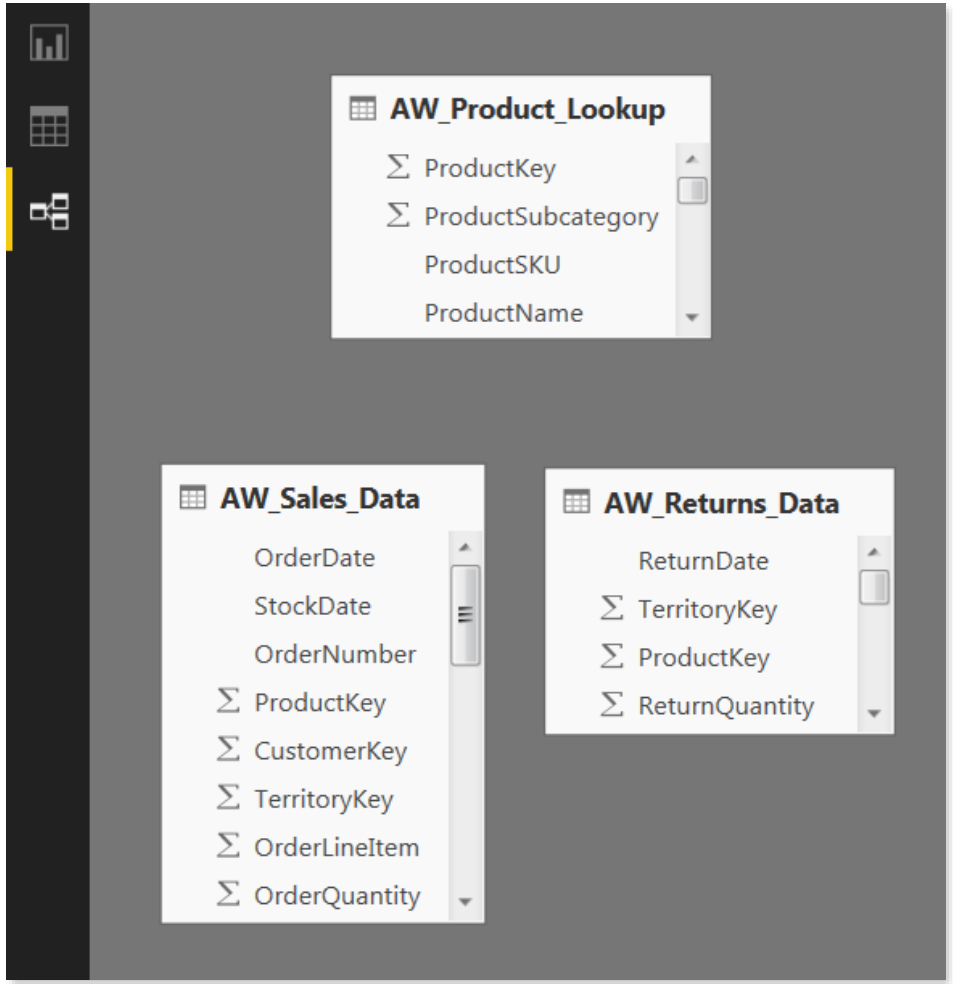
CREATING A DATA MODEL

WHAT'S A "**DATA MODEL**"?

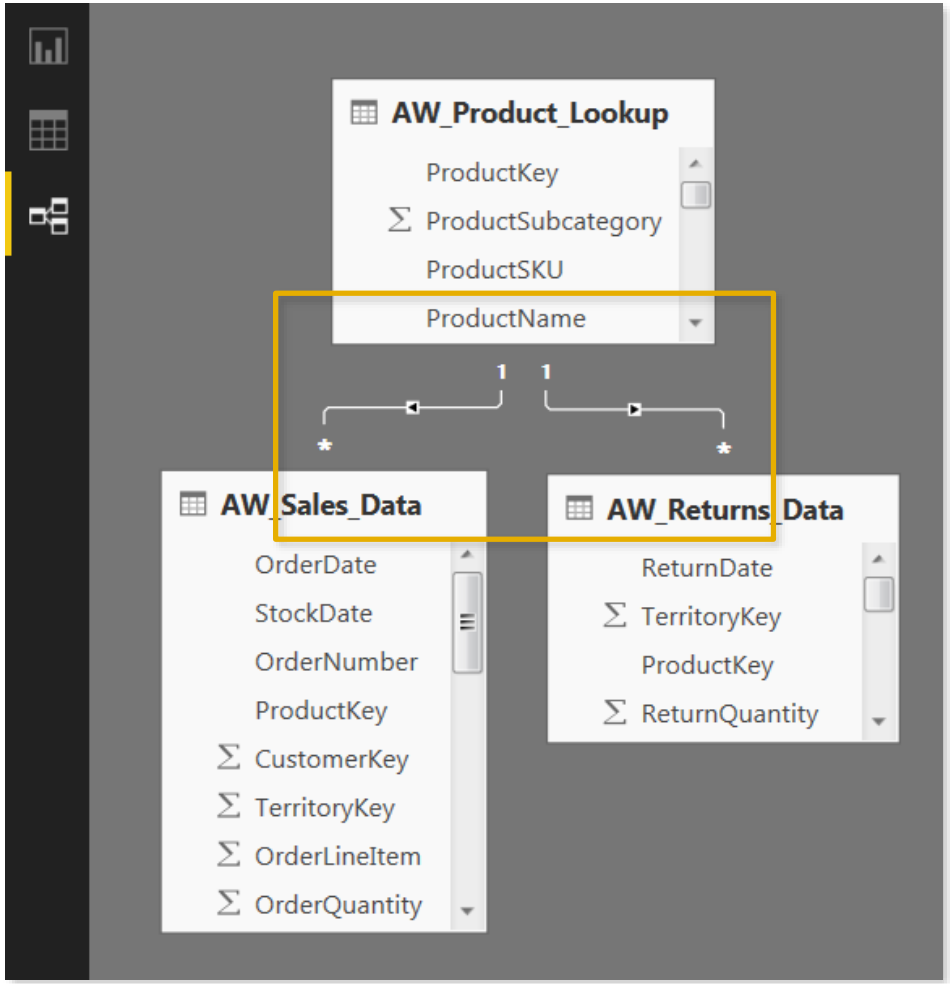


This **IS NOT** a data model ☹

- This is a collection of independent tables, which share no connections or relationships
- If you tried to visualize **Orders** and **Returns** by **Product**, this is what you'd get

ProductName	OrderQuantity	ReturnQuantity
All-Purpose Bike Stand	84,174	1,828
AWC Logo Cap	84,174	1,828
Bike Wash - Dissolver	84,174	1,828
Cable Lock	84,174	1,828
Chain	84,174	1,828
Classic Vest, L	84,174	1,828
Classic Vest, M	84,174	1,828
Classic Vest, S	84,174	1,828
Fender Set - Mountain	84,174	1,828
Total	**84,174**	**1,828**

WHAT'S A "**DATA MODEL**"?



This **IS** a data model!

- The tables are connected via relationships, based on the common *ProductKey* field
- Now the **Sales** and **Returns** tables know how to filter using fields from the **Product** table!

ProductName	OrderQuantity	ReturnQuantity
All-Purpose Bike Stand	234	8
AWC Logo Cap	4,151	46
Bike Wash - Dissolver	1,706	25
Classic Vest, L	182	4
Classic Vest, M	182	7
Classic Vest, S	157	8
Fender Set - Mountain	3,960	54
Half-Finger Gloves, L	840	18
Half-Finger Gloves, M	918	16
Total	**84,174**	**1,828**

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_weigh
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

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*

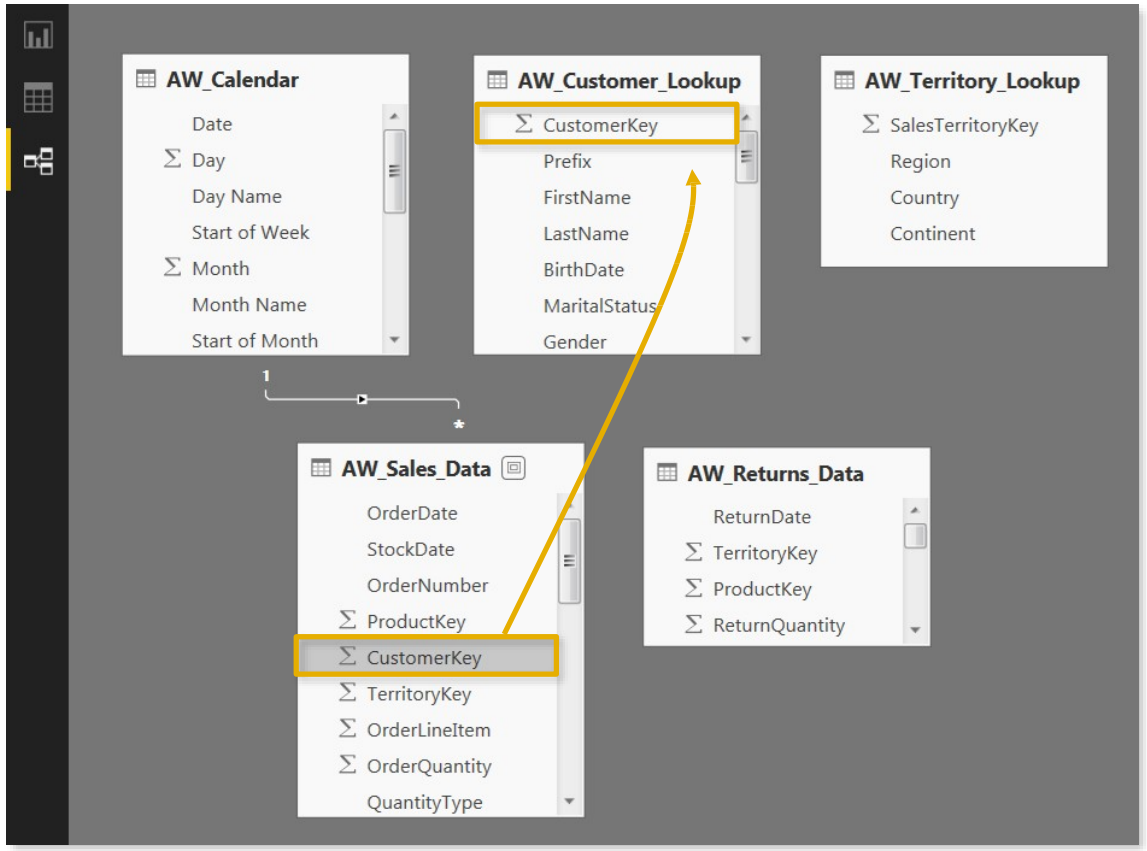
date	product_id	quantity	day_of_month	month	year	weekday	month_name	quarter	product_brand	product_name	product_sku	product_weight
1/1/1997	869	5	1	1	1997	Wednesday	January	Q1	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/7/1997	869	2	7	1	1997	Tuesday	January	Q1	Nationeel	Nationeel Grape Fruit 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!***

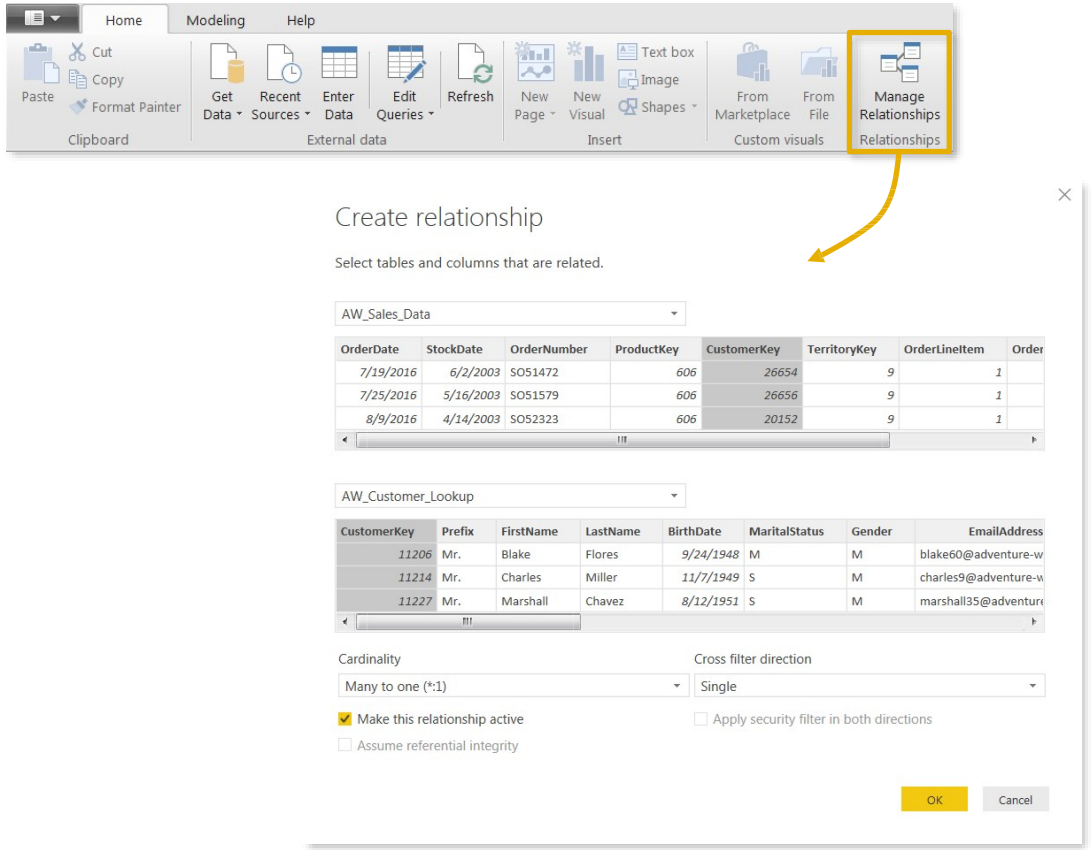
- Merging data in this way creates **redundant data** and utilizes **significantly more memory and processing power** than creating relationships between multiple small tables

CREATING TABLE RELATIONSHIPS

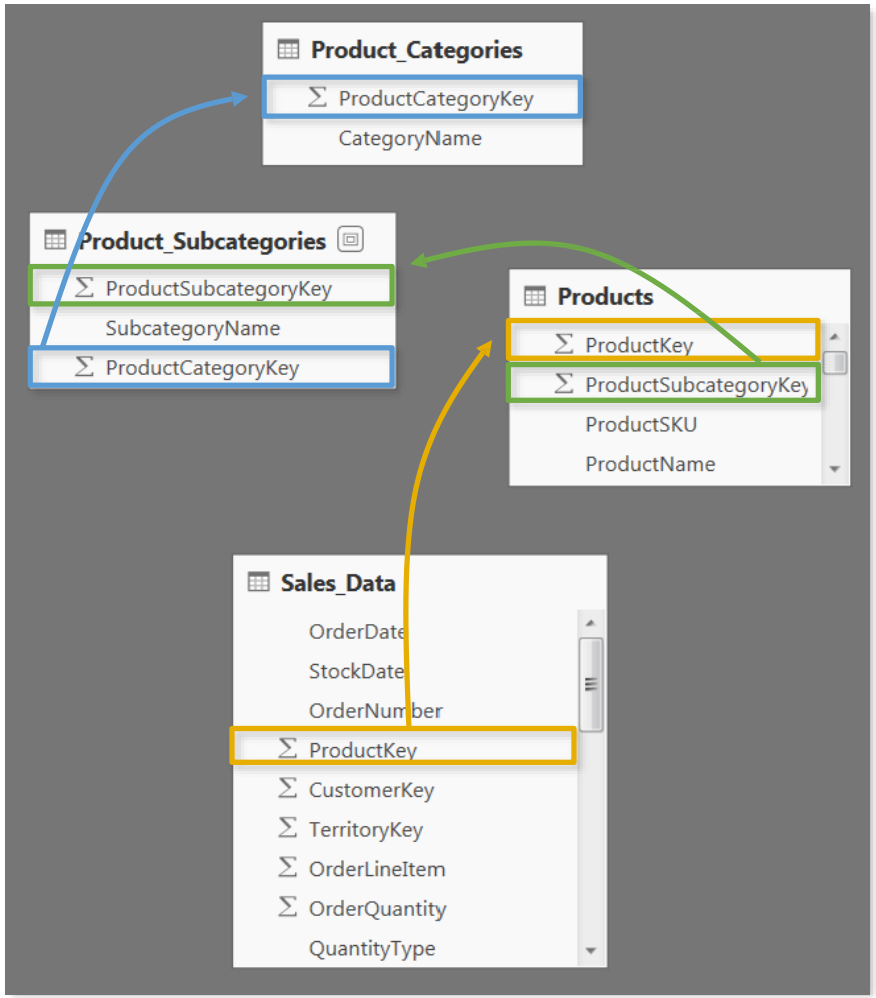
Option 1: Click and drag to connect primary and foreign keys within the **Relationships** pane



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



CREATING “SNOWFLAKE” SCHEMAS



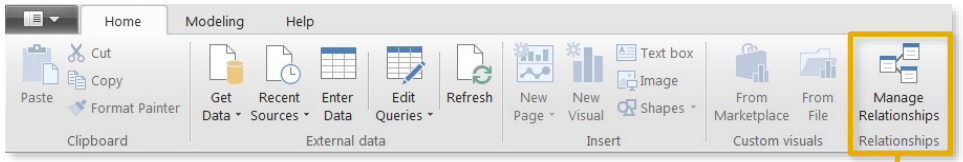
The **Sales_Data** table can connect to **Products** using the ***ProductKey*** field, but cannot connect directly to the **Subcategories** or **Categories** tables

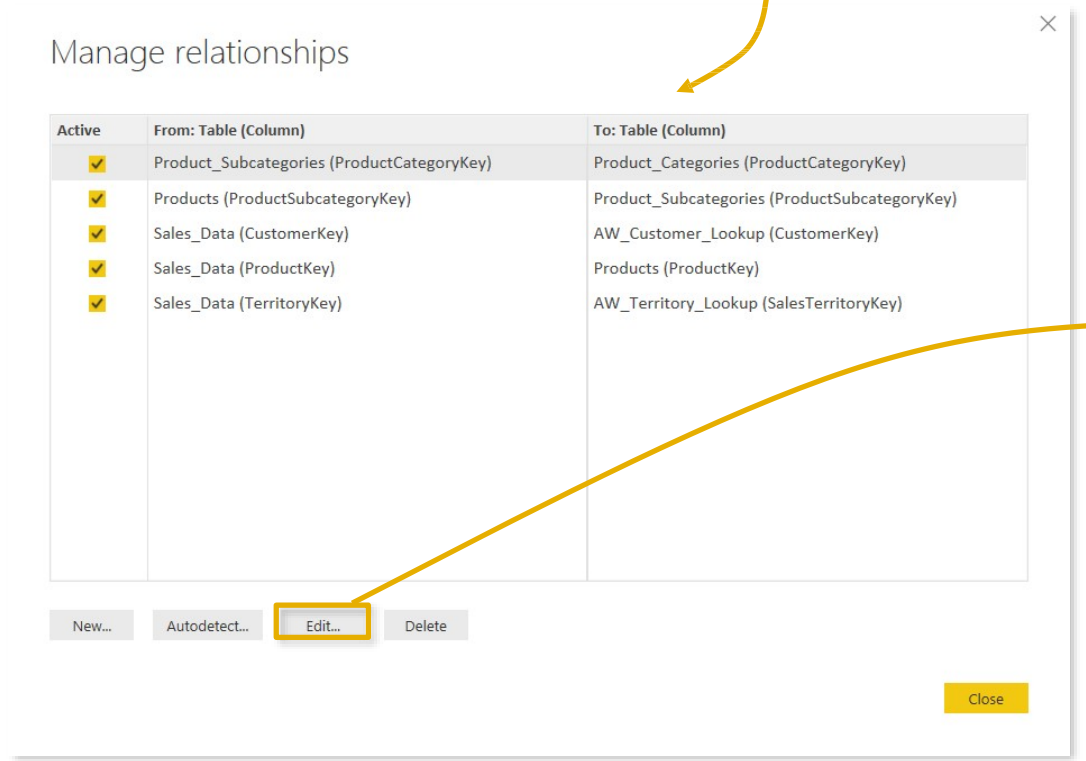
By creating relationships from **Products** to **Subcategories** (using ***ProductSubcategoryKey***) and **Subcategories** to **Categories** (using ***ProductCategoryKey***), we have essentially connected **Sales_Data** to each lookup table; filter context will now flow all the way down the chain

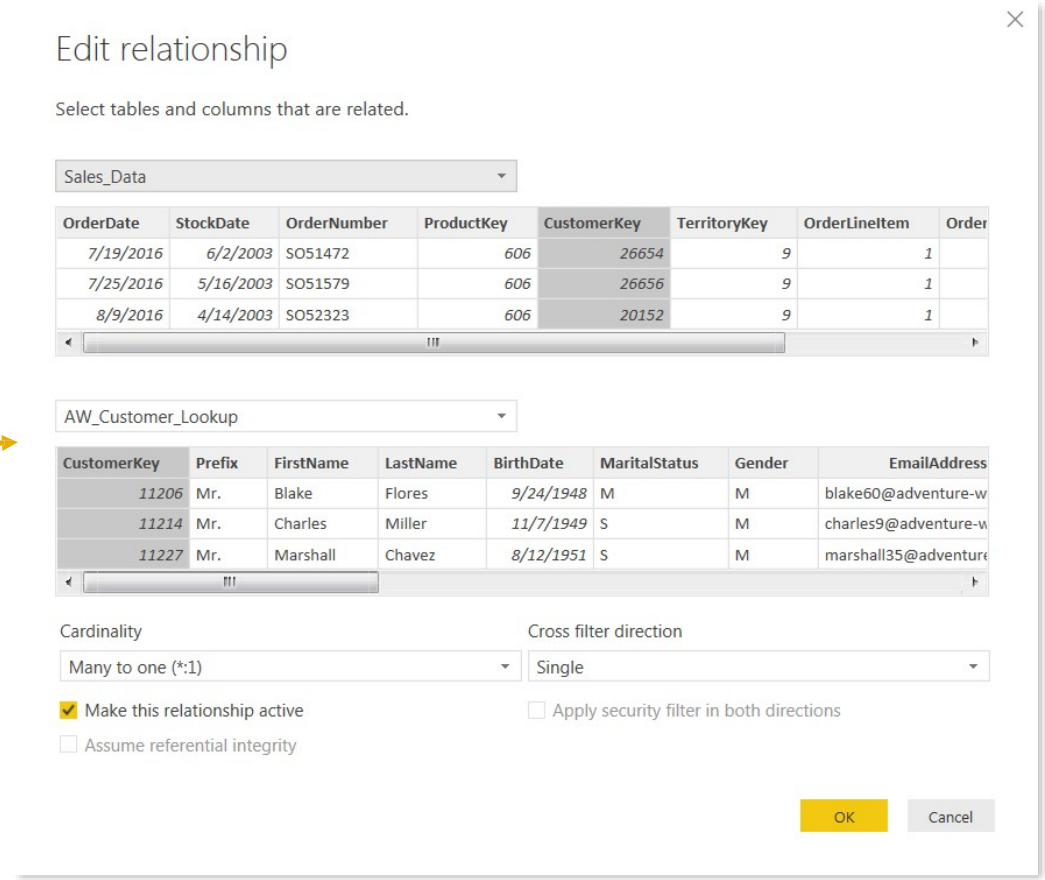
PRO TIP:

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

MANAGING & EDITING RELATIONSHIPS



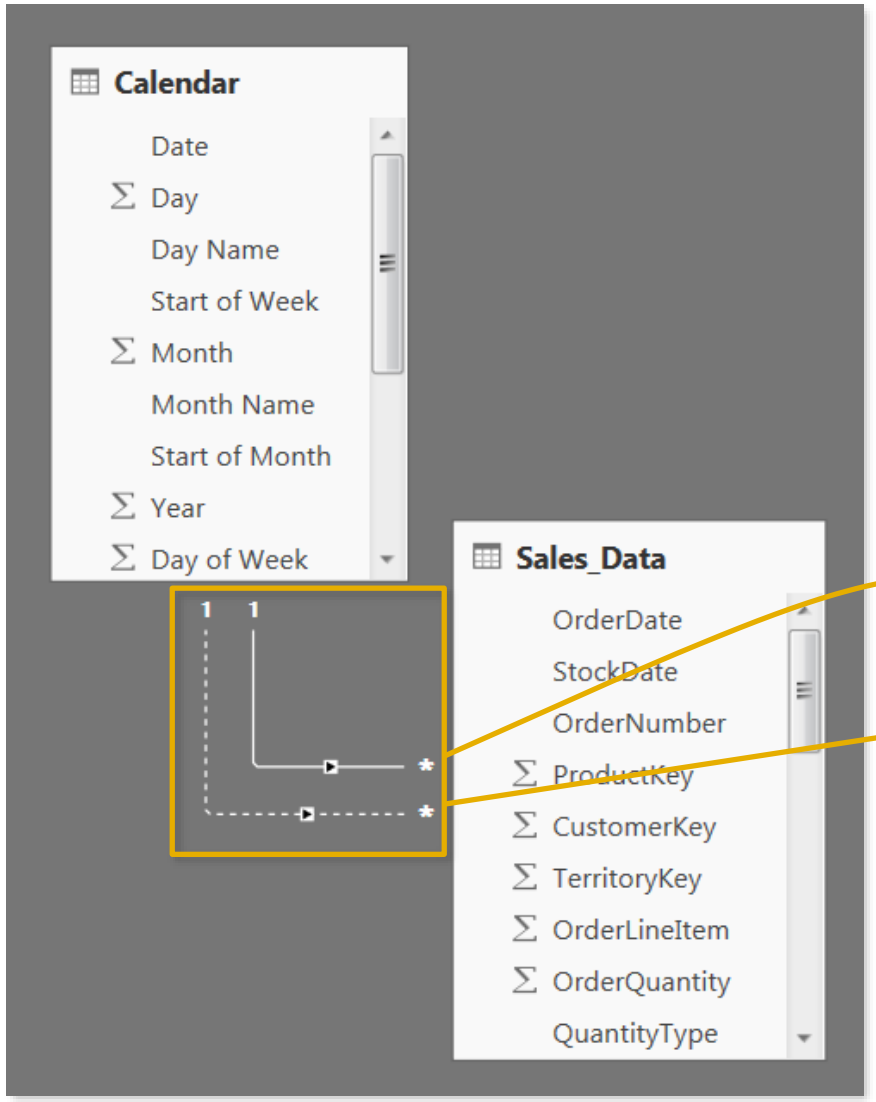


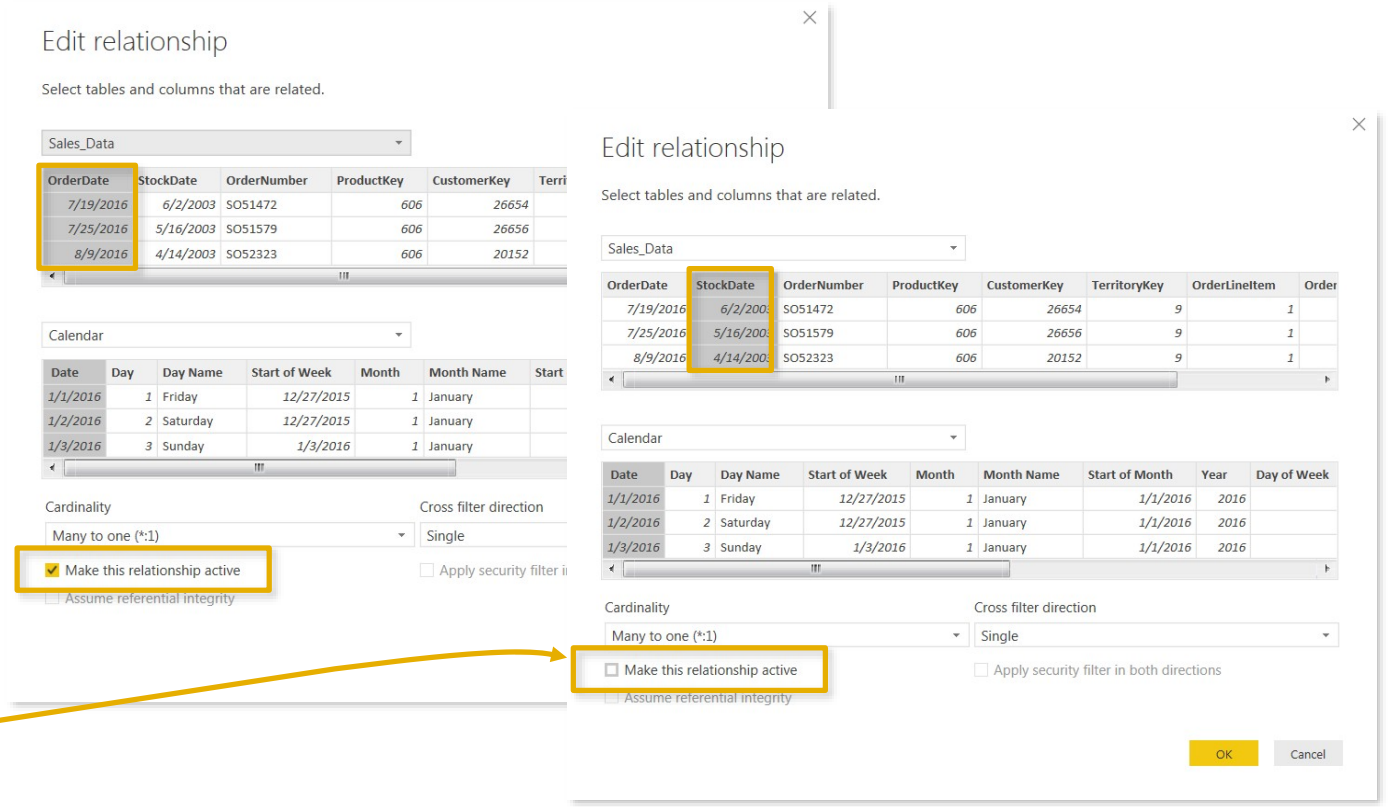


*The "**Manage Relationships**" dialog box allows you to **add**, **edit**, or **delete** table relationships*

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

ACTIVE VS. INACTIVE RELATIONSHIPS

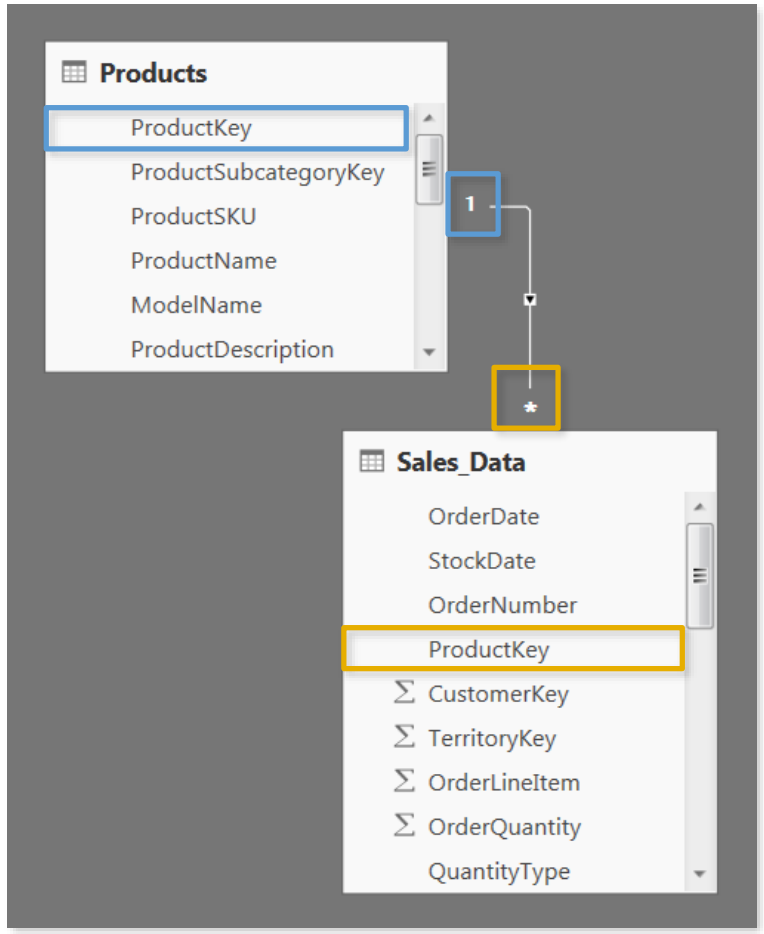




The **Sales_Data** table contains two date fields (***OrderDate*** & ***StockDate***), but there can only be one *active* relationship to the Date field in the Calendar table

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)

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*

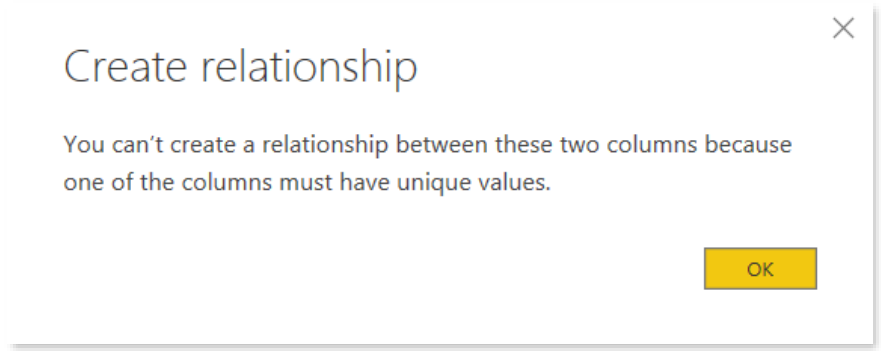
*In this case, there is only **ONE instance of each ProductKey** in the Products table (noted by the "**1**"), since each row contains **attributes of a single product** (Name, SKU, Description, Retail Price, etc)*

*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

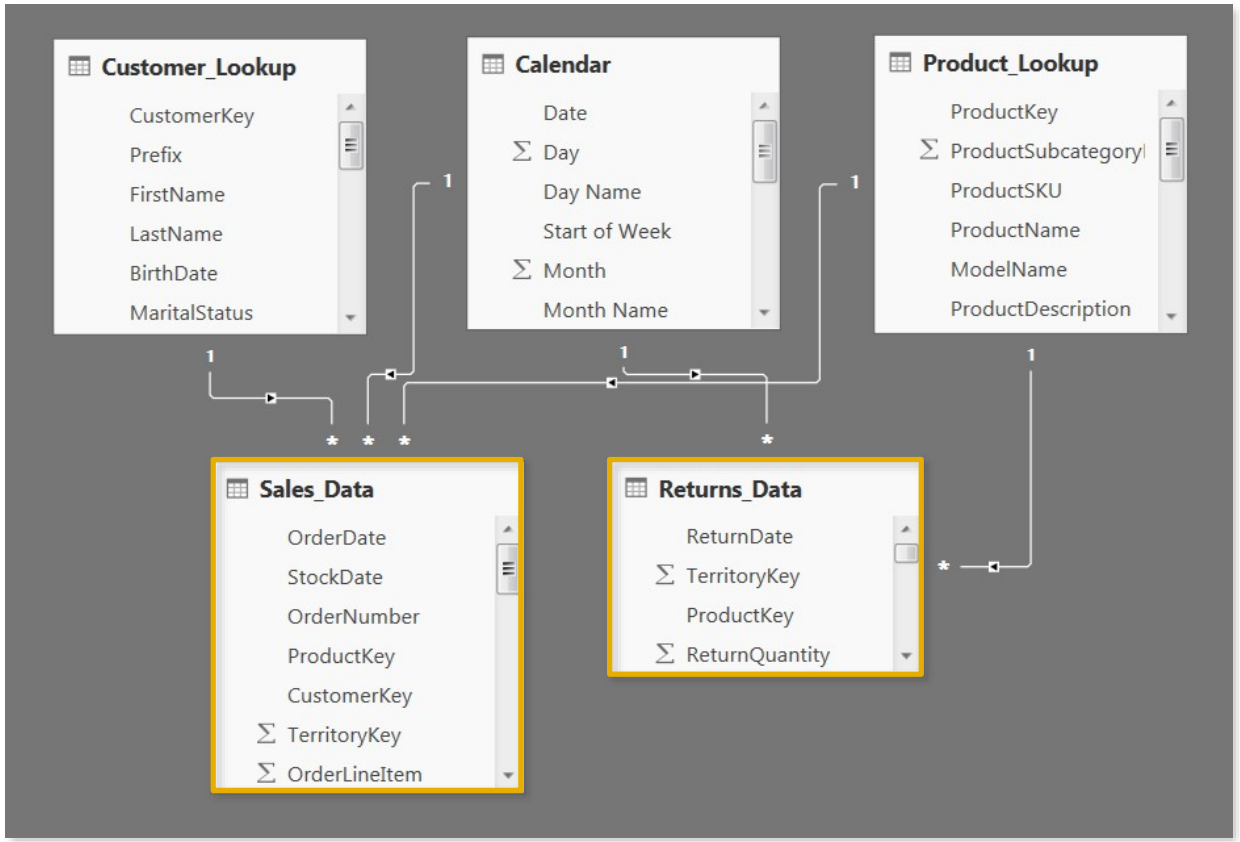
- Connecting the two tables above using the **product_id** field creates a ***one-to-one relationship***, since each ID only appears once in each table
- Unlike many-to-many, there is nothing *illegal* about this relationship; it's just **inefficient**

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



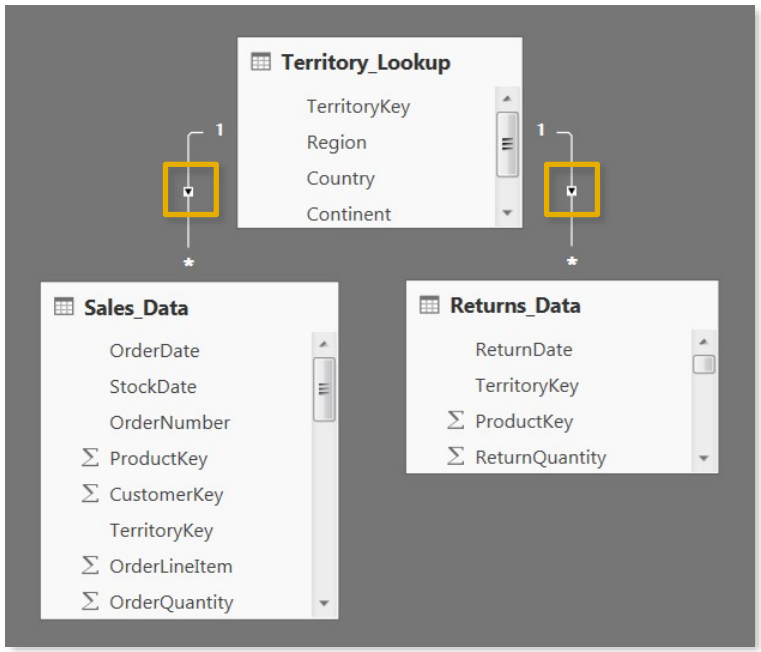
This model contains two data tables: **Sales_Data** and **Returns_Data**

- 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***

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)*

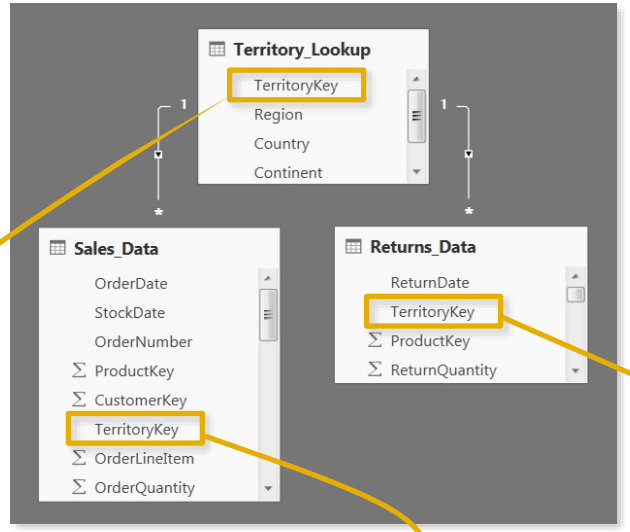
- When you filter a table, that filter context is passed along to all related "*downstream*" tables (following the direction of the arrow)
- Filters **cannot** flow "*upstream*" (against the direction of the arrow)

PRO TIP:

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

**In some cases filters may default to "two-way" depending on your Power BI Desktop settings*

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*

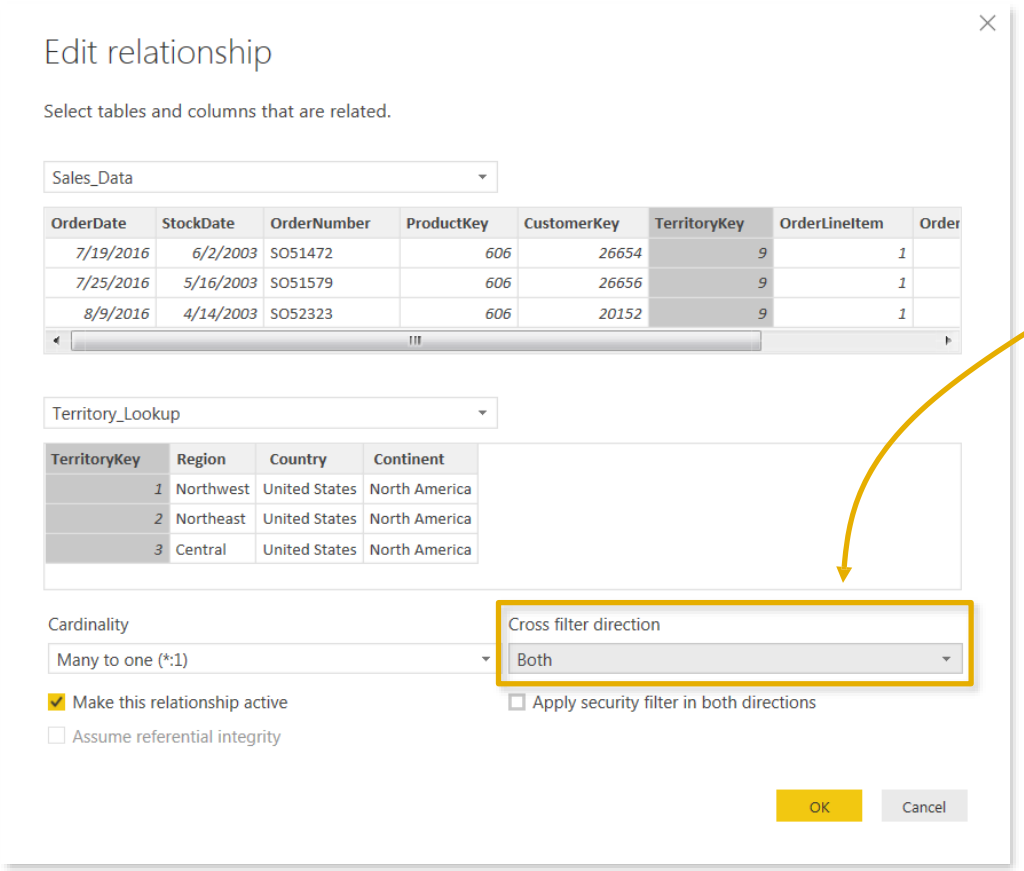
TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	1,828
2	40	1,828
3	30	1,828
4	17,191	1,828
5	49	1,828
6	894	1,828
7	7,862	1,828
8	7,950	1,828
9	17,951	1,828
10	9,694	1,828
Total	**84,174**	**1,828**

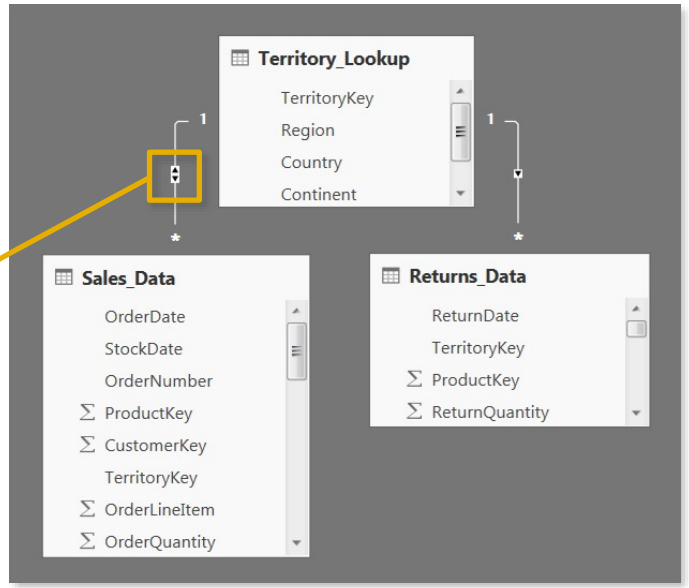
*2) Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	84,174	270
4	84,174	362
5	84,174	1
6	174	238
7	4,174	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



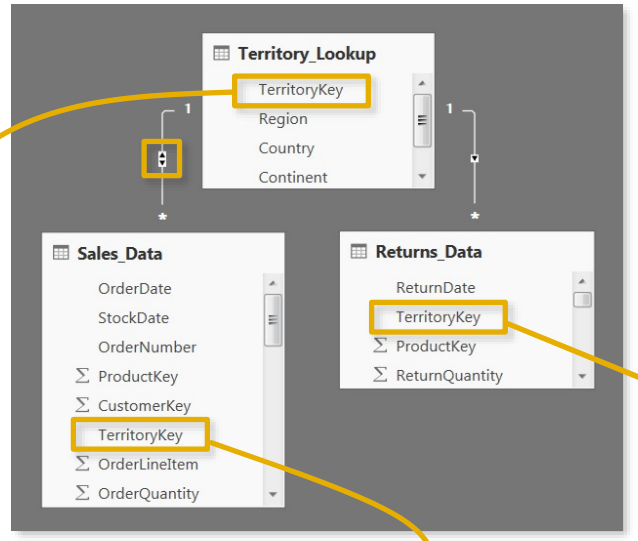


Updating the filter direction between **Sales** and **Territory** from "*Single*" to "*Both*" allows filter context to flow both ways

- 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*

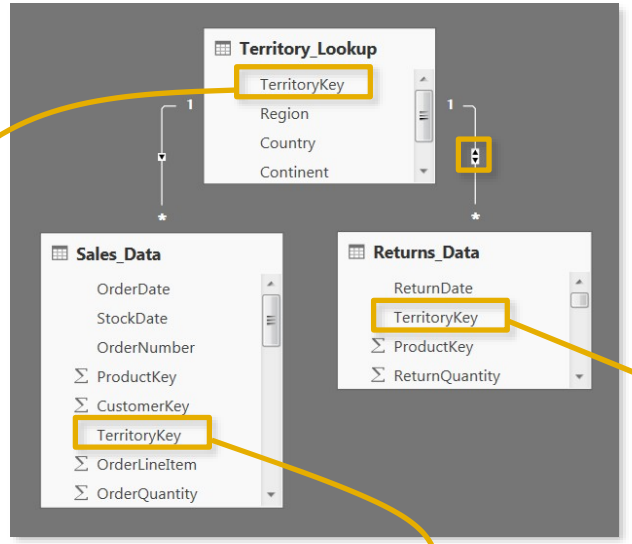
TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

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	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*

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**

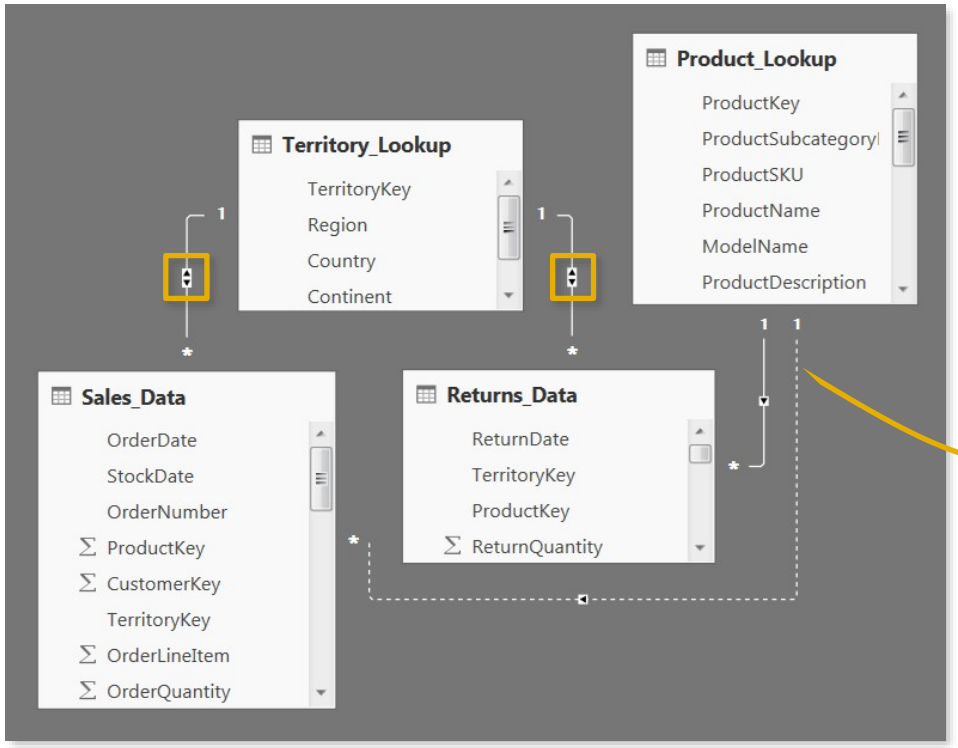
2) *Filtering using TerritoryKey from the **Sales_Data** table*

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.

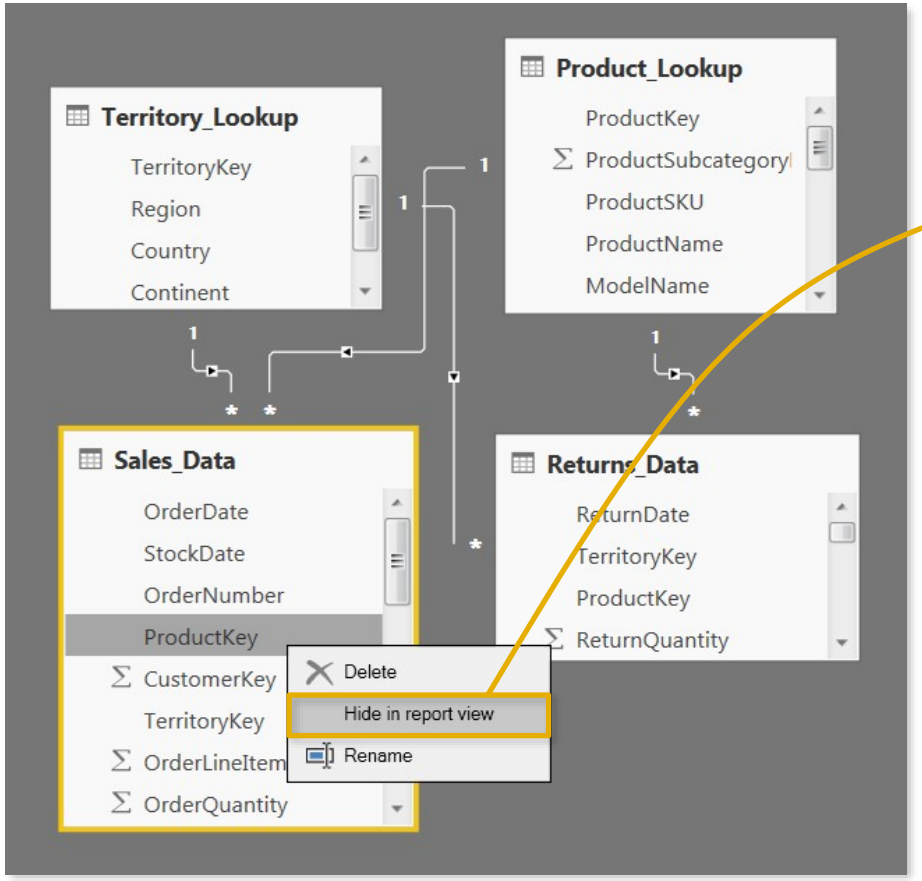
*In this model, filter context from the **Product_Lookup** table can pass down to **Returns_Data** and up to **Territory_Lookup**, which would filter accordingly based on the TerritoryKey values passed from the Returns table*

*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*