
E0101	Yousuf	100000	1/1/2020 12:00:00 AM	10000	1
E0102	Paula	100000	1/1/2020 12:00:00 AM	10000	null
E0103	Paula	100000	1/1/2020 12:00:00 AM	null	2
E0104	muhobath	100000	1/1/2020 12:00:00 AM	10000	1

DepID	DepName
1	IT
2	HR
3	Audit
null	General

CREATING “SNOWFLAKE” SCHEMAS



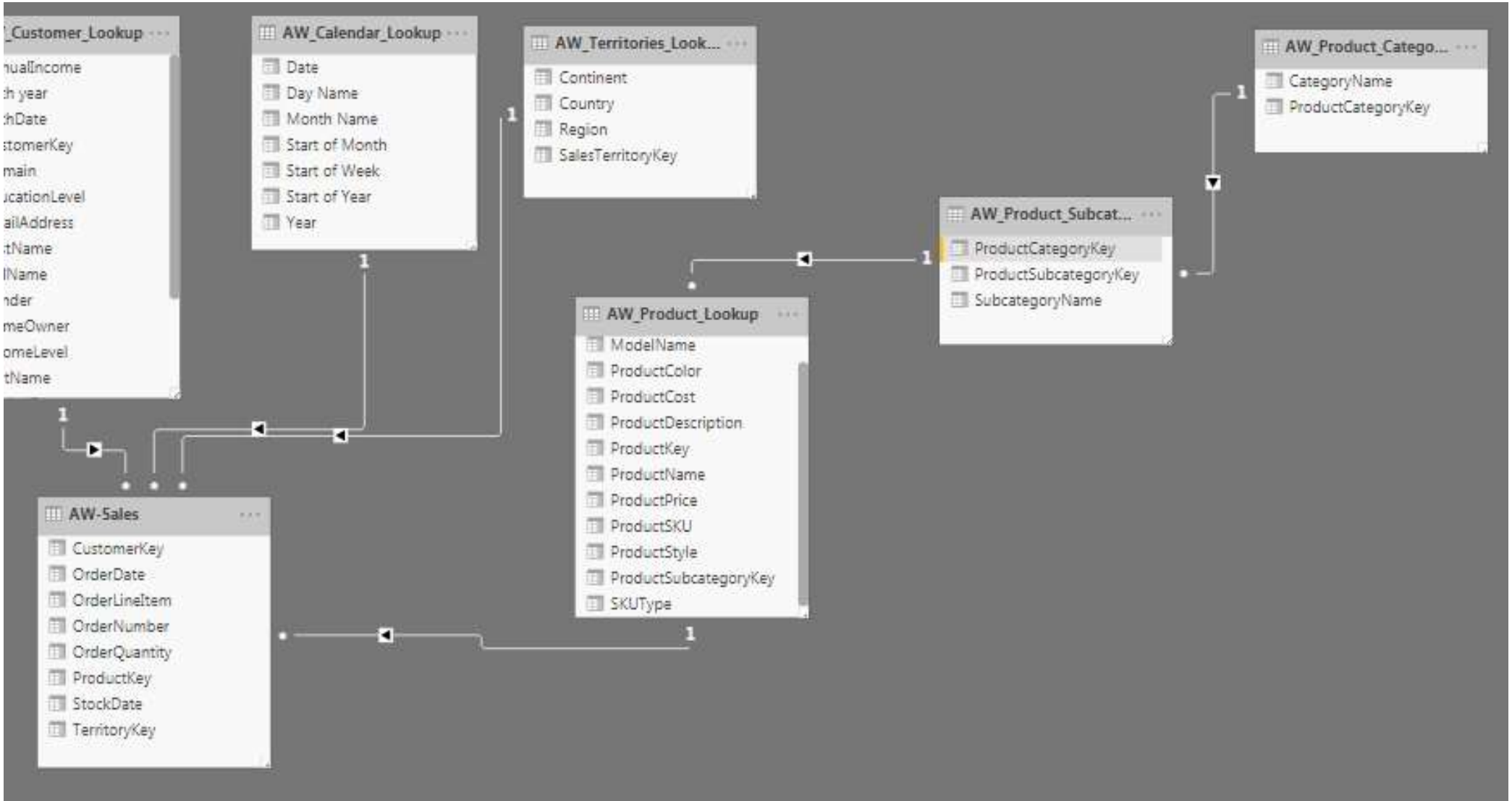
The **Sales_Data** table can connect to **Products** using the ***ProductKey*** field, but cannot connect directly to the **Subcategories** or **Categories** tables

By creating relationships from **Products** to **Subcategories** (using ***ProductSubcategoryKey***) and **Subcategories** to **Categories** (using ***ProductCategoryKey***), we have essentially connected **Sales_Data** to each lookup table; filter context will now flow all the way down the chain

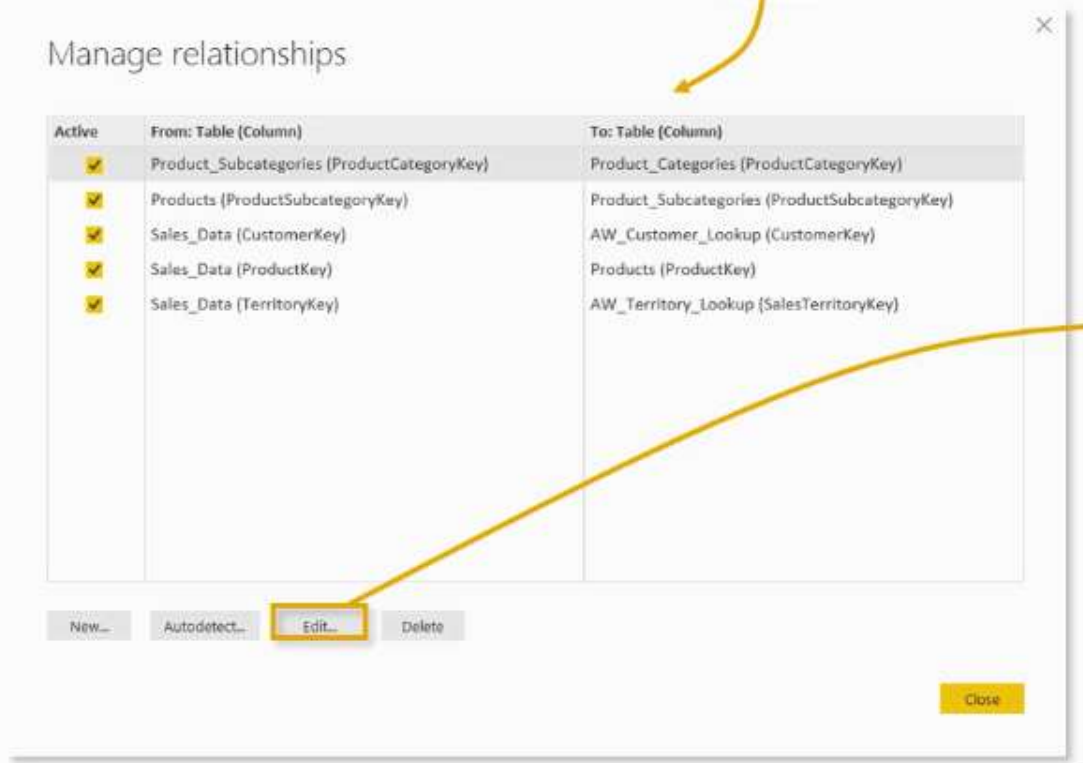
PRO TIP:

*Models with chains of dimension tables are often called “**snowflake**” schemas (whereas “**star**” schemas usually have individual lookup tables surrounding a central data table)*

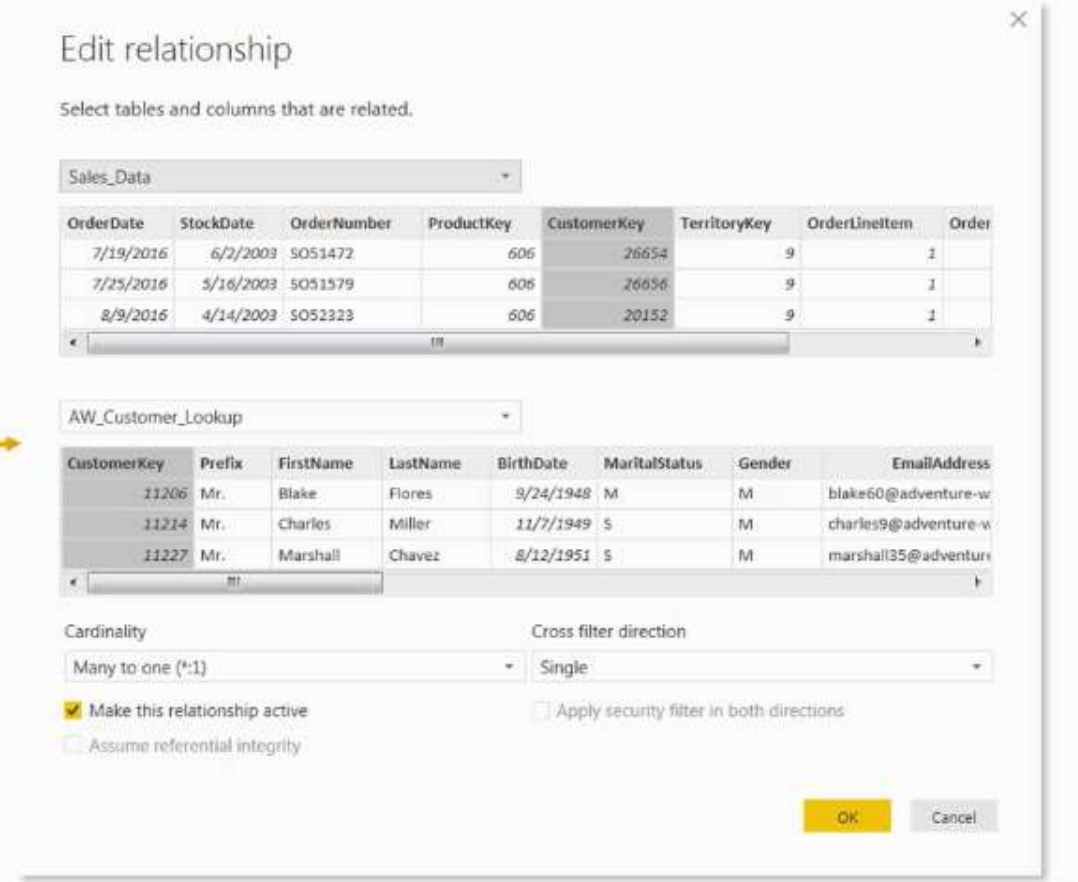
Demo


_Customer_Lookup
nualIncome
h year
hDate
tomerKey
main
ucationLevel
ailAddress
tName
lName
nder
meOwner
omeLevel
tName
AW_Calendar_Lookup
Date
Day Name
Month Name
Start of Month
Start of Week
Start of Year
Year
AW_Territories_Look...
Continent
Country
Region
SalesTerritoryKey
AW_Product_Catego...
CategoryName
ProductCategoryKey
AW_Product_Subcat...
ProductCategoryKey
ProductSubcategoryKey
SubcategoryName
AW_Product_Lookup
ModelName
ProductColor
ProductCost
ProductDescription
ProductKey
ProductName
ProductPrice
ProductSKU
ProductStyle
ProductSubcategoryKey
SKUType
AW-Sales
CustomerKey
OrderDate
OrderLineItem
OrderNumber
OrderQuantity
ProductKey
StockDate
TerritoryKey
1
1
1
1
1
1

MANAGING & EDITING RELATIONSHIPS



*The "**Manage Relationships**" dialog box allows you to **add**, **edit**, or **delete** table relationships*



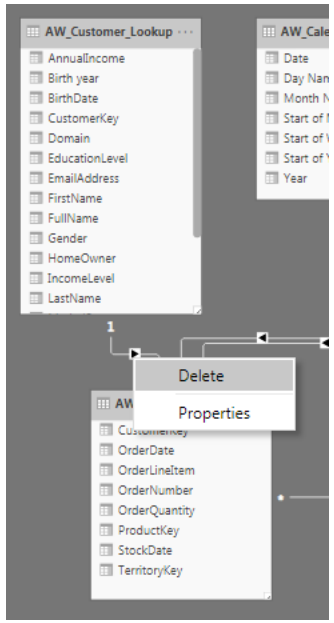
*Editing tools allow you to **activate/deactivate** relationships, view **cardinality**, and modify the **cross filter direction** (stay tuned!)*

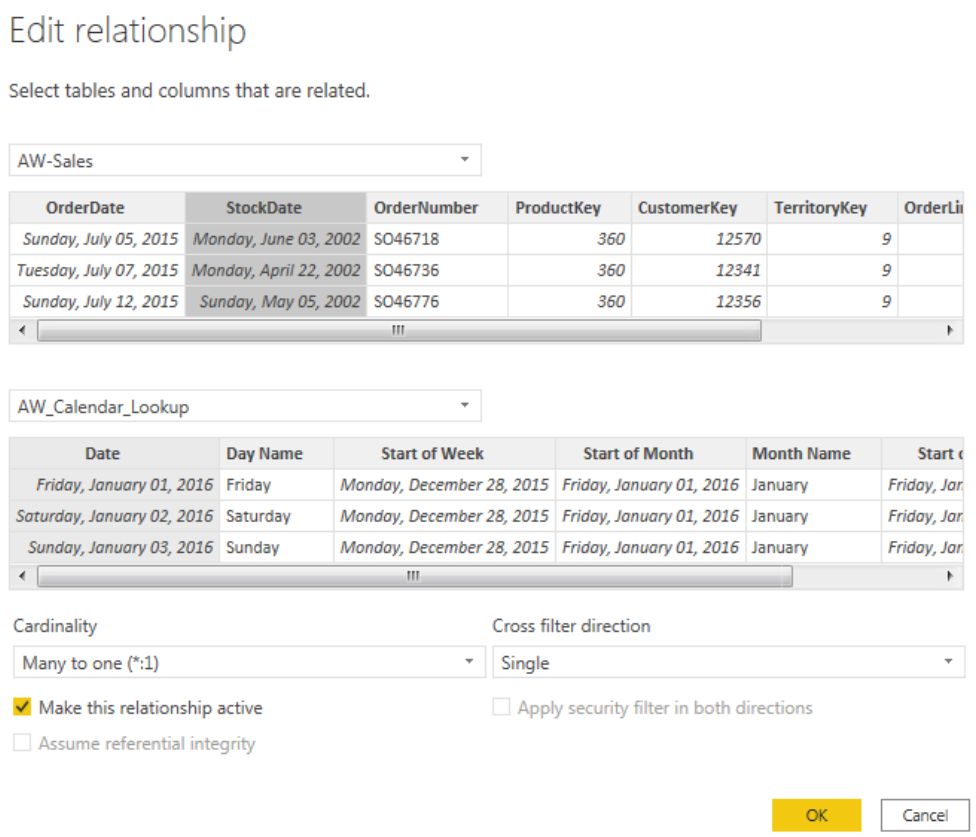
Demo

- Try to use (Manage Relationship) to edit sales date relation ship to and change (Order Date) to (Stock Date).
- Also we can delete the unwanted

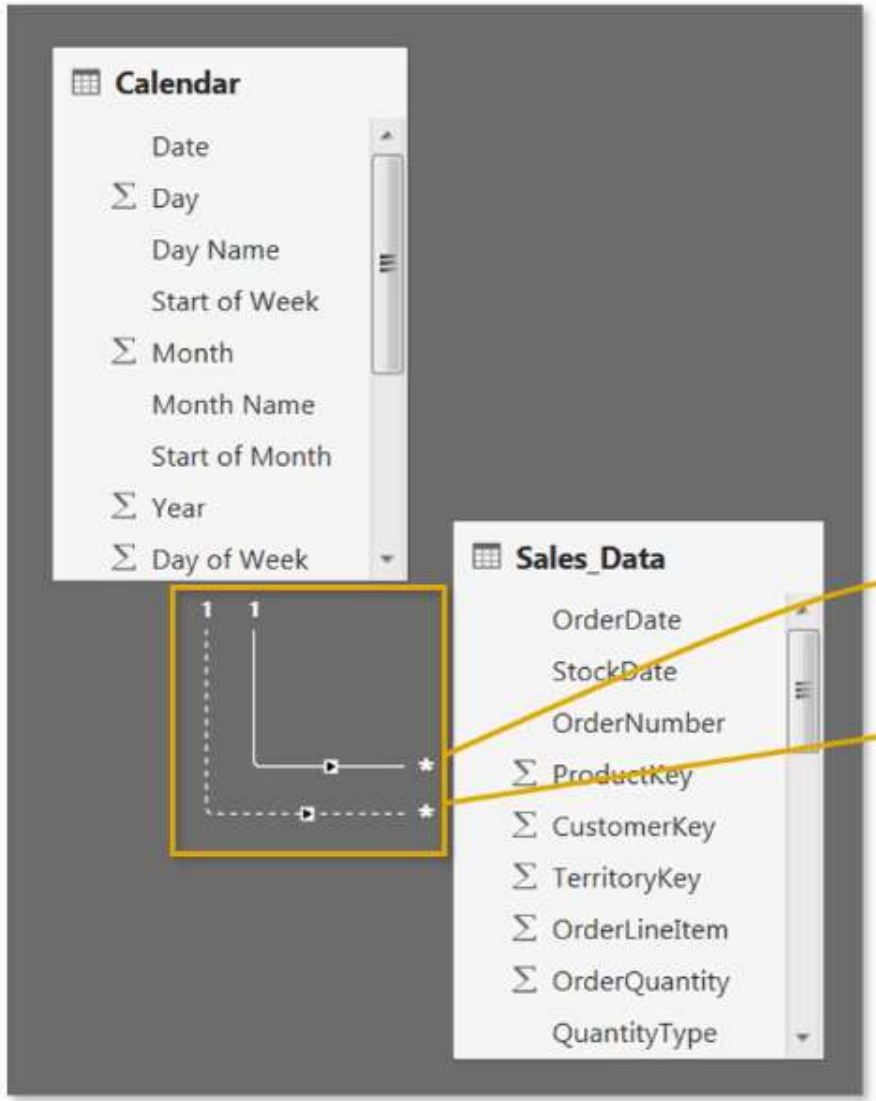
Relationship by simply clicking on it and

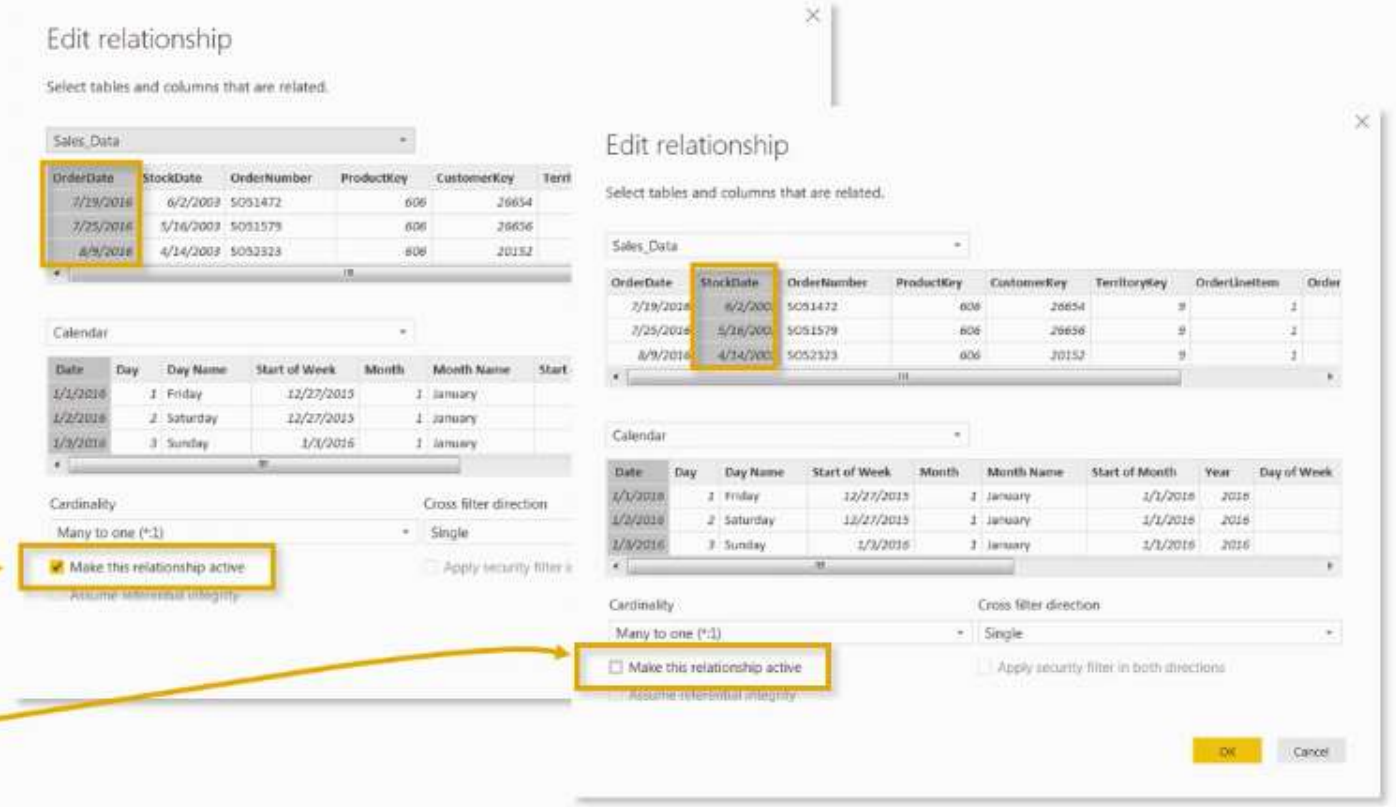
Choose delete.





ACTIVE VS. INACTIVE RELATIONSHIPS





The **Sales_Data** table contains two date fields (***OrderDate*** & ***StockDate***), but there can only be one *active* relationship to the Date field in the Calendar table

Double-click the relationship line, and check the "***Make this relationship active***" box to toggle (note that you have to deactivate one in order to activate another)

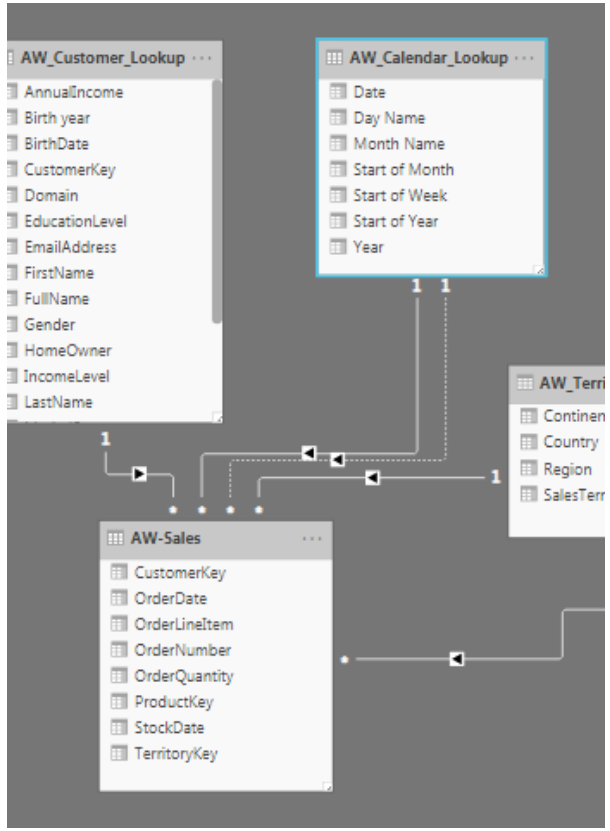
Demo

- Add relationship between stock date (sales table) with date from (calender_lookup)
- Notice that one line is solid and other not, this is

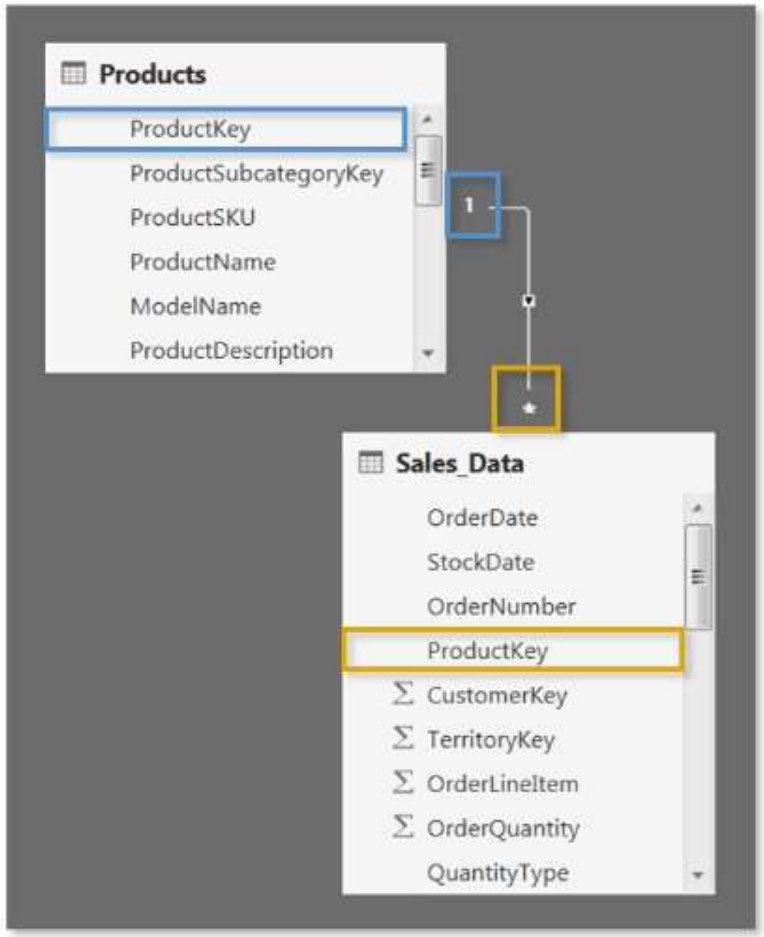
Because only one relationship can be active at a

Time so by default one relation ship is active and other

One inactive.



RELATIONSHIP CARDINALITY



Cardinality refers to the *uniqueness of values* in a column

- For our purposes, all relationships in the data model should follow a "**one-to-many**" cardinality; **one** instance of each *primary key*, but potentially **many** instances of each *foreign key*

*In this case, there is only **ONE instance of each ProductKey** in the Products table (noted by the "**1**"), since each row contains **attributes of a single product** (Name, SKU, Description, Retail Price, etc)*

*There are **MANY instances of each ProductKey** in the Sales_Data table (noted by the asterisk *****), since there are **multiple sales associated with each product***

CARDINALITY CASE STUDY: MANY-TO-MANY

product_id	product_name	product_sku
4	Washington Cream Soda	64412155747
4	Washington Diet Cream Soda	81727382373
5	Washington Diet Soda	85561191439
7	Washington Diet Cola	20191444754
8	Washington Orange Juice	89770532250

date	product_id	transactions
1/1/2017	4	12
1/2/2017	4	9
1/3/2017	4	11
1/1/2017	5	16
1/2/2017	5	19
1/1/2017	7	11

- If we try to connect these tables using **product_id**, we'll get a "***many-to-many relationship***" error since there are multiple instances of each ID in both tables
- Even if we *could* create this relationship, how would you know which product was actually sold on each date – ***Cream Soda*** or ***Diet Cream Soda***?

CARDINALITY CASE STUDY: ONE-TO-ONE

product_id	product_name	product_sku
4	Washington Cream Soda	64412155747
5	Washington Diet Soda	85561191439
7	Washington Diet Cola	20191444754
8	Washington Orange Juice	89770532250

product_id	product_price
4	$3.64
5	$2.19
7	$2.61
8	$2.59

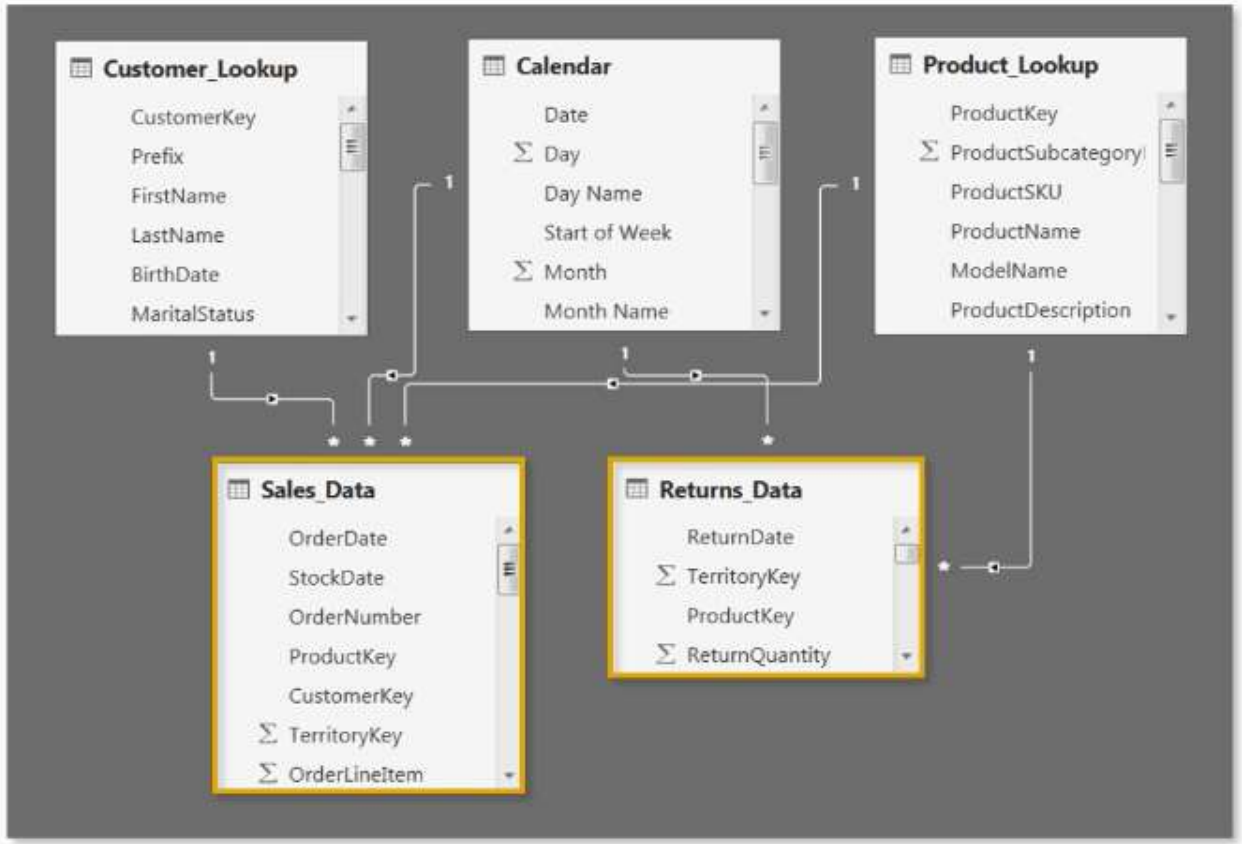
- Connecting the two tables above using the **product_id** field creates a ***one-to-one relationship***, since each ID only appears once in each table
- Unlike many-to-many, there is nothing *illegal* about this relationship; it's just **inefficient**

To eliminate the inefficiency, you could simply ***merge the two tables*** *into a single, valid lookup*

NOTE: *this still respects the laws of normalization, since all rows are unique and capture attributes related to the primary key*

product_id	product_name	product_sku	product_price
4	Washington Cream Soda	64412155747	$3.64
5	Washington Diet Soda	85561191439	$2.19
7	Washington Diet Cola	20191444754	$2.61
8	Washington Orange Juice	89770532250	$2.59

CONNECTING MULTIPLE DATA TABLES



This model contains two data tables: **Sales_Data** and **Returns_Data**

- Note that the **Returns** table connects to **Calendar** and **Product_Lookup** just like the **Sales** table, but without a *CustomerKey* field it cannot be joined to **Customer_Lookup**
- This allows us to analyze sales and returns within the same view, ***but only if we filter or segment the data using shared lookups***
  - In other words, we know which ***product*** was returned and on which ***date***, but nothing about which ***customer*** made the return

HEY THIS IS IMPORTANT!

*In general, never create **direct relationships** between data tables; instead, **connect them through shared lookups***

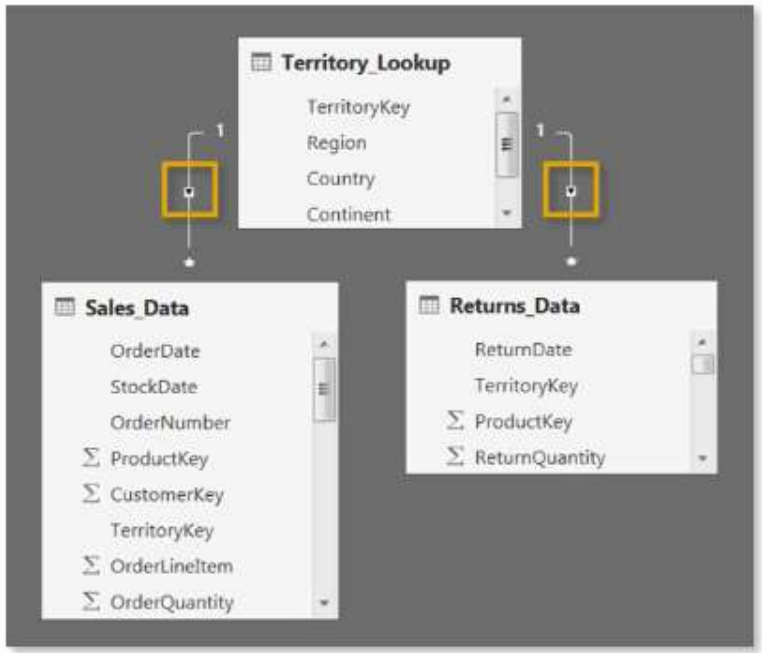
Demo

- Import (AW_return) table
- Form a relationship between (AW_return) and (Territories_lookup,Calende_lookup ,Product_lookup)
- Go to report view tab
- Select the previous matrix
- Drag (return quantity) to values field with (order quantity)

Next: Delete (Product column) from value field and replace it with Gender from (customer_lookup), you will notice that return quantity values is fixed for all rows as there is no relationship between (customer_lookup) and (return data)

Note: Don’t form relationship between two data tables as it will be many-to-many relationship

FILTER FLOW



Here we have two data tables (**Sales_Data** and **Returns_Data**), connected to **Territory_Lookup**

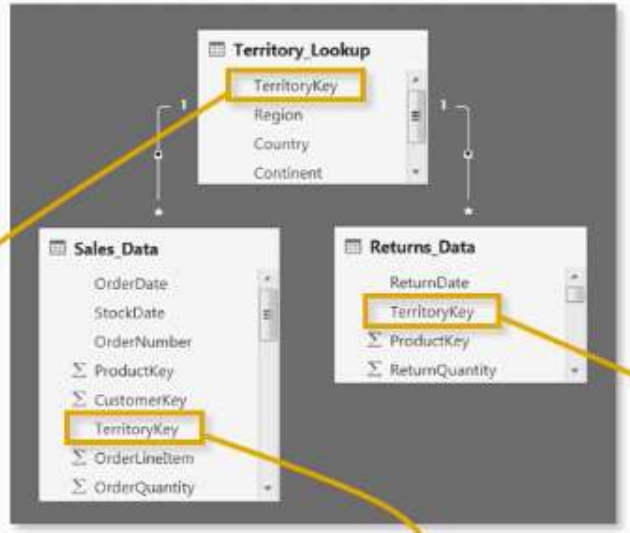
Note the filter directions (shown as arrows) in each relationship; by default, **these will point from the "one" side of the relationship** *(lookups)* **to the "many" side** *(data)*

- When you filter a table, that filter context is passed along to all related "*downstream*" tables (following the direction of the arrow)
- Filters **cannot** flow "*upstream*" (against the direction of the arrow)

PRO TIP:

*Arrange your lookup tables **above** your data tables in your model as a visual reminder that filters flow "**downstream**"*

FILTER FLOW (CONT.)



In this case, the only valid way filter both **Sales** and **Returns** data by Territory is to use the *TerritoryKey* field from the **Territory_Lookup** table, which is upstream and related to *both* data tables

- Filtering using *TerritoryKey* from the **Sales** table yields incorrect **Returns** values, since the filter context ***cannot flow upstream*** to either one of the other tables
- Similarly, filtering using *TerritoryKey* from the **Returns** table yields incorrect **Sales** data; in addition, ***only territories that registered returns are visible in the table*** (even though they registered sales)

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*1) Filtering using TerritoryKey from the **Territory_Lookup** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	1,828
2	40	1,828
3	30	1,828
4	17,191	1,828
5	49	1,828
6	894	1,828
7	7,862	1,828
8	7,950	1,828
9	17,951	1,828
10	9,694	1,828
Total	**84,174**	**1,828**

*2) Filtering using TerritoryKey from the **Sales_Data** table*

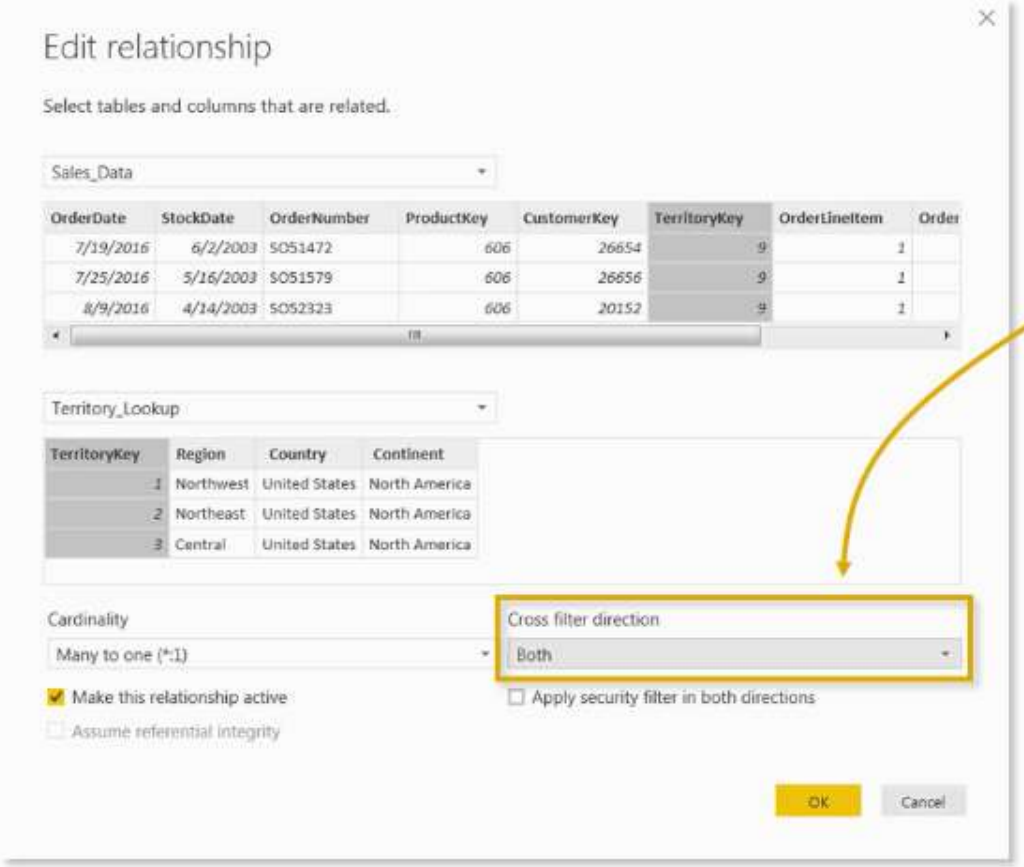
TerritoryKey	OrderQuantity	ReturnQuantity
1	84,174	270
4	84,174	362
5	84,174	1
6	84,174	238
7	84,174	186
8	84,174	163
9	84,174	404
10	84,174	204
Total	**84,174**	**1,828**

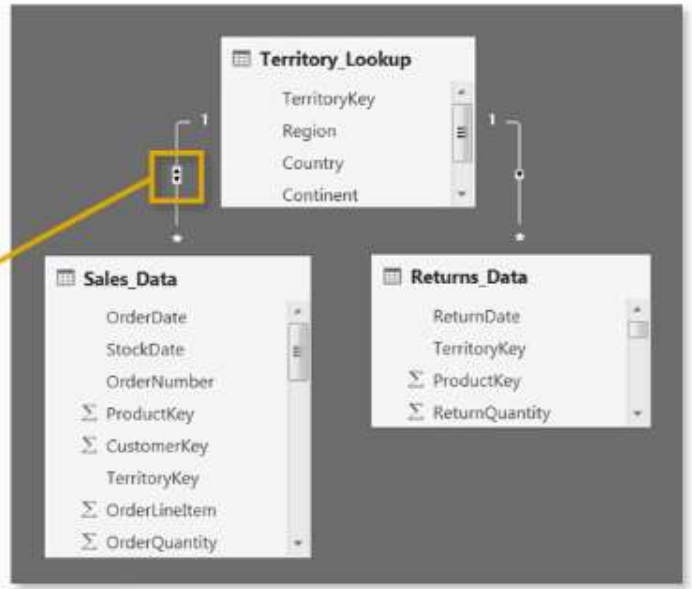
*3) Filtering using TerritoryKey from the **Returns_Data** table*

Demo

- Replace gender column from matrix to territory key from territory_lookup table, you will notice that both order quantities, and return quantities are responding as they are connected to territory_lookup table.

TWO-WAY FILTERS



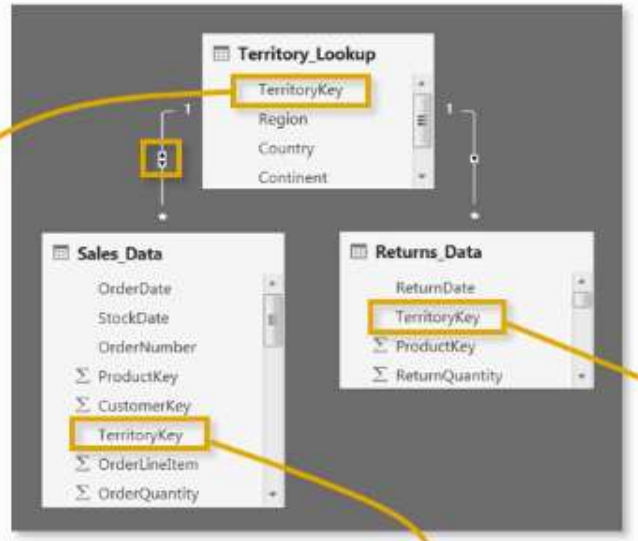


Updating the filter direction between **Sales** and **Territory** from "*Single*" to "*Both*" allows filter context to flow both ways

- This means that filters applied to the **Sales_Data** table will pass to the lookup, and then down to the **Returns_Data** table

NOTE: *The "Apply security filter in both directions" option relates to row-level security (RLS) settings, which are not covered in this course*

TWO-WAY FILTERS (CONT.)



With two-way cross-filtering enabled between the **Sales** and **Territory** tables, we now see correct values using *TerritoryKey* from either table

- The filter context for **Sales_Data[TerritoryKey]** now passes *up* to the **Territory_Lookup**, and then *down* to the **Returns_Data** table
- Note that we still see incorrect values when filtering using *TerritoryKey* from the **Returns** table, since the filter context is isolated to that single table

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*1) Filtering using TerritoryKey from the **Territory_Lookup** table*

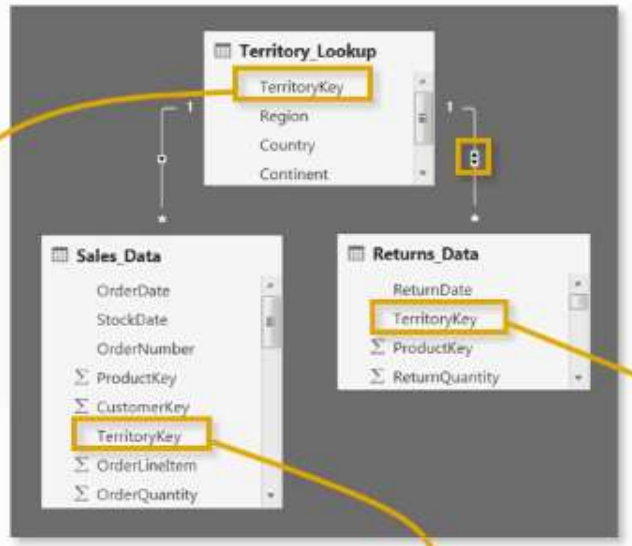
TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*2) Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	84,174	270
4	84,174	362
5	84,174	1
6	174	238
7	174	186
8	84,174	163
9	84,174	404
10	84,174	204
Total	**84,174**	**1,828**

*3) Filtering using TerritoryKey from the **Returns_Data** table*

TWO-WAY FILTERS (CONT.)



In this case, we've enabled two-way cross-filtering between the **Returns** and **Territory** tables

- As expected, we now see incorrect values when filtering using *TerritoryKey* from the **Sales** table, since the filter context is isolated to that single table
- While the values *appear* to be correct when filtering using *TerritoryKey* from the **Returns** table, we're ***missing sales data*** from any territories that didn't register returns (*specifically Territories **2** & **3***)

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	10,894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

1) *Filtering using TerritoryKey from the **Territory_Lookup** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	1,828
2	40	1,828
3	30	1,828
4	17,191	1,828
5	49	1,828
6	10,894	1,828
7	7,862	1,828
8	7,950	1,828
9	17,951	1,828
10	9,694	1,828
Total	**84,174**	**1,828**

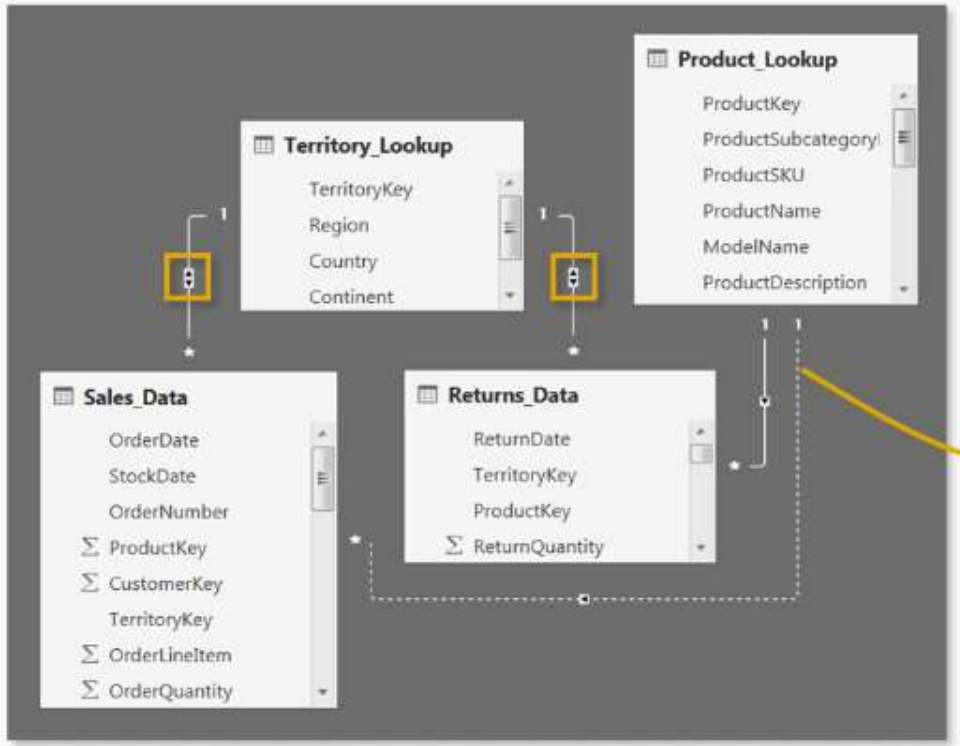
2) *Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
4	17,191	362
5	49	1
6	10,894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

3) *Filtering using TerritoryKey from the **Returns_Data** table*

*Since no information about Territory 2 or 3 is passed from the **Returns_Data** table to **Territory_Lookup**, they get filtered out of the lookup, and subsequently filtered out of the **Sales_Data***

TWO-WAY FILTERS: **A WORD OF WARNING**



Use two-way filters carefully, and **only when necessary***

- If you try to use multiple two-way filters in a more complex model, you run the risk of creating "***ambiguous relationships***" by introducing multiple filter paths between tables:

> You can't create a direct active relationship between Sales_Data and Product_Lookup because that would introduce ambiguity between tables Product_Lookup and Territory_Lookup. To make this relationship active, deactivate or delete one of the relationships between Product_Lookup and Territory_Lookup first.

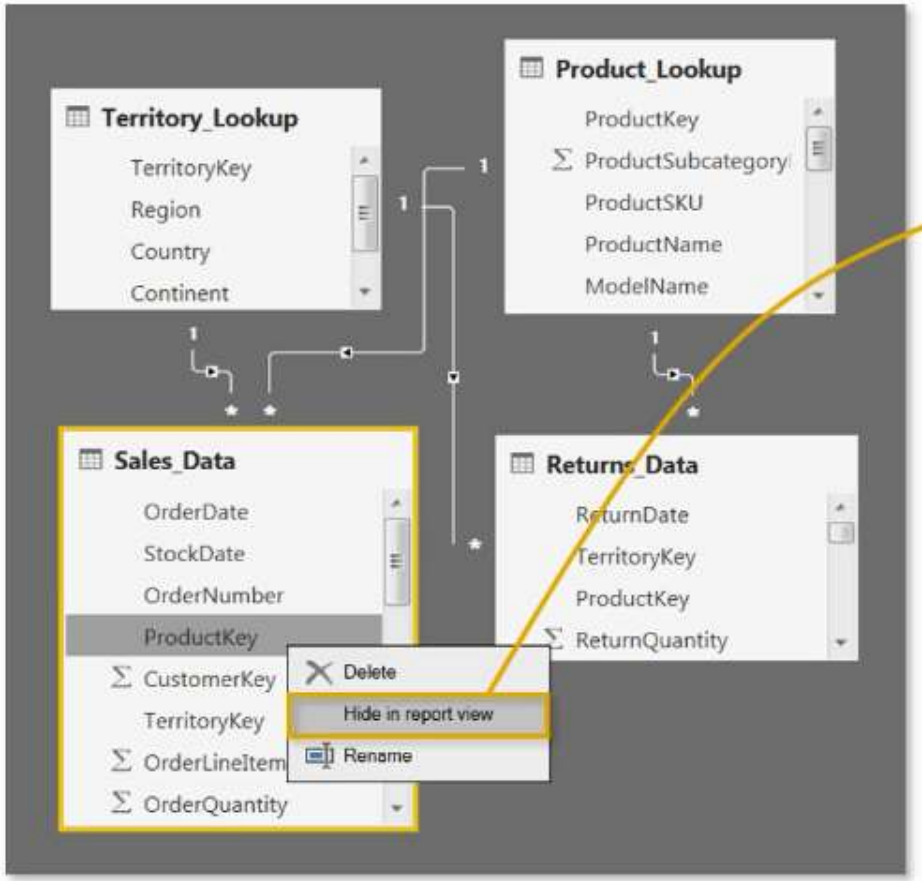
*In this model, filter context from the **Product_Lookup** table can pass down to **Returns_Data** and up to **Territory_Lookup**, which would filter accordingly based on the TerritoryKey values passed from the Returns table*

*If we were able to activate the relationship between **Product_Lookup** and **Sales_Data** as well, filters could pass from the **Product_Lookup** table through EITHER the Sales or Returns table to reach the **Territory_Lookup**, which could yield conflicting filter context*

PRO TIP:

*Design your models with **one-way filters** and **1-to-Many cardinality**, unless more complex relationships are necessary*

HIDING FIELDS FROM REPORT VIEW



Hiding fields from Report View makes them inaccessible from the Report tab (*although they can still be accessed within the **Data** or **Relationships** views*)

This is commonly used to prevent users from filtering using invalid fields, or to hide irrelevant metrics from view

PRO TIP:

*Hide the **foreign key columns** in your data tables to force users to filter using the **primary keys** in the lookup tables*

BEST PRACTICES: DATA MODELING

Focus on building a normalized model from the start

- *Make sure that each table in your model serves a single, distinct purpose*
- *Use relationships vs. merged tables; long & narrow tables are better than short & wide*

Organize lookup tables *above* data tables in the diagram view

- *This serves as a visual reminder that filters flow "**downstream**"*

Avoid complex cross-filtering unless absolutely necessary

- *Don't use two-way filters when 1-way filters will get the job done*

Hide fields from report view to prevent invalid filter context

- *Recommend hiding foreign keys from data tables, so that users can only access valid fields*

Quiz

Question 1:

Why might you normalize a data model?

- ◯ To make it more efficient
- ◯ To minimize the risk of error during data modifications
- ◯ To simplify queries and analysis
- ◯ All of the above

Question 2:

Which type of table generally contains foreign keys?

- ◯ Lookup Tables
- ◯ Data Tables
- ◯ Dimension Tables
- ◯ All Tables

Question 3:

TRUE or FALSE: You can create multiple active relationships from one table to the same primary key

- True
- False

Question 4:

Which type of cardinality describes the relationships in our Adventure Works data model?

- ◯ One-to-One
- ◯ One-to-Many
- ◯ Many-to-Many

Question 5:

Generally speaking, what's the best way to "connect" two data tables within the same model?

- ◯ Deduplicate the foreign keys in one table to create a one-to-many relationship
- ◯ Deduplicate the foreign keys in both tables
- ◯ Match the row-level granularity in order to define a one-to-one relationship
- ◯ Relate them through shared lookup tables

Question 6:

Why is it sometimes helpful to arrange lookup tables *above* data tables in the Relationships view?

- ◯ To remind you that filters flow "downstream"
- ◯ Because they generally contain the most fields
- ◯ Because it makes it easier to create new relationships
- ◯ Because lookup tables contain the fewest number of fields

Question 8:

What can you do to prevent users from filtering with invalid fields from a data model?

- ◯ Politely (but firmly) ask them to cut it out
- ◯ Threaten them with public humiliation
- ◯ Tell them that "there will be consequences" if they do it again, then slowly back out of the room without breaking eye contact
- ◯ Activate the "Hide in Report View" option

Question 9:

Which type of database schema could be described as containing a central data table surrounded by individual lookup tables?

- ◯ Star
- ◯ Snowflake
- ◯ Snowball
- ◯ Spiral

Exercise: Creating Table Relationships & Data Models in Power BI

- Using your Adventure Works report file, complete the following:

- **1)** Navigate to the **RELATIONSHIPS** view, and perform the following actions
- Right-click to delete each relationship between **AW_Sales**, **AW_Customer_Lookup** and **AW_Calendar_Lookup** (*including both date fields*)
- Use the **Manage Relationships** tool to delete *all* remaining relationships between all tables

- **2)** Recreate all table relationships (*using any method you prefer*), and confirm the following :
- Cardinality is **1-to-Many** for all relationships
- Filters are all **One-Way** (*no two-way filters*)
- Filter direction correctly flows "**downstream**" to data tables
- Data tables are **not connected** directly to one another
- Both data tables are connected to **all valid lookup tables**
- Product-related tables follow a **snowflake schema**

- **3)** Return to the **REPORT** view, and complete the following:
- Edit (*or insert*) the matrix visual to show **ReturnQuantity** (*values*) by **CategoryName** (*rows*) from the **AW_Product_Category_Lookup** table
  - *Which category saw the highest volume of returns? How many?*
- Replace **CategoryName** with **Year** from the **AW_Calendar_Lookup** table
  - *How many returns do you see in 2015 vs. 2016?*
- Replace **Year** with **FullName** from the **AW_Customer_Lookup** table
  - *What do you see, and why?*
- Update the matrix to show both **OrderQuantity** and **ReturnQuantity** (*values*) by **ProductKey** (*rows*) fro m the **AW_Product_Lookup** table
  - *What was the total OrderQuantity for Product #338?*

- **4)** Unhide the **ProductKey** field from the **AW_Returns** tables (*using either the **DATA** or **RELATIONSHIPS** view*):
- In the matrix,
  replace **ProductKey** from **AW_Product_Lookup** with **ProductKey** from the **AW_Returns** table
  - *Why do we the same repeating values for OrderQuantity?*
- Edit the relationship between **AW_Returns** and **AW_Product_Lookup** to change the cross filter direction from *Single* to *Both*
  - *Why does the visual now show OrderQuantity values by product, even though we are using **ProductKey** from **AW_Returns**?*
  - *How many orders do we see now for Product #338? What's going on here?*

- **5)** Complete the following:
- Change the cross filter direction between **AW_Returns** and **AW_Product_Lookup** back to *single* (*One-Way*)
- Hide the **ProductKey** field in the **AW_Returns** table from report view (*and any other foreign keys, if necessary*)
- Update the matrix to show **ProductKey** from the **AW_Product_Lookup**, rather than **AW_Returns**
- **Recommendation:** Save a separate backup copy of the .pbix file (*i.e. "AdventureWorks_Report_Backup"*)

Section 4: Analyzing Data with DAX Calculations in Power BI

MEET **DAX**

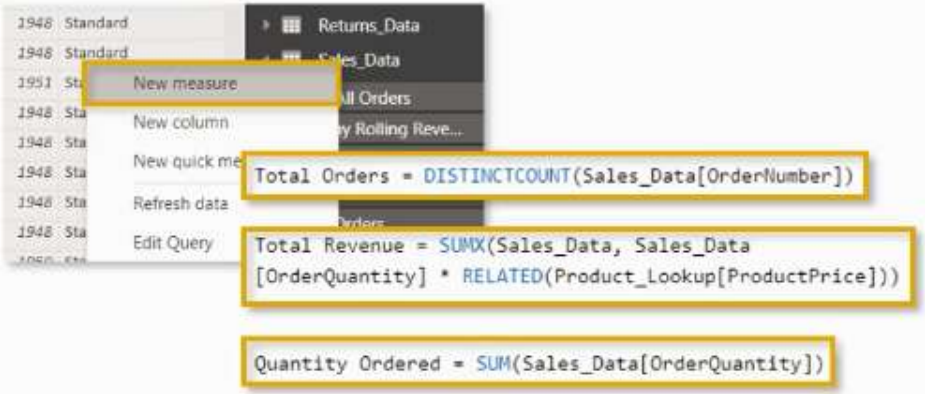
Data Analysis Expressions, commonly known as **DAX**, is the formula language that drives Power BI. With DAX, you can:

- Add ***calculated columns*** and ***measures*** to your model, using intuitive syntax
- Go beyond the capabilities of traditional "grid-style" formulas, with powerful and flexible functions built specifically to work with relational data models

Two ways to use DAX

1) *Calculated Columns*

2) *Measures*



CALCULATED COLUMNS

Calculated columns allow you to add new, formula-based columns to tables

- No "A1-style" references; calculated columns refer to **entire tables** or **columns**
- Calculated columns generate values for each row, which are **visible within tables in the Data view**
- Calculated columns understand **row context**; they're great for defining properties based on information in each row, but generally useless for aggregation (*SUM, COUNT, etc*)

HEY THIS IS IMPORTANT!

As a rule of thumb, use calculated columns when you want to "stamp" static, fixed values to each row in a table (*or use the Query Editor!*)

DO NOT use calculated columns for aggregation formulas, or to calculate fields for the "Values" area of a visualization (use **measures** instead)

PRO TIP:

*Calculated columns are typically used for **filtering** data, rather than creating numerical values*

CALCULATED COLUMNS (EXAMPLES)

In this case we've added a **calculated column** named **"Parent"**, which equals "***Yes***" if the [TotalChildren] field is greater than 0, and "***No***" otherwise (*just like Excel!*)

- Since calculated columns understand **row context**, a new value is calculated in each row based on the value in the [TotalChildren] column
- This is a **valid use** of calculated columns; it creates a new row "property" that we can now use to filter or segment any related data within the model

Here we're using an aggregation function (SUM) to calculate a new column named **TotalQuantity**

- Since calculated columns do not understand **filter context**, the same grand total is returned in *every single row* of the table
- This is **not a valid use** of calculated columns; these values are statically "stamped" onto the table and can't be filtered, sliced, subdivided, etc.

Demo

- Go to data view
- Select AW_sales table → create new column (QuantityType)
- Write the following DAX formula.

```
QuantityType = IF(AW_Sales[OrderQuantity]>1,"Multiple Items","Single Item")
```

- Select AW_sales table → create new column (TotalQuantity)
- Write the following DAX aggregated formula

```
TotalQuantity = SUM(AW_Sales[OrderQuantity])
```

Note: Notice that we got same value repeated in each row.

MEASURES

Measures are DAX formulas used to generate new calculated values

- Like calculated columns, measures reference **entire tables** or **columns** (*no A1-style or "grid" references*)
- *Unlike* calculated columns, **measure** values aren't visible within tables; they can only be "*seen*" within a visualization like a chart or matrix (*similar to a calculated field in an Excel pivot*)
- Measures are evaluated based on **filter context**, which means they recalculate when the fields or filters around them change (*like when new row or column labels are pulled into a matrix or when new filters are applied to a report*)

HEY THIS IS IMPORTANT!

As a rule of thumb, use measures (*vs. calculated columns*) when a single row can't give you the answer (*in other words, when you need to **aggregate***)

PRO TIP:

*Use measures to create **numerical, calculated values** that can be analyzed in the "**values**" field of a report visual*

RECAP: CALCULATED COLUMNS VS. MEASURES

CALCULATED COLUMNS

- Values are calculated based on information from each row of a table (has **row context**)
- Appends static values to each row in a table and stores them in the model (*which increases file size*)
- Recalculate on data source refresh or when changes are made to component columns
- Primarily used as **rows**, **columns**, **slicers** or **filters**

*Calculated columns "live" in **tables***

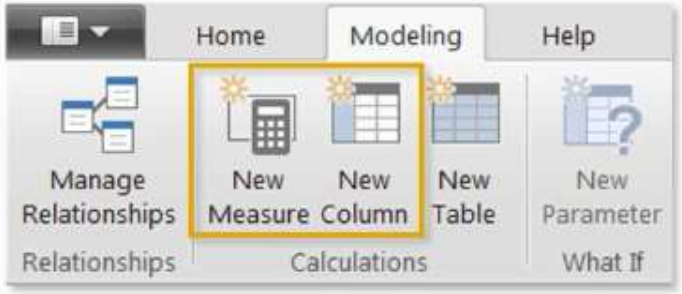
MEASURES

- Values are calculated based on information from any filters in the report (has **filter context**)
- Does not create new data in the tables themselves (*doesn't increase file size*)
- Recalculate in response to any change to filters within the report
- Almost *always* used within the **values** field of a visual

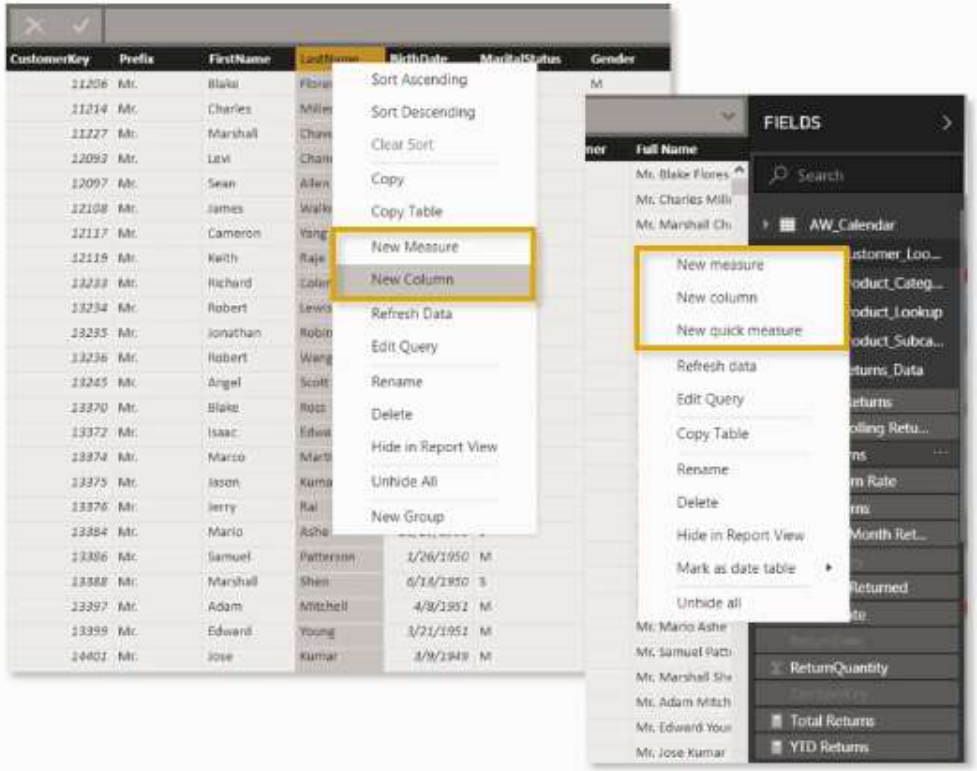
*Measures "live" in **visuals***

ADDING COLUMNS & MEASURES

Option 1: Select "**New Measure**" or "**New Column**" from the **Modeling** tab



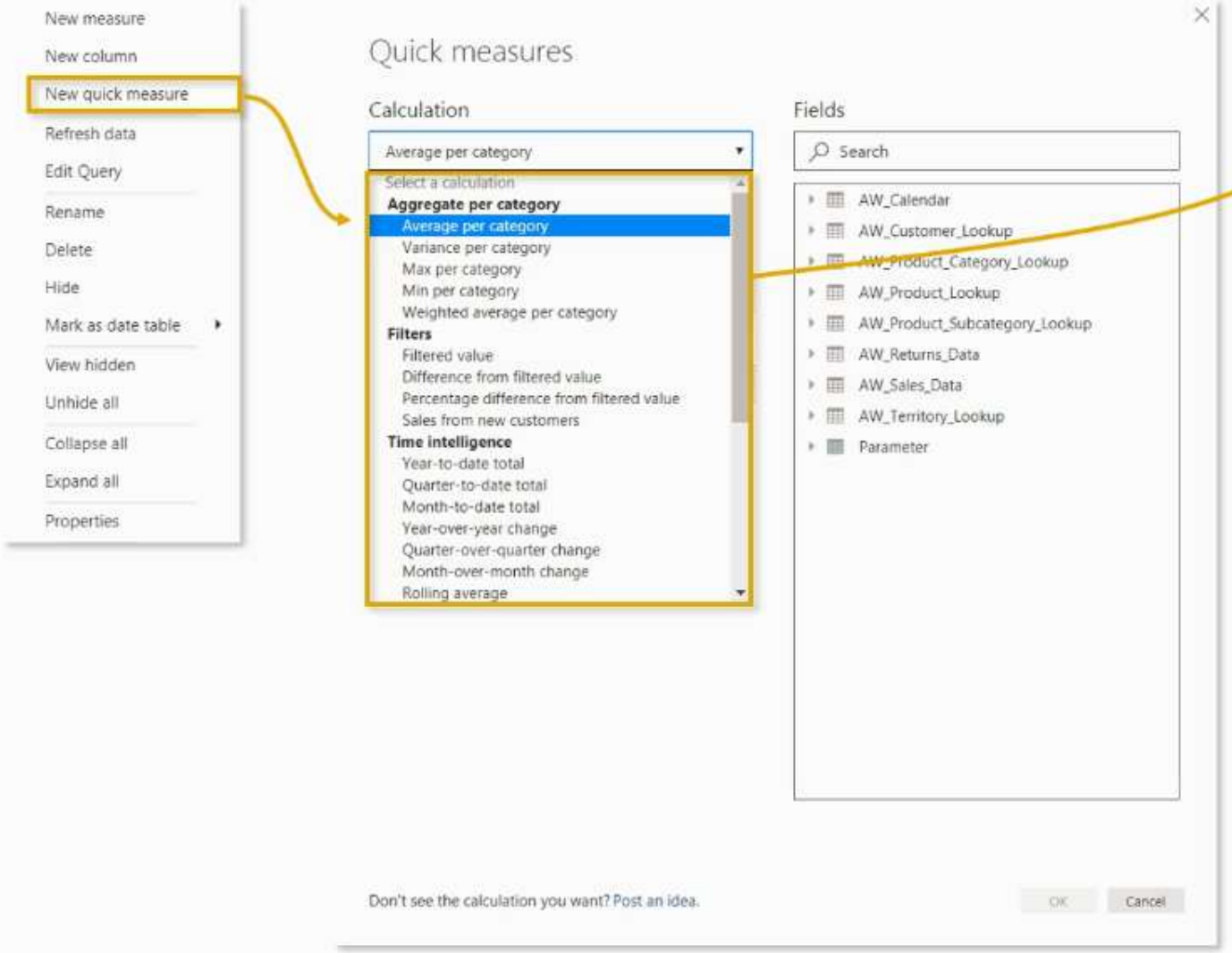
Option 2: Right-click within the **table** (in the **Data** view) or the **Field List** (in either the **Data** or **Report** view)



When you insert Columns or Measures using the **Modeling** tab (Option 1), they are assigned to whichever table is *currently selected*, or the *first table in the field list* by default

- Measures can be reassigned to new "Home" tables (under the "*Properties*" options in the **Modeling** tab), but the Option 2 allows you to be more deliberate about placing them
  - ***Note:*** *Assigning measures to specific tables doesn't have any impact on functionality – it's just a way to keep them organized*

QUICK MEASURES



Quick Measures are pre-built formula templates that allow you to drag and drop fields, rather than write DAX from scratch

While these tools can be helpful for defining more complex measures (*like weighted averages or time intelligence formulas*), they encourage laziness and don't help you understand the fundamentals of DAX

PRO TIP:

*Just say "**NO**" to quick measures*
(you're better than that)

Demo

- Go to report view tab
- Select AW_salse and then write click then

Select new measure

- Write the following DAX

```
Quantity Sold = SUM(AW_Sales[OrderQuantity])
```

- Select the new quantity sold field under

AW_Sales and drag it to value.

ProductKey	OrderQuantity
214	2099
215	1940
220	1995
223	4151
226	392
229	408
232	424
235	381
310	169
311	139
312	179
313	168
314	157
320	65
322	39
324	72
Total	84174

STEP-BY-STEP MEASURE CALCULATION

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

How *exactly* is this measure calculated?

- **REMEMBER:** This all happens *instantly* behind the scenes, every time the filter context changes

STEP 1

Filter context is detected & applied

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

Product[**CategoryName**] = "Accessories"

STEP 2

Filters flow "downstream" to all related tables

1 1

* *

STEP 3

Measure formula evaluates against the filtered table

```
Total Returns = COUNTROWS(AW_Returns_Data)
```

*Count of rows in the **AW_Returns_Data** table, filtered down to only rows where the product category is "Accessories"*

= **1,115**

DAX SYNTAX

MEASURE NAME

- ***Note:*** *Measures are always surrounded in brackets (i.e.* ***[Total Quantity]****) when referenced in formulas, so spaces are OK*

Referenced **TABLE NAME**

Referenced **COLUMN NAME**

Total Quantity: =SUM(Transactions[quantity])

Note: *This is a* ***"fully qualified"*** *column, since it's preceeded by the table name -- table names with spaces must be surrounded by* ***single quotes****:*

- *Without a space:* ***Transactions****[quantity]*
- *With a space:* ***'Transactions Table'****[quantity]*

FUNCTION NAME

- Calculated columns don't always use functions, but measures do:
  - In a **Calculated Column**, =Transactions[quantity] returns the value from the quantity column in each row (*since it evaluates one row at a time*)
  - In a **Measure**, =Transactions[quantity] will return an ***error*** since Power BI doesn't know how to translate that as a single value (*you need some sort of aggregation*)

PRO TIP:

For ***column*** *references, use the fully qualified name (i.e.* ***Table[Column]****)*
For ***measure*** *references, just use the measure name (i.e.* ***[Measure]****)*

DAX OPERATORS

Arithmetic Operator	Meaning	Example
+	Addition	2 + 7
-	Subtraction	5 – 3
*	Multiplication	2 * 6
/	Division	4 / 2
^	Exponent	2 ^ 5

Comparison Operator	Meaning	Example
=	Equal to	[City]="Boston"
>	Greater than	[Quantity]>10
<	Less than	[Quantity]<10
>=	Greater than or equal to	[Unit_Price]>=2.5
<=	Less than or equal to	[Unit_Price]<=2.5
<>	Not equal to	[Country]<>"Mexico"

Text/Logical Operator	Meaning	Example
&	Concatenates two values to produce one text string	[City] & " " & [State]
&&	Create an AND condition between two logical expressions	([State]="MA") && ([Quantity]>10)
|| (double pipe)	Create an OR condition between two logical expressions	([State]="MA") || ([State]="CT")
IN	Creates a logical OR condition based on a given list (using curly brackets)	'Store Lookup'[State] IN { "MA", "CT", "NY" }

DAX OPERATORS

Arithmetic Operator	Meaning	Example
+	Addition	2 + 7
-	Subtraction	5 – 3
*	Multiplication	2 * 6
/	Division	4 / 2
^	Exponent	2 ^ 5

Comparison Operator	Meaning	Example
=	Equal to	[City]="Boston"
>	Greater than	[Quantity]>10
<	Less than	[Quantity]<10
>=	Greater than or equal to	[Unit_Price]>=2.5
<=	Less than or equal to	[Unit_Price]<=2.5
<>	Not equal to	[Country]<>"Mexico"

Pay attention to these!

Text/Logical Operator	Meaning	Example
&	Concatenates two values to produce one text string	[City] & " " & [State]
&&	Create an AND condition between two logical expressions	([State]="MA") && ([Quantity]>10)
|| (double pipe)	Create an OR condition between two logical expressions	([State]="MA") || ([State]="CT")
IN	Creates a logical OR condition based on a given list (using curly brackets)	'Store Lookup'[State] IN { "MA", "CT", "NY" }

COMMON FUNCTION CATEGORIES

MATH & STATS Functions

*Basic **aggregation** functions as well as "**iterators**" evaluated at the row-level*

Common Examples:

- SUM
- AVERAGE
- MAX/MIN
- DIVIDE
- COUNT/COUNTA
- COUNTROWS
- DISTINCTCOUNT

Iterator Functions:

- SUMX
- AVERAGEX
- MAXX/MINX
- RANKX
- COUNTX

LOGICAL Functions

*Functions for returning information about values in a given **conditional expression***

Common Examples:

- IF
- IFERROR
- AND
- OR
- NOT
- SWITCH
- TRUE
- FALSE

TEXT Functions

*Functions to manipulate **text strings** or **control formats** for dates, times or numbers*

Common Examples:

- CONCATENATE
- FORMAT
- LEFT/MID/RIGHT
- UPPER/LOWER
- PROPER
- LEN
- SEARCH/FIND
- REPLACE
- REPT
- SUBSTITUTE
- TRIM
- UNICHAR

FILTER Functions

***Lookup** functions based on related tables and **filtering** functions for dynamic calculations*

Common Examples:

- CALCULATE
- FILTER
- ALL
- ALLEXCEPT
- RELATED
- RELATEDTABLE
- DISTINCT
- VALUES
- EARLIER/EARLIEST
- HASONEVALUE
- HASONEFILTER
- ISFILTERED
- USERELATIONSHIP

DATE & TIME Functions

*Basic **date and time** functions as well as advanced **time intelligence** operations*

Common Examples:

- DATEDIFF
- YEARFRAC
- YEAR/MONTH/DAY
- HOUR/MINUTE/SECOND
- TODAY/NOW
- WEEKDAY/WEEKNUM

Time Intelligence Functions:

- DATESYTD
- DATESQTD
- DATESMTD
- DATEADD
- DATESINPERIOD

BASIC DATE & TIME FUNCTIONS

DAY/MONTH/ YEAR()	*Returns the day of the month (1-31), month of the year (1-12), or year of a given date*	=**DAY/MONTH/YEAR**(Date)
HOUR/MINUTE/ SECOND()	*Returns the hour (0-23), minute (0-59), or second (0-59) of a given datetime value*	=**HOUR/MINUTE/SECOND**(Datetime)
TODAY/NOW()	*Returns the current date or exact time*	=**TODAY/NOW**()
WEEKDAY/ WEEKNUM()	*Returns a weekday number from 1 (Sunday) to 7 (Saturday), or the week # of the year*	=**WEEKDAY/WEEKNUM**(Date, *[ReturnType]*)
EOMONTH()	*Returns the date of the last day of the month, +/- a specified number of months*	=**EOMONTH**(StartDate, Months)
DATEDIFF()	*Returns the difference between two dates, based on a selected interval*	=**DATEDIFF**(Date1, Date2, Interval)

Demo

- Go to table view → select calendar table
- Create new column
- Write DAX formula

```
Day of week = WEEKDAY(AW_Calendar_Lookup[Date],1)
```

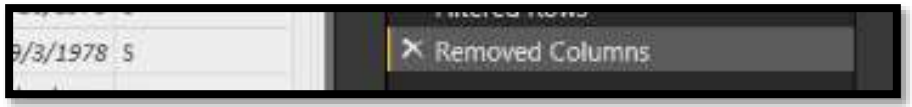
- Go to (Customer_Lookup) table (Birth Year)column , create same (birth year) column using DAX and remove the original.

```
BirthYear_CC = year(AW_Customer_Lookup[BirthDate])
```

- Also we can get same column values without using DAX functions by

```
BirthYear_CC = AW_Customer_Lookup[BirthDate].[Year]
```

- Delete (BirthYear) column generate by Query Editor and see if this step will be added.



Demo (Cont) get the age of the customers

- Create new column
- Write DAX formula to calculate the (customer age) using

```
Current Age = DATEDIFF(AW_Customer_Lookup[BirthDate],TODAY(),YEAR)
```

BASIC LOGICAL FUNCTIONS (IF/AND/OR)

IF()	*Checks if a given condition is met, and returns one value if the condition is TRUE, and another if the condition is FALSE*	=IF(LogicalTest, ResultIfTrue, *[ResultIfFalse]*)
IFERROR()	*Evaluates an expression and returns a specified value if the expression returns an error, otherwise returns the expression itself*	=IFERROR(Value, ValueIfError)
AND()	*Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE, otherwise returns FALSE*	=AND(Logical1, Logical2)
OR()	*Checks whether one of the arguments is TRUE to return TRUE, and returns FALSE if both arguments are FALSE*	=OR(Logical1, Logical2)

Note: *Use the **&&** and **||** operators if you want to include more than two conditions!*

Demo, check wither the customer is parent or not?

- Go to (AW_customer_lookup) , create new column
- Write DAX formula to check wither a customer is parent or not

```
parent = if(AW_Customer_Lookup[TotalChildren]>0,"Yes","No")
```

Demo , Identify weekend day from Day of Week column

- Go to (AW_Calender_lookup)
- Write the following to check wither the day is week end or not

```
weekend = if(or(AW_Calendar_Lookup[Day of week]=6,AW_Calendar_Lookup[Day of week]=7),"weekend","No")
```

TEXT FUNCTIONS

LEN()	*Returns the number of characters in a string*	=**LEN**(Text)
CONCATENATE()	*Joins two text strings into one*	=**CONCATENATE**(Text1, Text2)
LEFT/MID/ RIGHT()	*Returns a number of characters from the start/middle/end of a text string*	=**LEFT/RIGHT**(Text, *[NumChars]*) =**MID**(Text, StartPosition, NumChars)
UPPER/LOWER/ PROPER()	*Converts letters in a string to upper/lower/proper case*	=**UPPER/LOWER/PROPER**(Text)
SUBSTITUTE()	*Replaces an instance of existing text with new text in a string*	=**SUBSTITUTE**(Text, OldText, NewText, *[InstanceNumber]*)
SEARCH()	*Returns the position where a specified string or character is found, reading left to right*	=**SEARCH**(FindText, WithinText, *[StartPosition], [NotFoundValue]*)

Note: *Use the* ***&*** *operator as a shortcut, or to combine more than two strings!*

Demo

- Go to AW_Csutomer_lookup
- Create new column
- Write DAX formula to Concatenate (prefix , First name, last name)

```
fullname_CC = AW_Customer_Lookup[Prefix] &
" " & AW_Customer_Lookup[FirstName] & " " &
AW_Customer_Lookup[LastName]
```

Demo

- Go to (AW_Calendar_lookup) table
- Write DAX formula to get a shortcut for the month name (first 3 char of each month)

```
month name _CC = left(AW_Calendar_Lookup[Month Name],3)
```

- Change the values in (month name) column to upper case

```
month name _CC = upper(left(AW_Calendar_Lookup[Month Name],3))
```

BASIC MATH & STATS FUNCTIONS

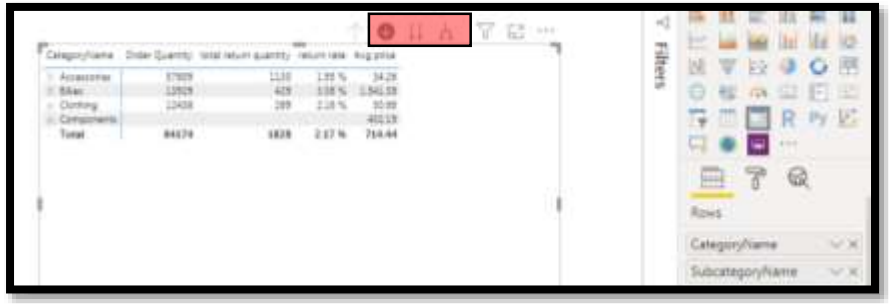
SUM()	*Evaluates the sum of a column*	=SUM(ColumnName)
AVERAGE()	*Returns the average (arithmetic mean) of all the numbers in a column*	=AVERAGE(ColumnName)
MAX()	*Returns the largest value in a column or between two scalar expressions*	=MAX(ColumnName) *or* =MAX(Scalar1, *[Scalar2]*)
MIN()	*Returns the smallest value in a column or between two scalar expressions*	=MIN(ColumnName) *or* =MIN(Scalar1, *[Scalar2]*)
DIVIDE()	*Performs division and returns the alternate result (or blank) if div/0*	=DIVIDE(Numerator, Denominator, *[AlternateResult]*)

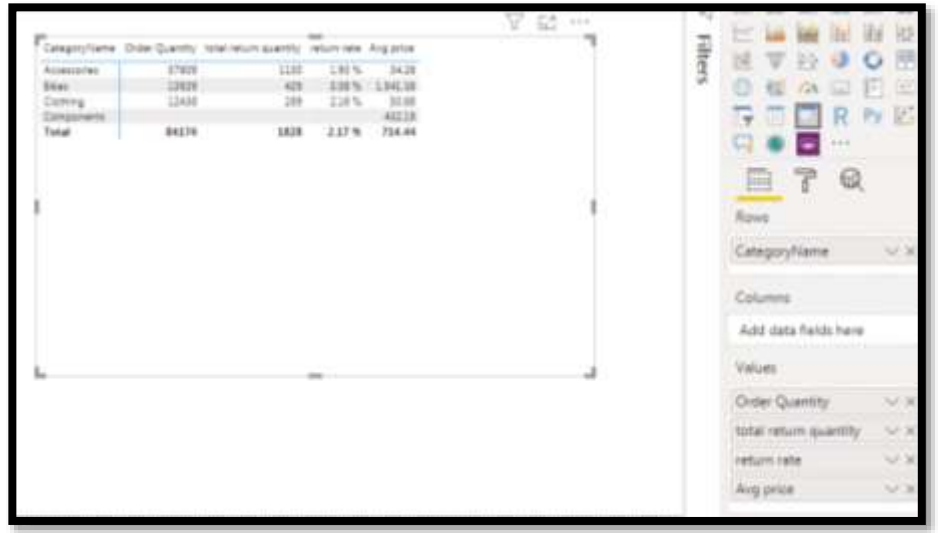
Demo : calculate the Avg of product price

- Go the report view
- Go to product table tab and create new measure.
- Write the followings DAX

```
Avg price = AVERAGE(AW_Product_Lookup[ProductPrice])
```

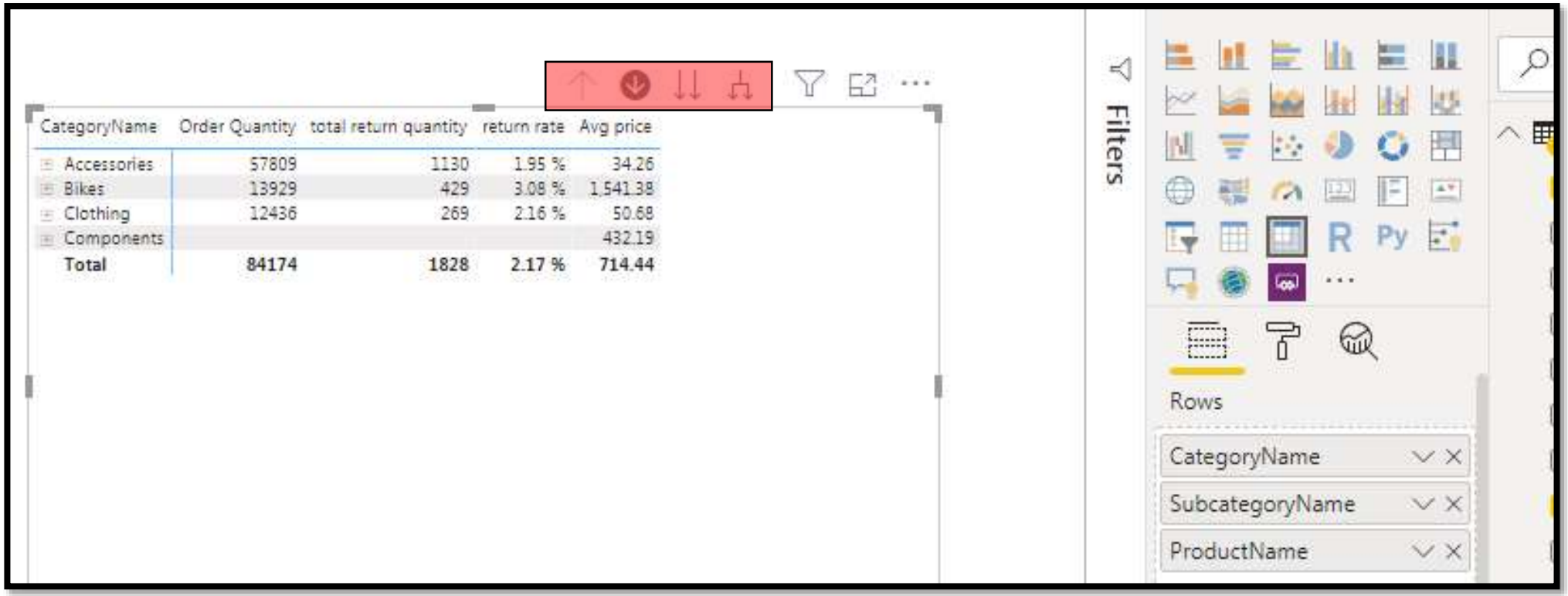
- Drage (Avg price) to the value field from the matrix
- Format the (price) of (Avg price) to currency
- Drag the (subcategory name)to the column





Demo (Cont)

- Drag the product name to the (row) field of matrix



Quiz

Q1.

- **Which two types of fields can DAX be used to create**

- ◯ Calculated Columns & Calculated Fields
- ◯ Calculated Columns & Measures
- ◯ Calculated Fields & Measures
- ◯ Measures & Calculated Items

- **What's the difference between calculated columns and measures**

  - ◯ Calculated columns understand row context
  - ◯ Measures understand filter context
  - ◯ Calculated column values are stored in tables
  - ◯ All of the above

- **Which of the following formulas would make the *most* sense as a calculated column?**

  - ◯ =SUM(Sales[quantity])
  - ◯ =AVERAGE(Products[RetailPrice])
  - ◯ =DISTINCTCOUNT(Customers[CustomerName])
  - ◯ =Products[RetailPrice] * 0.75

- **TRUE or FALSE: The *Grand Total* cell in a Power BI visualization calculates by summing the measure values in the rows above it**

True

False

- **When would you need to use the && operator?**

  - ◯ To concatenate two text strings
  - ◯ To create an AND condition between two expressions
  - ◯ To concatenate more than two text strings
  - ◯ To create an AND condition between more than two expressions

Exercise: Analyzing Data with DAX Calculations in Power BI

- Using the Adventure Works report, complete the following:
- **1)** In the **DATA** view, create the following **calculated columns**:
- In the **AW_Customer_Lookup** table, add a new column named "**Customer Priority**" that equals "*Priority*" for customers who are under 50 years old and have an annual income of greater than $100,000, and "*Standard*" otherwise
- In the **AW_Product_Lookup** table, add a new column named "**Price Point**", based on the following criteria
- *If the product price is greater than $500,* ***Price Point*** *= "High"*
  - *If the product price is between $100 and $500,* ***Price Point*** *= "Mid-Range"*
  - *If the product price is less than or equal to $100,* ***Price Point*** *= "Low"*
- In the **AW_Calendar_Lookup** table, add a new column named "**Short Day**" to extract and capitalize the first three letters from the **Day Name** column
- In the **AW_Product_Lookup** table, add a column named "**SKU Category**" to extract the first two characters from the **ProductSKU** field
  - ***BONUS:*** *Modify the* ***SKU Category*** *function to return any number of characters up to the first dash (****Hint:*** *You may need to "search" long and hard for that dash...)*

2) In the **REPORT** view, create the following **measures** (*Use a matrix visual to match the "**spot check**" values provided*)

- Create a measure named "**Product Models**" to calculate the number of unique product model names
- ***Spot check:*** *You should see a total of **119** unique product models*
- Create a measure named "**ALL Returns**" to calculate the grand total number of returns, regardless of the filter context
  - ***Spot check:*** *You should see a total of **1,809** returns*
- Create a measure to calculate "**% of All Returns**"
  - ***Spot check:*** *You should see a value of **61.64%** for the **Accessories** product category*
- Create a measure named "**Bike Returns**" to calculate total returns for bikes specifically
  - ***Spot check:*** *You should see a total of **427** bike returns*

- Create a measure named "**Total Cost**", by multiplying order quantities by product costs at the row-level
  - ***Spot check:*** *You should see a total cost of* ***$14,456,986.32***
- Once you've calculated **Total Cost**, create a new measure for "**Total Profit**", defined as the total revenue minus the total cost
- ***Spot check:*** *You should see a total profit of* ***$10,457,580.86***

- Total Cost = sumx('AW-Sales', 'AW-Sales'[OrderQuantity] * related(AW_Product_Lookup[ProductCost]))
- total profit = [total revenue] - [Total Cost]