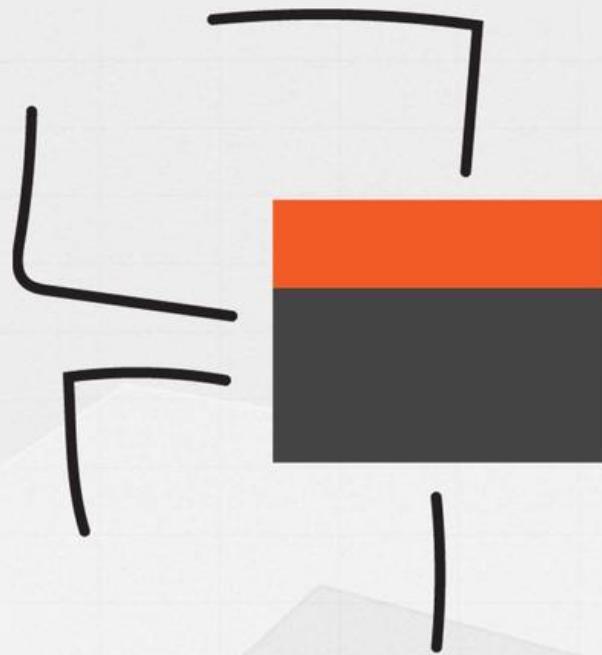


[www.sqlbi.com](http://www.sqlbi.com)

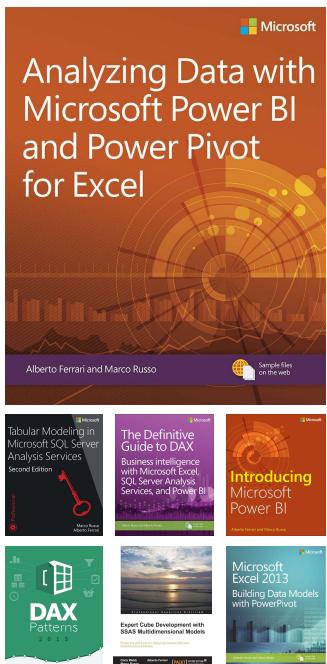




# DATA MODELING

For Power BI

We write  
**Books**



We teach  
**Courses**



We provide  
**Consulting**

- Remote Consulting
- Power BI/SSAS Optimization
- BI Architectural Review
- On-Site Consulting
- Custom Training & Mentoring

We are recognized  
**BI Experts**



# Data Modeling for Power BI

- Dimensions and facts
- Header / detail tables, multiple fact tables
- Date and time, intervals
- Using snapshots
- Many-to-many relationships
- Different granularities
- Segmentation data models, multiple currencies

# Practice

- Answer to our questions
- Guided exercises
- Apply the techniques to your data models

Why is data modeling important?

## Introduction to data modeling



# Working with a single table



- In Excel, you work with a single table
- As simple as it is, it is already a data model
- It comes with several limitations
  - Number of rows: less than 1 Million
  - Speed and memory usage are not optimal
  - Can only perform basic calculations
- The limit on size becomes a limit on the data model

# Choosing columns

- The columns of your table define its expressivity

FullDateLabel	Manufacturer	BrandName	ProductSubcategoryName	ProductCategoryName	SalesQuantity	SalesAmount	TotalCost
2007-03-31	Adventure Works	Adventure Works	Coffee Machines	Home Appliances	55	14332.268	7651.84
2008-10-22	Contoso, Ltd	Contoso	Cell phones Accessories	Cell phones	2040	23504.88	12648.94
2009-01-31	Adventure Works	Adventure Works	Televisions	TV and Video	194	51593.106	28146.4
2009-01-21	Fabrikam, Inc.	Fabrikam	Camcorders	Cameras and camcorders	282	163007.2	76709.45
2007-12-31	Adventure Works	Adventure Works	Laptops	Computers	29	14008.43	7944.32
2007-06-22	Contoso, Ltd	Contoso	Cell phones Accessories	Cell phones	680	6107.24	3420.44
2007-06-22	Proseware, Inc.	Proseware	Projectors & Screens	Computers	86	71417.6	30786.94
2007-08-23	Adventure Works	Adventure Works	Laptops	Computers	43	22672.2	9954.6
2009-03-30	The Phone Company	The Phone Company	Touch Screen Phones	Cell phones	198	48500.37	24164.56
2008-03-24	Contoso, Ltd	Contoso	Home & Office Phones	Cell phones	306	7353.594	3914.64
2007-09-30	Fabrikam, Inc.	Fabrikam	Microwaves	Home Appliances	44	4805.604	2824.24
2007-11-13	Adventure Works	Adventure Works	Desktops	Computers	153	47357.97	28256.02
2008-12-06	Contoso, Ltd	Contoso	Projectors & Screens	Computers	32	10790.4	6477.2
2007-11-14	Contoso, Ltd	Contoso	Digital SLR Cameras	Cameras and camcorders	146	55397.5	25876
2009-12-30	Adventure Works	Adventure Works	Desktops	Computers	32	15107.75	7952.97
2009-03-13	Wide World Importers	Wide World Importers	Recording Pen	Audio	42	7990.92	3607.26
2009-08-11	Wide World Importers	Wide World Importers	Recording Pen	Audio	9	1466.1	749.16
2009-09-28	Contoso, Ltd	Contoso	Microwaves	Home Appliances	78	9955.268	5189.27
2008-02-18	A. Datum Corporation	A. Datum	Digital Cameras	Cameras and camcorders	345	70989.93	32872.58
2007-08-15	Litware, Inc.	Litware	Washers & Dryers	Home Appliances	69	112603.8	56472.35

# Analytical limits



- You can slice by category and brand
- But... can you slice by color?
- No, because color does not belong to your dataset

ProductCategoryName
Audio
Cameras and camcorders
Cell phones
Computers
Games and Toys
Home Appliances
Music, Movies and Audio Bo...
TV and Video

Row Labels	Sum of SalesAmount
Adventure Works	141,178,573.89
Contoso	85,468,758.14
Fabrikam	44,940,846.17
Proseware	173,760,754.90
Southridge Video	16,092,228.97
Wide World Importers	140,433,368.67
<b>Grand Total</b>	<b>601,874,530.73</b>



Row Labels	2007	2008	2009
Black	\$59,783,936.63	\$63,073,257.64	\$70,489,997.35
Blue	\$5,128,405.12	\$6,516,691.17	\$7,715,161.42
Brown	\$6,301,722.60	\$7,183,128.75	\$4,904,738.33
Gold		\$39,185.40	\$122,755.63
Green	\$1,747,237.19	\$1,566,822.01	\$2,159,190.14
Grey	\$6,318,419.68	\$5,924,762.67	\$6,410,704.05
Orange		\$12,800.63	\$84,766.80
Pink	\$20,078.44	\$43,870.83	\$228,526.86
Red	\$6,926,045.96	\$7,872,382.02	\$11,495,613.50
Silver	\$43,375,399.72	\$39,082,679.67	\$36,471,502.90
White	\$64,963,894.39	\$64,142,854.42	\$70,639,615.97
Yellow	\$260,512.33	\$330,165.82	\$537,704.67
<b>Grand Total</b>	<b>\$194,825,652.07</b>	<b>\$195,788,601.04</b>	<b>\$211,260,277.62</b>



# Granularity

- Granularity is the level of detail of your table
- The more the columns, the higher the granularity
- Higher granularity
  - More detailed information
  - More powerful model
  - Increase in the number of rows
- Lower granularity
  - Faster and smaller model
  - Less analytical power

# Granularity and table size

- Increasing granularity increase the size of the model
  - More columns → More rows
- You quickly hit the limit of 1M rows of Excel...

Category	Sales
Bikes	10,000
Helmets	5,000

Category	Subcategory	Sales
Bikes	Cross bikes	8,000
Bikes	Mountain bikes	2,000
Helmets	Colorful helmets	2,000
Helmets	Lightweight helmets	3,000

# Increasing the analytical power

- You can either add columns to a single table
- Or add more tables to the model
- In both scenarios, the analytical power increases

ProductCategoryName	Sum of SalesAmount	Column Labels	2007	2008	2009	Grand Total
Row Labels						
Audio			\$59,783,936.63	\$63,073,257.64	\$70,489,997.35	\$193,347,191.62
Cameras and camcorders			\$5,128,405.12	\$6,516,691.17	\$7,715,161.42	\$19,360,257.71
Cell phones			\$6,301,722.60	\$7,183,128.75	\$4,904,738.33	\$18,389,589.68
Computers				\$39,185.40	\$122,755.63	\$161,941.03
Games and Toys			\$1,747,237.19	\$1,566,822.01	\$2,159,190.14	\$5,473,249.35
Home Appliances			\$6,318,419.68	\$5,924,762.67	\$6,410,704.05	\$18,653,886.41
Music, Movies and Audio Bo...				\$12,800.63	\$84,766.80	\$97,567.43
TV and Video			\$20,078.44	\$43,870.83	\$228,526.86	\$292,476.13
Black						
Blue						
Brown						
Gold						
Green						
Grey						
Orange						
Pink						
Red						
Silver						
White						
Yellow						
Grand Total			\$194,825,652.07	\$195,788,601.04	\$211,260,277.62	\$601,874,530.73

# Introducing the data model



- Using Power Pivot or Power BI you leverage the data model
- No limit on the size of the tables
- You can define granularity at will, even with a single table
- With the data model
  - Load multiple tables in a single model
  - Build relationships between tables
  - Leverage the powerful DAX language
- But... you need to learn data modeling basics

# Scattered information



- Higher granularity is not always the best choice
- Too high is as bad as too low
- Example: yearly income repeated on every row

ProductCategoryName
□ Audio
□ Cameras and camcorders
□ Cell phones
□ Computers
□ Games and Toys
■ Home Appliances
□ Music, Movies and Audio Books
□ TV and Video

BrandName	AverageYearlyIncome
Wide World Importers	\$9,765,456.65
Proseware	\$9,586,214.41
Northwind Traders	\$2,230,398.67
Litware	\$9,170,201.49
Fabrikam	\$9,461,956.24
Contoso	\$8,307,093.90
Adventure Works	\$9,614,894.80
<b>Total</b>	<b>\$8,957,859.39</b>

# Averaging scattered information

Since the YearlyIncome is scattered, you need to rebuild the correct granularity before performing the aggregation

CorrectAverage :=

```
AVERAGEX (
    SUMMARIZE (
        Sales,
        Sales[CustomerKey],
        Sales[YearlyIncome]
    ),
    Sales[YearlyIncome]
)
```

BrandName	AverageYearlyIncome	Correct Average
Wide World Importers	\$9,765,456.65	1,035,131.95
Proseware	\$9,586,214.41	491,908.56
Northwind Traders	\$2,230,398.67	151,583.50
Litware	\$9,170,201.49	265,677.30
Fabrikam	\$9,461,956.24	361,924.73
Contoso	\$8,307,093.90	262,307.94
Adventure Works	\$9,614,894.80	535,593.62
<b>Total</b>	<b>\$8,957,859.39</b>	<b>260,183.91</b>

## Leveraging the data model

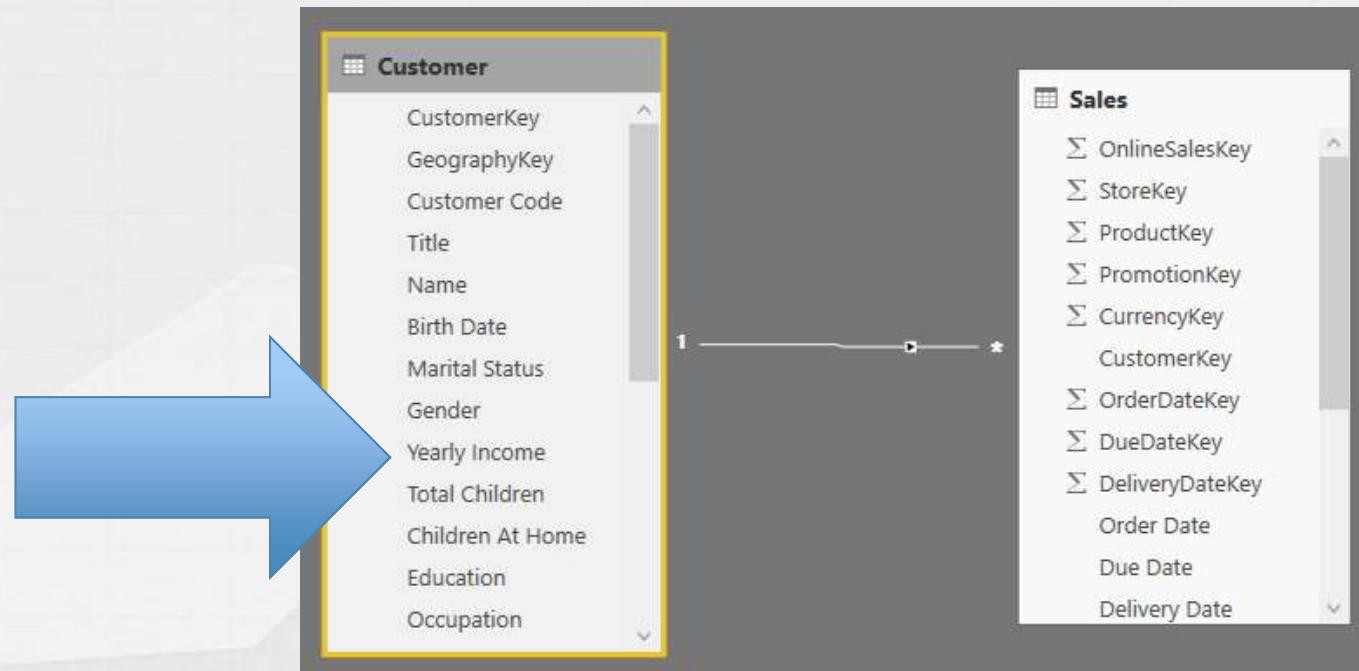


- Using the data model, you can load multiple tables
- Load Customers and Sales as separate tables
- Two tables need to be linked through a relationship
- Sales[CustomerKey] = Customer[CustomerKey]

If YearlyIncome is a customer related information, then you need a separate Customer table to store it.

# Customer is a business entity

- Being a business entity, it deserves a table by itself



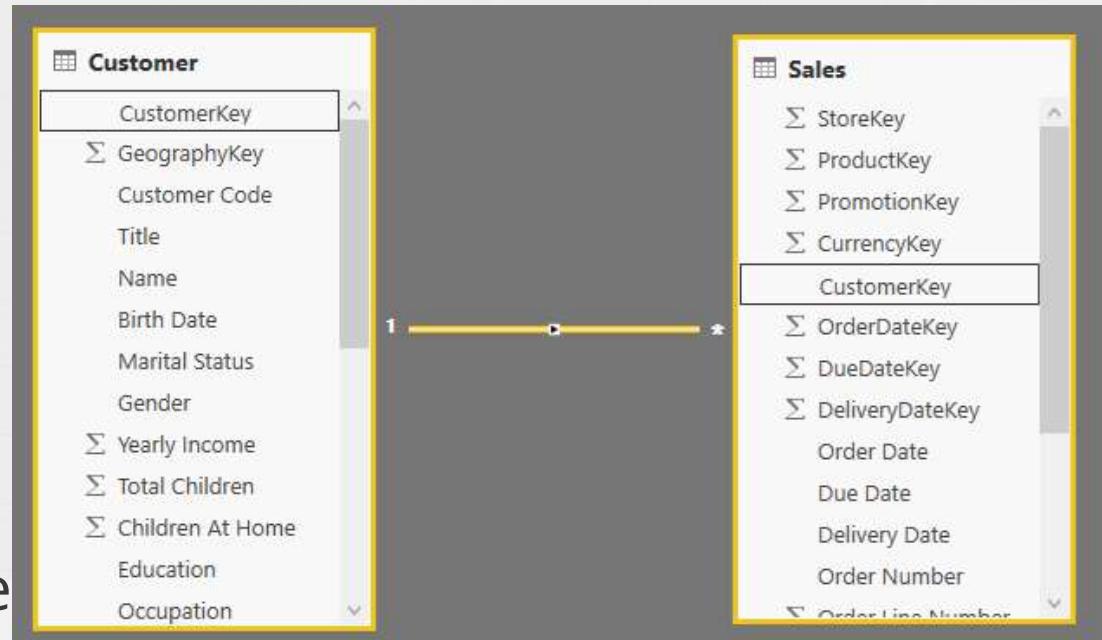
# Business Entities

- Each business has different entities
  - Products, Customers, Resellers
  - Patients, Medications, Doctors
  - Claims, Customers
  - Teams, Workers, Buildings, Projects
  - Software, Features, Bugs, Customers
- Each business entity has unique characteristics

1 Entity = 1 Table

# Relationship

- Many tables need relationships
- Links two tables
- Has a direction
  - Customer: one side
  - Sales: many side
  - Many sales, one customer
- Best practice: same column name in both tables



## Granularity with multiple tables

- With multiple tables, granularity is a different topic
- Each table has its own granularity
  - Customers: at the customer level
  - Date: at the date level
  - Product: at the product level
- Sales has granularity defined by related tables
  - Customer, Date and Product level
  - If you have those three tables
- We will come back to granularity pretty often...

Adding and removing tables is the key skill of any data modeler

## Normalization and denormalization



# Normalization vs Denormalization

- Normalization is the process of organizing the columns (attributes) and tables of a database to reduce data redundancy and improve data integrity
- Denormalization is the opposite of normalization, that is increasing data redundancy, with the goal of improving the understanding of the model
- Let us see the concept with some examples

# Normalization of a table

Manufacturer	BrandName	ProductSubcategoryName
Adventure Works	Adventure Works	Coffee Machines
Contoso, Ltd	Contoso	Cell phones Accessories
Adventure Works	Adventure Works	Televisions
Fabrikam, Inc.	Fabrikam	Camcorders
Adventure Works	Adventure Works	Laptops
Contoso, Ltd	Contoso	Cell phones Accessories
Proseware, Inc.	Proseware	Projectors & Screens
Adventure Works	Adventure Works	Laptops
The Phone Company	The Phone Company	Touch Screen Phones
Contoso, Ltd	Contoso	Home & Office Phones
Fabrikam, Inc.	Fabrikam	Microwaves
Adventure Works	Adventure Works	Desktops
Contoso, Ltd	Contoso	Projectors & Screens
Contoso, Ltd	Contoso	Digital SLR Cameras
Adventure Works	Adventure Works	Desktops
Wide World Importers	Wide World Importers	Recording Pen
Wide World Importers	Wide World Importers	Recording Pen
Contoso, Ltd	Contoso	Microwaves
A. Datum Corporation	A. Datum	Digital Cameras
Litware, Inc.	Litware	Washers & Dryers

Brand Code	Brand Name
1	Adventure Works
2	Contoso
3	Fabrikam
4	Proseware
5	The Phone Company
...	...

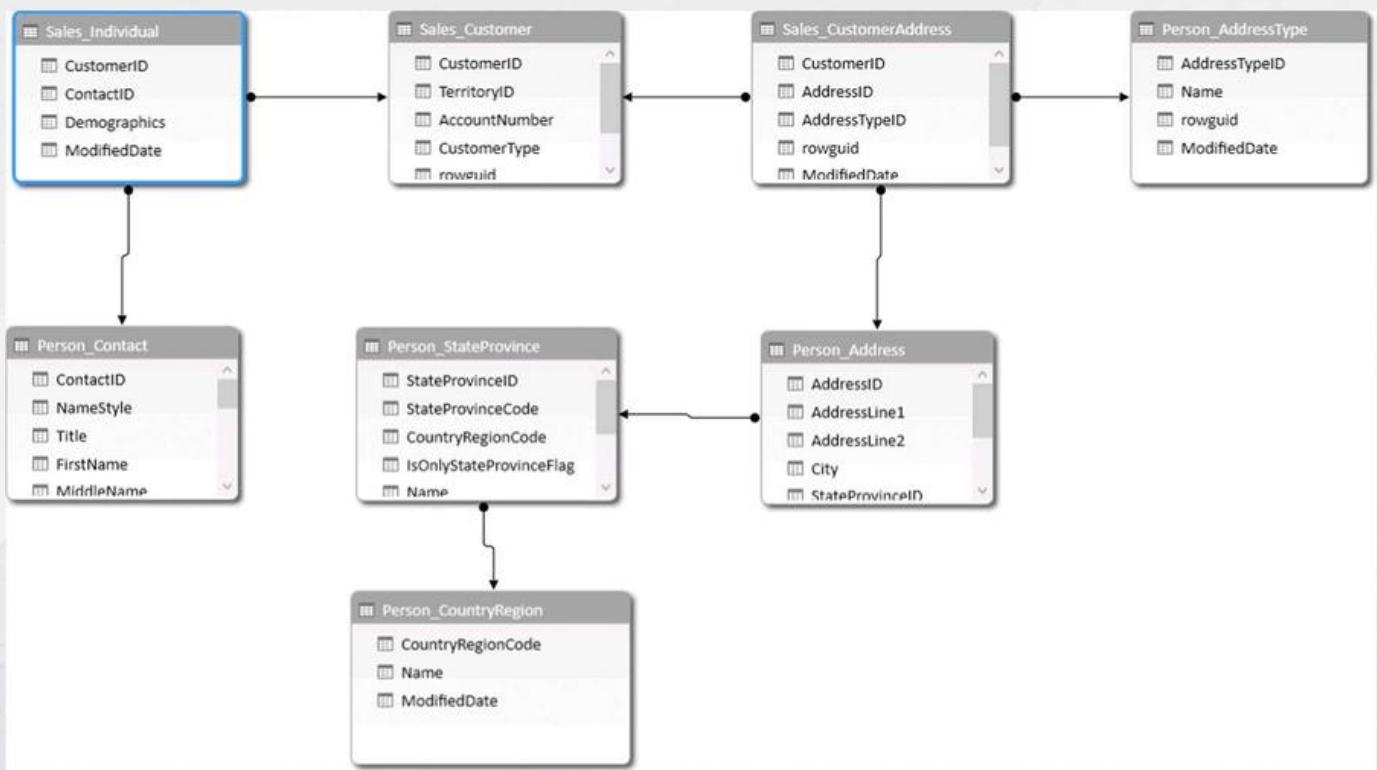
# Working with a single table

- All columns are denormalized

FullDateLabel	Manufacturer	BrandName	ProductSubcategoryName	ProductCategoryName	SalesQuantity	SalesAmount	TotalCost
2007-03-31	Adventure Works	Adventure Works	Coffee Machines	Home Appliances	55	14332.268	7651.84
2008-10-22	Contoso, Ltd	Contoso	Cell phones Accessories	Cell phones	2040	23504.88	12648.94
2009-01-31	Adventure Works	Adventure Works	Televisions	TV and Video	194	51593.106	28146.4
2009-01-21	Fabrikam, Inc.	Fabrikam	Camcorders	Cameras and camcorders	282	163007.2	76709.45
2007-12-31	Adventure Works	Adventure Works	Laptops	Computers	29	14008.43	7944.32
2007-06-22	Contoso, Ltd	Contoso	Cell phones Accessories	Cell phones	680	6107.24	3420.44
2007-06-22	Proseware, Inc.	Proseware	Projectors & Screens	Computers	86	71417.6	30786.94
2007-08-23	Adventure Works	Adventure Works	Laptops	Computers	43	22672.2	9954.6
2009-03-30	The Phone Company	The Phone Company	Touch Screen Phones	Cell phones	198	48500.37	24164.56
2008-03-24	Contoso, Ltd	Contoso	Home & Office Phones	Cell phones	306	7353.594	3914.64
2007-09-30	Fabrikam, Inc.	Fabrikam	Microwaves	Home Appliances	44	4805.604	2824.24
2007-11-13	Adventure Works	Adventure Works	Desktops	Computers	153	47357.97	28256.02
2008-12-06	Contoso, Ltd	Contoso	Projectors & Screens	Computers	32	10790.4	6477.2
2007-11-14	Contoso, Ltd	Contoso	Digital SLR Cameras	Cameras and camcorders	146	55397.5	25876
2009-12-30	Adventure Works	Adventure Works	Desktops	Computers	32	15107.75	7952.97
2009-03-13	Wide World Importers	Wide World Importers	Recording Pen	Audio	42	7990.92	3607.26
2009-08-11	Wide World Importers	Wide World Importers	Recording Pen	Audio	9	1466.1	749.16
2009-09-28	Contoso, Ltd	Contoso	Microwaves	Home Appliances	78	9955.268	5189.27
2008-02-18	A. Datum Corporation	A. Datum	Digital Cameras	Cameras and camcorders	345	70989.93	32872.58
2007-08-15	Litware, Inc.	Litware	Washers & Dryers	Home Appliances	69	112603.8	56472.35

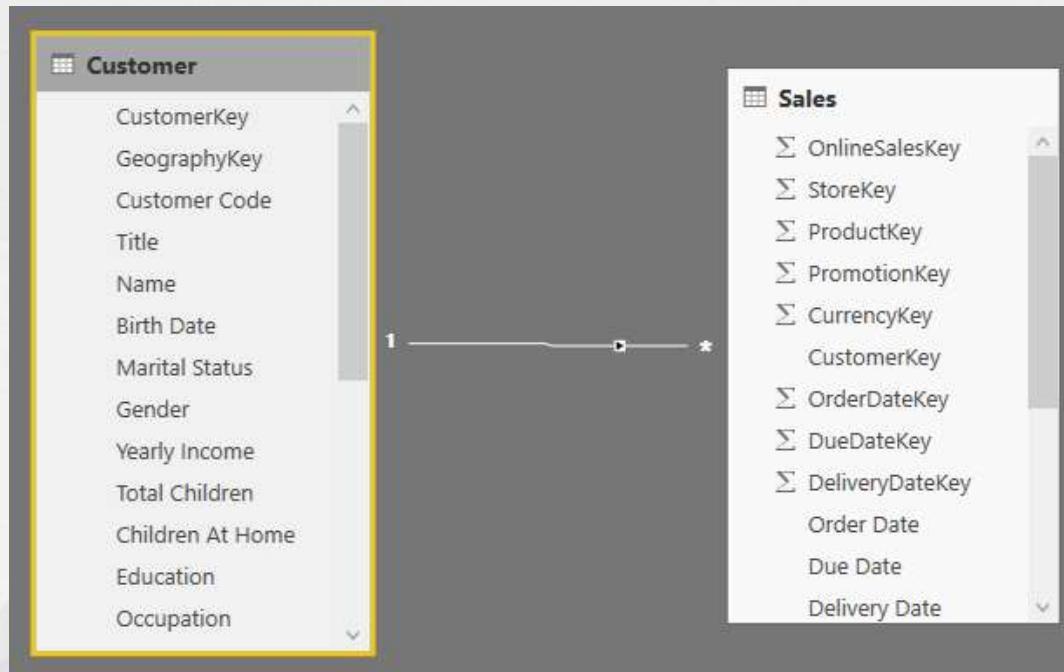
## Normalized models (OLTP)

- This whole model represents a customer in an OLTP database
- There are a lot of different tables...
- Not a good model for queries



# Denormalized model (BI model)

- Denormalization is welcome, to make the model easier



The difference between facts and dimensions is the core of star schemas and BI modeling

## Facts and dimensions



# Separation between facts and dimensions

- Different entities need different ways of handling
- Fact: *something that happened*
  - The sale of a product to a customer
  - A cash withdrawal on an ATM machine
  - The signature of an order
  - The prescription of a medical treatment
- Dimension: *something that describes a fact*
  - Attribute of a fact
  - The name of the customer, or of the patient
  - The date when the fact happened
  - The currency of the cash withdrawal

## Some important notes about facts

- The fact table is typically the largest one
  - You can have billions of sales
  - But it is unlikely you have billions of customers
- Fact are associated with numbers
  - Amount withdrawn, quantity of the product sold
- Not all numbers belong to facts
  - Price of a product: belongs to the dimension
  - Quantity of products: belongs to the fact table
- Measures are typically aggregation of numbers from the fact table
  - $\text{SUM} (\text{Sales[Quantity]})$

# Some important notes about dimensions

- Dimensions are typically small tables
  - 1 million rows, for a dimension, is already a large number
- Dimensions contain a lot of strings
  - Customer name, address, gender, education
  - Product color, size, brand, manufacturer
- You typically slice by dimensions
- When you compute something, you count them
  - Number of customers
  - Number of products
- Dimensions are not related each other

## What makes a dimension?

- One business entity = one table
- Attributes of an entity in the same table
- Customer is a business entity
  - Attributes: city, country, region, education, gender, age
- Usually Country is not an entity
  - It is an attribute of other dimensions
  - Country of customer, country of store
- Exception: demographic data
  - Measure: population (fact table)
  - Dimension: country (which is an entity in this model)

# Helper columns and tables

- Sometimes you create columns, tables or relationships that are only useful for the model, not for the user
- These are “helper objects”
  - Helper tables
  - Helper columns
  - Helper relationships
- As an example, think at the “sort by column” feature
- The sorting column is a helper column

Star schemas are the most popular way of modeling data in Business Intelligence

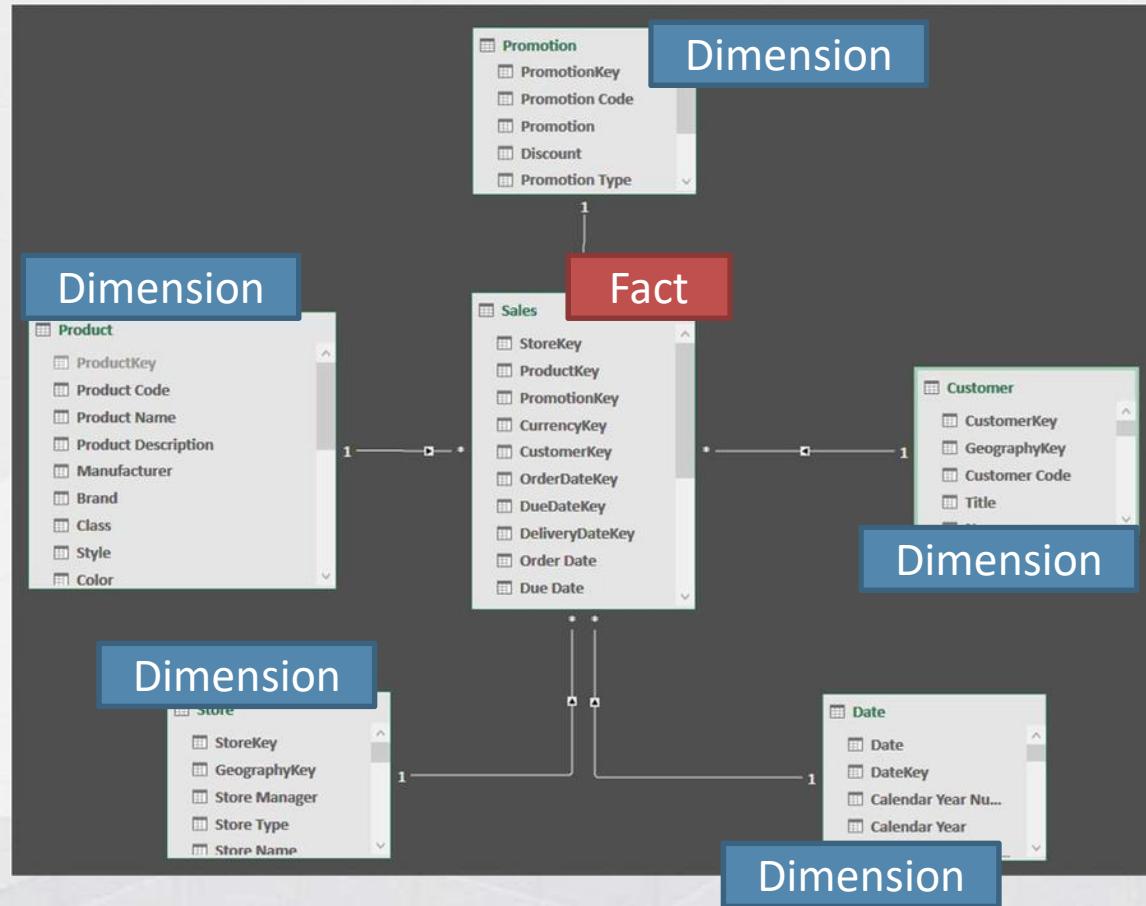
## Introducing star schemas



# Placing tables in a diagram

- Fact table
  - Stands, alone, in the center
- Dimensions
  - All around the fact table
  - Directly linked to it
- The figure that appears looks like a star
- Hence, the name: star schema

# Introducing star schemas



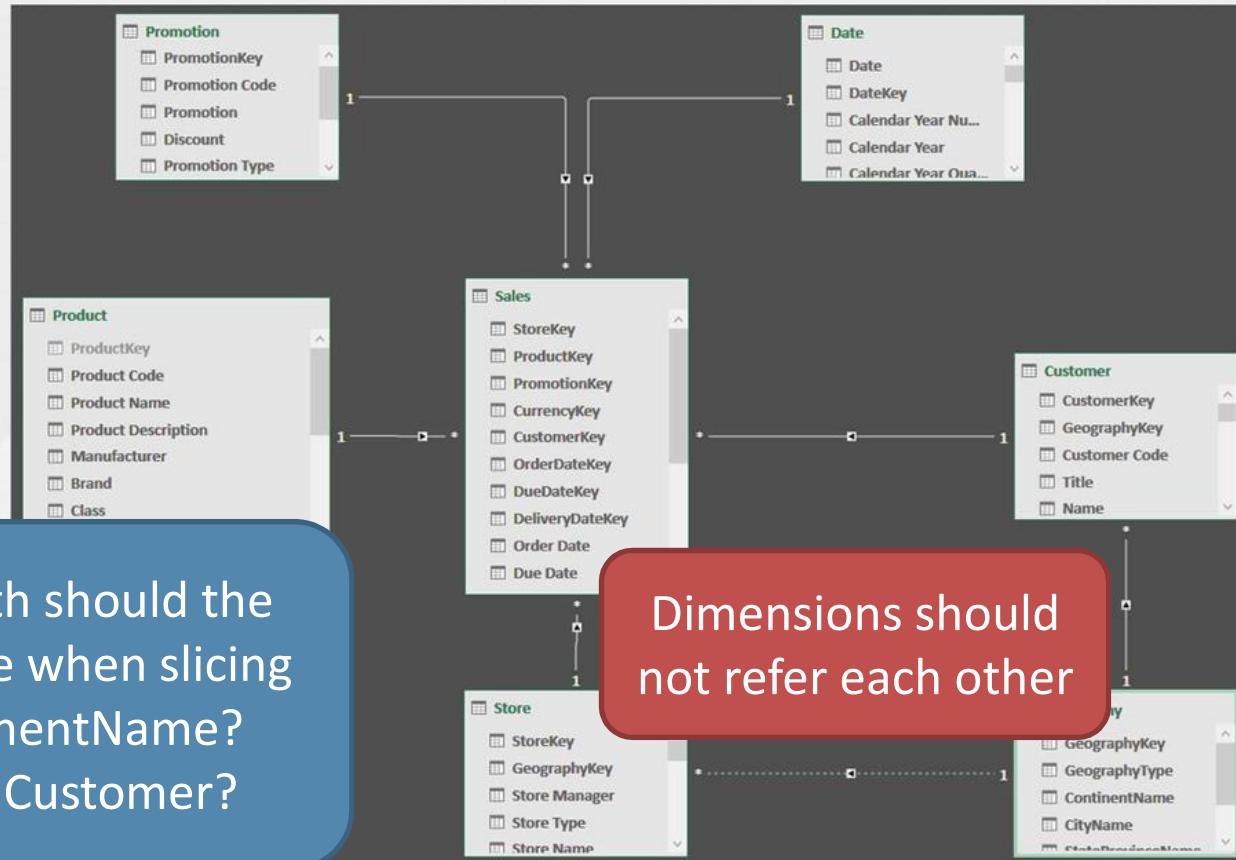
# Star schemas

- Very easy to understand at first glance
  - You slice by dimensions and aggregate facts
  - There is no ambiguity
  - One level of indirection makes it easy to see roles of tables
- Very fast
  - Modern engines are optimized for star schemas
- Drive a clean modeling path
  - Numbers go in the fact table
  - Strings go in the dimension
  - Everything else... we need to understand what it is

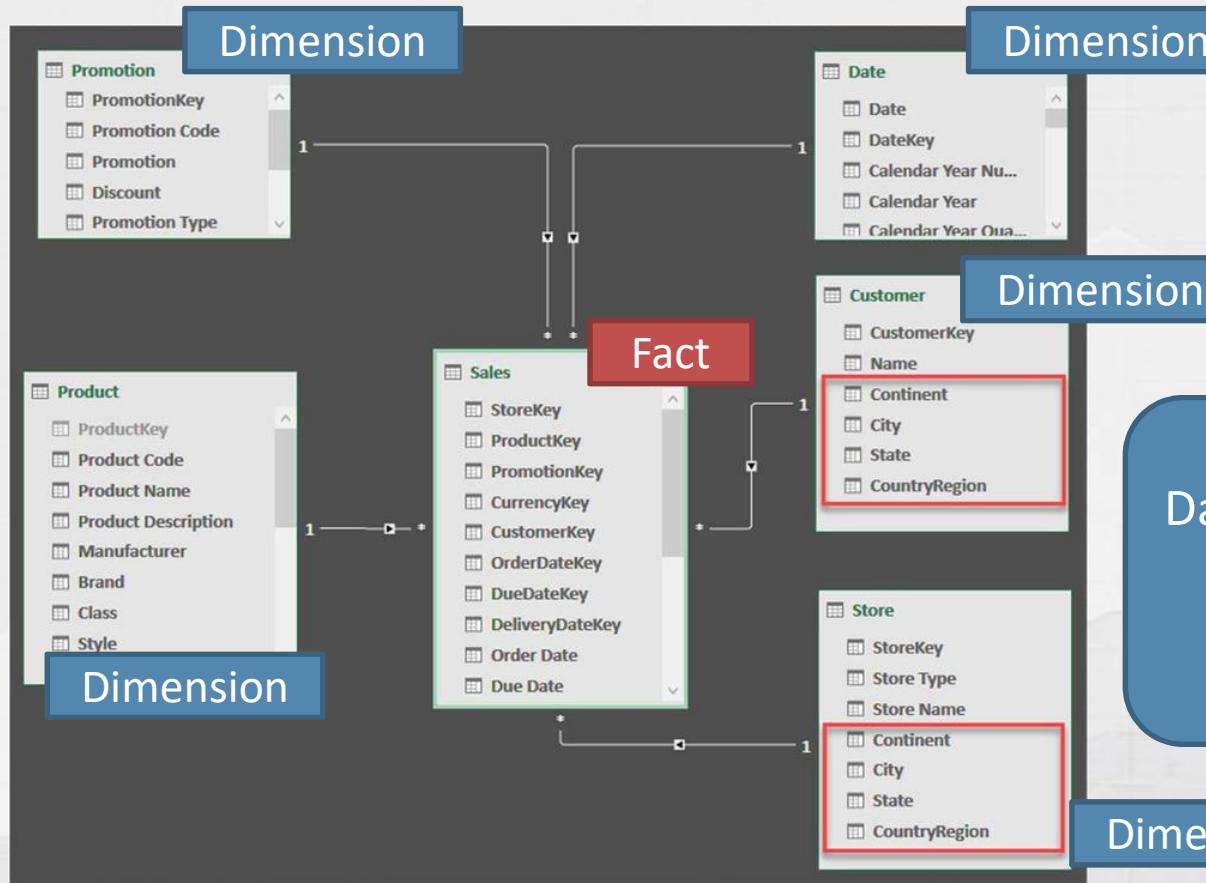
## If you don't have a star schema

- Most of the times, you are in trouble
- Any model change towards a star schema is a good step
- We will see several examples of this
- Your model is not different from all the other ones
  - As anybody else, you have a “special” model
  - With special requirements and special calculations
  - However, a star schema will fit it well!
- If you are unable to identify facts and dimensions
  - It is likely you still have to understand well the model

# Ambiguous model: not a star schema

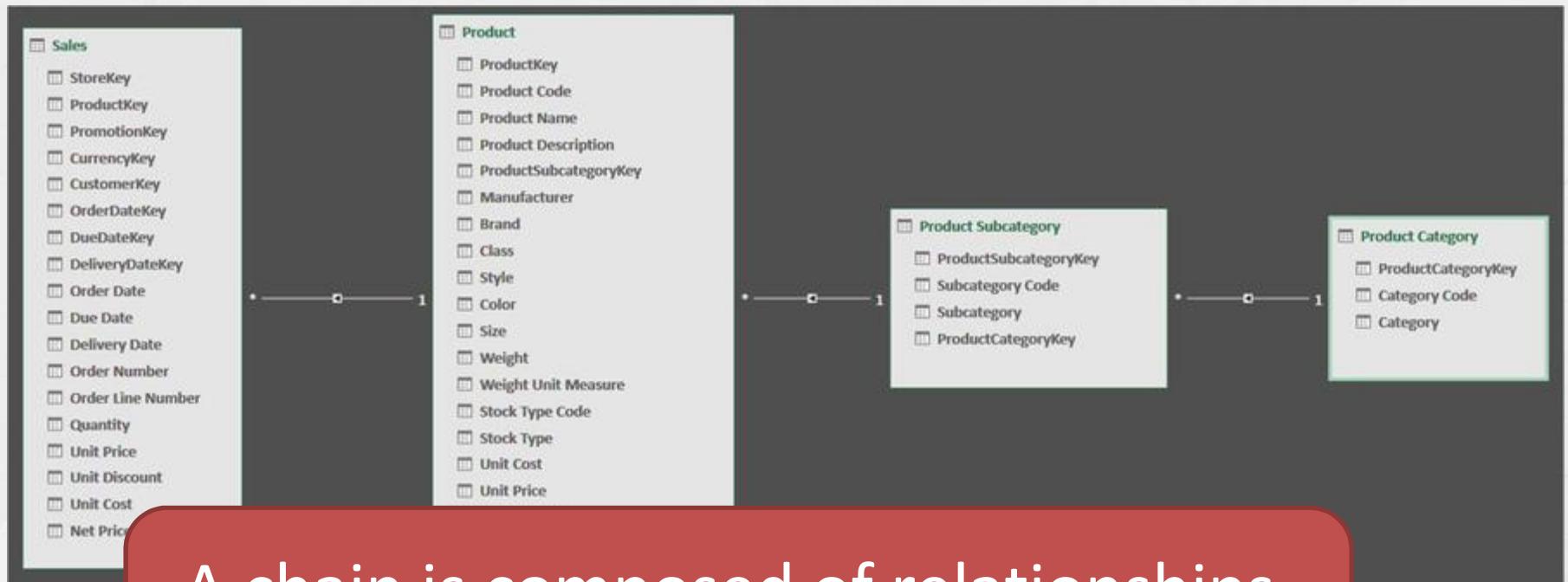


# Solving ambiguity: back to a star schema



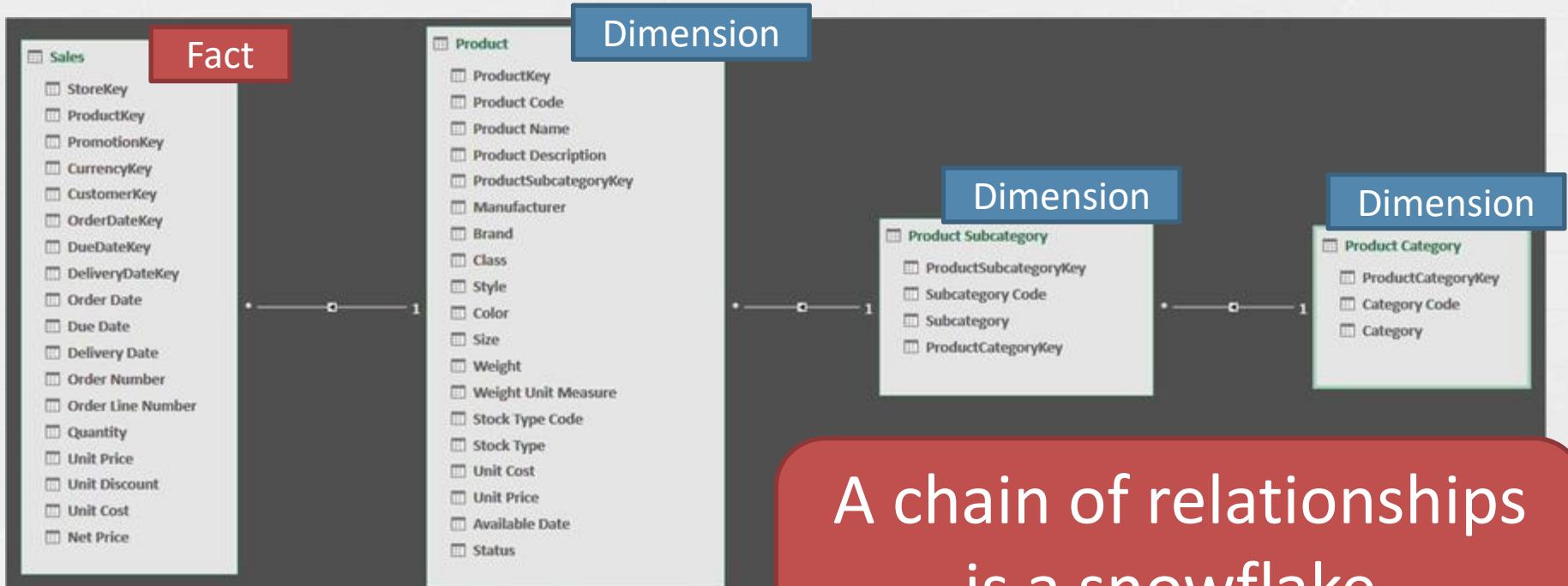
Data is duplicated, but the overall model is much simpler to use

# Chains of relationships



A chain is composed of relationships,  
all in the same direction

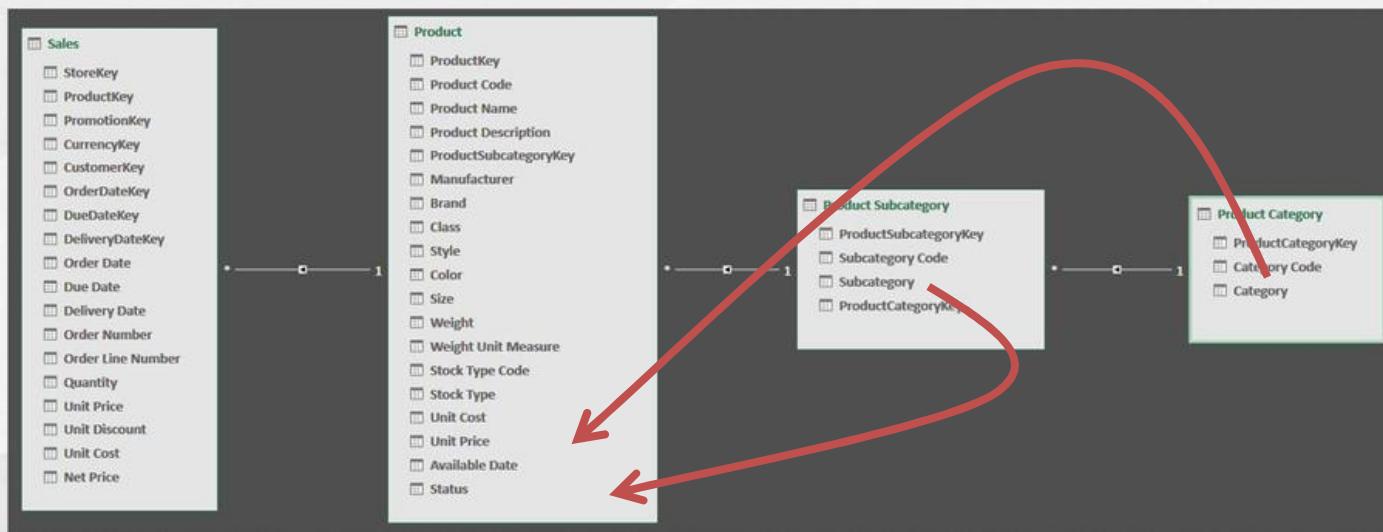
# Snowflakes



A chain of relationships  
is a snowflake.  
No ambiguity

# Avoiding snowflakes

- You can avoid a snowflake by properly denormalize values in dimensions
- Anyway, snowflakes are not bad, they only need some more attention to details



# How many dimensions?

- One entity = one dimension
  - More dimensions are often useless and confusing
- Rules of thumb:
  - Number of dimensions
    - 10 dimensions is good
    - 20 dimensions is probably wrong
    - 30 dimensions is totally wrong
  - Dimension / attribute ratio:
    - 1 dimension with 20 attributes is good
    - 20 dimensions with 1 attribute is bad

# How many fact tables?

Multiple events merged

Order No	Customer	Date	Type	Amount
1	Contoso	01/01/2017	Order	100.00
1	Contoso	01/15/2017	Ship	60.00
1	Contoso	01/15/2017	Ship	40.00
2	Imageware	01/16/2017	Order	90.00
3	Imageware	01/20/2017	Order	45.00
4	Contoso	01/22/2017	Order	25.00

Computing the orders not yet shipped in this model is a complex operation

Correct model

Orders			
Order No	Customer	Order Date	Amount
1	Contoso	01/01/2017	100.00
2	Imageware	01/16/2017	90.00
3	Imageware	01/20/2017	45.00
4	Contoso	01/22/2017	25.00

Shipments

Order No	Date	Amount
1	01/15/2017	60.00
1	01/15/2017	40.00

# Why data modeling is useful



# Why choosing a different model?

- If the model is not the right one
  - DAX code tend to be very complex
  - Formulas are hard to think at
  - Complexity turns into performance issues
- With the correct model
  - DAX code is simple, as it should be
  - Performance are great
- Building the right model requires experience

# Tasks of a data modeler

- Data modeling means
  - Knowing several patterns
  - Being able to match your model to a pattern
  - Apply the pattern
  - Adapt the small differences appearing in custom models
- You learn patterns with experience
- In this course, we present multiple patterns
- The goal is not learning them, but seeing them in action
  - Learning requires time, you will do it later

## Is your model a different one?

- At the beginning, you always feel your model is different than the standard ones
- 99.9% of the times, this is not the case
- Do not deviate from standard modeling, unless you really know what you are doing
- Business Intelligence was born in 1958
- In 60 years, we analyzed nearly any existing model
- And we found star schemas to be the best option

# Basic data modeling



Time to practice and work on some models.

In this lab you start to distinguish between dimensions and values, you discover where to store information to obtain the desired result and move the first steps in the world of data modeling.

Refer to **lab number 1** on the hands-on manual.

Let us see a first deviation from star schemas

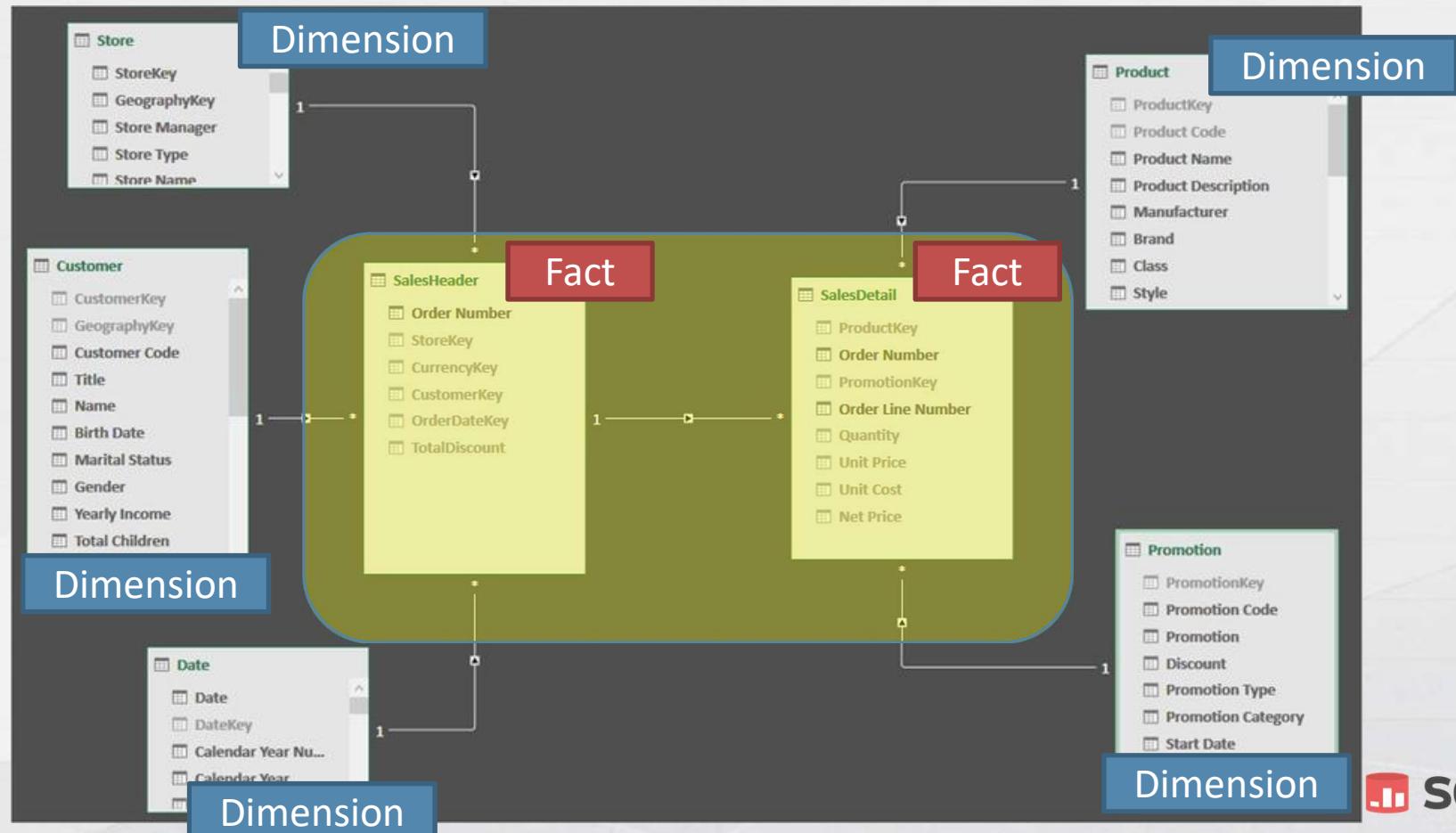
## Header / detail tables



## Introducing header/detail schemas

- Two fact tables, linked through a relationship
  - Invoices / lines of invoice
  - Orders / lines of the order
  - Teams / Individuals
- The model appears when you link fact tables
- Linking dimensions in hierarchies possible
  - Even if not a best practice
- Linking fact tables, increases the complexity and it is usually a very bad idea

# Sales headers and sales details



# Header/details issues



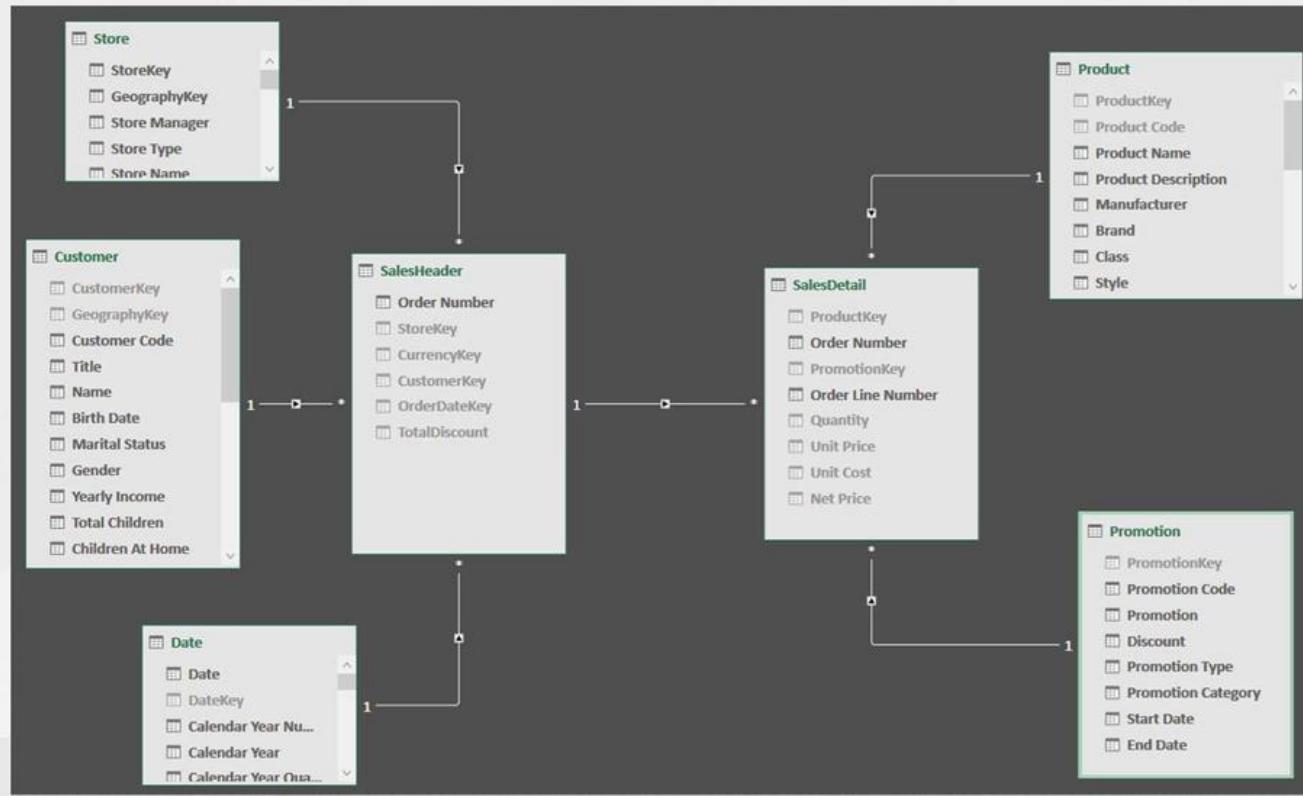
- Being two fact tables, both tables store information
- Aggregating the header produces incorrect results, if sliced by any dimension not linked to it

Continent	CY 2007	CY 2008	CY 2009	Total
Asia	\$56,282.74	\$52,457.75	\$42,562.23	<b>\$151,302.72</b>
Europe	\$52,153.03	\$25,881.80	\$32,700.45	<b>\$110,735.27</b>
North America	\$50,070.90	\$42,118.73	\$41,404.66	<b>\$133,594.29</b>
<b>Total</b>	<b>\$158,506.66</b>	<b>\$120,458.29</b>	<b>\$116,667.34</b>	<b>\$395,632.29</b>

Color	CY 2007	CY 2008	CY 2009	Total
Azure	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Black	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Blue	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Brown	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Gold	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Green	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Grey	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Orange	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
Pink	\$158,506.66	\$120,458.29	\$116,667.34	<b>\$395,632.29</b>
<b>Total</b>	<b>\$158,506.66</b>	<b>\$120,458.29</b>	<b>\$116,667.34</b>	<b>\$395,632.29</b>



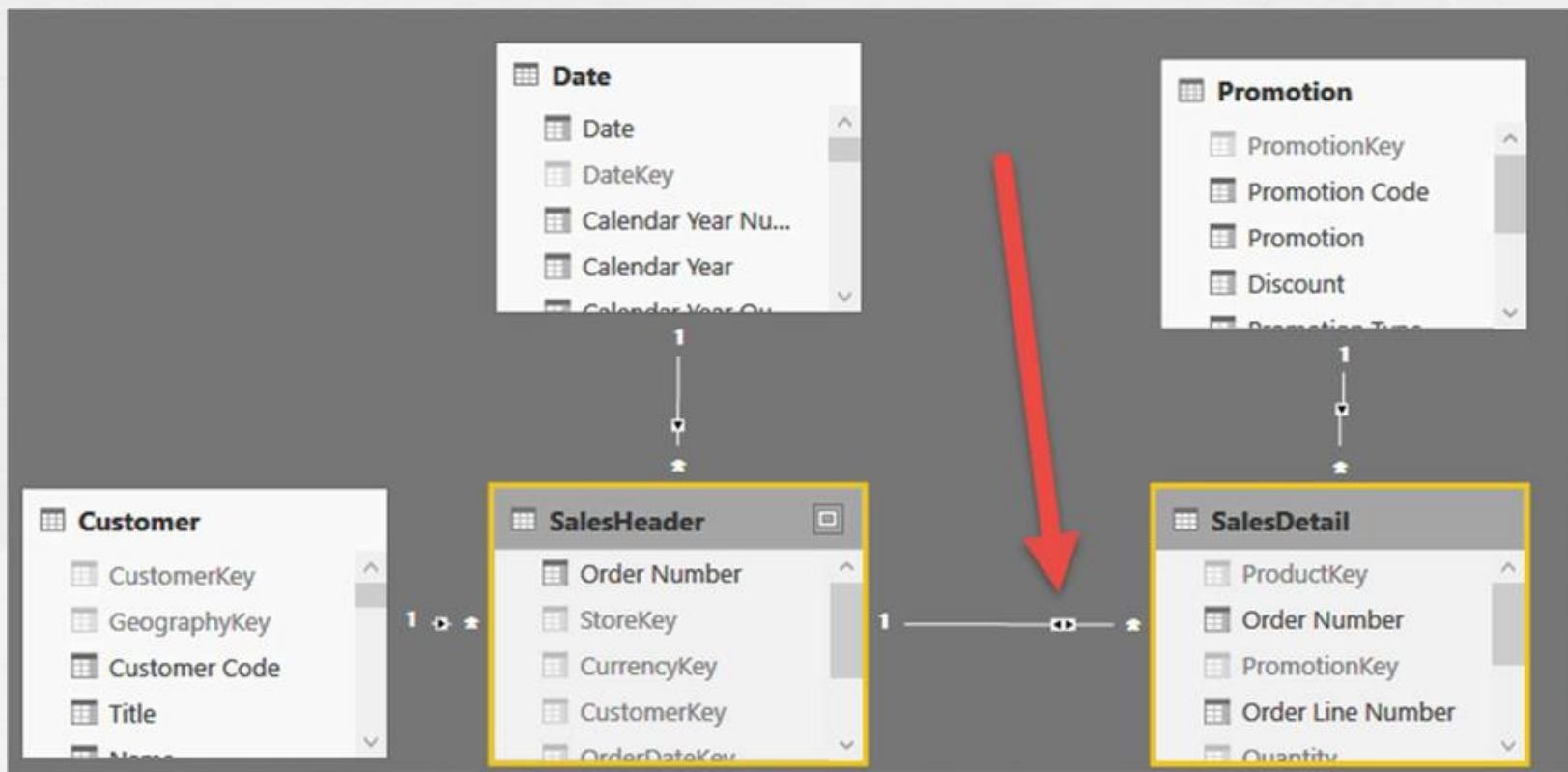
# Header / Detail



How do I avoid repeating the same header value for all the products?



# Bidirectional filtering is not the way to go



# Numbers are still incorrect



- What is the column total showing?
- It is using the many-to-many pattern, very hard to read

Brand	CY 2007	CY 2008	CY 2009	Total
A. Datum	\$23,396.50	\$6,625.26	\$6,818.09	<b>\$36,839.84</b>
Adventure Works	\$31,014.00	\$11,358.74	\$10,899.31	<b>\$53,272.05</b>
Contoso	\$51,975.46	\$24,946.38	\$28,580.97	<b>\$105,502.81</b>
Fabrikam	\$14,426.09	\$22,265.30	\$17,478.13	<b>\$54,169.52</b>
Litware	\$13,876.12	\$17,539.56	\$14,796.98	<b>\$46,212.66</b>
Northwind Traders	\$21,013.75	\$3,754.67	\$5,361.97	<b>\$30,130.39</b>
Proseware	\$12,812.25	\$9,740.69	\$15,318.93	<b>\$37,871.87</b>
Southridge Video	\$23,171.83	\$4,380.71	\$4,788.64	<b>\$32,341.19</b>
Tailspin Toys	\$1,029.51	\$707.81	\$1,775.78	<b>\$3,513.10</b>
The Phone Company	\$4,027.90	\$7,067.17	\$6,330.28	<b>\$17,425.35</b>
Wide World Importers	\$7,125.39	\$19,366.80	\$14,494.74	<b>\$40,986.93</b>
<b>Total</b>	<b>\$158,506.66</b>	<b>\$120,458.29</b>	<b>\$116,667.34</b>	<b>\$395,632.29</b>

SUM is **458,265.70**

## Header / detail

- Values in the detail
  - Can be sliced by any dimension
  - Lead to standard, additive behavior
- Values in the header
  - Can be sliced by dimensions linked to the header
  - But they produce wrong results with dimensions of detail
    - Always the same number
    - or -
    - Non-additive (wrong) values

# Computing the discount as a percentage

At the header level, the discount can be computed as a percentage, instead of a fixed value.

In this way, we allocate the discount proportionally.

Calculated columns work fine for this.

SalesHeader[DiscountPct] =

```
DIVIDE (
    SalesHeader[TotalDiscount],
    SUMX (
        RELATEDTABLE ( SalesDetail ),
        SalesDetail[Unit Price] * SalesDetail[Quantity]
    )
)
```

Order Number	StoreKey	CurrencyKey	CustomerKey	OrderDateKey	TotalDiscount	DiscountPct
20080604724008	307	1	13009	20080604	€ 0.54	10.00%
200805105CS561	307	1	19098	20080510	€ 13.99	10.00%
20070605820430	307	1	9431	20070605	€ 181.80	10.00%
20070510215734	307	1	4735	20070510	€ 32.90	10.00%
200801156CS531	307	1	19092	20080115	€ 311.99	15.00%
200704013CS473	307	1	19082	20070401	€ 698.60	20.00%
20071115726159	307	1	15160	20071115	€ 1.33	15.00%
200905028CS712	307	1	19122	20090502	€ 97.80	10.00%
20070422714011	307	1	3012	20070422	€ 5.59	20.00%
200902218CS699	307	1	19116	20090221	€ 335.98	20.00%
20070213824162	307	1	13163	20070213	€ 307.36	20.00%
200902076CS697	307	1	19115	20090207	€ 12.20	20.00%
20071227722905	307	1	11906	20071227	€ 13.91	15.00%
20080414822856	307	1	11857	20080414	€ 363.78	20.00%

# Computing the discount on the detail



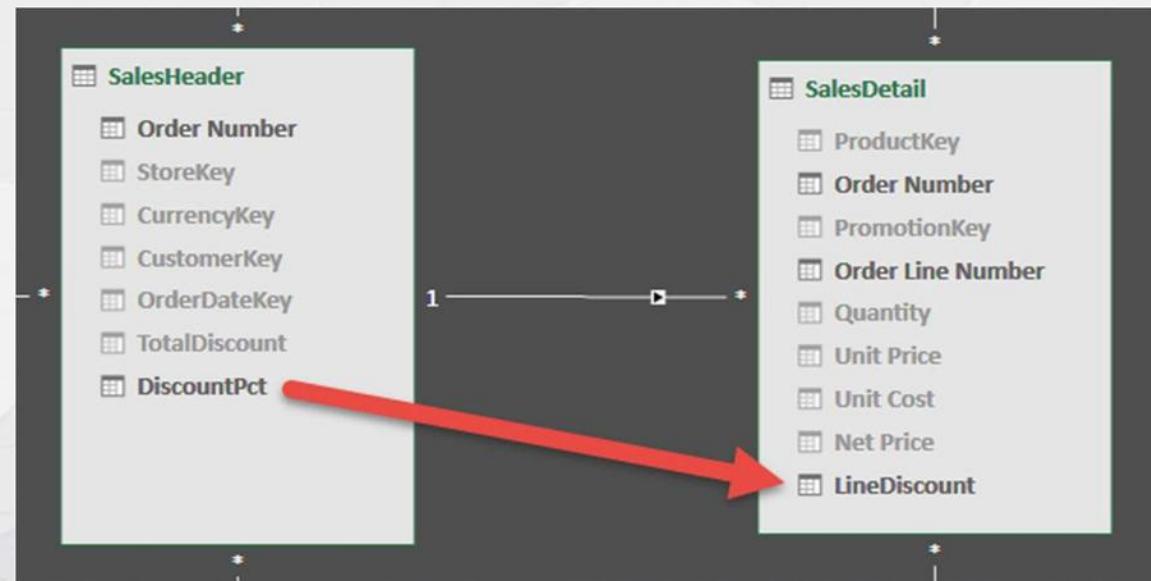
To correctly compute the discount, you aggregate the detail rows and – for each row – you compute the discount using the percentage stored in the header.

```
[DiscountValueCorrect] =  
SUMX (   
    SalesDetail,  
    RELATED ( SalesHeader[DiscountPct] )  
    * SalesDetail[Unit Price]  
    * SalesDetail[Quantity]  
)
```

Brand	▲ DiscountValue	DiscountValueCorrect
A. Datum	\$36,839.84	\$26,489.32
Adventure Works	\$53,272.05	\$47,608.89
Contoso	\$105,502.81	\$89,994.41
Fabrikam	\$54,169.52	\$49,618.90
Litware	\$46,212.66	\$43,991.90
Northwind Traders	\$30,130.39	\$29,794.45
Proseware	\$37,871.87	\$34,436.21
Southridge Video	\$32,341.19	\$17,100.27
Tailspin Toys	\$3,513.10	\$3,513.10
The Phone Company	\$17,425.35	\$17,287.23
Wide World Importers	\$40,986.93	\$35,797.62
<b>Total</b>	<b>\$395,632.29</b>	<b>\$395,632.29</b>

## Denormalizing the discount

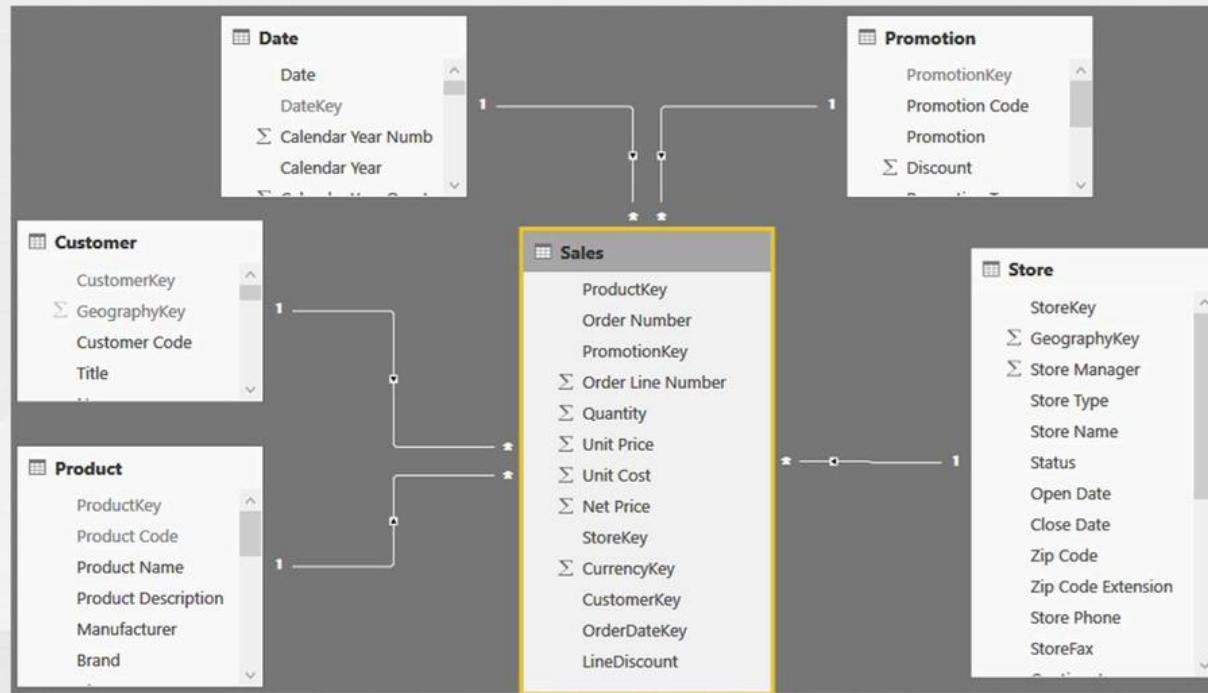
- The discount can be stored in the detail table
- This, turns the SalesHeader table into a dimension



# Back to a star schema



Once correctly denormalized, the model becomes a star schema again.



Once you identify the set of facts and dimensions, you no longer need header/detail tables.

Star schemas are always the best choice

## Allocation factor

- In the example, we allocated the discount as a percentage
- Thus, we used the price as the allocation factor
- Other scenarios might be different
  - For example: shipping allocated by weight
- Find the correct allocation factor
- Denormalize the values in the fact table
- Most header/detail models can be flattened easily, that is, denormalize the columns in a single fact table

# Conclusions

- Header and detail are both fact tables
- Aggregating from the header is troublesome
  - Numbers do not aggregate correctly
  - Aggregation at the wrong granularity
- Bidirectional filtering is of no help here
- Allocating the numbers flattens the model into a star schema
  - Need to define the allocation factor
  - Price, weight, number of items... it depends on the business

## Header / Detail



Time to practice and work on some models.

You will need to solve the header/detail pattern by yourself, following the instructions on the exercise manual.

Once the scenario is solved, you will need to go a few steps further, of course!

Refer to **lab number 2** on the hands-on manual.

Computing over multiple star schemas

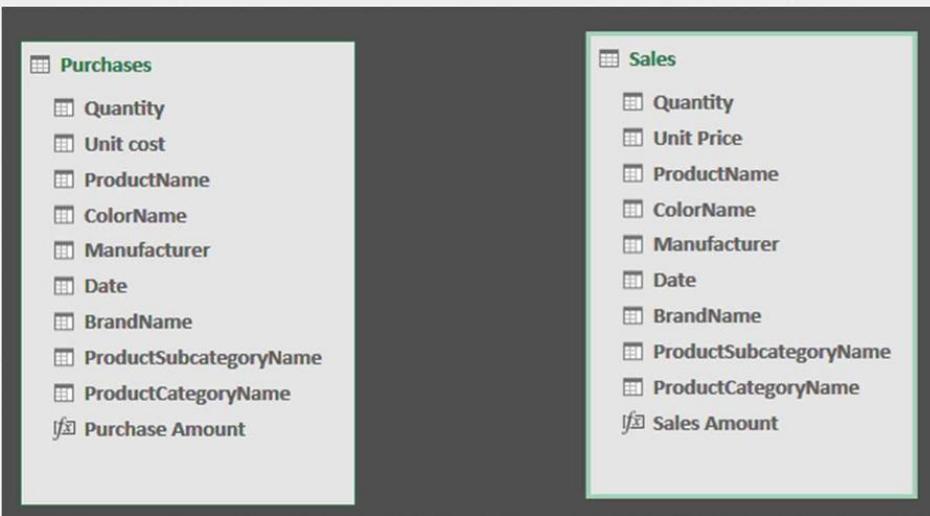
## Multiple fact tables



# Using multiple fact tables

- Very common scenario
  - Sales and purchases
  - Orders and shipments
  - Sales and weather information
- What we cover in this section
  - Build the correct set of dimensions
  - Use one fact table to filter the other one(s)

# Denormalized fact tables



BrandName	Purchase Amount	Sales Amount
A. Datum	2,533,963.42	30,202,685.54
Adventure Works	6,048,167.59	30,202,685.54
Contoso	12,314,395.68	30,202,685.54
Fabrikam	10,003,071.13	30,202,685.54
Litware	6,377,548.93	30,202,685.54
Northwind Traders	1,713,836.80	30,202,685.54
Proseware	5,305,305.29	30,202,685.54
Southridge Video	2,199,989.35	30,202,685.54
Tailspin Toys	646,571.47	30,202,685.54
The Phone Company	3,045,608.33	30,202,685.54
Wide World Importers	4,151,139.81	30,202,685.54
<b>Total</b>	<b>54,339,597.80</b>	<b>30,202,685.54</b>

# Why data is already denormalized

- Denormalized models do not appear in databases
- They often come from:
  - Reports
  - Views
  - Excel files
  - Flat files (CSV)
- Example: need to join two different reports

# Multiple fact tables

Purchases
Quantity
Unit cost
ProductName
ColorName
Manufacturer
Date
BrandName
ProductSubcategoryName
ProductCategoryName
Purchase Amount

Sales
Quantity
Unit Price
ProductName
ColorName
Manufacturer
Date
BrandName
ProductSubcategoryName
ProductCategoryName
Sales Amount

	BrandName	Purchase Amount	Sales Amount
	A. Datum	2,533,963.42	30,202,685.54
	Adventure Works	6,048,167.59	30,202,685.54
	Contoso	12,314,395.68	30,202,685.54
	Fabrikam	10,003,071.13	30,202,685.54
	Litware	6,377,548.93	30,202,685.54
	Northwind Traders	1,713,836.80	30,202,685.54
	Proseware	5,305,305.29	30,202,685.54
	Southridge Video	2,199,989.35	30,202,685.54
	Tailspin Toys	646,571.47	30,202,685.54
	The Phone Company	3,045,608.33	30,202,685.54
	Wide World Importers	4,151,139.81	30,202,685.54
	<b>Total</b>	<b>54,339,597.80</b>	<b>30,202,685.54</b>

How do I filter both tables at the same time in a report?



# Moving filters with DAX



You can use INTERSECT, CONTAINS or TREATAS to move a filter from one table to the other one, none of these options results in optimal performance

Sales Amount Filtered :=

```
CALCULATE (
    [Sales Amount],
    TREATAS (
        VALUES ( Purchases[BrandName] ),
        Sales[BrandName]
    )
)
```

BrandName	Purchase Amount	Sales Amount	Sales Amount Filtered
A. Datum	2,533,963.42	30,202,685.54	1,966,583.30
Adventure Works	6,048,167.59	30,202,685.54	4,022,462.56
Contoso	12,314,395.68	30,202,685.54	6,722,804.20
Fabrikam	10,003,071.13	30,202,685.54	5,040,864.58
Litware	6,377,548.93	30,202,685.54	3,425,045.95
Northwind Traders	1,713,836.80	30,202,685.54	1,205,185.61
Proseware	5,305,305.29	30,202,685.54	2,656,623.00
Southridge Video	2,199,989.35	30,202,685.54	1,463,471.36
Tailspin Toys	646,571.47	30,202,685.54	333,143.41
The Phone Company	3,045,608.33	30,202,685.54	1,293,603.00
Wide World Importers	4,151,139.81	30,202,685.54	2,072,898.57
<b>Total</b>	<b>54,339,597.80</b>	<b>30,202,685.54</b>	<b>30,202,685.54</b>

# Not the best option, anyway...

With an increasing number of attributes, using DAX does not look appealing, because the code starts to be complex and you need to repeat it in all the measures

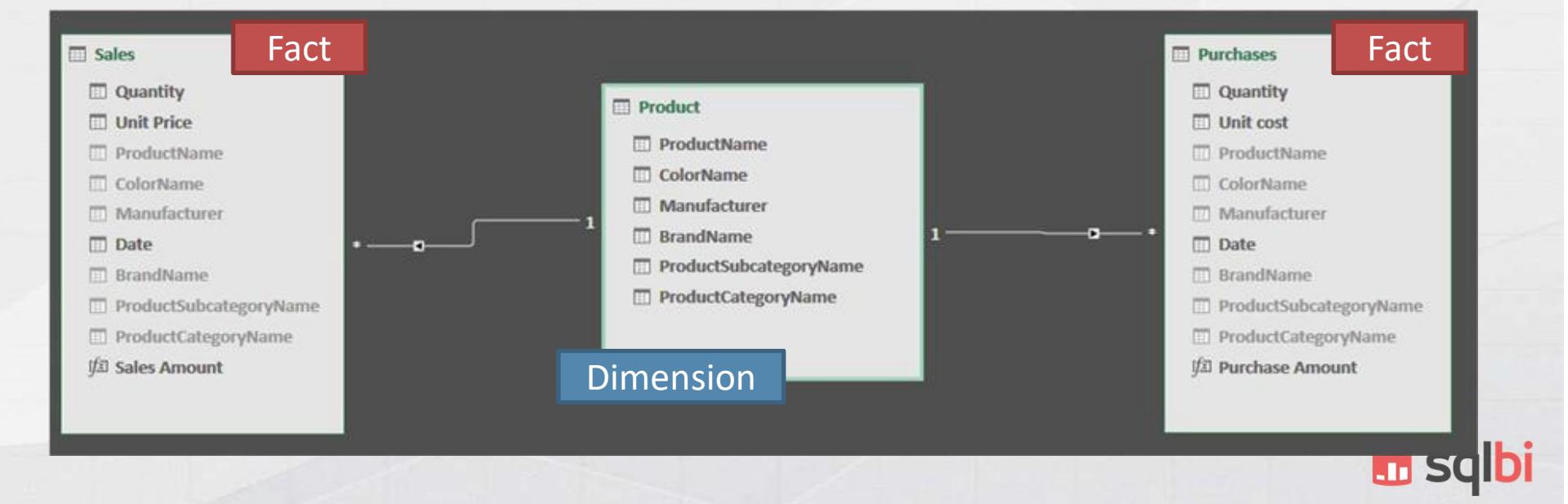
Sales Amount Filtered :=

```
CALCULATE (
    [Sales Amount],
    TREATAS ( VALUES ( Purchases[BrandName] ), Sales[BrandName] ),
    TREATAS ( VALUES ( Purchases[ColorName] ), Sales[ColorName] ),
    TREATAS ( VALUES ( Purchases[Manufacturer] ), Sales[Manufacturer] ),
    TREATAS (
        VALUES ( Purchases[ProductCategoryName] ),
        Sales[ProductCategoryName]
    ),
    TREATAS (
        VALUES ( Purchases[ProductSubcategoryName] ),
        Sales[ProductSubcategoryName]
    )
)
```

# Building a star schema



- A proper star schema is nearly always the best choice
- But how do we build the Product table?



## Options to build the new dimension

- Use an SQL view, if feasible
- Use M code in Power Query
  - Available in Excel and Power BI
- Use DAX code and build a calculated table
  - Available in Power BI and SSAS 2016
- You need a key for the new dimension
  - Easy in SQL
  - Harder in M or DAX, if primary key not already available

# Example of DAX code for the dimension

Here you can see an example of the new dimension built with DAX.

In this case, we will use the product name for the key, because nothing better is available.

```
Products =  
DISTINCT (  
    UNION (  
        ALL (  
            Sales[ProductName],  
            Sales[ColorName],  
            Sales[Manufacturer],  
            Sales[BrandName],  
            Sales[ProductCategoryName],  
            Sales[ProductSubcategoryName]  
        ),  
        ALL (  
            Purchases[ProductName],  
            Purchases[ColorName],  
            Purchases[Manufacturer],  
            Purchases[BrandName],  
            Purchases[ProductCategoryName],  
            Purchases[ProductSubcategoryName]  
        )  
    )  
)
```

How to properly use multiple fact tables, if present

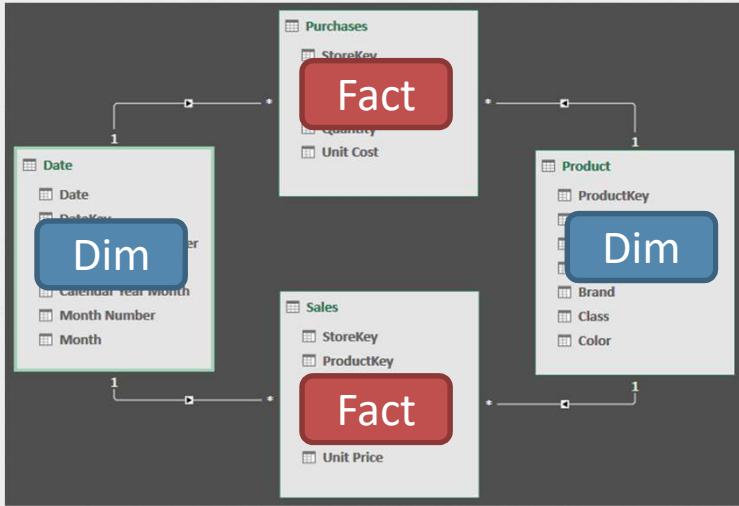
## Star schema, multiple fact tables



## Multiple fact tables

- Using multiple star schemas is straightforward
  - Sales
  - Purchases
- You can slice both by using intermediate dimensions
- Multiple fact tables provides some hidden power
  - By mixing filters on both tables
  - Using one fact table to filter the other ones
  - With one or multiple dimensions in the middle

# Star schema with multiple fact tables



Slicing sales and purchases by product and year is a simple task

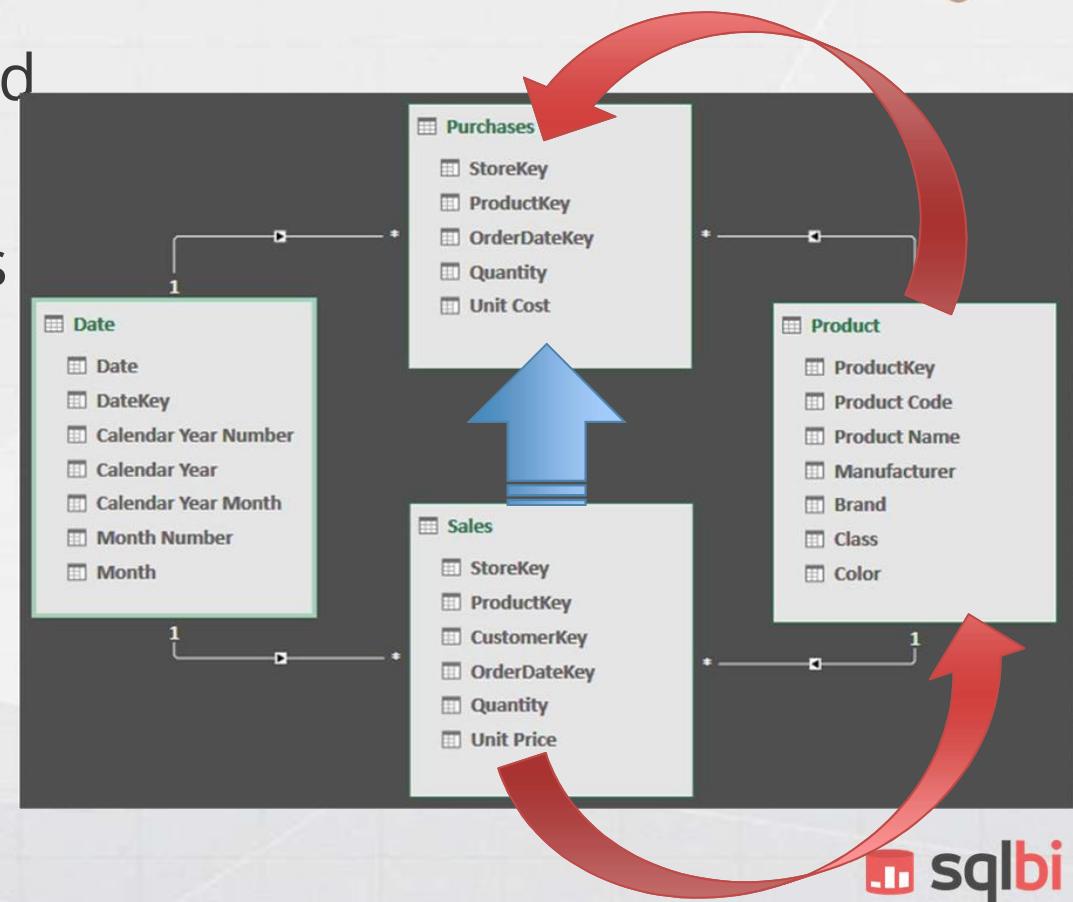
Year	BrandName	2007		2008	
		Purchase Amount	Sales Amount	Purchase Amount	Sales Amount
A. Datum		1,023,679.78	1,157,725.70	818,043.27	406,315.70
Adventure Works		1,870,557.41	2,039,391.33	2,038,461.28	986,358.80
Contoso		4,619,805.74	2,512,313.30	4,145,623.47	1,971,091.04
Fabrikam		3,888,713.45	1,536,602.49	3,148,920.81	1,636,343.31
Litware		2,157,328.85	893,749.66	2,303,339.22	1,276,017.29
Northwind Traders		880,347.77	523,238.11	580,451.12	417,964.34
Proseware		2,225,536.45	916,830.85	1,626,193.19	854,115.95
Southridge Video		840,767.62	711,938.95	673,846.03	373,918.52
Tailspin Toys		170,143.99	85,328.29	168,487.05	77,445.98
The Phone Company		1,340,335.89	429,561.00	943,104.43	413,104.00
Wide World Importers		1,192,712.27	438,136.25	1,522,052.76	766,375.56
<b>Total</b>		<b>20,209,929.22</b>	<b>11,244,815.93</b>	<b>17,968,522.63</b>	<b>9,179,050.49</b>
Year	BrandName	2007		2008	
		Purchase Amount	Sales Amount	Purchase Amount	Sales Amount
A. Datum		1,023,679.78	1,157,725.70	818,043.27	406,315.70
Adventure Works		1,870,557.41	2,039,391.33	2,038,461.28	986,358.80
Contoso		4,619,805.74	2,512,313.30	4,145,623.47	1,971,091.04
Fabrikam		3,888,713.45	1,536,602.49	3,148,920.81	1,636,343.31
Litware		2,157,328.85	893,749.66	2,303,339.22	1,276,017.29
Northwind Traders		880,347.77	523,238.11	580,451.12	417,964.34
Proseware		2,225,536.45	916,830.85	1,626,193.19	854,115.95
Southridge Video		840,767.62	711,938.95	673,846.03	373,918.52
Tailspin Toys		170,143.99	85,328.29	168,487.05	77,445.98
The Phone Company		1,340,335.89	429,561.00	943,104.43	413,104.00
Wide World Importers		1,192,712.27	438,136.25	1,522,052.76	766,375.56
<b>Total</b>		<b>20,209,929.22</b>	<b>11,244,815.93</b>	<b>17,968,522.63</b>	<b>9,179,050.49</b>



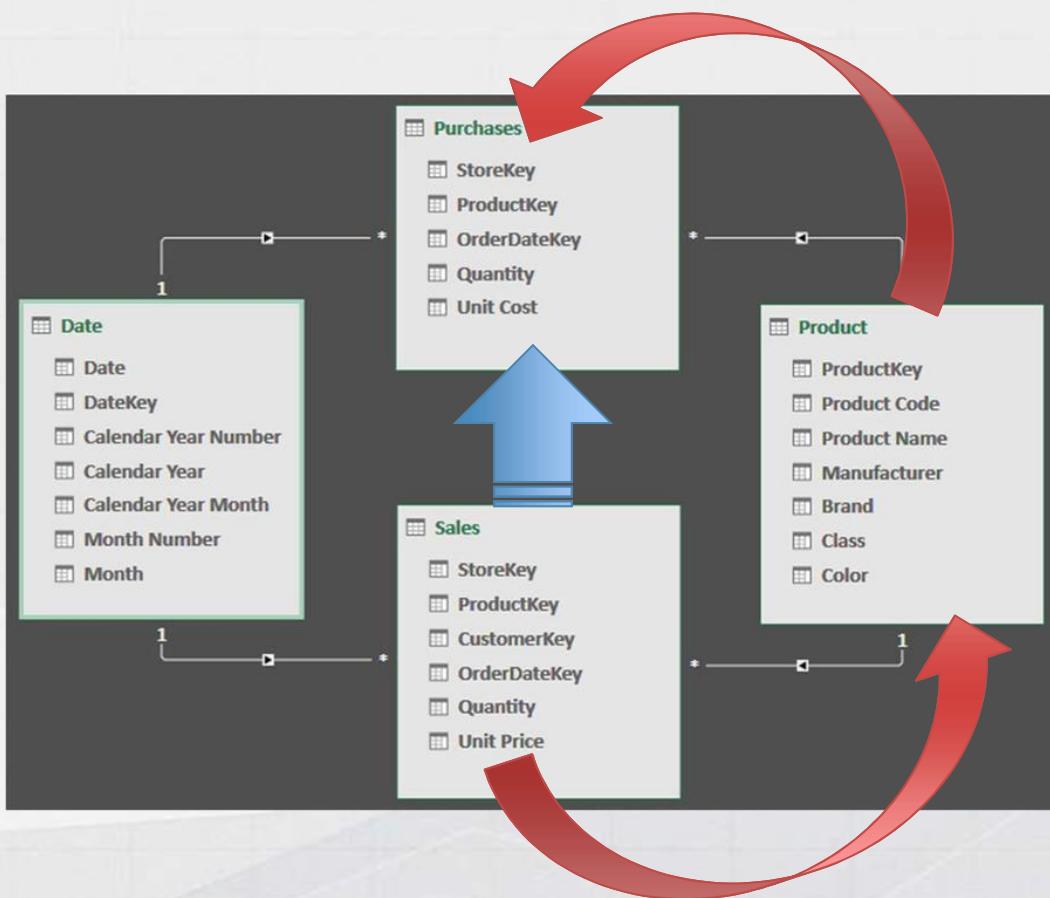
# Purchases of products sold



- Filter purchases based on sales
- How many purchases of only the products sold in a given period?
- Bidirectional filtering looks promising here...



# Moving filters



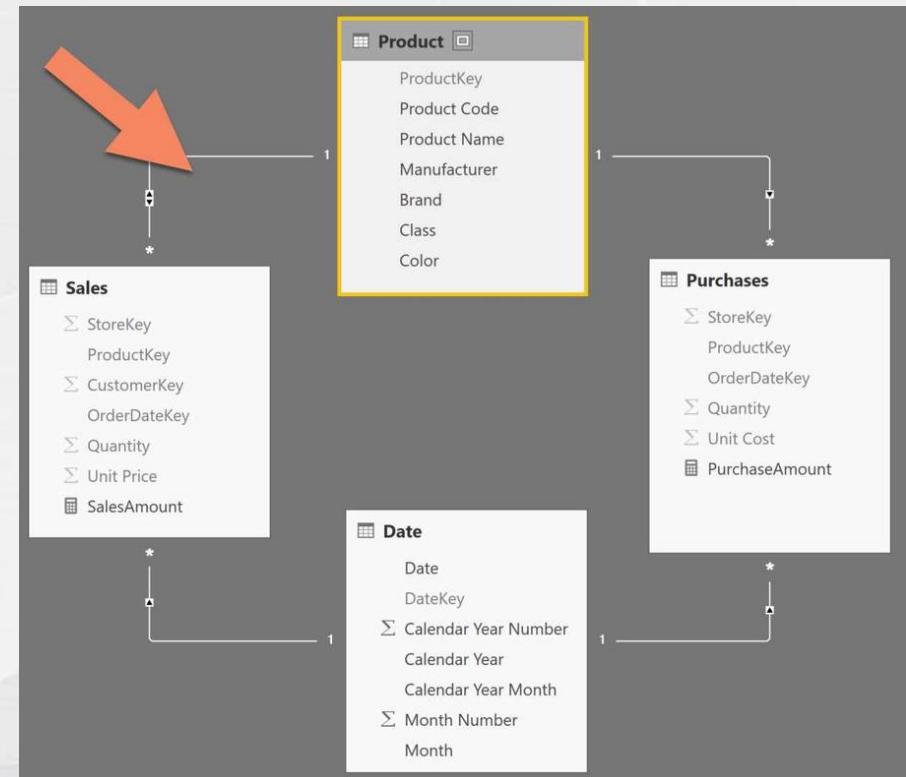
How do I move a filter from Sales to Purchases?



# Trying bidirectional filtering



- Unfortunately, you cannot use bidirectional filtering in the model
- The model would become ambiguous
- Sales filters Product, but then Product cannot filter Purchases anymore



# DAX to activate bidirectional filters



You can activate bidirectional filtering on a relationship by using the DAX CROSSFILTER function. You maintain the model coherent, and activate a relationship as bidirectional on demand. It also works on Excel 2016.

PurchaseOfSoldProducts :=

```
CALCULATE (
    [PurchaseAmount],
    CALCULATETABLE (
        Product,
        CROSSFILTER ( Sales[ProductKey], Product[ProductKey], BOTH )
        CROSSFILTER ( Purchase[OrderDateKey], Date[DateKey], NONE )
    )
)
```

A small digression to understand what model ambiguity is

## Model ambiguity



## What is ambiguity?

- If there are multiple paths between tables, then the model is ambiguous
- It easily appears with bidirectional filtering activated
- Even if it is not only because of bidirectional filtering
- Ambiguous models are not permitted
- You are required to deactivate some relationships

# Ambiguity

- Multiple paths not permitted between any two tables
- Behavior already visible using just two tables



RELATED always uses  
the active relationship.

This is why you can have  
only one active

```
Sales[Year] = RELATED ( 'Date'[Calendar Year] )
```

# Activating relationships on demand

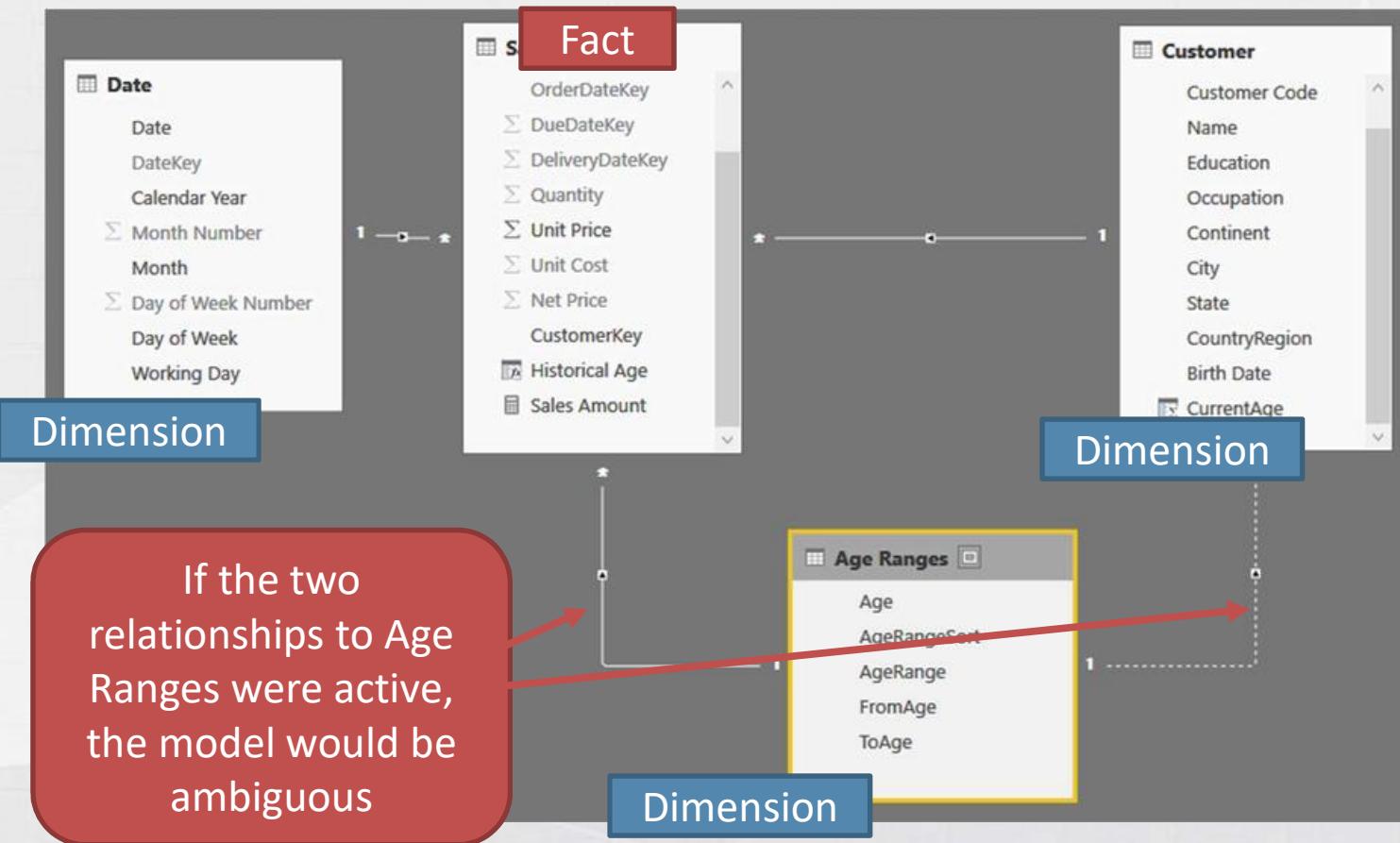
With USERELATIONSHIP you can activate a relationship on demand, deactivating the conflicting ones, for the duration of CALCULATE.

```
Shipped2009 :=
```

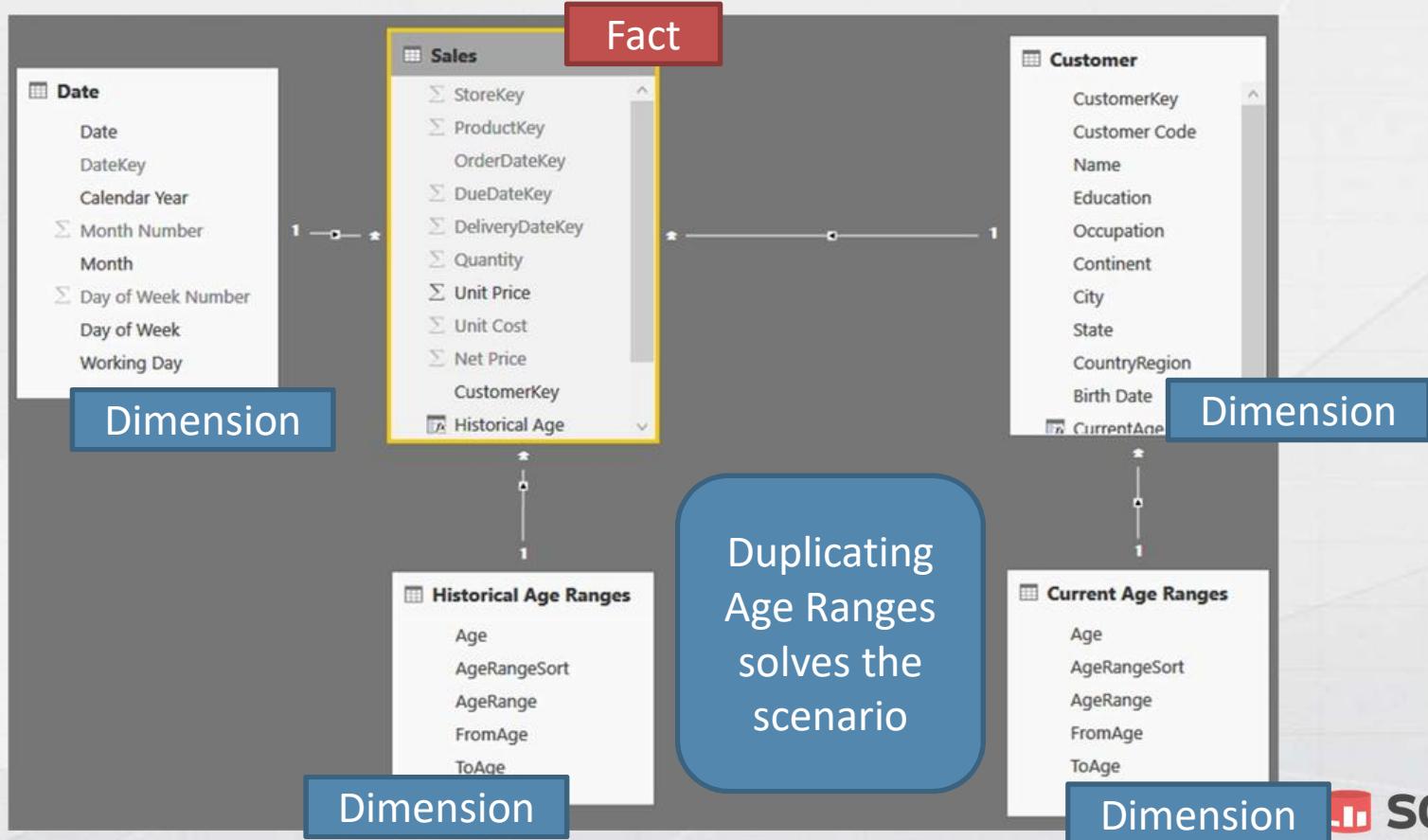
```
CALCULATE (
    [Sales Amount],
    'Date'[Calendar Year] = "CY 2009",
    USERELATIONSHIP (
        'Date'[DateKey],
        Sales[DeliveryDateKey]
    )
)
```



# Ambiguity with intermediate dimensions



# Solving ambiguity



## Role dimensions

- In the previous example, Age is a role dimension
- The same dimension acts with different roles
  - Current age
  - Historical age
- Using DAX, role dimensions need to be duplicated
- Pay attention to the column names and content to make the report clear for the users
- Optionally denormalize the content in the original table

# Multiple fact tables



Time to practice and work on some models.

In these labs you work with multiple fact tables. You will prepare proper dimensions and check some details about the correctness of data.

Refer to **lab number 3** on the hands-on manual.

Time intelligence calculations require a dimension and some more details

## Working with date and time

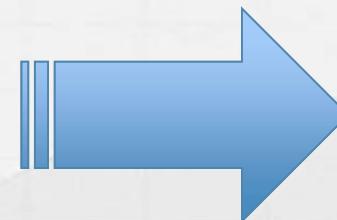
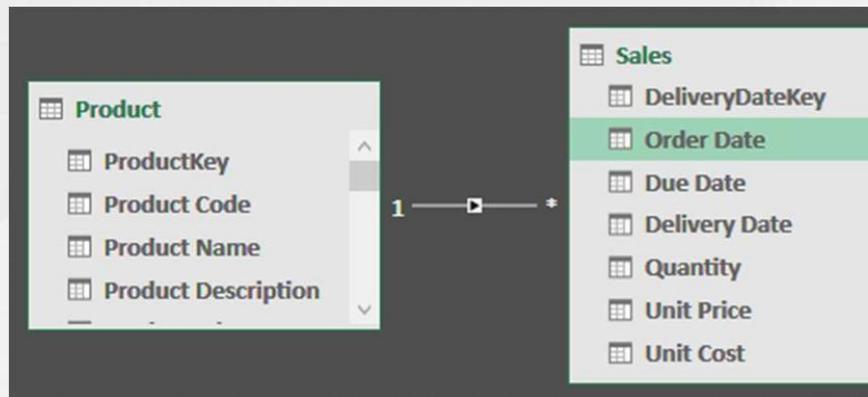


# Date attributes in the fact table



You can slice by year, month and date by adding a proper set of attributes to your fact table.  
This is not a best practice, because all the calculations tend to be much more complex.

```
Sales[Year] = YEAR ( Sales[Order Date] )  
Sales[Month] = FORMAT ( Sales[Order Date], "mmmm" )  
Sales[MonthNumber] = MONTH ( Sales[Order Date] )
```



Year	Sales Amount
2007	<b>1,459,215.95</b>
January	101,097.12
February	108,553.20
March	119,707.83
April	121,085.74
May	123,413.41
June	121,707.44
July	139,381.00
August	87,384.31
September	155,275.94
October	99,872.65
November	122,522.86
December	159,214.45
2008	<b>1,122,535.05</b>

# YTD with attributes in the fact table



A simple YTD calculation results in quite complex DAX code and a careful treatment of filters on individual columns. Definitely not a best practice.

Sales YTD :=

```
VAR CurrentYear = MAX ( Sales[Year] )
VAR CurrentDate = MAX ( Sales[Order Date] )
```

RETURN

```
CALCULATE (
    [Sales Amount],
    Sales[Order Date] <= CurrentDate,
    Sales[Year] = CurrentYear,
    ALL ( Sales[Month] ),
    ALL ( Sales[MonthNumber] )
)
```

Year	Sales	Amount	Sales YTD
2007	<b>1,459,215.95</b>	<b>1,459,215.95</b>	
January	101,097.12	101,097.12	
February	108,553.20	209,650.32	
March	119,707.83	329,358.16	
April	121,085.74	450,443.90	
May	123,413.41	573,857.31	
June	121,707.44	695,564.75	
July	139,381.00	834,945.75	
August	87,384.31	922,330.06	
September	155,275.94	1,077,606.00	
October	99,872.65	1,177,478.64	
November	122,522.86	1,300,001.50	
December	159,214.45	1,459,215.95	

## Attributes in the fact table

- The code tends to be complex to author
- You cannot slice multiple fact tables
- You cannot leverage built-in DAX time intelligence functions
- Limited analytical capabilities
- It is an example of scattered information
  - Time is an entity of your model
  - It requires a dimension by itself

Learn how to correctly build a date dimension

# Building a date dimension



# Creating a date dimension

- Multiple options
  - Power Query and the M language
  - DAX code
  - Load from a data source
- Needed properties
  - All dates in the year need to be present (01/01-12/31)
  - One datetime column (better if it is the primary key)
- If the table is not properly set, calculations are wrong

# CALENDAR



Returns a table with a single column named “Date” containing a contiguous set of dates in the given range, inclusive.

```
CALENDAR (
    DATE ( 2005, 1, 1 ),
    DATE ( 2015, 12, 31 )
)
```

```
CALENDAR (
    MIN ( Sales[Order Date] ),
    MAX ( Sales[Order Date] )
)
```

# CALENDAR

If you have multiple fact tables, you need to compute the correct values.

Date =

```
CALENDAR (
    MIN (
        MIN ( Sales[Order Date] ),
        MIN ( Purchases[Purchase Date] )
    ),
    MAX (
        MAX ( Sales[Order Date] ),
        MAX ( Purchases[Purchase Date] )
    )
)
```

# CALENDARAUTO



Automatically creates a calendar table based on the database content. Optionally you can specify the last month (useful for fiscal years).

```
= CALENDARAUTO ( )  
-- The optional parameter is the last month  
-- of the fiscal year  
--
```

```
= CALENDARAUTO (  
    6  
)
```

Beware: CALENDARAUTO uses all the dates in your model, excluding only calculated columns and tables

# Building a date table in DAX



Using simple DAX code you can write a DAX expression that dynamically generates a calendar table

```
Date =  
ADDCOLUMNS (   
    FILTER (   
        CALENDARAUTO (),  
        AND (   
            YEAR ( [Date] ) >= YEAR ( MIN ( Sales[Order Date] ) ),  
            YEAR ( [Date] ) <= YEAR ( MAX ( Sales[Order Date] ) )  
        )  
    ),  
    "Year", YEAR ( [Date] ),  
    "Month", FORMAT ( [Date], "mmmm" ),  
    "Month Number", MONTH ( [Date] )  
)
```

## The model with a date dimension

- Build a relationship based on OrderDateKey
- Slice by any attribute in the Date dimension
- Single place for all the attributes
- Much better support to author your measures



# Year-to-date with a date dimension

Using a date dimension you can leverage DATESYTD and the many time-intelligence predefined calculations available in DAX.

```
Sales YTD :=
```

```
CALCULATE (
    [Sales Amount],
    DATESYTD ( 'Date'[Date] )
)
```

Year	Sales	Amount	Sales YTD
2007	<b>1,459,215.95</b>	<b>1,459,215.95</b>	
January	101,097.12	101,097.12	
February	108,553.20	209,650.32	
March	119,707.83	329,358.16	
April	121,085.74	450,443.90	
May	123,413.41	573,857.31	
June	121,707.44	695,564.75	
July	139,381.00	834,945.75	
August	87,384.31	922,330.06	
September	155,275.94	1,077,606.00	
October	99,872.65	1,177,478.64	
November	122,522.86	1,300,001.50	
December	159,214.45	1,459,215.95	

# Handling fiscal calendars



Some time intelligence functions handle fiscal calendars with an additional parameter.

```
Sales YTD Fiscal :=  
CALCULATE (  
    [Sales Amount],  
    DATESYTD ( 'Date'[Date], "06/30" )  
)
```

Need a new column for the fiscal month, sorted in a different way (July...June)

Fiscal Year	Sales Amount	Sales YTD
2006	<b>695,564.75</b>	<b>695,564.75</b>
January	101,097.12	101,097.12
February	108,553.20	209,650.32
March	119,707.83	329,358.16
April	121,085.74	450,443.90
May	123,413.41	573,857.31
June	121,707.44	695,564.75
2007	<b>1,286,923.00</b>	<b>1,286,923.00</b>
July	139,381.00	139,381.00
August	87,384.31	226,765.31
September	155,275.94	382,041.25
October	99,872.65	481,913.89
November	122,522.86	604,436.75
December	159,214.45	763,651.19

# Same period last year

Same period in previous year. CALCULATE is needed  
Specialized version of DATEADD.

Sales SPLY :=

```
CALCULATE (
    SUM ( Sales[SalesAmount] ),
    SAMEPERIODLASTYEAR ( 'Date'[Date] )
)
```

Excel and Power BI have a feature to make time browsing easier, but using it is not a best practice

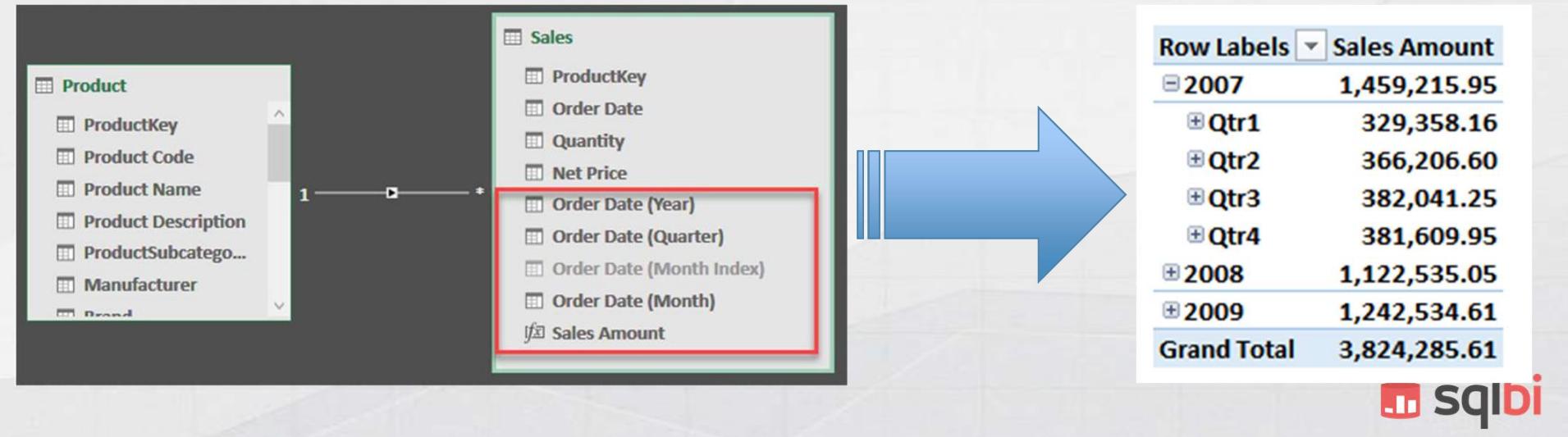
## Automatic date grouping in Excel and Power BI



# Automatic date grouping in Excel



- Excel performs automatic date grouping
  - Adding new columns to the fact table
  - Which we said is the wrong way...



# Automatic date grouping in Power BI



- Power BI offers a similar feature
- It builds a calendar table for each date column
- Slightly better than Excel, still not optimal
- No single calendar table
  - One for each column in the model
  - No consolidated slicing by date
  - The tables are hidden
  - Uses a “hidden” feature of DAX to reference columns:  
*column variations*

# Quick calculations in Power BI Desktop



Quick calculations provide an easy solution to many time-intelligence requirements, but they use a technique that is not a best practice for DAX.

Sales Amount YTD :=

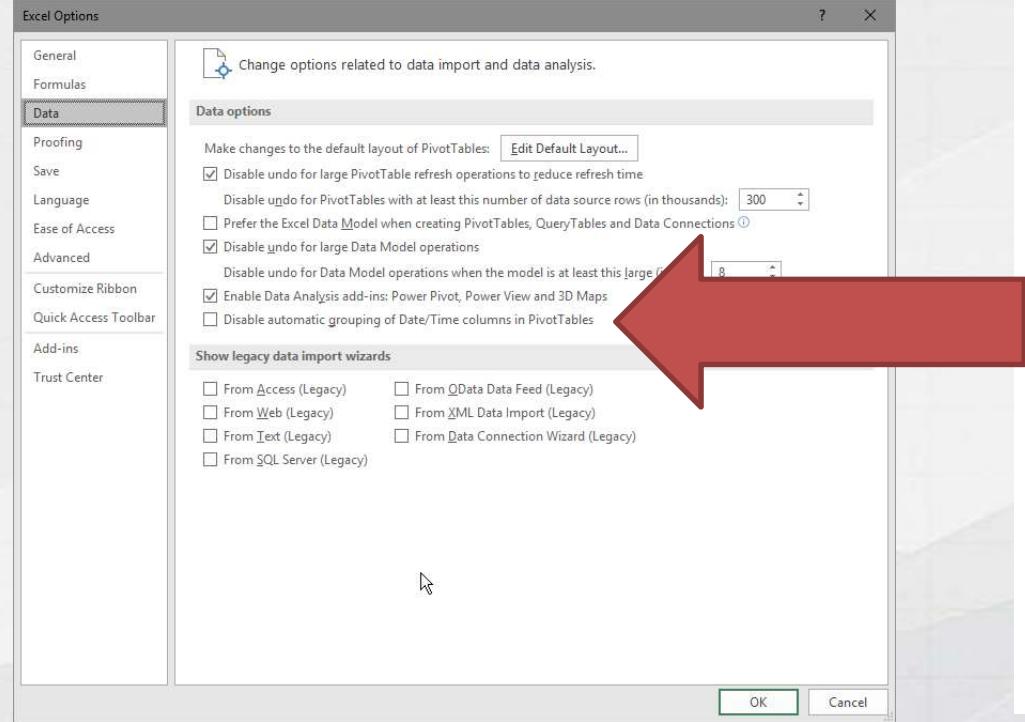
```
IF (
    ISFILTERED ('Sales'[Order Date]),
    ERROR ( "blah blah blah" ),
    TOTALYTD (
        'Sales'[Sales Amount],
        'Sales'[Order Date].[Date]
    )
)
```

Column Variation

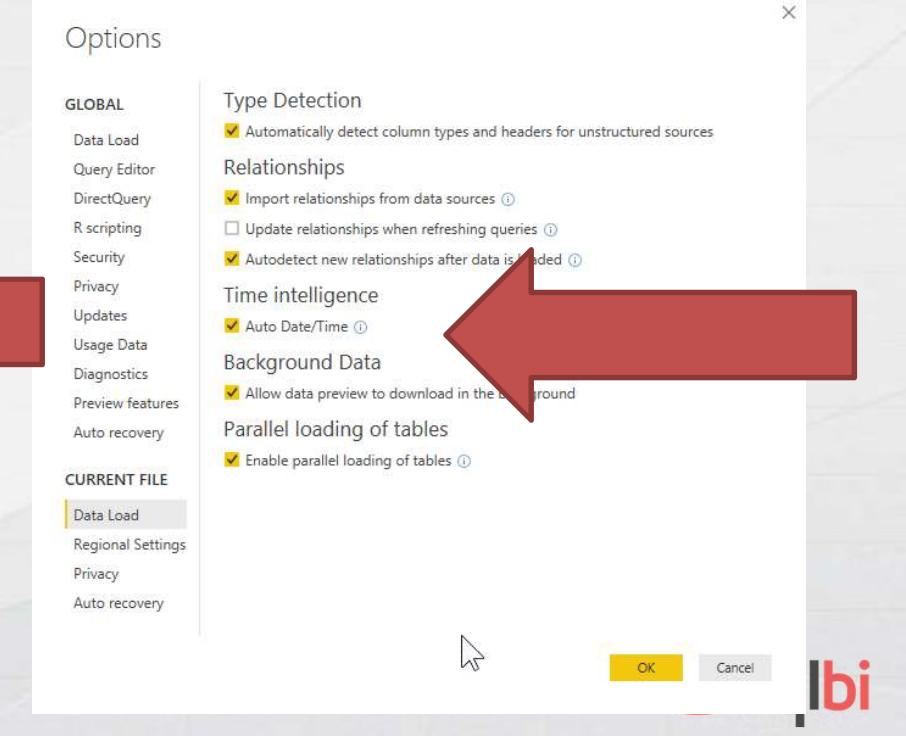
Year	Quarter	Month	Day	Sales Amount	Sales Amount YTD
2009	Qtr 1	January	1	2,198.95	2,198.95
			2	1,325.89	3,524.84
			3	1,775.52	5,300.35
			4	2,167.90	7,468.25
			5	511.70	7,979.95
			6	907.89	8,887.84
			7	332.37	9,220.21
			8	4,605.52	13,825.73
			9	4,442.14	18,267.87
			10	82.70	18,350.58
			11	60.44	18,411.01
			12	1,771.18	20,182.19

# Disable automatic date columns

Excel



Power BI



In a fact table you might have multiple dates, how should you handle them?

## Handling multiple dates



# Multiple Dates



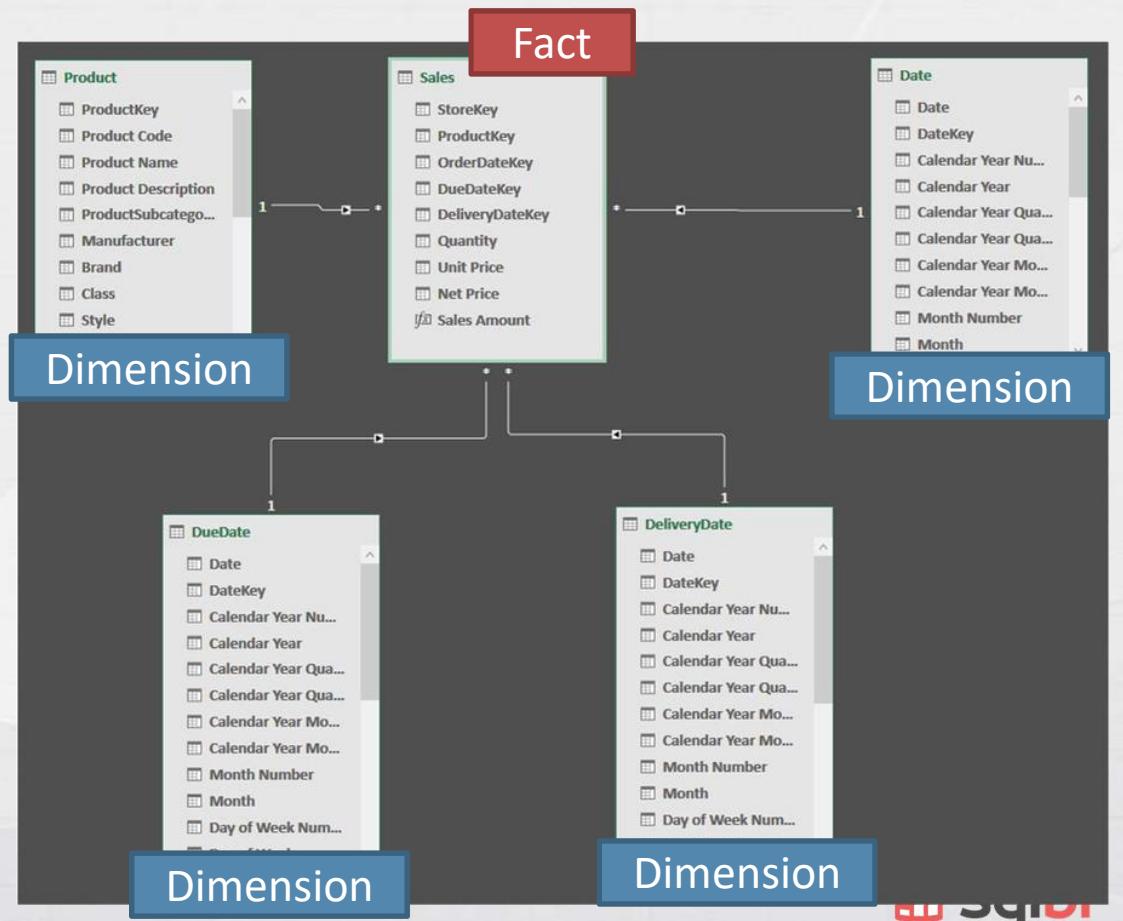
- Date is often a role dimension
  - Many roles for a date
  - Many date tables
- How many date tables?
  - Try to use only one table
  - Use many, only if you need to intersect multiple dates
  - Many date tables lead to confusion
    - And issues when slicing
- Use proper naming convention

# Inactive relationships

- Inactive relationships cannot be activated by the user
- No option in any of the current client tools
- Why using them?
  - Extreme optimizations: they are faster than filters
  - Can be activated on demand
- Can we solve the issue with only standard, active relationships?

# Multiple date tables

- Multiple date tables
- Single fact table
- The model becomes more complicated
- Slicing multiple fact tables becomes troublesome
- Not a best practice



## Use proper naming convention



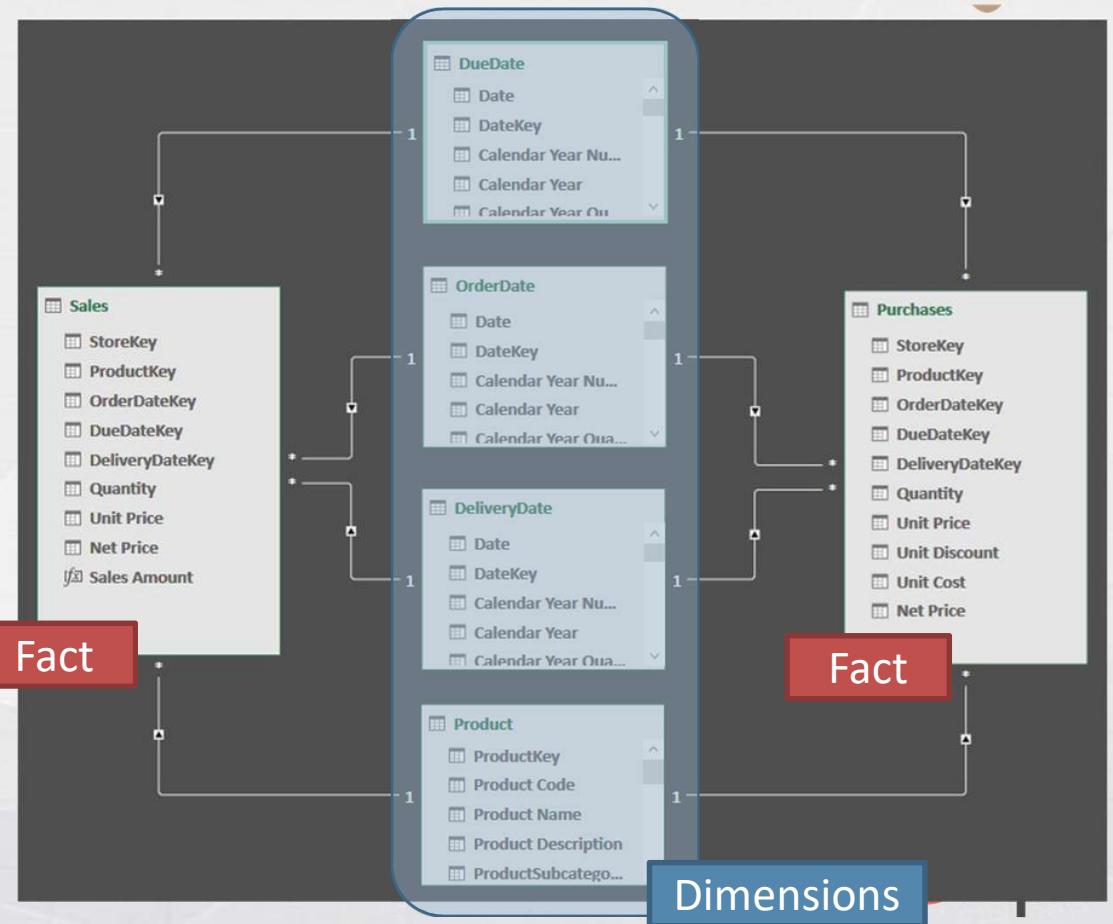
- Without proper names, the reports become hard to read
- Users want insights, not to spend time understanding what's on rows and columns...

Year	2007	2008	2009	2010	Total
2007	1,410,787.80	48,428.15			<b>1,459,215.95</b>
2008		1,096,993.59	25,541.46		<b>1,122,535.05</b>
2009			1,196,025.44	46,509.18	<b>1,242,534.61</b>
<b>Total</b>	<b>1,410,787.80</b>	<b>1,145,421.73</b>	<b>1,221,566.90</b>	<b>46,509.18</b>	<b>3,824,285.61</b>

# Multiple date tables with multiple fact tables



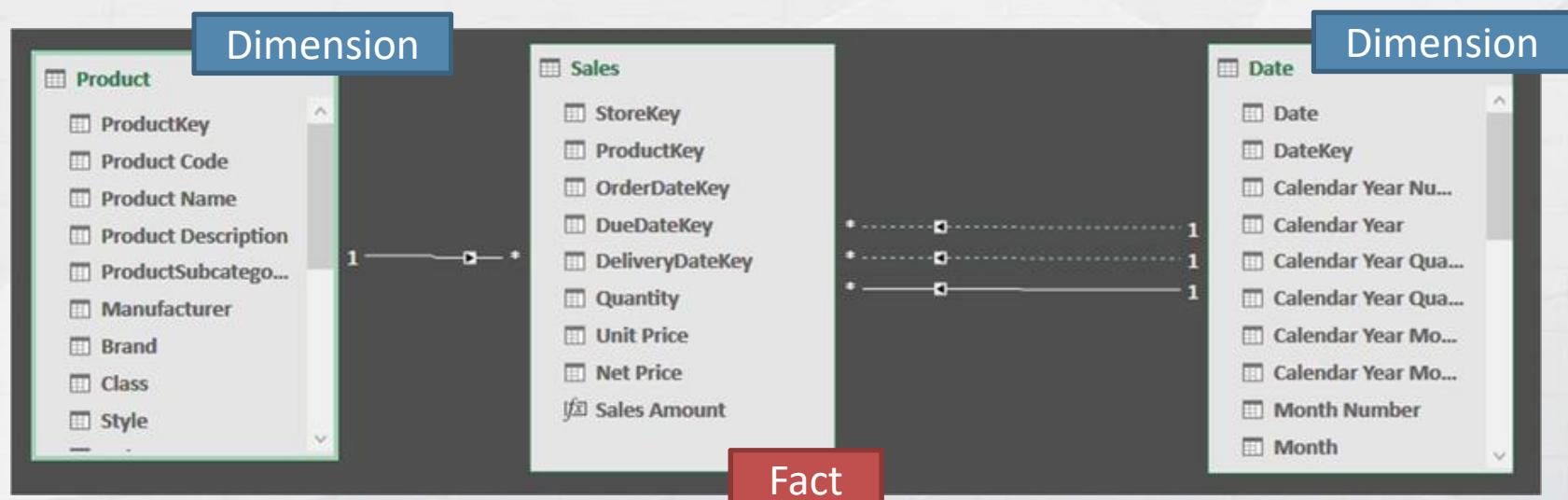
- Multiple fact tables
- Multiple date tables
- Slicing becomes much more complex
- Values are hard to define and compute



# Multiple relationships with date



- One table, multiple relationships
- Only one relationship can be active



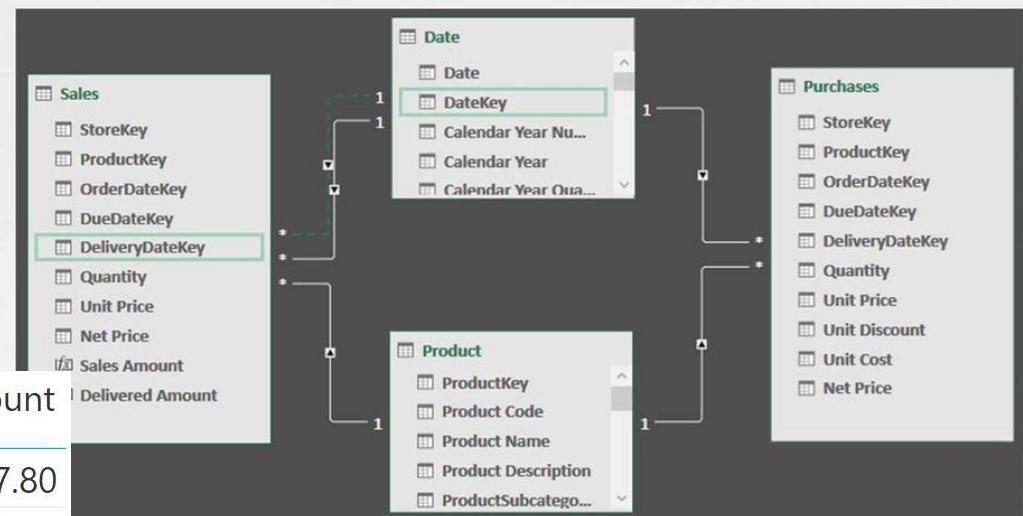
# Multiple relationships, single date table

By authoring code, you can activate a relationship on demand, so to rely on inactive relationships for specific calculations, still keeping a single date dimension in the model.

```
Delivered Amount :=
```

```
CALCULATE (
    [Sales Amount],
    USERELATIONSHIP (
        Sales[DeliveryDateKey],
        'Date'[DateKey]
    )
)
```

	Year	Sales Amount	Delivered Amount
	2007	1,459,215.95	1,410,787.80
	2008	1,122,535.05	1,145,421.73
	2009	1,242,534.61	1,221,566.90
	2010		46,509.18
<b>Total</b>		<b>3,824,285.61</b>	<b>3,824,285.61</b>



# Handling date and time



- Don't use a single column with date and time
- Two columns
  - Date (date dimension)
  - Time (time dimension)
- Build them with M code
- Browsing and DAX authoring is much easier

Time	Hour	Minute	TimelIndex	HourMinute
00.00.00	00	00	0	00:00
00.01.00	00	01	1	00:01
00.02.00	00	02	2	00:02
00.03.00	00	03	3	00:03
00.04.00	00	04	4	00:04
00.05.00	00	05	5	00:05
00.06.00	00	06	6	00:06
00.07.00	00	07	7	00:07
00.08.00	00	08	8	00:08
00.09.00	00	09	9	00:09
00.10.00	00	10	10	00:10
00.11.00	00	11	11	00:11
00.12.00	00	12	12	00:12
00.13.00	00	13	13	00:13
00.14.00	00	14	14	00:14
00.15.00	00	15	15	00:15

Some days are not working days

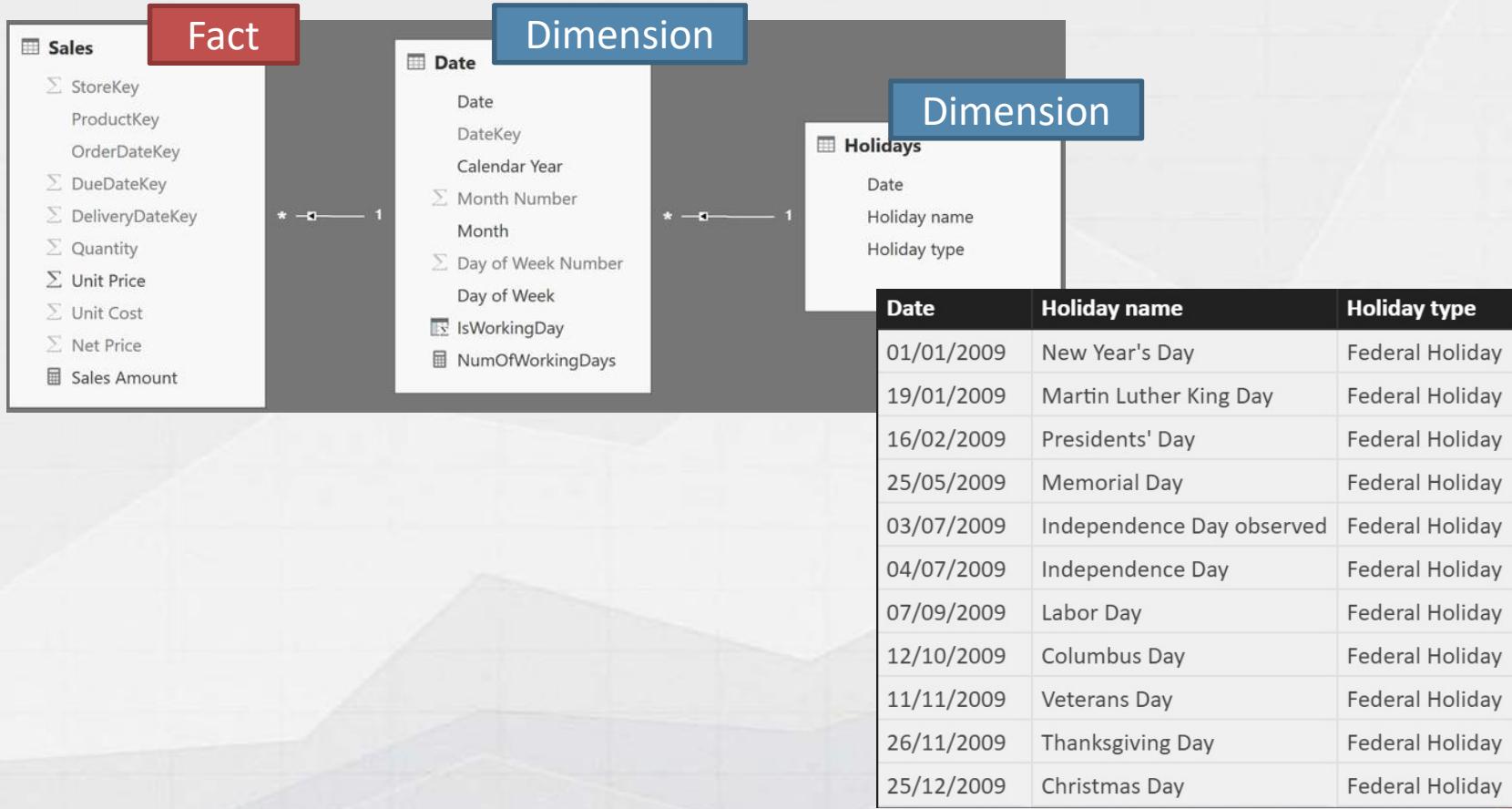
## Computing with working days



# Computing with working days

- Define what a working day is
  - Weekends (easy to compute)
  - Holidays (requires an additional dimension)
- Holidays are not the same all around the world
- Two scenarios
  - Holidays in a single country/region
  - Holidays in multiple countries/regions

# Create a holidays table (one country)



# IsWorkingDay calculated column

A simple calculated column does the trick.

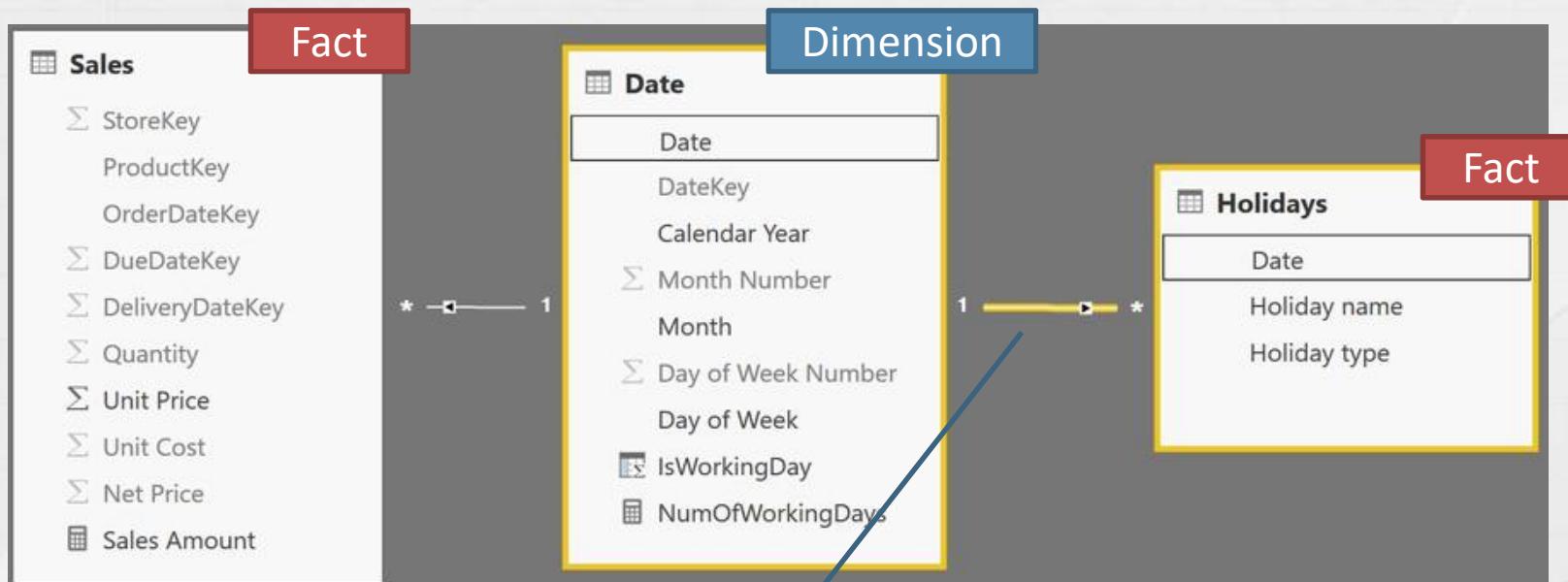
The RELATED function retrieves the presence of a holidays related to the current date.

```
'Date'[IsWorkingDay] =  
INT (  
    AND (  
        NOT ( 'Date'[Day of Week Number] IN { 1, 7 } ),  
        ISBLANK ( RELATED ( Holidays[Date] ) )  
    )  
)
```

It is not a weekend

It is not a holiday

# If Date is not a key in Holidays?



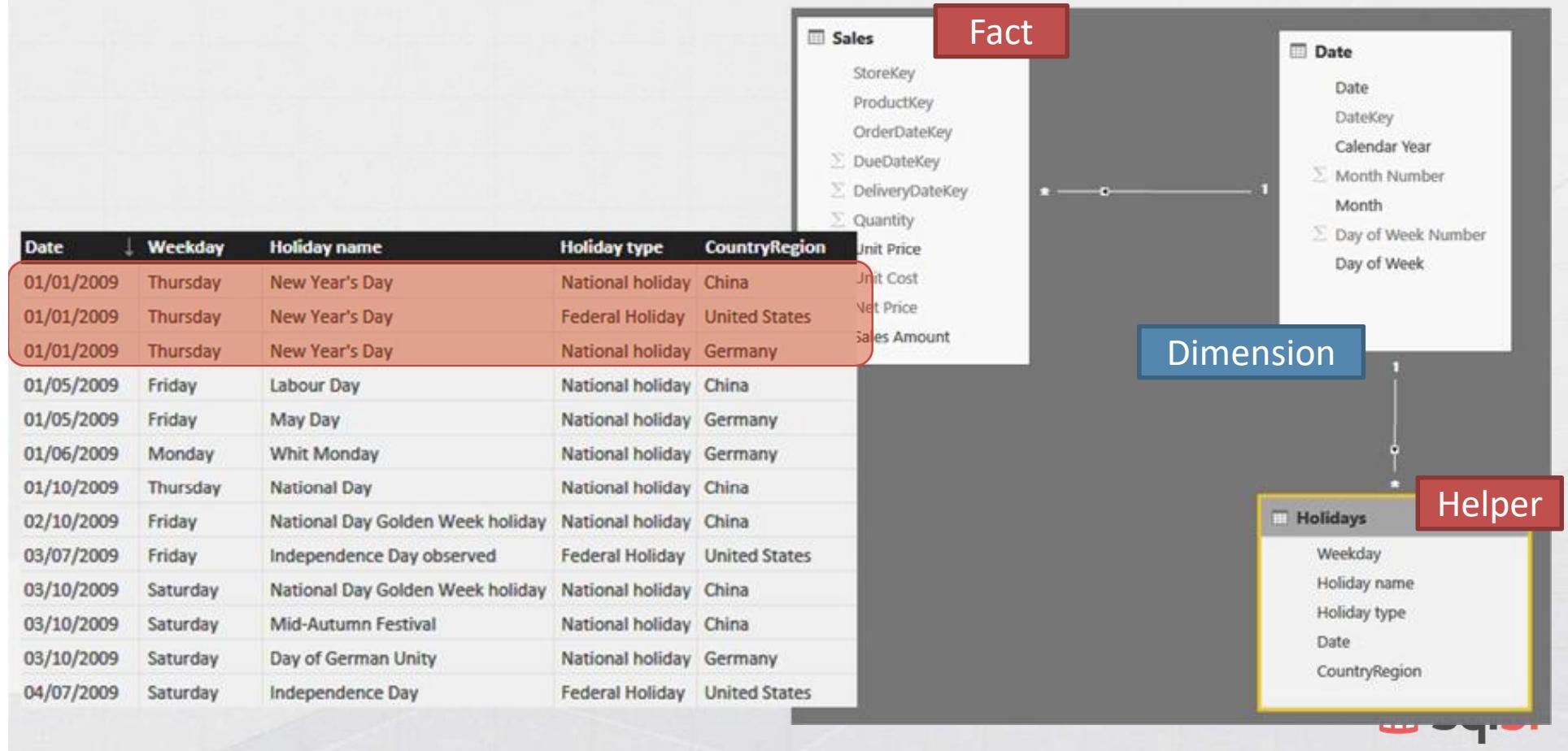
For sure, it is a key in Date, so you can reverse the relationship

# IsWorkingDay (reversed relationship)

If DateKey is not a key in the Holidays table (multiple holidays on the same date), then you can reverse the relationship and modify the calculated column:

```
'Date'[IsWorkingDay] =  
  
INT (  
    AND (  
        NOT ( 'Date'[Day of Week Number] IN { 1, 7 } ),  
        ISEMPTY ( RELATEDTABLE ( Holidays ) )  
    )  
)
```

# Create a holidays table (multiple countries)



# Create a holidays table (multiple countries)

Date	Weekday	Holiday name	Holiday type	CountryRegion
01/01/2009	Thursday	New Year's Day	National holiday	China
01/01/2009	Thursday	New Year's Day	Federal Holiday	United States
01/01/2009	Thursday	New Year's Day	National holiday	Germany
01/05/2009	Friday	Labour Day	National holiday	China
01/05/2009	Friday	May Day	National holiday	Germany
01/06/2009	Monday	Whit Monday	National holiday	Germany
01/10/2009	Thursday	National Day	National holiday	China
02/10/2009	Friday	National Day Golden Week holiday	National holiday	China
03/07/2009	Friday	Independence Day observed	Federal Holiday	United States
03/10/2009	Saturday	National Day Golden Week holiday	National holiday	China
03/10/2009	Saturday	Mid-Autumn Festival	National holiday	China
03/10/2009	Saturday	Day of German Unity	National holiday	Germany
04/07/2009	Saturday	Independence Day	Federal Holiday	United States

Month	China	Germany	United States	Total
January	4	1	2	7
February			1	1
April	1	1		2
May	2	2	1	5
June		1		1
July			2	2
September			1	1
October	4	1	1	6
November			2	2
December		2	1	3
Total	11	8	11	30

# How many holidays in a month?

- It is a “wrong” question
- Holidays cannot be summed, or averaged
- Because the number of days is always the same

Month	China	Germany	United States	Total
January	4	1	2	7
February			1	1
April	1	1		2
May	2	2	1	5
June		1		1
July			2	2
September			1	1
October	4	1	1	6
November			2	2
December		2	1	3
<b>Total</b>	<b>11</b>	<b>8</b>	<b>11</b>	<b>30</b>

# NumOfWorkingDays with multiple countries

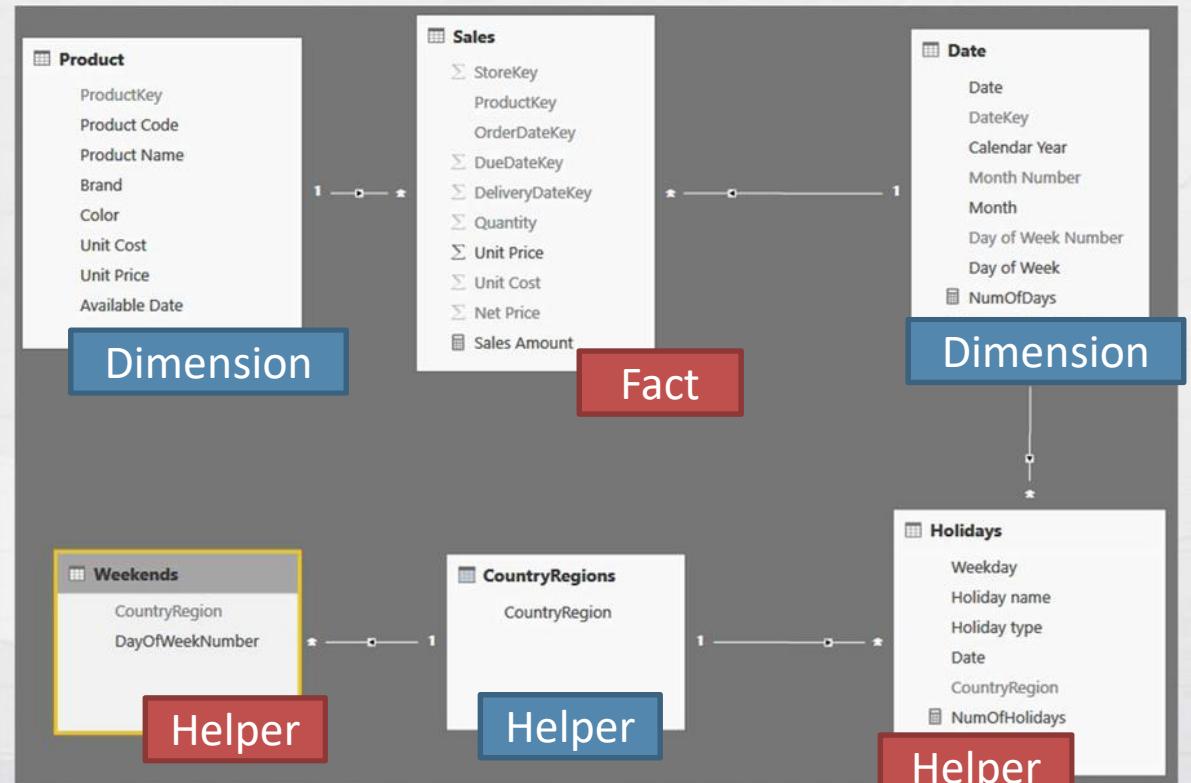
The code need to be fixed to avoid showing wrong values when multiple countries are selected and to take care of a holiday falling during a weekend.

```
NumOfWorkingDays :=  
  
IF (  
    OR (  
        HASONEVALUE ( Holidays[CountryRegion] ),  
        ISEMPTY ( Holidays )  
    ),  
    CALCULATE (  
        COUNTROWS ( 'Date' ),  
        NOT ( 'Date'[Day of Week Number] IN { 1, 7 } )  
        EXCEPT ( VALUES ( 'Date'[Date] ), VALUES ( Holidays[Date] ) )  
    )  
)
```

# Weekends are not always the same



- Weekend might fall in different days
- It depends on the territory
- Additional model activities needed



# Working days with different weekends

If weekends are stored in a separate table, the pattern need to be adjusted accordingly.

```
NumOfWorkingDays :=  
  
IF (  
    HASONEVALUE ( CountryRegions[CountryRegion] ),  
    CALCULATE (  
        COUNTROWS ( 'Date' ),  
        EXCEPT (  
            VALUES ( 'Date'[Day of Week Number] ),  
            VALUES ( Weekends[Day of Week Number] )  
        ),  
        EXCEPT (  
            VALUES ( 'Date'[Date] ),  
            VALUES ( Holidays[Date] )  
        )  
    )  
)
```

Special periods are time frames that might change year over year

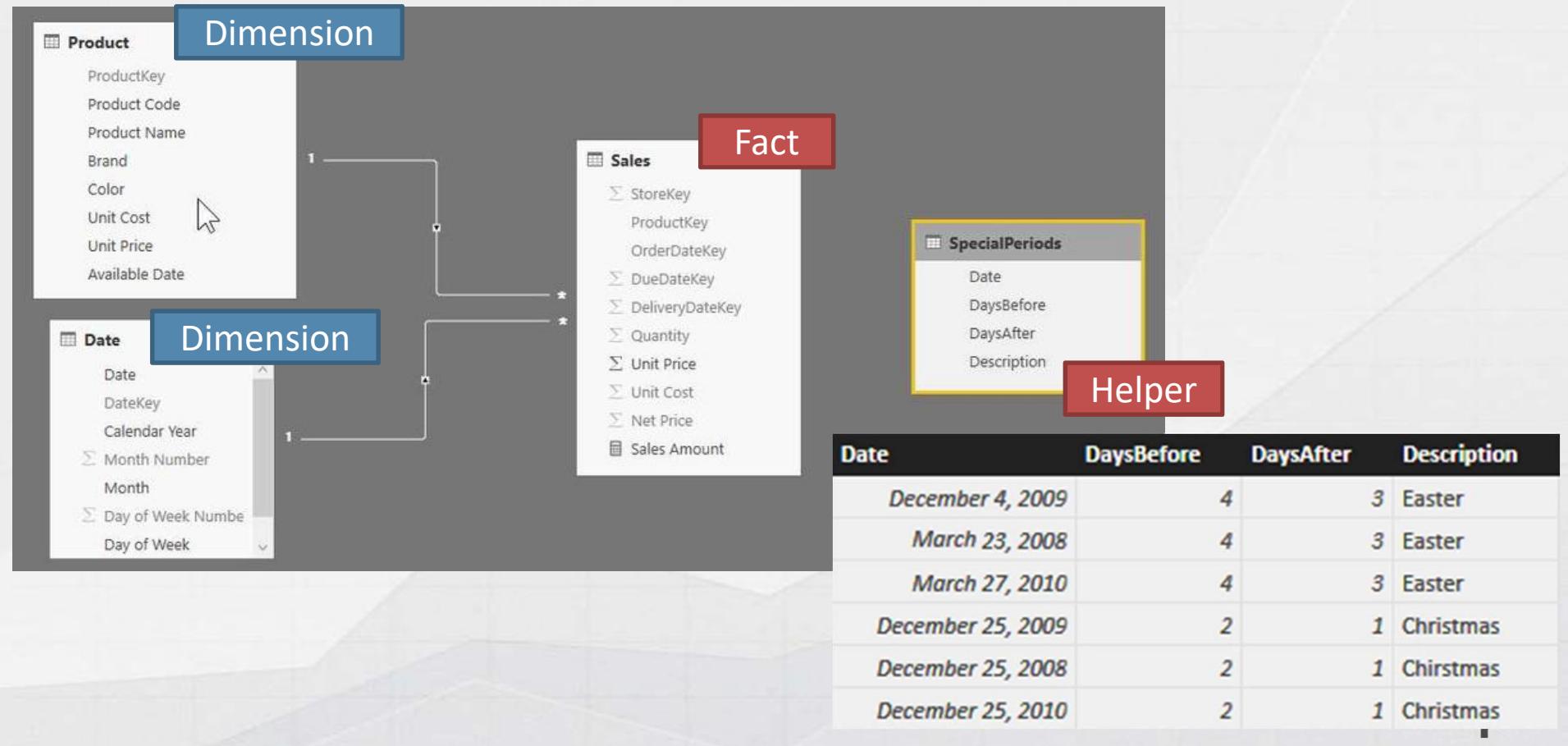
## Handling special periods of the year



# Special periods of the year

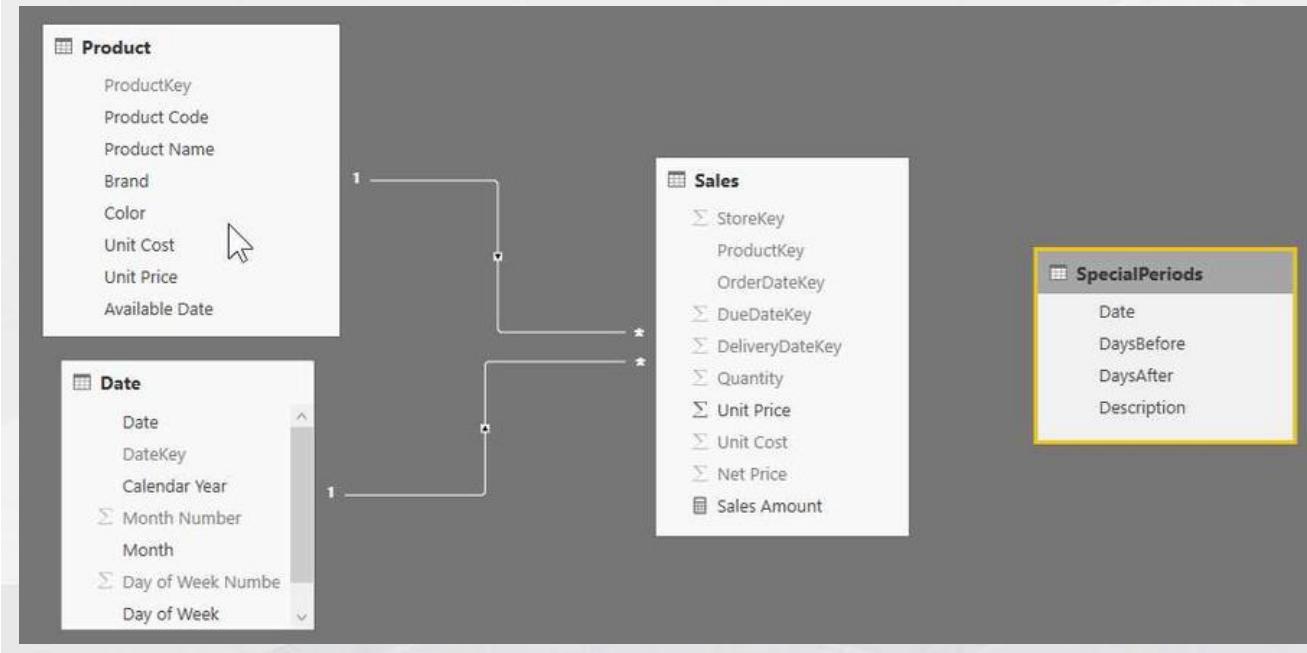
- Periods of time with a name
  - Summer holidays
  - Easter holidays
  - Christmas time
  - Seasons: spring, summer...
  - Sales seasons
- They do not happen in the same period every year
- You want to compare them between different years

# Non overlapping periods: the model



# Non-overlapping periods

How can I make  
SpecialPeriods to  
filter Sales?



# Non-overlapping special periods



With a special periods table, disconnected from the model, you can tag any date in the Date table with its special period description, using a simple calculated column.

```
'Date'[SpecialPeriod] =  
  
CALCULATE (  
    VALUES ( SpecialPeriods[Description] ),  
    FILTER (  
        SpecialPeriods,  
        AND (  
            SpecialPeriods[Date] - SpecialPeriods[DaysBefore] <= 'Date'[Date],  
            SpecialPeriods[Date] + SpecialPeriods[DaysAfter] > 'Date'[Date]  
        )  
    )  
)
```

Date	DaysBefore	DaysAfter	Description
December 4, 2009	4	3	Easter
March 23, 2008	4	3	Easter
March 27, 2010	4	3	Easter
December 25, 2009	2	1	Christmas
December 25, 2008	2	1	Chirstmas
December 25, 2010	2	1	Christmas

# Querying non-overlapping special periods

Date	DateKey	Calendar Year	Month Number	Month	Day of Week Number	Day of Week	SpecialPeriod
3/11/2008	20080311	CY 2008		3 March		3 Tuesday	
3/12/2008	20080312	CY 2008		3 March		4 Wednesday	
3/13/2008	20080313	CY 2008		3 March		5 Thursday	
3/14/2008	20080314	CY 2008		3 March		6 Friday	
3/15/2008	20080315	CY 2008		3 March		7 Saturday	
3/16/2008	20080316	CY 2008		3 March		1 Sunday	
3/17/2008	20080317	CY 2008		3 March		2 Monday	
3/18/2008	20080318	CY 2008		3 March		3 Tuesday	
3/19/2008	20080319	CY 2008		3 March		4 Wednesday	Easter
3/20/2008	20080320	CY 2008		3 March		5 Thursday	Easter
3/21/2008	20080321	CY 2008		3 March		6 Friday	Easter
3/22/2008	20080322	CY 2008		3 March		7 Saturday	Easter
3/23/2008	20080323	CY 2008		3 March		1 Sunday	Easter
3/24/2008	20080324	CY 2008		3 March		2 Monday	Easter
3/25/2008	20080325	CY 2008		3 March		3 Tuesday	Easter
3/26/2008	20080326	CY 2008		3 March			
3/27/2008	20080327	CY 2008		3 March			

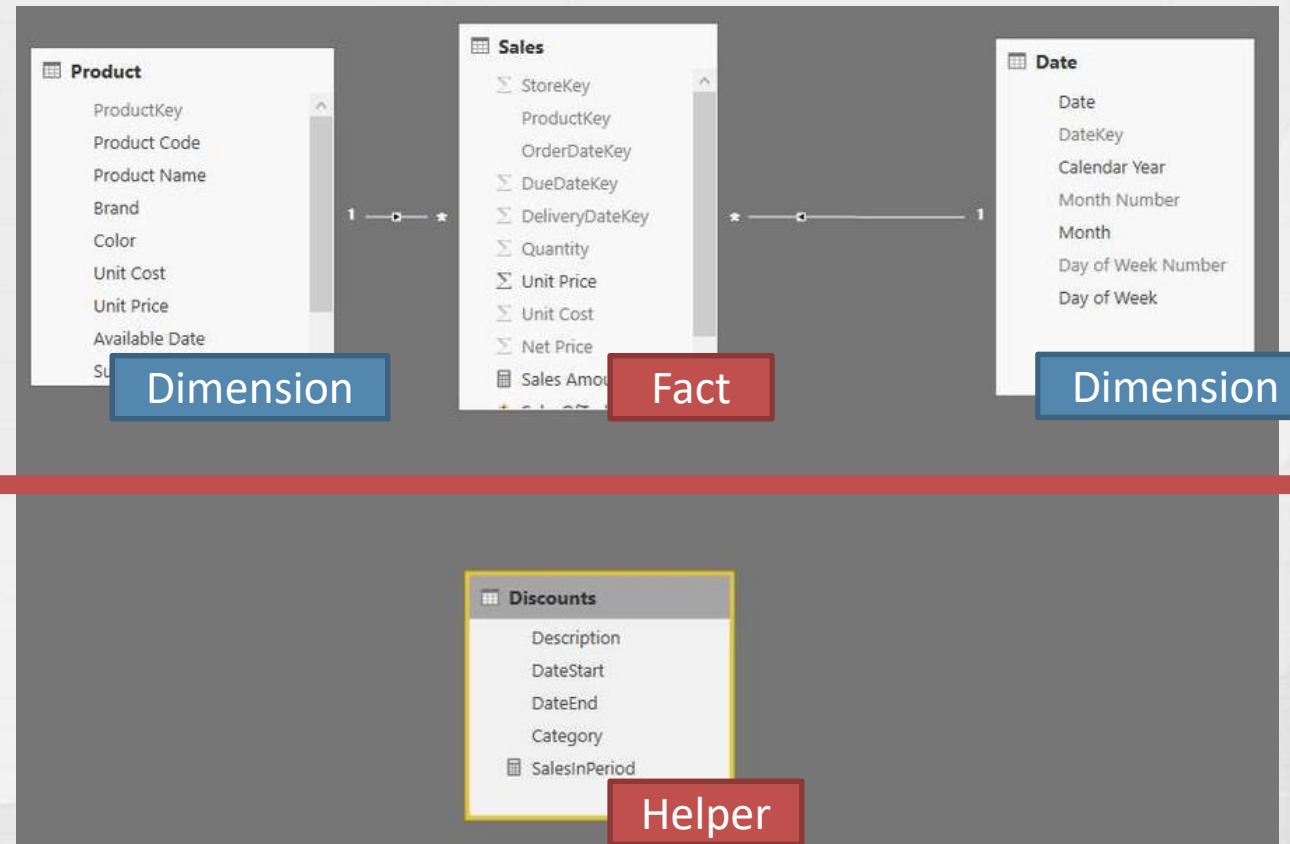
SpecialPeriod	CY 2008	CY 2009	Total
Christmas	6,940.20	30,625.28	<b>37,565.48</b>
Easter	18,426.43	32,875.65	<b>51,302.07</b>
<b>Total</b>	<b>25,366.63</b>	<b>63,500.93</b>	<b>88,867.55</b>

# Overlapping special periods

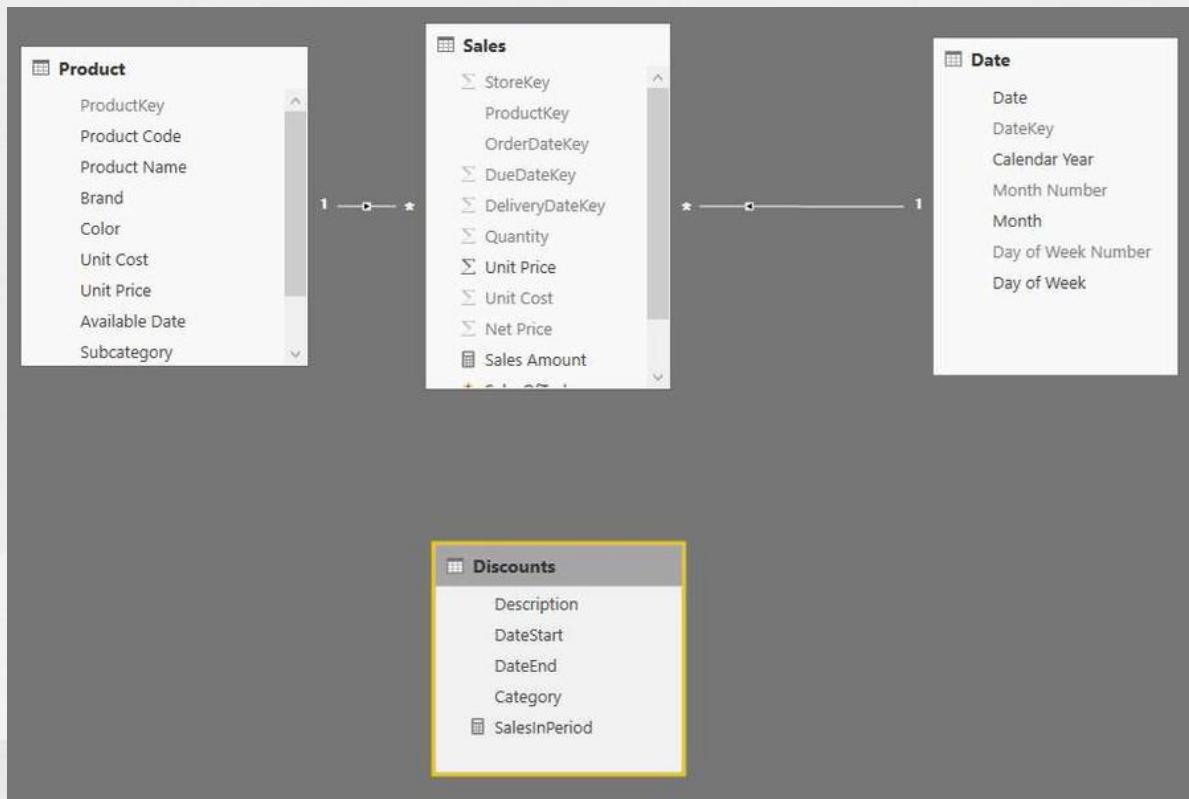
- When periods are not separated each other, they overlap
- Calculated columns are not an option, measures are required

Description	DateStart	DateEnd	Category
January Sales	1/1/2007	1/31/2007	Computers
January Sales	1/1/2008	1/31/2008	Computers
Start with Audio	1/1/2007	1/15/2007	Audio
Start with Audio	1/1/2008	1/15/2008	Audio
Summer Music	8/1/2007	8/15/2007	Audio
Summer Music	8/1/2008	8/15/2008	Audio
Holidays calls home	7/15/2007	8/15/2007	Cell phones
Holidays calls home	7/15/2007	8/15/2007	Cell phones

# No physical relationships with discounts table



# Overlapping periods



I cannot use a calculated column this time...



# Overlapping period measure

You compute with a non-additive measure the sales made for each discounted period.

```
SalesInPeriod =  
SUMX (Discounts,  
    VAR CurrentCategory  
        = Discounts[Category]  
    RETURN  
        CALCULATE (  
            [Sales Amount],  
            KEEPFILTERS (  
                DATESBETWEEN ('Date'[Date],  
                    Discounts[DateStart],  
                    Discounts[DateEnd]  
                )  
            ),  
            'Product'[Category] = CurrentCategory  
        )
```

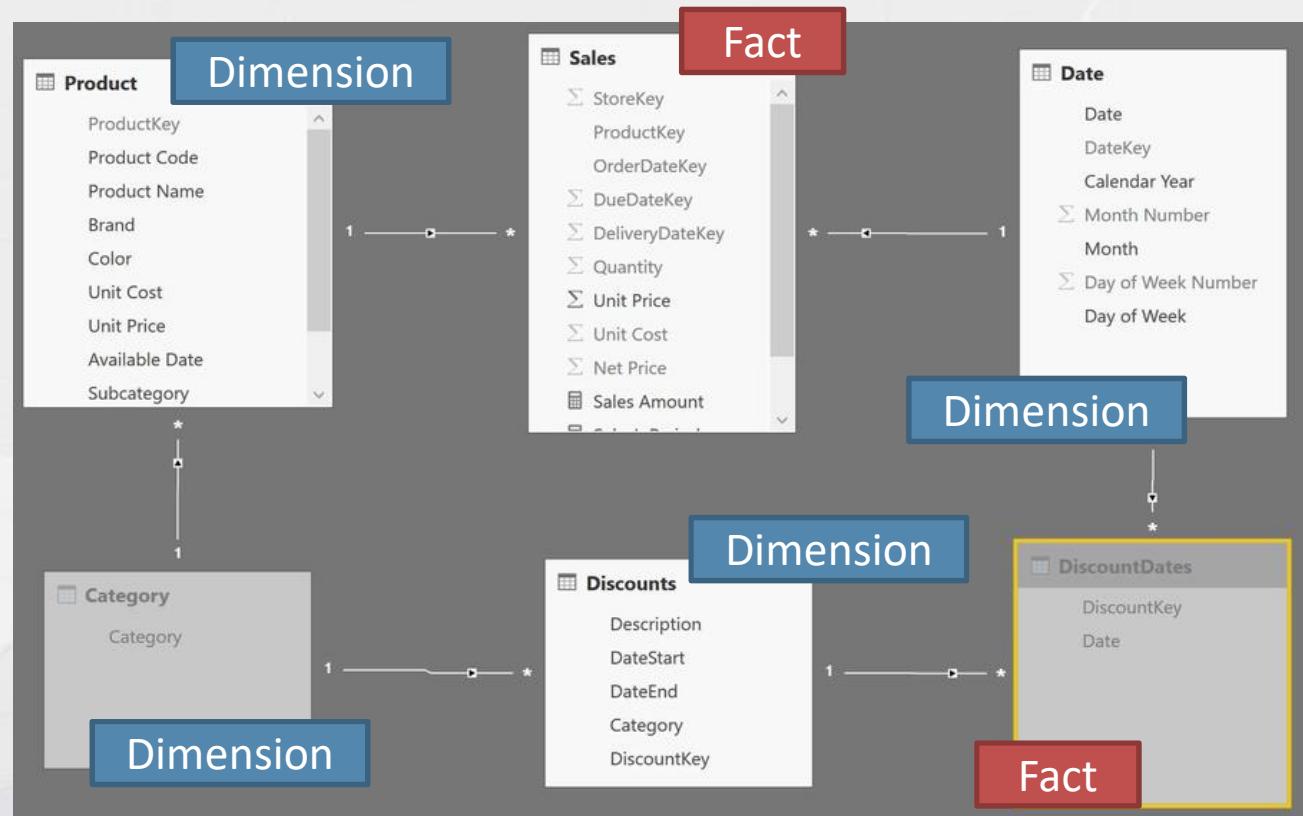
Category	Description	CY 2007	CY 2008	Total
Computers	January Sales	\$9,842.11	\$20,450.43	\$30,292.54
	<b>Total</b>	<b>\$9,842.11</b>	<b>\$20,450.43</b>	<b>\$30,292.54</b>
Cell phones	Holidays calls home	\$1,556.69		\$1,556.69
	<b>Total</b>	<b>\$1,556.69</b>		<b>\$1,556.69</b>
Audio	Start with Audio	\$492.22		\$492.22
	Summer Music	\$35.07		\$35.07
	<b>Total</b>	<b>\$527.30</b>		<b>\$527.30</b>
		\$11,926.09	\$20,450.43	\$32,376.53

# Overlapping periods – many-to-many



Build a bridge table to link discounts and date

Build a new dimension to link discounts and products



# Code for the bridge table

The bridge table can be created with DAX

```
DiscountDates =  
  
SELECTCOLUMNS (  
    GENERATE (  
        Discounts,  
        DATESBETWEEN (  
            'Date'[Date],  
            Discounts[DateStart],  
            Discounts[DateEnd]  
        )  
    ),  
    "DiscountKey", [DiscountKey],  
    "Date", [Date]  
)
```

DiscountKey	Date
1	1/1/2007 12:00:00 AM
2	1/1/2007 12:00:00 AM
5	1/1/2007 12:00:00 AM
1	1/2/2007 12:00:00 AM
2	1/2/2007 12:00:00 AM
5	1/2/2007 12:00:00 AM
1	1/3/2007 12:00:00 AM
2	1/3/2007 12:00:00 AM
5	1/3/2007 12:00:00 AM
1	1/4/2007 12:00:00 AM
2	1/4/2007 12:00:00 AM

# Overlapping periods M2M: the formula

Leveraging many-to-many relationships, the code is much simpler, although somewhat complex to understand, in its simplicity

Sales Amount M2M =

CALCULATE (  
[Sales Amount],  
DiscountDates  
)

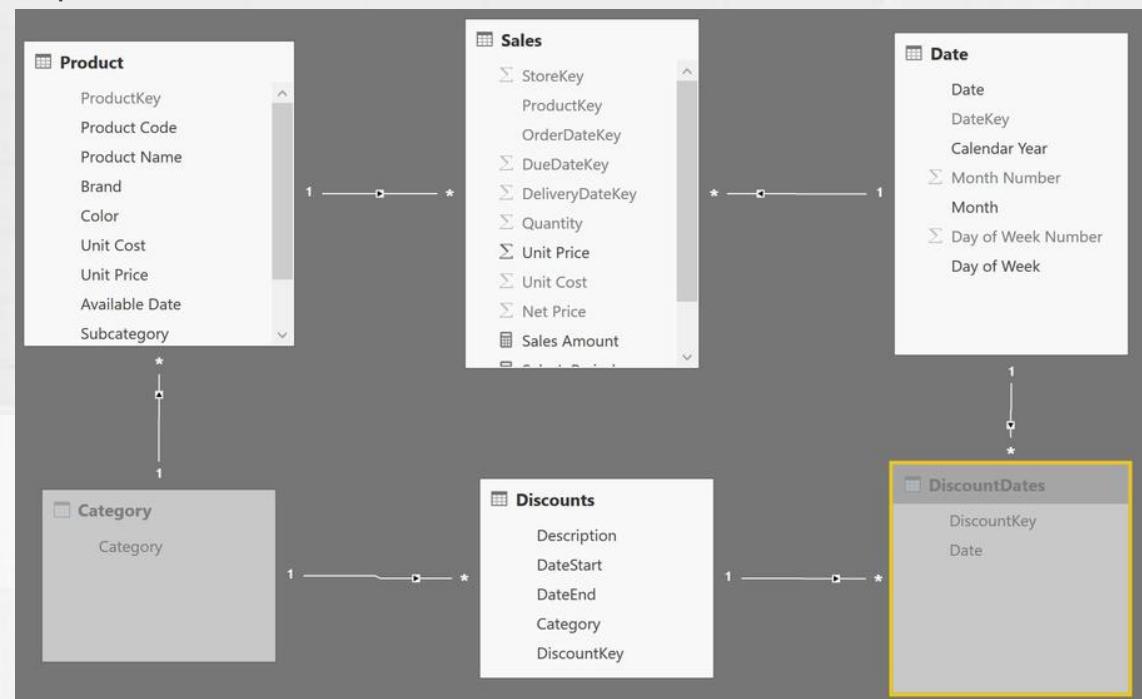
Description	Category	Month	CY 2007	CY 2008	Total
Holidays calls home	Cell phones	July	421.27	3,022.89	<b>3,444.15</b>
		August	357.07	2,520.49	<b>2,877.56</b>
		<b>Total</b>	<b>778.34</b>	<b>5,543.37</b>	<b>6,321.72</b>
January Sales	Audio	January	958.35	1,025.47	<b>1,983.82</b>
		<b>Total</b>	<b>958.35</b>	<b>1,025.47</b>	<b>1,983.82</b>
January Sales	Audio	January	958.35	1,025.47	<b>1,983.82</b>
Start with Audio	Audio	January	<b>10,800.46</b>	<b>21,475.90</b>	<b>32,276.36</b>
			958.35	1,025.47	<b>1,983.82</b>
		<b>Total</b>	<b>958.35</b>	<b>1,025.47</b>	<b>1,983.82</b>
			958.35	1,025.47	<b>1,983.82</b>
		<b>Total</b>	<b>10,800.46</b>	<b>21,475.90</b>	<b>32,276.36</b>
Start with Audio	Audio	January	958.35	1,025.47	<b>1,983.82</b>
		<b>Total</b>	<b>958.35</b>	<b>1,025.47</b>	<b>1,983.82</b>
Summer Music	Audio	August	35.07		<b>35.07</b>
		<b>Total</b>	<b>35.07</b>		<b>35.07</b>
				35.07	<b>35.07</b>
		<b>Total</b>		<b>11,613.88</b>	<b>27,019.28</b>
					<b>38,633.15</b>

# Overlapping periods M2M: relationships

The flow of filters is somewhat complex to read

Sales Amount M2M =

```
CALCULATE (
    [Sales Amount],
    DiscountDates
)
```



# Working with date and time



Time to practice and work on some models.

You will create a proper date table to browse purchases and sales, and then you will modify a model to analyze events by time.

Refer to **lab number 4** on the hands-on manual.

# Tracking historical attributes



## Attributes change over time

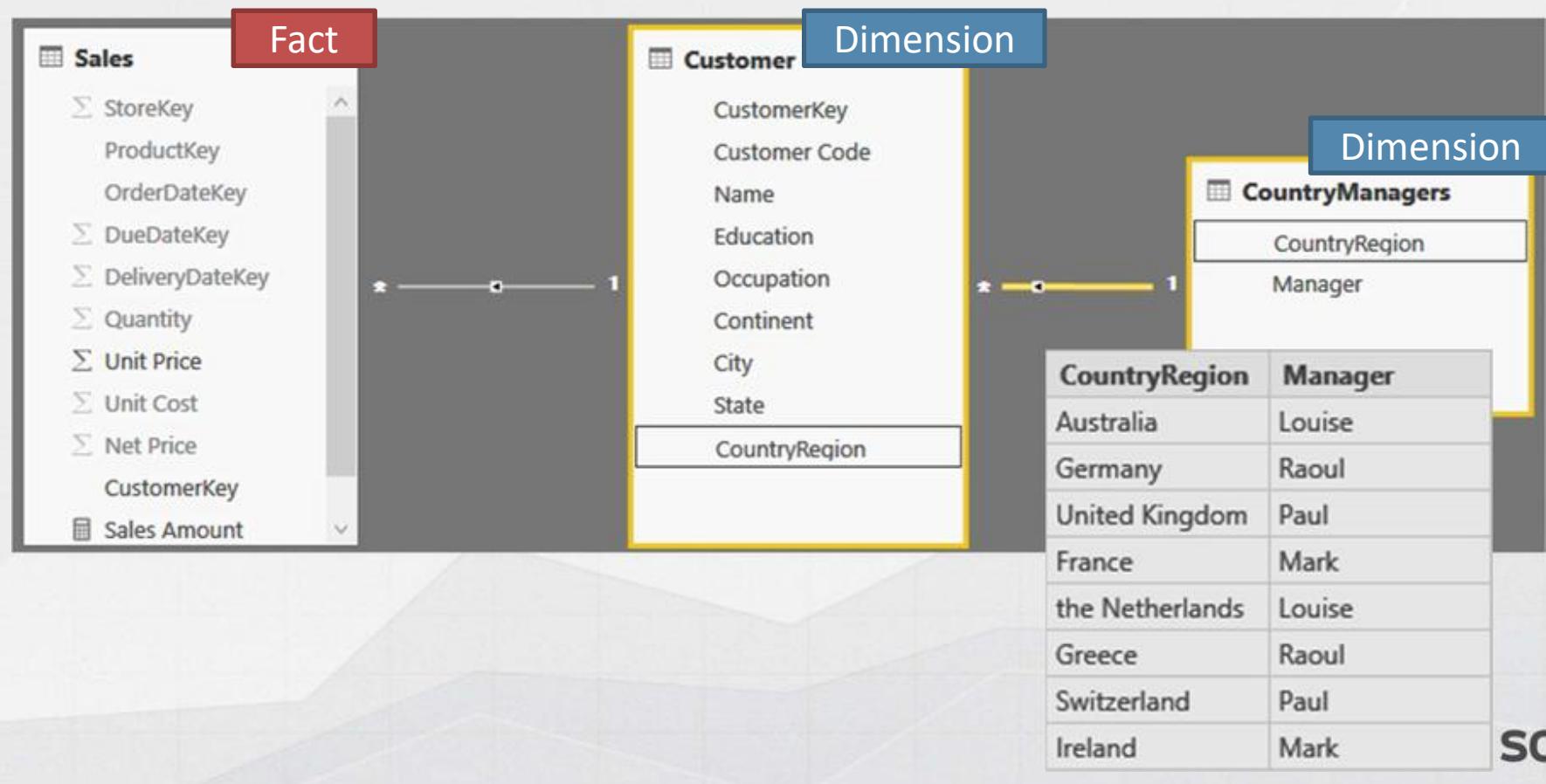
- Several dimensions' attributes might change over time
- Customer
  - Address, Category, Ranking, Age
- Product
  - Color, Brand, Manufacturer
- What happens when an attribute changes its value?
- Either you overwrite it, or you keep track of the update

## Example of slowly changing dimension

- Each country has a manager
- Country manager changes over time
- Customers belong to a given country
  - Their country manager changes
- The country is one of the attributes of the customer
- We start with a regular dimension

CountryRegion	Manager
Australia	Louise
Germany	Raoul
United Kingdom	Paul
France	Mark
the Netherlands	Louise
Greece	Raoul
Switzerland	Paul
Ireland	Mark

# Introducing slowly changing dimensions



sqlbi

# All sales are assigned to the current manager

Manager	Continent	CY 2007	CY 2008	CY 2009	Total
Louise	Asia	488.792,75	419.828,25	131.572,42	<b>1.040.193,42</b>
	Europe		21.331,60	70.311,61	<b>91.643,21</b>
	North America	61.577,18	39.782,80	11.658,93	<b>113.018,92</b>
	<b>Total</b>	<b>550.369,94</b>	<b>480.942,65</b>	<b>213.542,96</b>	<b>1.244.855,55</b>
Mark	Asia			65.381,4	<b>CountryRegion</b>
	Europe	53.587,61	40.480,90	66.078,7	Australia
	North America	457.710,49	444.308,45	405.745,6	Louise
	<b>Total</b>	<b>511.298,10</b>	<b>484.789,36</b>	<b>537.205,8</b>	Germany
Paul	Asia			60.640,0	United Kingdom
	Europe	249.495,88	101.363,69	168.773,2	Paul
	<b>Total</b>	<b>249.495,88</b>	<b>101.363,69</b>	<b>229.413,3</b>	France
	Asia			205.557,3	the Netherlands
Raoul	Europe	148.052,02	55.439,35	56.815,1	Raoul
	<b>Total</b>	<b>148.052,02</b>	<b>55.439,35</b>	<b>262.372,5</b>	Greece
<b>Total</b>		<b>1.459.215,95</b>	<b>1.122.535,05</b>	<b>1.242.534,6</b>	Switzerland
					Ireland
					Mark

## What if you do not handle SCD?



- All sales are assigned to the current manager
  - Past sales are not assigned to historical manager
- Current manager
  - Useful to analyze future sales, budget and forecasting
- Historical manager
  - Useful to analyze past sales and give bonuses
  - If not handled, then reports change over time
- Both (current and historical) are useful information

# Handling variations over time

- Add two columns to the table, to reflect years of validity
- CountryRegion is no longer a key
- The relationship can no longer be created the easy way

CountryRegion	Manager	FromYear	ToYear
Australia	Paul	2007	2008
United States	Louise	2007	2008
Canada	Louise	2007	2008
United States	Paul	2008	2009
Australia	Mark	2008	2008
United States	Louise	2009	2010
Canada	Raoul	2009	2010
Australia	Louise	2009	2010
Germany	Raoul	2007	2010
United Kingdom	Paul	2007	2010
France	Mark	2007	2010
the Netherlands	Louise	2007	2010

## The correct report

Continent	Historical Manager	CY 2007	CY 2008	CY 2009	Total
Asia	Louise			131.572,42	131.572,42
	Mark		419.828,25	65.381,46	485.209,71
	Paul	488.792,75		60.640,05	549.432,80
	Raoul			205.557,35	205.557,35
	<b>Total</b>	<b>488.792,75</b>	<b>419.828,25</b>	<b>463.151,29</b>	<b>1.371.772,29</b>
Europe	Louise		21.331,60	70.311,61	91.643,21
	Mark	53.587,61	40.480,90	66.078,71	160.147,22
	Paul	249.495,88	101.363,69	168.773,28	519.632,85
	Raoul	148.052,02	55.439,35	56.815,17	260.306,54
	<b>Total</b>	<b>451.135,51</b>	<b>218.615,55</b>	<b>361.978,75</b>	<b>1.031.729,82</b>
North America	Louise	519.287,68			519.287,68
	Mark			405.745,64	405.745,64
	Paul		444.308,45		444.308,45
	Raoul		39.782,80	11.658,93	51.441,74
	<b>Total</b>	<b>519.287,68</b>	<b>484.091,26</b>	<b>417.404,57</b>	<b>1.420.783,51</b>
<b>Total</b>		<b>1.459.215,95</b>	<b>1.122.535,05</b>	<b>1.242.534,61</b>	<b>3.824.285,61</b>

## Current versus historical attributes

- Mix current and historical attributes in a single report
- Not easy to use, but very powerful

Current Manager	Historical Manager	CY 2007	CY 2008	CY 2009	Total
Mark	Louise	457,710.49			<b>457,710.49</b>
	Mark		405,745.64		<b>405,745.64</b>
	Paul		444,308.45		<b>444,308.45</b>
	<b>Total</b>	<b>457,710.49</b>	<b>444,308.45</b>	<b>405,745.64</b>	<b>1,307,764.58</b>
Raoul	Louise	61,577.18			<b>61,577.18</b>
	Raoul		39,782.80	11,658.93	<b>51,441.74</b>
	<b>Total</b>	<b>61,577.18</b>	<b>39,782.80</b>	<b>11,658.93</b>	<b>113,018.92</b>
<b>Total</b>		<b>519,287.68</b>	<b>484,091.26</b>	<b>417,404.57</b>	<b>1,420,783.51</b>

# Slowly changing attributes or dimensions?

- We speak about slowly changing dimensions (SCD)
- In reality, the changes are in attributes (columns)
  - Customer address
  - Product manufacturer
  - Product color
- More accurate definition: slowly changing attributes
- Complexity is not in the model, it is in loading SCDs

# The customer dimension as SCD

Customer Code	CustomerKey	Name	CountryRegion	Actual Manager	Historical Manager	Year
11000	12007	Yang, Jon	Australia	Louise	Paul	2007
11000	12008	Yang, Jon	Australia	Louise	Mark	2008
11000	12009	Yang, Jon	Australia	Louise	Louise	2009
11001	22007	Huang, Eugene	Australia	Louise	Paul	2007
11001	22008	Huang, Eugene	Australia	Louise	Mark	2008
11001	22009	Huang, Eugene	Australia	Louise	Louise	2009
11002	32007	Torres, Ruben	Australia	Louise	Paul	2007
11002	32008	Torres, Ruben	Australia	Louise	Mark	2008
11002	32009	Torres, Ruben	Australia	Louise	Louise	2009
11003	42007	Zhu, Christy	Australia	Louise	Paul	2007
11003	42008	Zhu, Christy	Australia	Louise	Mark	2008
11003	42009	Zhu, Christy	Australia	Louise	Louise	2009
11004	52007	Johnson, Elizabeth	Australia	Louise	Paul	2007
11004	52008	Johnson, Elizabeth	Australia	Louise	Mark	2008

# Customer Code is no longer a unique key

Customer Code	CustomerKey	Name	CountryRegion	Actual Manager	Historical Manager	Year
11000	12007	Yang, Jon	Australia	Louise	Paul	2007
11000	12008	Yang, Jon	Australia	Louise	Mark	2008
11000	12009	Yang, Jon	Australia	Louise	Louise	2009
11001	22007	Huang, Eugene	Australia	Louise	Paul	2007
11001	22008	Huang, Eugene	Australia	Louise	Mark	2008
11001	22009	Huang, Eugene	Australia	Louise	Louise	2009
11002	32007	Torres, Ruben	Australia	Louise	Paul	2007
11002	32008	Torres, Ruben	Australia	Louise	Mark	2008
11002	32009	Torres, Ruben	Australia	Louise	Louise	2009
11003	42007	Zhu, Christy	Australia	Louise	Paul	2007
11003	42008	Zhu, Christy	Australia	Louise	Mark	2008
11003	42009	Zhu, Christy	Australia	Louise	Louise	2009
11004	52007	Johnson, Elizabeth	Australia	Louise	Paul	2007
11004	52008	Johnson, Elizabeth	Australia	Louise	Mark	2008

# Counting customers with SCD



When using SCD, you can no longer use a simple DISTINCTCOUNT to count the customers, you need to rely on bidirectional filtering to compute a correct value, otherwise you count versions, not customers.

```
NumOfBuyingCustomers := DISTINCTCOUNT ( Sales[CustomerKey] )
```

```
NumOfBuyingCustomersCorrect :=
```

```
CALCULATE (
    DISTINCTCOUNT ( Customers[Customer Code] ),
    CROSSFILTER (
        Sales[CustomerKey],
        Customer[CustomerKey],
        BOTH
    )
)
```

Brand	NumOfBuyingCustomers	NumOfBuyingCustomersCorrect
A. Datum	189	189
Adventure Works	392	392
Contoso	820	815
Fabrikam	173	173
Litware	201	199
Northwind Traders	151	151
Proseware	127	125
Southridge Video	862	855
Tailspin Toys	594	594
The Phone Company	76	76
Wide World Importers	133	133
<b>Total</b>	<b>2.395</b>	<b>2.353</b>

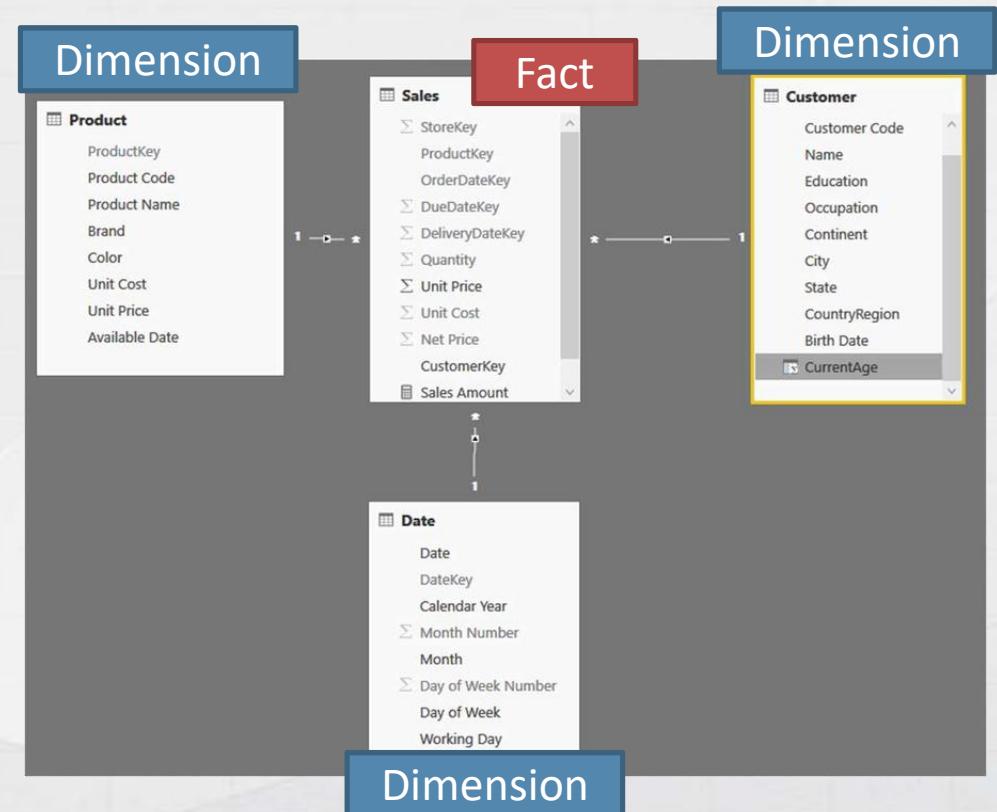
Sometimes attributes change very frequently

## Rapidly changing dimensions

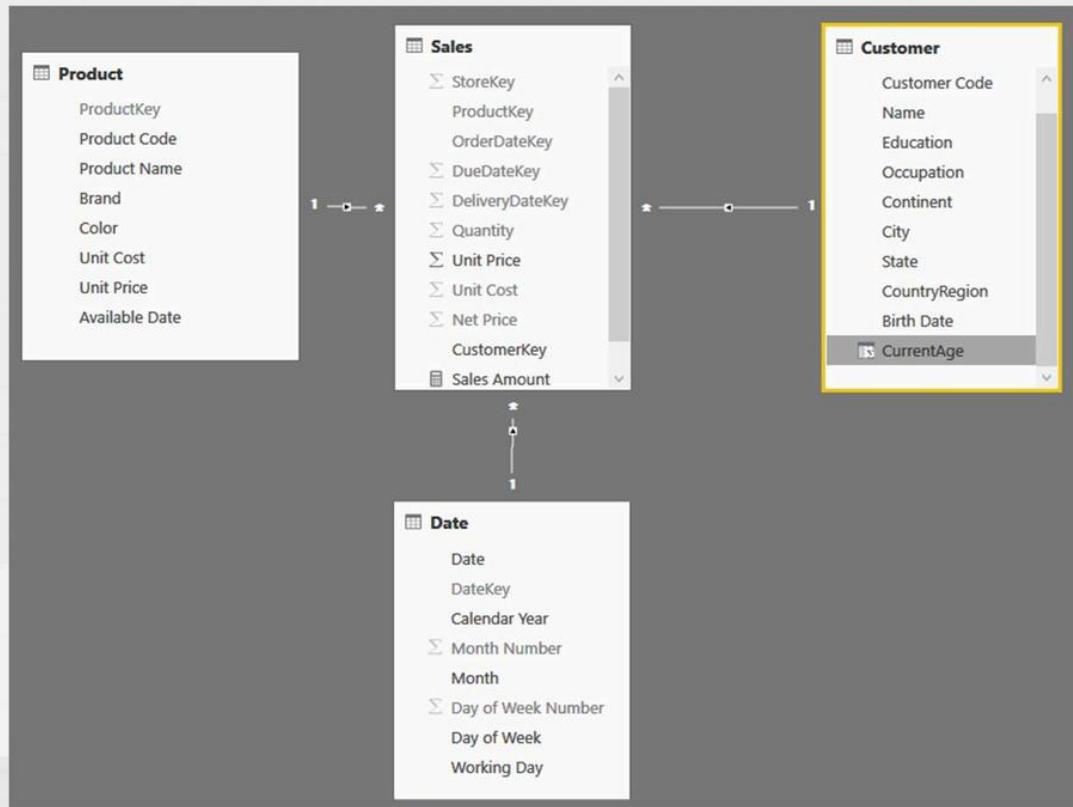


# Rapidly changing dimensions

- Age changes every year
- Is it fine to store it in the customer table?
- Doing so, it would generate a new customer version every year



# Slowly Changing Attributes...



Can I handle an SCA  
without creating an  
SCD?

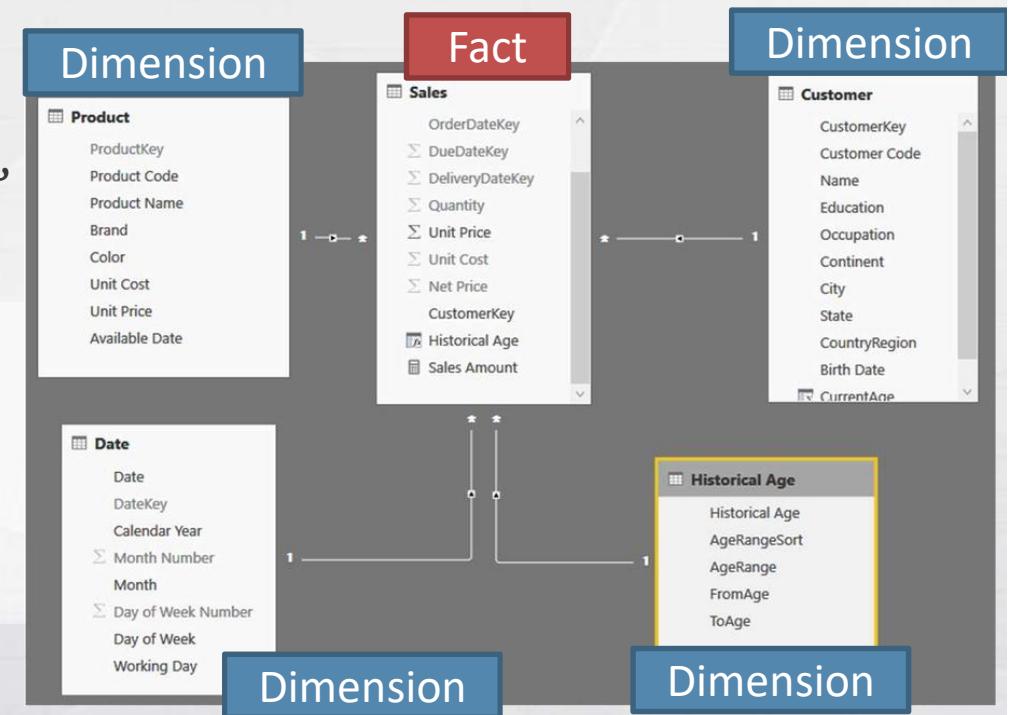


# Attributes in the fact table

You can store the historical age straight in the fact table by using a simple calculated column.  
In this way, customer is no longer an SCD and its handling is much simpler

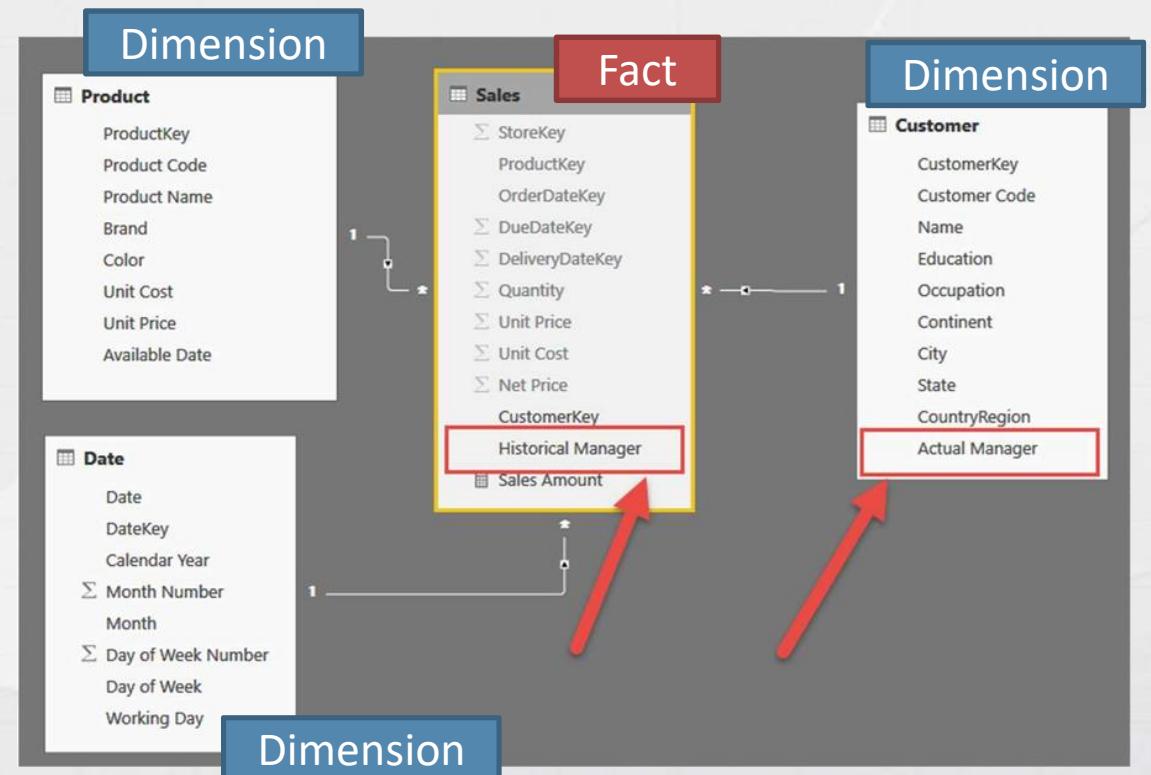
Sales[Historical Age] =

```
DATEDIFF (
    RELATED ( Customer[Birth Date] ),
    RELATED ( 'Date'[Date] ),
    YEAR
)
```



# Which technique is the best one?

- Small set of SCD attributes
  - Denormalize in the fact table
- Complex SCD
  - Handle the dimension as an SCD



Snapshots are powerful tools to analyze facts in a given scenario

## Using snapshots



# What is a snapshot?

- A fact is typically an event
  - An event happens at some point in time
  - The value associated can be aggregated
- Sometimes, a fact is not an event
  - Measure taken at some point in time
  - For example
    - Temperature of an engine
    - Number of customers served in one day
    - Current balance of an account
  - As such, aggregating it is much harder

# Natural versus derived snapshots

- Natural snapshots
  - Natural snapshots need to be handled as they are
  - Examples
    - Temperature of an engine
    - Exchange rate of a currency
    - Average age of your customers
- Derived snapshots
  - Derived snapshots can be computed from transactional data
  - Examples
    - Current balance of an account
    - Average number of orders handled daily

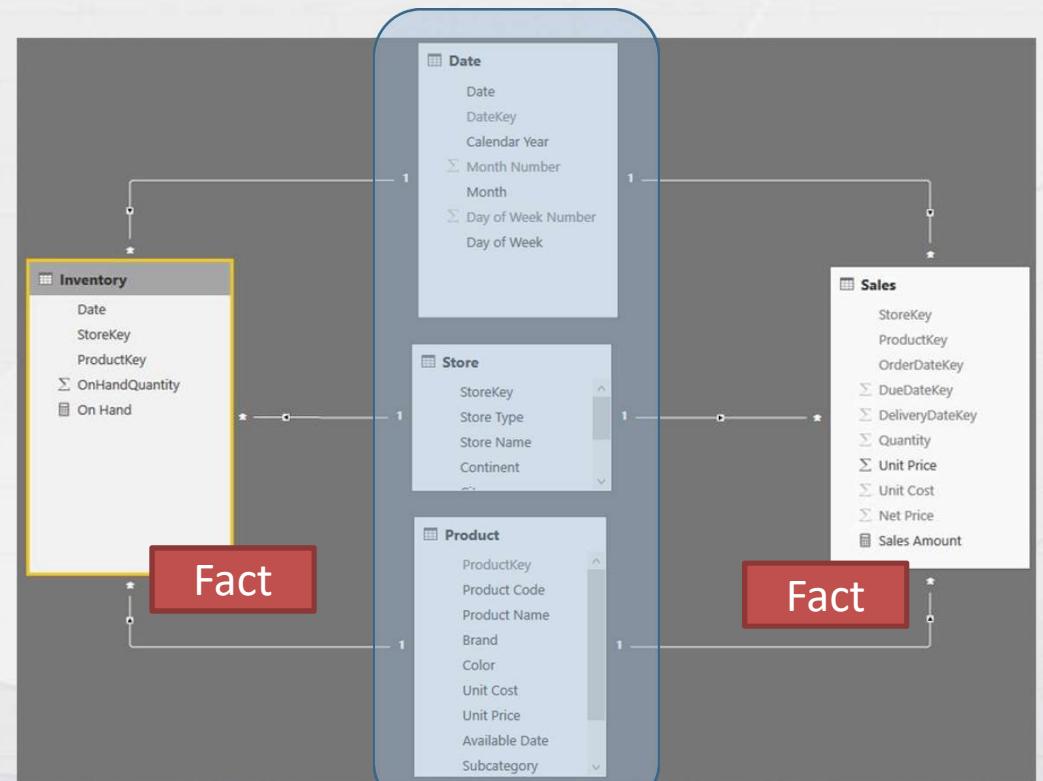
# Sales versus Inventory in the same model

## Sales

- Transaction table
- Aggregated by simple SUM

## Inventory

- Snapshot
- Cannot be aggregated using SUM



Dimensions

# Aggregating Inventory using SUM



Product Name
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Black
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Red
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Silver
<input type="checkbox"/> NT Bluetooth Active Headphones E202 White
<input checked="" type="checkbox"/> NT Bluetooth Stereo Headphones E52 Black
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Blue
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Pink
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Yellow
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E30...

Calendar Year	Month	Giebelstadt Store	Munich Store	obamberg Store	Total
CY 2007	January		6		6
	February		18		18
	March			8	8
	April	8	8	24	40
	May	24	18		42
	June			6	6
	July			20	20
	October	6	6	6	18
	November	18	18	18	54
	<b>Total</b>	<b>56</b>	<b>74</b>	<b>82</b>	<b>212</b>
	May		34		34
	June	18	63		81
CY 2008	July	42	42		84
	<b>Total</b>	<b>60</b>	<b>139</b>		<b>199</b>
	November			20	20
	December			27	27
	<b>Total</b>	<b>116</b>	<b>213</b>	<b>129</b>	<b>458</b>

Aggregating snapshots  
using SUM leads to  
wrong totals

## Non-additive measures

- Snapshots generate non-additive measures
- What are non-additive measure?
  - Additive measures aggregate by using SUM over all dimensions
  - Non-additive measures do not use SUM
  - Semi-additive measures use SUM on some dimensions, and different aggregations on other dimensions
- Typically, semi-additive are non-additive against time

# Non-Additive calculations

What numbers should I display in the red lines? And how do I accomplish that?

Product Name	Calendar Year	Month	Giebelstadt Store	Munich Store	obamberg Store	Total
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Black	CY 2007	January		6		6
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Red		February		18		18
<input type="checkbox"/> NT Bluetooth Active Headphones E202 Silver		March			8	8
<input type="checkbox"/> NT Bluetooth Active Headphones E202 White		April	8	8	24	40
<input checked="" type="checkbox"/> NT Bluetooth Stereo Headphones E52 Black		May	24	18		42
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Blue		June			6	6
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Pink		July			20	20
<input type="checkbox"/> NT Bluetooth Stereo Headphones E52 Yellow		October	6	6	6	18
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...		November	18	18	18	54
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...		<b>Total</b>	<b>56</b>	<b>74</b>	<b>82</b>	<b>212</b>
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E10...						
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E30...	CY 2008	May		34		34
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E30...		June	18	63		81
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E30...		July	42	42		84
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones E30...		<b>Total</b>	<b>60</b>	<b>139</b>		<b>199</b>
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones M4...	CY 2009	November			20	20
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones M4...		December			27	27
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones M4...		<b>Total</b>			<b>47</b>	<b>47</b>
<input type="checkbox"/> NT Wireless Bluetooth Stereo Headphones M4...			<b>116</b>	<b>213</b>	<b>129</b>	<b>458</b>



# LASTDATE does not work here



Standard solution for semi-additive is the usage of the LASTDATE table filter.  
Unfortunately, this is not an option: the totals disappear and the numbers are wrong

```
On Hand :=  
CALCULATE (  
    SUM ( Inventory[OnHandQuantity] ),  
    LASTDATE ( 'Date'[Date] )  
)
```

Calendar Year	Month	Date	Giebelstadt Store	Munich Store	obamberg Store	Total
CY 2007	January	1/27/2007		6		6
		<b>Total</b>				
	February	2/3/2007		6		6
		2/10/2007		6		6
		2/17/2007		6		6
		<b>Total</b>				
	March	3/3/2007			8	8
		<b>Total</b>				
	April	4/7/2007			6	6
		4/14/2007			6	6
		4/21/2007	8	8	6	22
		4/28/2007			6	6
		<b>Total</b>				

# The correct code is more complex



The correct aggregation is by the last available date, and it requires some DAX skills

```
CALCULATE (
    SUM ( Inventory[OnHandQuantity] ),
    LASTNONBLANK(
        'Date'[Date],
        CALCULATE (
            COUNTROWS ( Inventory ),
            ALLEXCEPT (
                Sales,
                'Date'
            )
        )
    )
)
```

Calendar Year	Month	Date	Giebelstadt Store	Munich Store	obamberg Store	Total
CY 2007	January	1/27/2007		6		6
		<b>Total</b>		<b>6</b>		<b>6</b>
	February	2/3/2007		6		6
		2/10/2007		6		6
		2/17/2007		6		6
		<b>Total</b>		<b>6</b>		<b>6</b>
	March	3/3/2007			8	8
		<b>Total</b>			<b>8</b>	<b>8</b>
	April	4/7/2007			6	6
		4/14/2007			6	6
		4/21/2007	8	8	6	22
		4/28/2007			6	6
		<b>Total</b>			<b>6</b>	<b>6</b>
	May	5/5/2007	8	6		14

# Optimizing performance



The previous formula tends to be slow on large models  
You can optimize it by creating a calculated helper column in Date

```
Date[RowsInInventory] = CALCULATE ( NOT ISEMPTY ( Inventory ) )
```

On Hand :=

```
CALCULATE (
    SUM ( Inventory[OnHandQuantity] ),
    CALCULATETABLE (
        LASTDATE ( 'Date'[Date] ),
        'Date'[RowsInInventory] = TRUE
    )
)
```

Typically, snapshots have a different granularity than regular fact tables

## Snapshots and granularity



# Derived snapshots and granularity

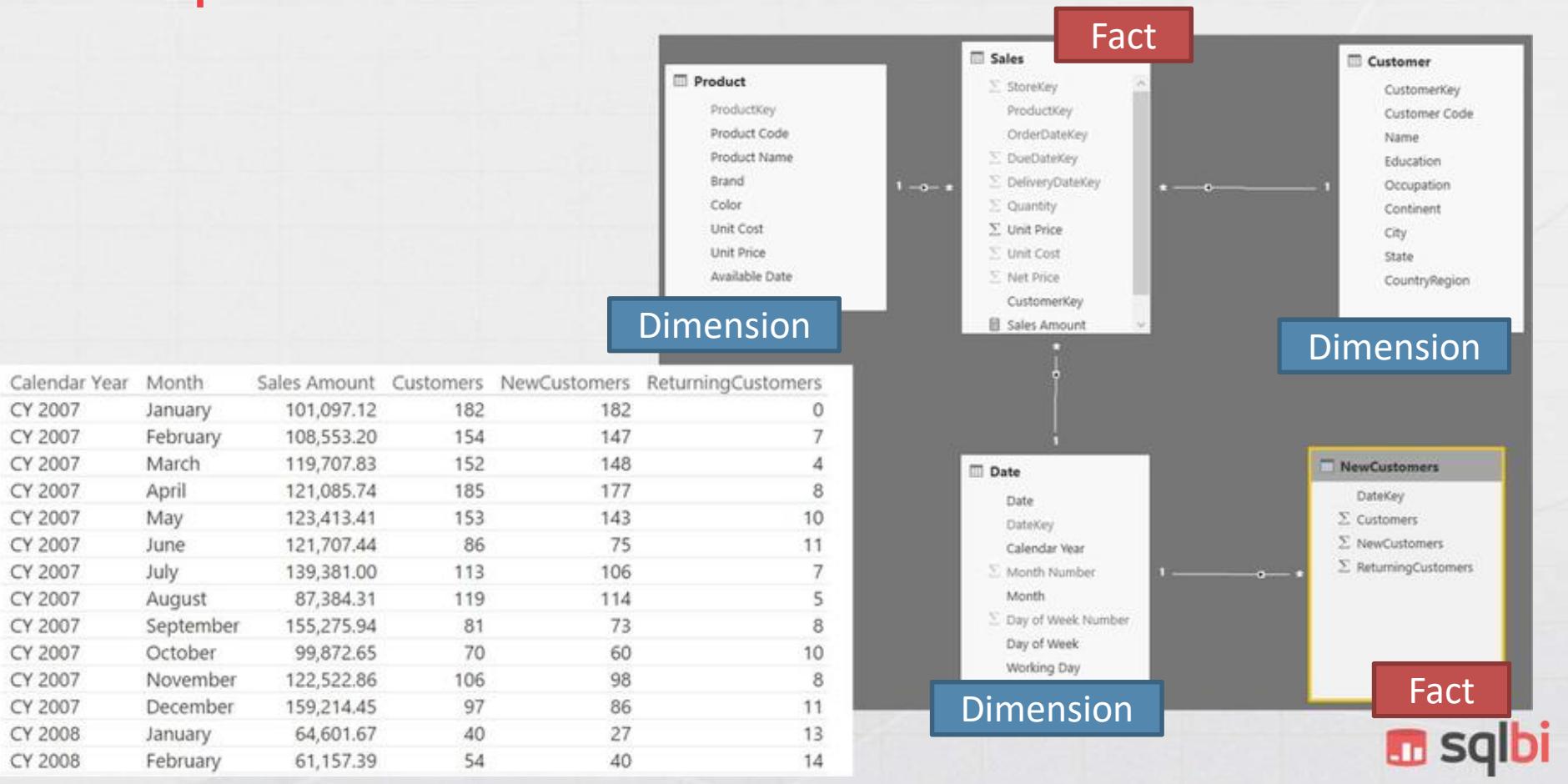
- Derived snapshots are good for performance
- Precomputing values makes the system faster
- But you need to pay attention to granularity
- Granularity, for a snapshot, is defined once and forever
- Let us see this with an example

# Computing new and returning customers

- Derived snapshot
- Storing new and returning customers by day
- Granularity is monthly, which becomes evident once you put the table in the model

DateKey	Customers	NewCustomers	ReturningCustomers
20070131	182	182	0
20080131	40	27	13
20090131	21	10	11
20070228	154	147	7
20080229	54	40	14
20090228	47	39	8
20070331	152	148	4
20080331	61	56	5
20090331	49	42	7
20070430	185	177	8
20080430	100	94	6
20090430	26	19	7
20070531	153	143	10
20080531	43	38	5

# Snapshot for new customers



# Snapshots and granularity

- Granularity at the daily level has side effects:
  - Cannot generate subtotals
    - Monthly customers?
    - Quarterly returning customers?
  - Cannot slice by attributes
    - New customers by country?
    - Returning customers by education?
  - Granularity is fixed: speed versus flexibility
- Need for different snapshots if you need different granularities

Advanced usage of snapshots to create a powerful data model

# Transition matrix



# Transition Matrix

- First, rank the customers based on a configuration

Rating	MinSale	MaxSale
Low	0	100
Medium	100	500
High	500	999999999

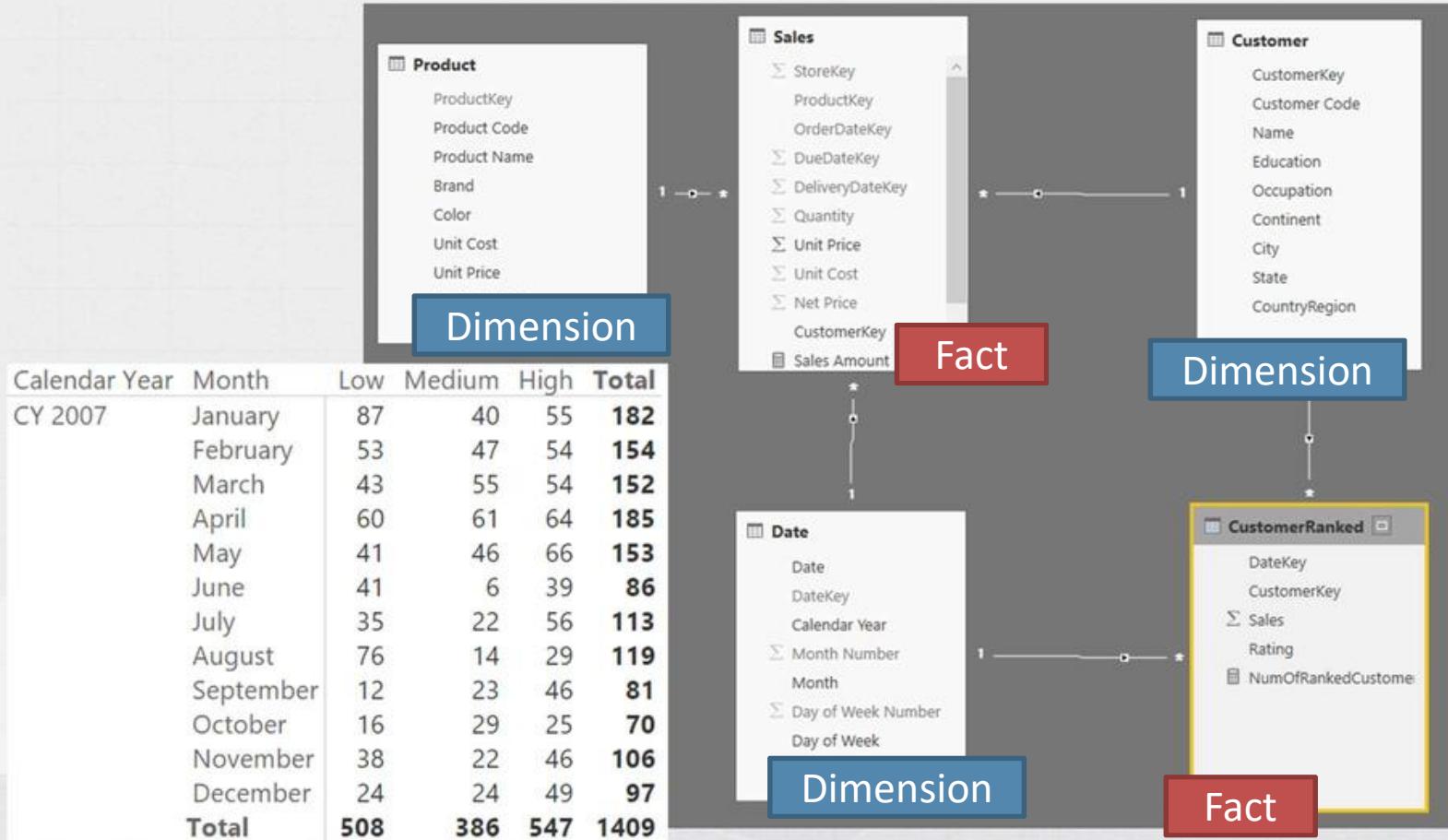
- A monthly ranking is assigned to each customer
- How is the ranking of customers evolving over time?
  - For example: how many of the customers ranked High in January are still ranked High in August?

# Ranking Snapshot

- Snapshot granularity
  - Month
  - Customer
- Values
  - Sales Amount
  - Historical rating

DateKey	CustomerKey	Sales	Rating
20070228	11567	\$1,136.46	High
20070228	11601	\$2,302.68	High
20070228	4550	\$607.96	High
20070228	6126	\$627.00	High
20070228	10056	\$3,383.15	High
20070228	6124	\$313.50	Medium
20070228	6125	\$313.50	Medium
20070228	6336	\$313.50	Medium
20070228	6337	\$313.50	Medium
20070228	6341	\$313.50	Medium
20070228	956	\$103.55	Medium

# The data model for ranking



# Transition Matrix

- Selection of customers:
  - Having a rating (Medium) evaluated in a month (Jan 2007)
- Business question:
  - How did selected customers evolved in the next months?

Year Month		Rating			
January 2007		Medium			
Calendar Year	Month	Low	Medium	High	Total
CY 2007	January		40		40
	April			1	1
	May	1			1
	November			1	1
	<b>Total</b>	<b>1</b>	<b>40</b>	<b>2</b>	<b>40</b>
CY 2009	June	4			4
	<b>Total</b>	<b>4</b>			<b>4</b>
<b>Total</b>		<b>5</b>	<b>40</b>	<b>2</b>	<b>40</b>

## Parameter table

- Month and rating slicers require a parameter table
- No relationships with other tables
- Created with a calculated table

# Parameter table: the code

The parameter table can be created by a simple CROSSJOIN of the parameter and the rating, with some additional calculated columns

```
SnapshotParameters =  
  
SELECTCOLUMNS (  
    ADDCOLUMNS (  
        CROSSJOIN (  
            SUMMARIZE ( CustomerRanked, 'Date'[Calendar Year], 'Date'[Month] ),  
            DISTINCT ( CustomerRanked[Rating] )  
        ),  
        "DateKey", CALCULATE ( MAX ( 'Date'[DateKey] ) )  
    ),  
    "DateKey", [DateKey],  
    "Year Month", FORMAT ( CALCULATE ( MAX ( 'Date'[Date] ) ), "mmmm YYYY" ),  
    "Rating", [Rating]  
)
```

# Transition matrix: the formula



The formula moves the filter from the parameter table to the CustomerRanked one, using the TREATAS function.

Transition Matrix =

```
CALCULATE (
    DISTINCTCOUNT ( CustomerRanked[CustomerKey] ),
    CALCULATETABLE (
        VALUES ( CustomerRanked[CustomerKey] ),
        TREATAS (
            VALUES ( SnapshotParameters[DateKey] ),
            'Date'[DateKey]
        ),
        TREATAS (
            VALUES ( SnapshotParameters[Rating] ),
            CustomerRanked[Rating]
        ),
        ALL ( CustomerRanked[RatingSort] ),
        ALL ( 'Date' )
    )
)
```

Year Month		Rating		
Calendar Year	Month	Low	Medium	High
JANUARY 2007	January	40	1	40
	April		1	1
	May	1		1
	November		1	1
	Total	1	40	2
CY 2007				40
	June	4		4
	Total	4		4
CY 2009				
	Total	5	40	2
				40

## Transition matrix: conclusions

- Snapshot needed in this case, to perform a more complex analysis
- Granularity is fixed, but this is the desired effect
- Need for a parameter table to filter the snapshot
- Need to use TREATAS or INTERSECT (in previous versions of DAX) to move the filter to the snapshot
- Snapshots are very powerful, but they tend to be complex to use

# Using snapshots



Time to practice and work on some models.

In this lab you build a snapshot at different granularities and experience how the calculations are affected by the snapshot grain.

Refer to **lab number 5** on the hands-on manual.

# Analyzing date and time intervals

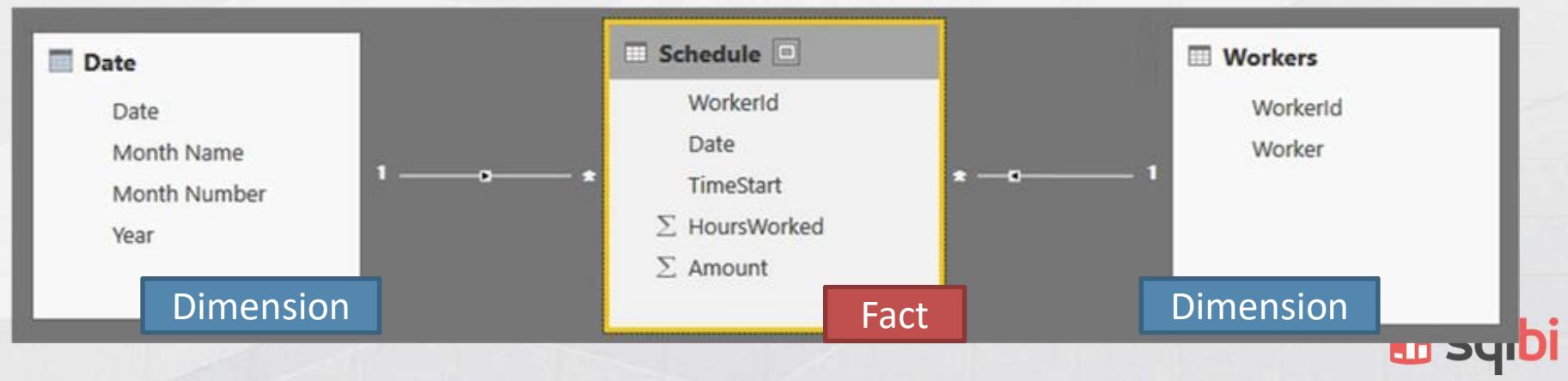


## What are intervals?

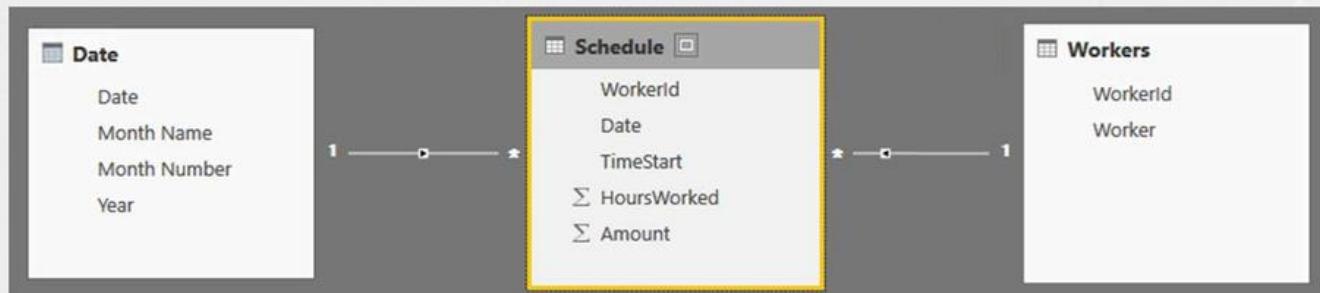
- Handling intervals is different from time intelligence
- Time intelligence analyzes events in different periods
- Intervals are not atomic events, they have a duration
  - The date is when the event started
  - You typically have a duration column (or an end date)
  - It might be seconds, minutes, days, years...
- Slicing by date it not enough, you need DAX code

# Working schedule

- A worker starts a shift that has a given duration
- Date and TimeStart is when the shift starts
- HoursWorked is the shift duration
- Amount is the corresponding cost



# Working schedule report



WorkerId	Date	TimeStart	HoursWorked	Amount
1	1/1/2016	9:00:00 AM	8	160
1	1/15/2016	6:00:00 PM	6	180
1	1/31/2016	9:00:00 PM	9	360
2	1/1/2016	9:00:00 AM	8	160
2	1/15/2016	6:00:00 PM	5	150
2	1/31/2016	9:00:00 PM	8	320
1	2/1/2016	9:00:00 AM	4	80
1	2/15/2016	6:00:00 PM	3	90
1	2/29/2016	9:00:00 PM	8	320
2	2/1/2016	9:00:00 AM	6	120
2	2/15/2016	6:00:00 PM	5	150
2	2/29/2016	9:00:00 PM	8	320

Year	Michelle	Paul	Total
2016	40	38	78
January	21	23	44
February	19	15	34
<b>Total</b>	<b>40</b>	<b>38</b>	<b>78</b>

# The desired output

- Show dates which are not present in the fact table
- Split working hours the right way

Year	Michelle	Paul	Total
2016	40	38	78
January	21	23	44
1/1/2016	8	8	16
1/15/2016	5	6	11
1/31/2016	8	9	17
February	19	15	34
2/1/2016	6	4	10
2/15/2016	5	3	8
2/29/2016	8	8	16
<b>Total</b>	<b>40</b>	<b>38</b>	<b>78</b>

Year	Michelle	Paul	Total
2016	40	38	78
January	16	17	33
1/1/2016	8	8	16
1/15/2016	5	6	11
1/31/2016	3	3	6
February	19	16	35
2/1/2016	11	10	21
2/15/2016	5	3	8
2/29/2016	3	3	6
March	5	5	10
3/1/2016	5	5	10
<b>Total</b>	<b>40</b>	<b>38</b>	<b>78</b>

# Working schedule

WorkerId	Date	TimeStart	HoursWorked	Amount
1	1/1/2016	9:00:00 AM	8	160
1	1/15/2016	6:00:00 PM	6	180
1	1/31/2016	9:00:00 PM	9	360
2	1/1/2016	9:00:00 AM	8	160
2	1/15/2016	6:00:00 PM	5	150
2	1/31/2016	9:00:00 PM	8	320
1	2/1/2016	9:00:00 AM	4	80
1	2/15/2016	6:00:00 PM	~	~
1	2/29/2016	9:00:00 PM	~	~
2	2/1/2016	9:00:00 AM	~	~
2	2/15/2016	6:00:00 PM	~	~
2	2/29/2016	9:00:00 PM	~	~

Year	Month Name	Date	Michelle	Paul	Total
2016	January	1/1/2016	8	8	16
		1/15/2016	5	6	11
		1/31/2016	3	3	6
		<b>Total</b>	<b>16</b>	<b>17</b>	<b>33</b>
	February	2/1/2016	11	10	21
		2/15/2016	5	3	8
		2/29/2016	3	3	6
		<b>Total</b>	<b>19</b>	<b>16</b>	<b>35</b>
	March	3/1/2016	5	5	10
		<b>Total</b>	<b>5</b>	<b>5</b>	<b>10</b>
<b>Total</b>			<b>40</b>	<b>38</b>	<b>78</b>
			<b>40</b>	<b>38</b>	<b>78</b>

How can I show data on the 1<sup>st</sup> of March if no row is related to the 1<sup>st</sup> of March?



sqlbi

# Solving with DAX... too complex!



Real Working Hours =

```
SUMX (
    Schedule,
    IF (
        Schedule[TimeStart]
            + Schedule[HoursWorked]
            * ( 1 / 24 )
        <= 1,
        Schedule[HoursWorked],
        ( 1 - Schedule[TimeStart] )
            * 24
    )
)
```

This is a single formula... we splitted it only to format it!

```
+ SUMX (
    VALUES ( 'Date'[Date] ),
    VAR CurrentDay = 'Date'[Date]
    RETURN
        CALCULATE (
            SUMX (
                Schedule,
                IF (
                    Schedule[TimeStart]
                        + Schedule[HoursWorked]
                        * ( 1 / 24 )
                    > 1,
                    Schedule[HoursWorked]
                        - ( 1 - Schedule[TimeStart] )
                        * 24
                )
            ),
            'Date'[Date] = CurrentDay - 1
        )
)
```

# Changing granularity

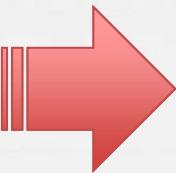
- The date represents when the event started
- If you want to split by date, then you can split an event: if the event crossed the date, you split it in two events:
  - Half of it when it started
  - The remaining part in the next day
- This increases the granularity and makes numbers harder to compute
- For example: How many events occurred in a certain period? Now it requires a distinct count measure

# Solving with ETL steps, still daily granularity



- ETL required (M code)

WorkerId	Date	TimeStart	HoursWorked	Amount
1	1/1/2016	9:00:00 AM	8	160
1	1/15/2016	6:00:00 PM	6	180
1	1/31/2016	9:00:00 PM	9	360
2	1/1/2016	9:00:00 AM	8	160
2	1/15/2016	6:00:00 PM	5	150
2	1/31/2016	9:00:00 PM	8	320
1	2/1/2016	9:00:00 AM	4	80
1	2/15/2016	6:00:00 PM	3	90
1	2/29/2016	9:00:00 PM	8	320
2	2/1/2016	9:00:00 AM	6	120
2	2/15/2016	6:00:00 PM	5	150
2	2/29/2016	9:00:00 PM	8	320



WorkerId	↓	Amount	Date	TimeStart	HoursWorked
1		160	1/1/2016	9:00:00 AM	8
1		180	1/15/2016	6:00:00 PM	6
1		360	1/31/2016	9:00:00 PM	3
1		360	2/1/2016	12:00:00 AM	6
1		80	2/1/2016	9:00:00 AM	4
1		90	2/15/2016	6:00:00 PM	3
1		320	2/29/2016	9:00:00 PM	3
1		320	3/1/2016	12:00:00 AM	5
2		160	1/1/2016	9:00:00 AM	8
2		150	1/15/2016	6:00:00 PM	5
2		320	1/31/2016	9:00:00 PM	3
2		320	2/1/2016	12:00:00 AM	5
2		120	2/1/2016	9:00:00 AM	6
2		150	2/15/2016	6:00:00 PM	5
2		320	2/29/2016	9:00:00 PM	3
2		320	3/1/2016	12:00:00 AM	5

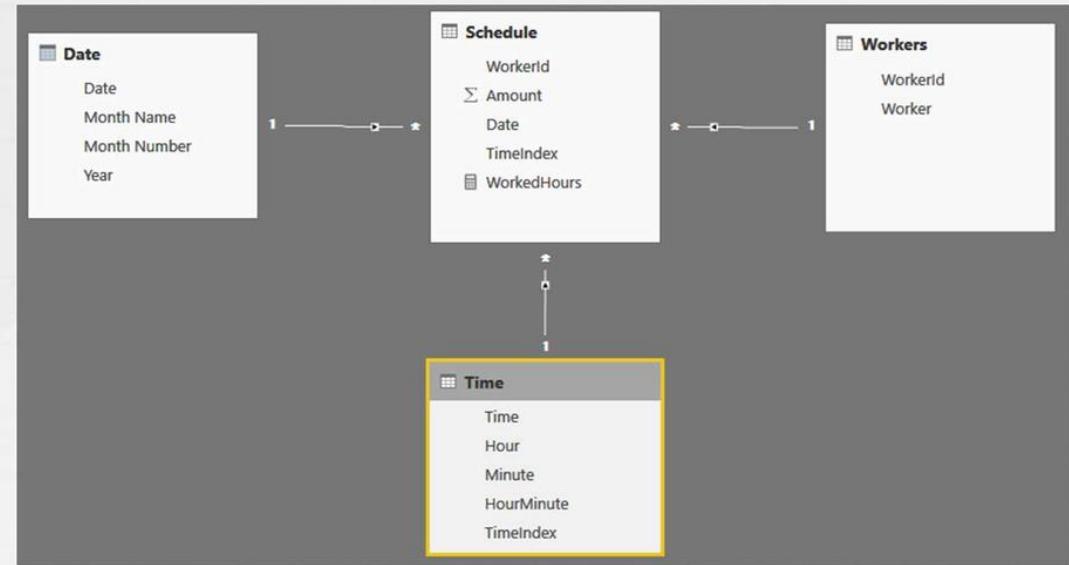
# Split hours AND amount!

- If you split hours, then you need to split the amount too
- The ETL code tends to be much more complex
- Hourly rate might be different in different hours

WorkerId	Amount	Date	TimeStart	HoursWorked
1	160	1/1/2016	9:00:00 AM	8
1	180	1/15/2016	6:00:00 PM	6
1	360	1/31/2016	9:00:00 PM	3
1	360	2/1/2016	12:00:00 AM	6
1	80	2/1/2016	9:00:00 AM	4
1	90	2/15/2016	6:00:00 PM	3
1	320	2/29/2016	9:00:00 PM	3
1	320	3/1/2016	12:00:00 AM	5
2	160	1/1/2016	9:00:00 AM	8
2	150	1/15/2016	6:00:00 PM	5
2	320	1/31/2016	9:00:00 PM	3
2	320	2/1/2016	12:00:00 AM	5
2	120	2/1/2016	9:00:00 AM	6
2	150	2/15/2016	6:00:00 PM	5
2	320	2/29/2016	9:00:00 PM	3
2	320	3/1/2016	12:00:00 AM	5

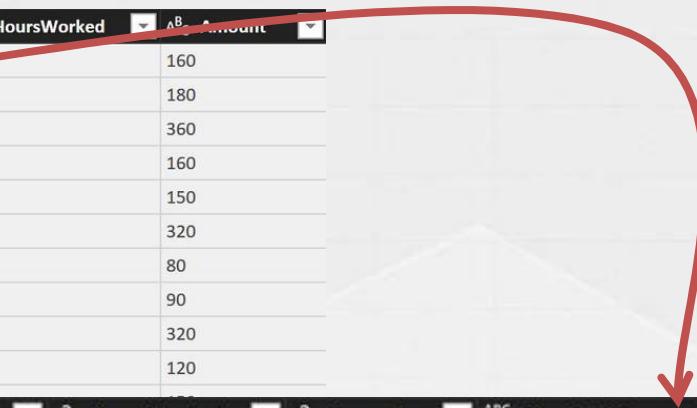
# Increase granularity

- You can increase granularity at the highest value: hours
- In this way, an event becomes atomic and the model becomes a simple star schema



WorkedHours := COUNTROWS ( Schedule )

# Increase granularity: step 1



The image shows two tables illustrating the process of increasing data granularity. A red arrow points from the first table to the second, indicating the transformation.

**Original Data (Top Table):**

	ABC WorkerId	Date	ABC TimeStart	ABC HoursWorked	ABC Amount
1	1	01/01/16	09:00 AM	8	160
2	1	01/15/16	06:00 PM	6	180
3	1	01/31/16	09:00 PM	9	360
4	2	01/01/16	09:00 AM	8	160
5	2	01/15/16	06:00 PM	5	150
6	2	01/31/16	09:00 PM	8	320
7	1	02/01/16	09:00 AM	4	80
8	1	02/15/16	06:00 PM	3	90
9	1	02/29/16	09:00 PM	8	320
10	2	02/01/16	09:00 AM	6	120

**Granularized Data (Bottom Table):**

	ABC WorkerId	Date	ABC TimeStart	ABC HoursWorked	ABC Amount	ABC StartDateTime	ABC EndDateTime	ABC NumOfDays
11	1	1/1/2016	9:00:00 AM	8	160	1/1/2016 9:00:00 AM	1/1/2016 4:59:59 PM	1
12	1	1/15/2016	6:00:00 PM	6	180	1/15/2016 6:00:00 PM	1/15/2016 11:59:59 PM	1
	1	1/31/2016	9:00:00 PM	9	360	1/31/2016 9:00:00 PM	2/1/2016 5:59:59 AM	2
	2	1/1/2016	9:00:00 AM	8	160	1/1/2016 9:00:00 AM	1/1/2016 4:59:59 PM	1
	2	1/15/2016	6:00:00 PM	5	150	1/15/2016 6:00:00 PM	1/15/2016 10:59:59 PM	1
	2	1/31/2016	9:00:00 PM	8	320	1/31/2016 9:00:00 PM	2/1/2016 4:59:59 AM	2
	1	2/1/2016	9:00:00 AM	4	80	2/1/2016 9:00:00 AM	2/1/2016 12:59:59 PM	1
	1	2/15/2016	6:00:00 PM	3	90	2/15/2016 6:00:00 PM	2/15/2016 8:59:59 PM	1
	1	2/29/2016	9:00:00 PM	8	320	2/29/2016 9:00:00 PM	3/1/2016 4:59:59 AM	2
	2	2/1/2016	9:00:00 AM	6	120	2/1/2016 9:00:00 AM	2/1/2016 2:59:59 PM	1
	2	2/15/2016	6:00:00 PM	5	150	2/15/2016 6:00:00 PM	2/15/2016 10:59:59 PM	1
	2	2/29/2016	9:00:00 PM	8	320	2/29/2016 9:00:00 PM	3/1/2016 4:59:59 AM	2

## Increase granularity: step 2

	Date	TimeStart	HoursWorked	Amount	StartDateTime	EndDateTime	NumOfDays
	WorkerId	Amount	Date	HoursWorked	TimeStart		
1	1/1/2016	9:00:00 AM		8	160	1/1/2016 9:00:00 AM	1/1/2016 4:59:59 PM
1	1/15/2016	6:00:00 PM		6	180	1/15/2016 6:00:00 PM	1/15/2016 11:59:59 PM
1	1/31/2016	9:00:00 PM		9	360	1/31/2016 9:00:00 PM	2/1/2016 5:59:59 AM
2	1/1/2016	9:00:00 AM		8	160	1/1/2016 9:00:00 AM	1/1/2016 4:59:59 PM
2	1/15/2016	1	160	1/1/2016	8	9:00:00 AM	1/16 10:59:59 PM
2	1/31/2016	1	160	1/15/2016	6	6:00:00 PM	1/16 11:59:59 PM
1	2/1/2016	1	180	1/15/2016	6	6:00:00 PM	1/16 11:59:59 PM
1	2/15/2016	1	360	1/31/2016	3	9:00:00 PM	1/16 4:59:59 PM
1	2/29/2016	1	360	1/31/2016	6	12:00:00 AM	1/16 4:59:59 AM
2	2/1/2016	2	160	1/1/2016	8	9:00:00 AM	1/16 2:59:59 PM
2	2/15/2016	2	150	1/15/2016	5	6:00:00 PM	1/16 10:59:59 PM
2	2/29/2016	2	320	1/31/2016	3	9:00:00 PM	1/16 4:59:59 AM
		2	320	2/1/2016	5	12:00:00 AM	
		1	80	2/1/2016	4	9:00:00 AM	
		1	90	2/15/2016	3	6:00:00 PM	
		1	320	2/29/2016	3	9:00:00 PM	
		1	320	3/1/2016	5	12:00:00 AM	
		2	120	2/1/2016	6	9:00:00 AM	

## Increase granularity: step 3

1 <sup>2</sup> 3	Amount	ABC	Date	ABC	HoursWorked	ABC	TimeStart				
160		123	1/1/2016	123	8	123	9:00:00 AM				
180		123	1/15/2016	123	6	123	6:00:00 PM				
360		123	1/31/2016	123	3	123	9:00:00 PM				
360		123	2/1/2016	123	6	123	12:00:00 AM				
160		123	1/1/2016	123	8	123	9:00:00 AM				
150		ABC	Date	ABC	TimeStart	\$	HoursWorked	1 <sup>2</sup> 3	Amount	ABC	HourStart
150		123	1/15/2016	123	9:00:00 AM	\$	8	123	160	123	9
320		123	1/31/2016	123	9:00:00 AM	\$	8	123	160	123	10
320		123	2/1/2016	123	9:00:00 AM	\$	8	123	160	123	11
80		123	2/1/2016	123	9:00:00 AM	\$	8	123	160	123	12
90		123	2/15/2016	123	9:00:00 AM	\$	8	123	160	123	13
320		123	2/29/2016	123	9:00:00 AM	\$	8	123	160	123	14
320		123	3/1/2016	123	9:00:00 AM	\$	8	123	160	123	15
120		123	2/1/2016	123	9:00:00 AM	\$	8	123	160	123	16
150		123	2/15/2016	123	9:00:00 AM	\$	8	123	160	123	17
320		123	2/29/2016	123	6:00:00 PM	\$	6	123	180	123	18
320		123	3/1/2016	123	6:00:00 PM	\$	6	123	180	123	19
			1/15/2016		6:00:00 PM	\$	6		180		20
			1/15/2016		6:00:00 PM	\$	6		180		21

sqlbi

# Size estimate for increased granularity

Entity	Value (min)	Value (max)
Employees	10,000	10,000
Hours / Year *	1,300	2,200
Years	5	5
Number of rows	65,000,000	110,000,000

- A company with 10,000 employees generates few tens of millions of rows
- Power BI can handle this size easily

\* Source: <https://stats.oecd.org/Index.aspx?DataSetCode=ANHRS>

## Better analysis at the hour level

- Moving to the hour level makes it possible to perform better analytical reports
- This would be hard with the original model

HourMinute	TimeIndex	Day Period
00:00	0	Night
01:00	60	Night
02:00	120	Night
03:00	180	Night
04:00	240	Night
05:00	300	Night
06:00	360	Night
07:00	420	Morning
08:00	480	Morning
09:00	540	Morning
10:00	600	Morning

Day Period	Michelle	Paul	Total
Morning	8	8	16
Afternoon	6	4	10
Evening	16	15	31
Night	10	11	21
<b>Total</b>	<b>40</b>	<b>38</b>	<b>78</b>

How many events are “active” during a given period of time?

## Analyzing active events

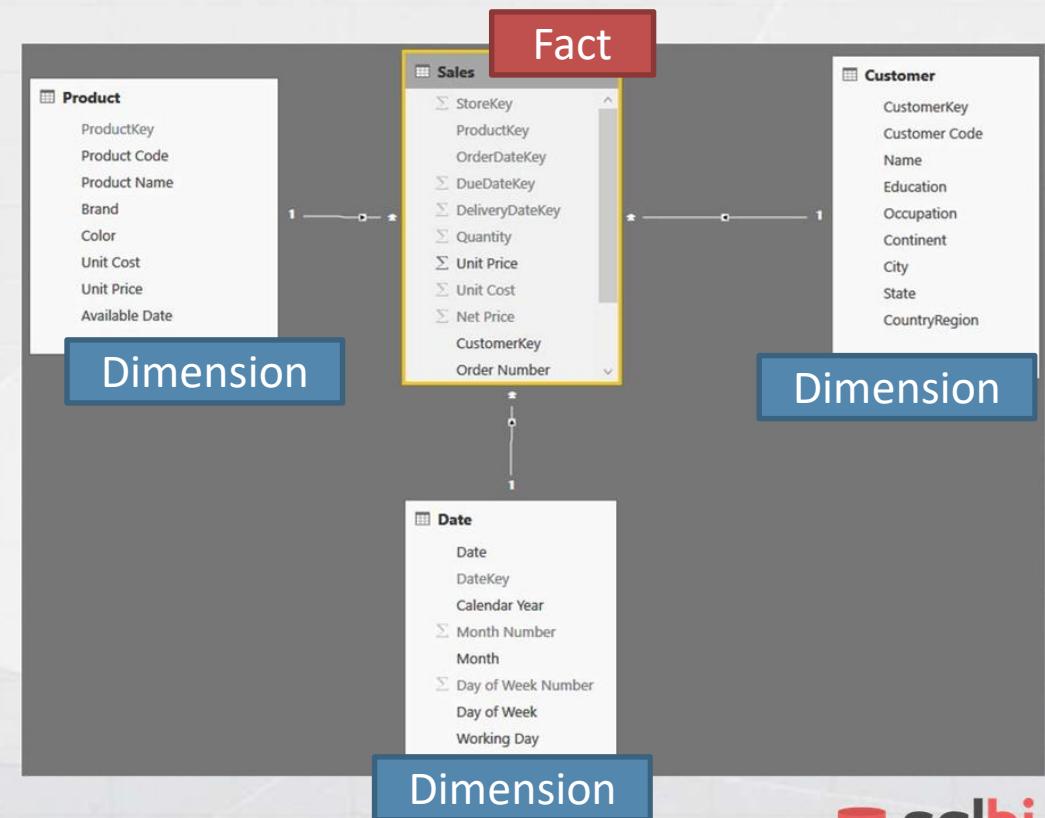


## Active events

- Very common pattern
  - How many orders are being processed?
  - How many active customers do we have?
  - How many claims processed a day?
  - How many passengers are we transporting?
- Whenever an event has a duration
- An event is active between its start and end date/time
- Requirement: count how many events are active in a given span of time

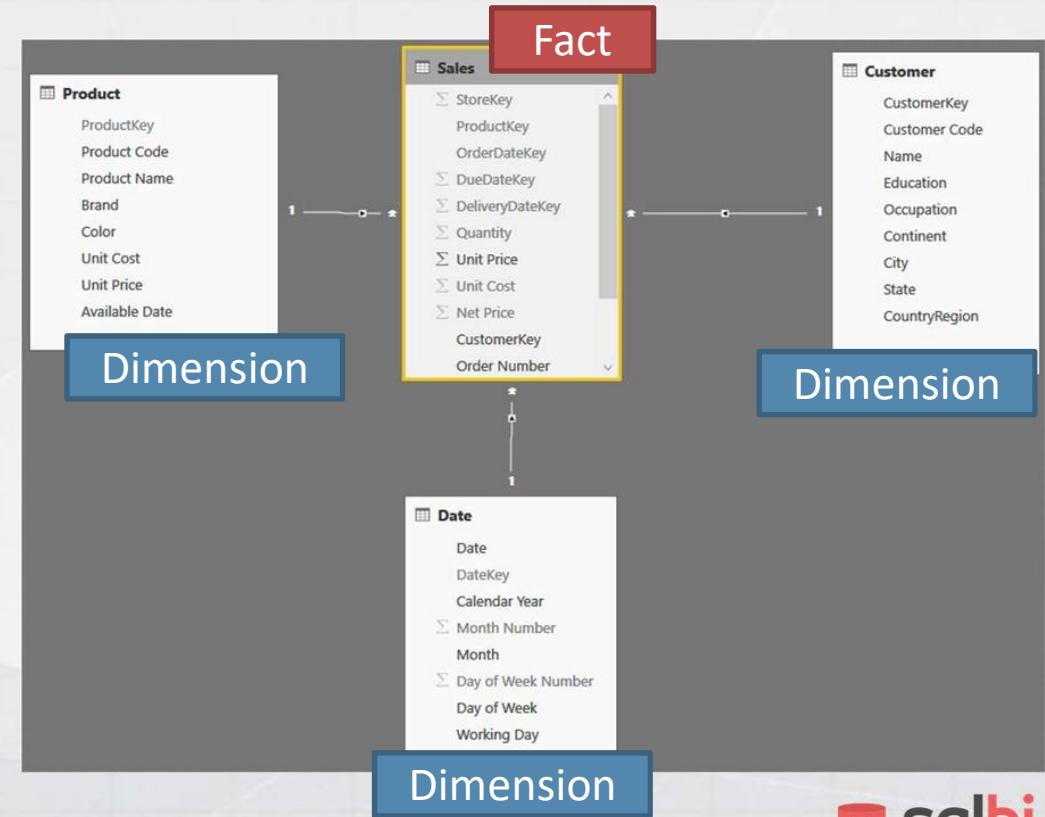
# Open orders: the starting model

- Given OrderDate and DeliveryDate
- How many orders are in the process of preparation (active) in a single day?
- A single order can be active for many days



# Open orders: multiple shipments

- DeliveryDate depends on the item shipped
- What is the actual delivery date if there are many of them?
- It is a business problem, which drives the data model

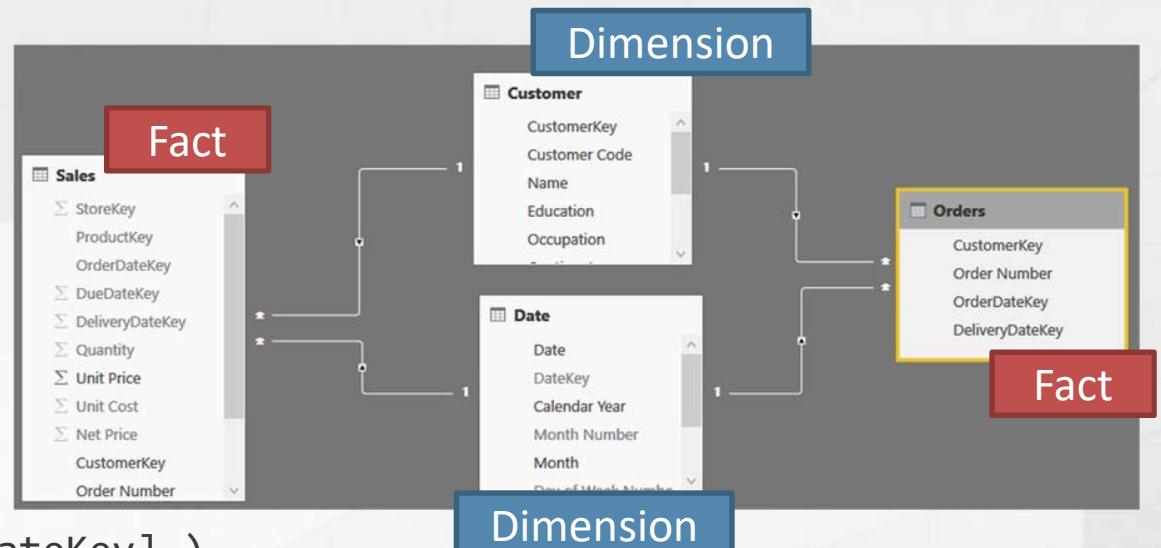


# Orders table

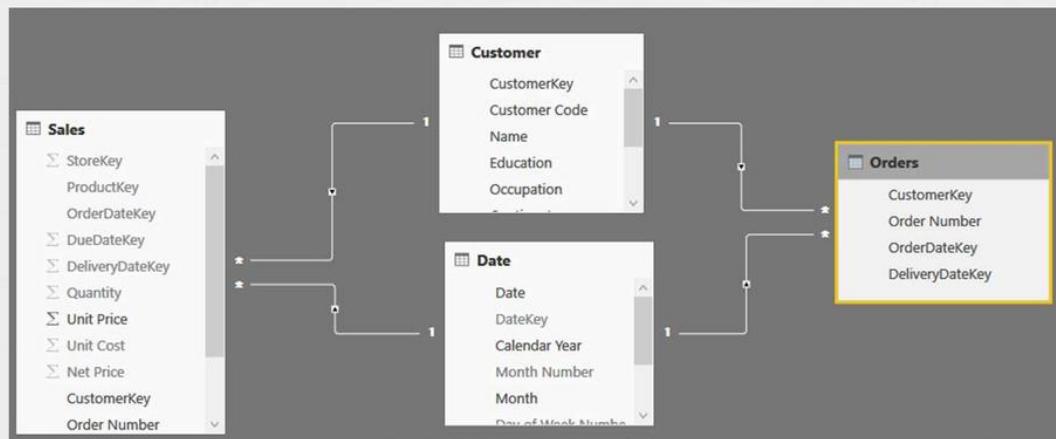


You can summarize the sales table into an Orders table, by taking the MAX of the delivery date, so to consider an order shipped when all of its lines have been delivered.

```
Orders =  
  
SUMMARIZECOLUMNS (  
    Sales[Order Number],  
    Sales[CustomerKey],  
    "OrderDateKey",  
        MIN ( Sales[OrderDateKey] ),  
    "DeliveryDateKey",  
        MAX ( Sales[DeliveryDateKey] )  
)
```



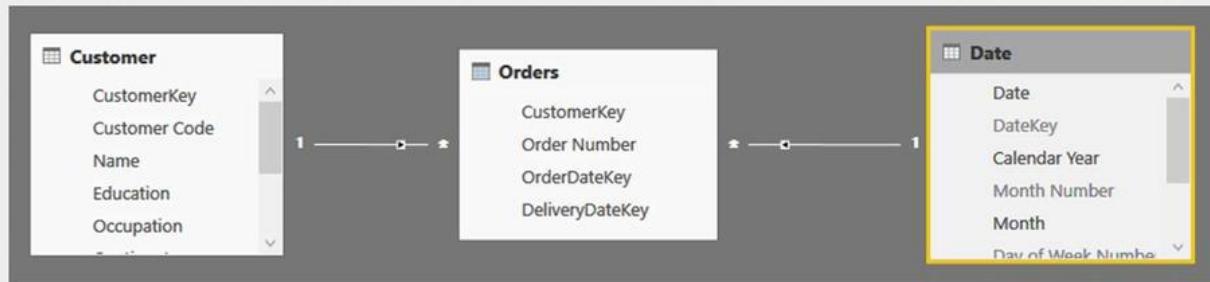
# Simplified view of orders



In this example, we derived the Orders table from Sales. From hereinafter, Orders is the main fact table



# Open orders



How many orders are open at a given date?

Calendar Year	Month	Date	OrdersReceived	OrdersDelivered	OpenOrders
CY 2007	January	2/1/07	10		10
		3/1/07	10		20
		4/1/07	15		35
		5/1/07	2		37
		6/1/07			37
		7/1/07	6		43
		8/1/07		1	42
		9/1/07	9	1	50
		10/1/07	15	4	61
		11/1/07	5	4	62
		12/1/07	9	5	66
		13/1/07	6	8	64
		14/1/07		4	60
		15/1/07	9	8	61



# Open orders with DAX



A simple DAX formula solves the scenario of open orders.

The only non-trivial part is ALL on Date, needed to remove the effect of the relationship.

```
OpenOrders =  
  
VAR DateStart = MIN ( 'Date'[DateKey] )  
VAR DateEnd = MAX ( 'Date'[DateKey] )  
  
RETURN  
  
CALCULATE (  
    COUNTROWS ( Orders ),  
    Orders[OrderDateKey] <= DateStart,  
    Orders[DeliveryDateKey] > DateEnd,  
    ALL ( 'Date' )  
)
```

Calendar Year	Month	Date	OrdersReceived	OrdersDelivered	OpenOrders
CY 2007	January	2/1/07	10		10
		3/1/07	10		20
		4/1/07	15		35
		5/1/07	2		37
		6/1/07			37
		7/1/07	6		43
		8/1/07			42
		9/1/07	9	1	50
		10/1/07	15	4	61
		11/1/07	5	4	62
		12/1/07	9	5	66
		13/1/07	6	8	64
		14/1/07			4
		15/1/07	9	8	61

## It works only at the day level...

Calendar Year	Month	Date	OrdersReceived	OrdersDelivered	OpenOrders
CY 2007	January	2/1/07	10	10	
		3/1/07	10	20	
		4/1/07	15	35	
		5/1/07	2	37	
		6/1/07		37	
		7/1/07	6	43	
		8/1/07		42	1
		9/1/07	9	1	50
		10/1/07	15	4	61
		11/1/07	5	4	62
		12/1/07	9	5	66
		13/1/07	6	8	64
		14/1/07		4	60
		15/1/07	9	8	61

At the month level, start and end are too far, all the orders are executed in-between

Calendar Year	Month	OrdersReceived	OrdersDelivered	OpenOrders
CY 2007	January	194	134	
	February	167	184	
	March	170	176	
	April	203	165	
	May	174	187	
	June	108	139	
	July	132	122	
	August	136	137	
	September	104	114	
	October	93	82	
	November	124	126	
	December	121	130	
	<b>Total</b>	<b>1726</b>	<b>1696</b>	

# Open orders with DAX (LASTDATE)



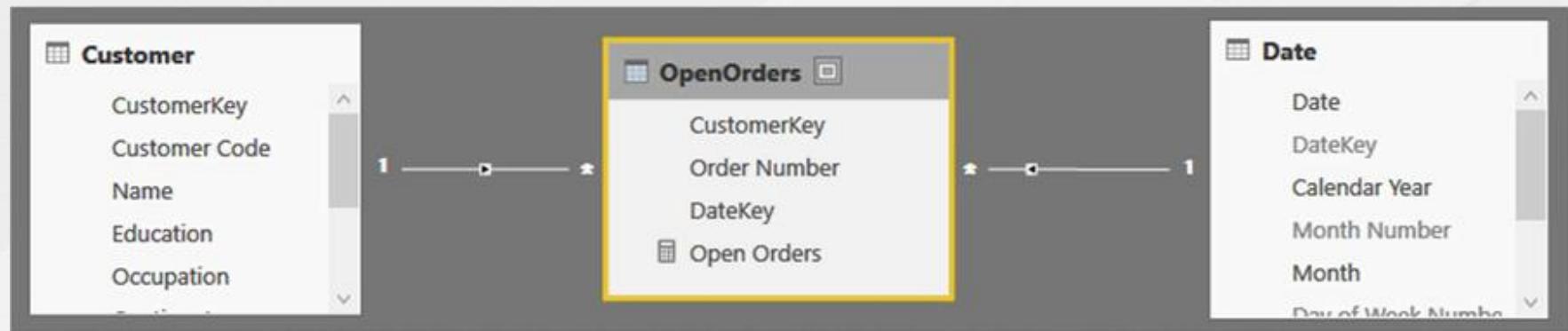
The value at the month level can be the value on the last day of the month, or an average of the individual dates. In this formula, we go for the value at the end of the month (or period)

```
OpenOrders =  
  
CALCULATE (  
    VAR DateStart = MIN ( 'Date'[DateKey] )  
    VAR DateEnd = MAX ( 'Date'[DateKey] )  
  
    RETURN  
        CALCULATE (  
            COUNTROWS ( Orders ),  
            Orders[OrderDateKey] <= DateStart,  
            Orders[DeliveryDateKey] > DateEnd,  
            ALL ( 'Date' )  
        ),  
        LASTDATE ( 'Date'[Date] )  
)
```

Calendar Year	Month	OrdersReceived	OrdersDelivered	OpenOrders
CY 2007	January	194	134	60
	February	167	184	43
	March	170	176	37
	April	203	165	75
	May	174	187	62
	June	108	139	31
	July	132	122	41
	August	136	137	40
	September	104	114	30
	October	93	82	41
	November	124	126	39
	December	121	130	30
	Total	1726	1696	30

# Changing the model

- Change the granularity to the “open order” level
- Now a fact states “on this date, the order was open”
- Much simpler model and code



Open Orders := DISTINCTCOUNT ( OpenOrders[Order Number] )

# All measures are much easier to author



With the updated model, all the calculations are faster and easier to author, at the expenses of an increase in the model size.

Open Orders EOM :=

```
CALCULATE (
    [Open Orders],
    LASTDATE ( 'Date'[Date] )
)
```

Open Orders AVG :=

```
AVERAGEX (
    VALUES ( 'Date'[DateKey] ),
    [Open Orders]
)
```

Calendar Year	Month	Open Orders	Open Orders EOM	Open Orders AVG
CY 2007	January	194	60	56
	February	218	43	61
	March	207	37	55
	April	236	75	60
	May	240	62	57
	June	166	31	42
	July	154	41	42
	August	173	40	44
	September	143	30	38
	October	119	41	30
	November	160	39	42
	December	154	30	39
	<b>Total</b>	<b>1726</b>	<b>30</b>	<b>47</b>

# Open orders is a snapshot table

- By creating the OpenOrders table
  - We built a snapshot
  - Measures whether the order is open at a date
- Complex detection logic is moved at data refresh time, by precomputing the values
  - Increase in the number of rows
  - Slower processing time
  - Faster query performance

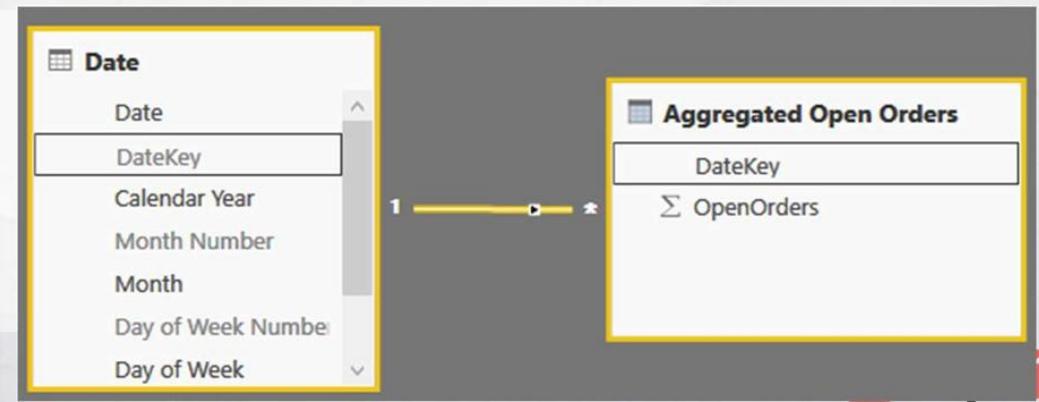
# Pre-aggregating the result



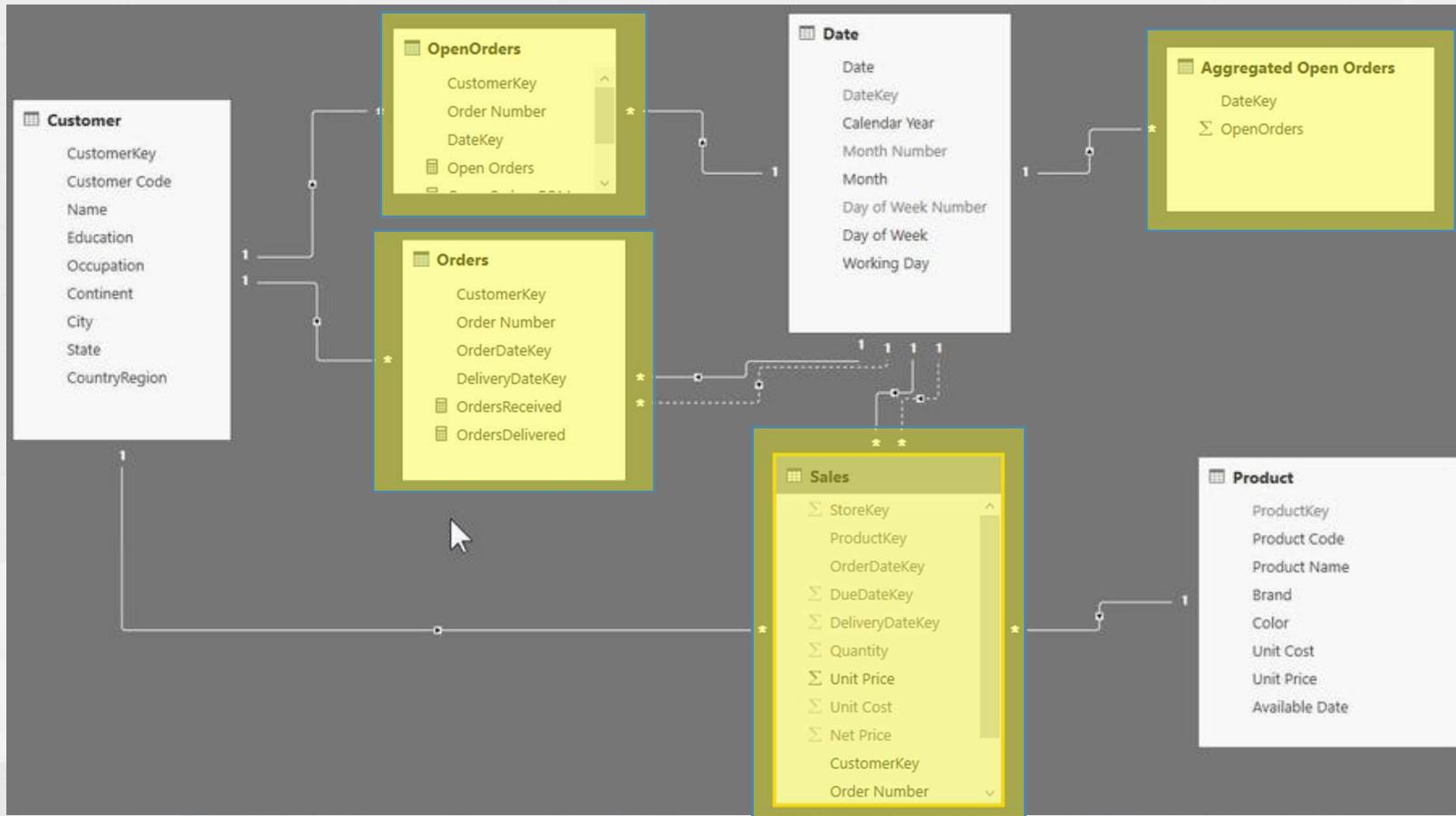
You can move one step further and pre-aggregate the number of open orders per day, reducing the granularity and obtaining a tiny table with all the values ready-to-use.

Aggregated Open Orders =

```
FILTER (
    ADDCOLUMNS (
        DISTINCT ( 'Date'[DateKey] ),
        "OpenOrders", [Open Orders]
    ),
    [Open Orders] > 0
)
```



# All solutions can coexist in the same model



Different events, different durations, different fact tables...

## Events with different durations



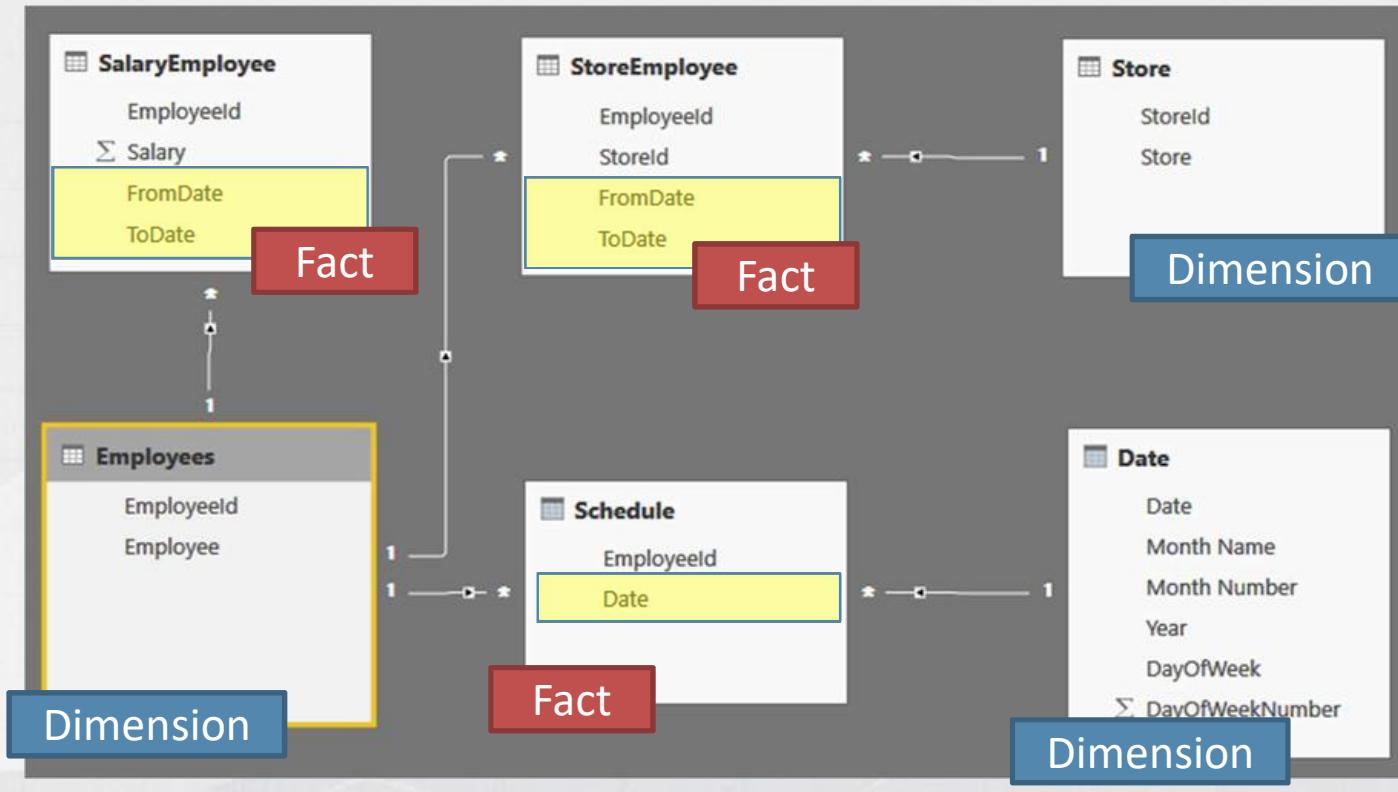
## Different durations

- This scenario happens when you have
  - Multiple fact tables
  - Each fact table contains some sort of event
  - The start date and the duration of different events is unrelated
- Example
  - Fact: hours worked by employees
  - Fact: the store where the employee is working
  - Fact: the salary of the employee, changing over time

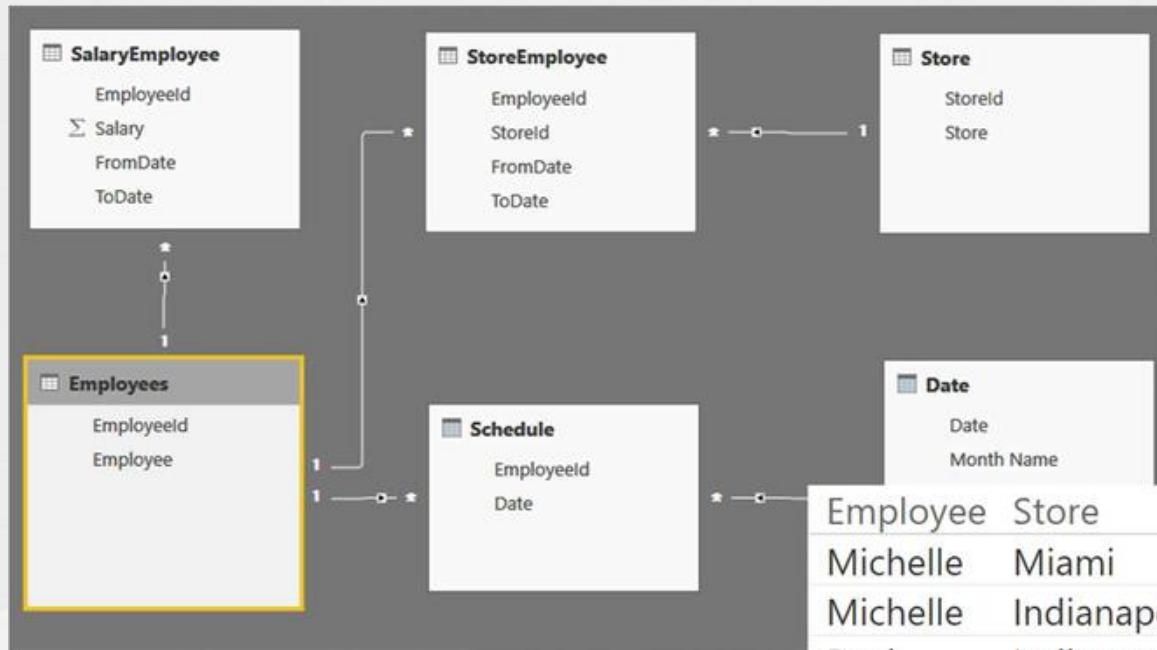
## The scenario

- SalaryEmployee
  - Salary of an employee
  - From date, to date
- StoreEmployee
  - Assignment of an employee to a given store
  - From date, to date
- Schedule
  - Working schedule of an employee
  - Daily granularity

# Employees, with salary and stores



# Different duration



Given a date, it is very hard to find the salary and the schedule active at that time...

Employee	Store	Year	SalaryPaid
Michelle	Miami	2015	\$99,928.00
Michelle	Indianapolis	2016	\$51,684.00
Paul	Indianapolis	2015	\$38,523.00
Paul	Miami	2015	\$42,938.00
Paul	Seattle	2016	\$16,851.00
<b>Total</b>			<b>\$249,924.00</b>



# Daily Salary (DAX)



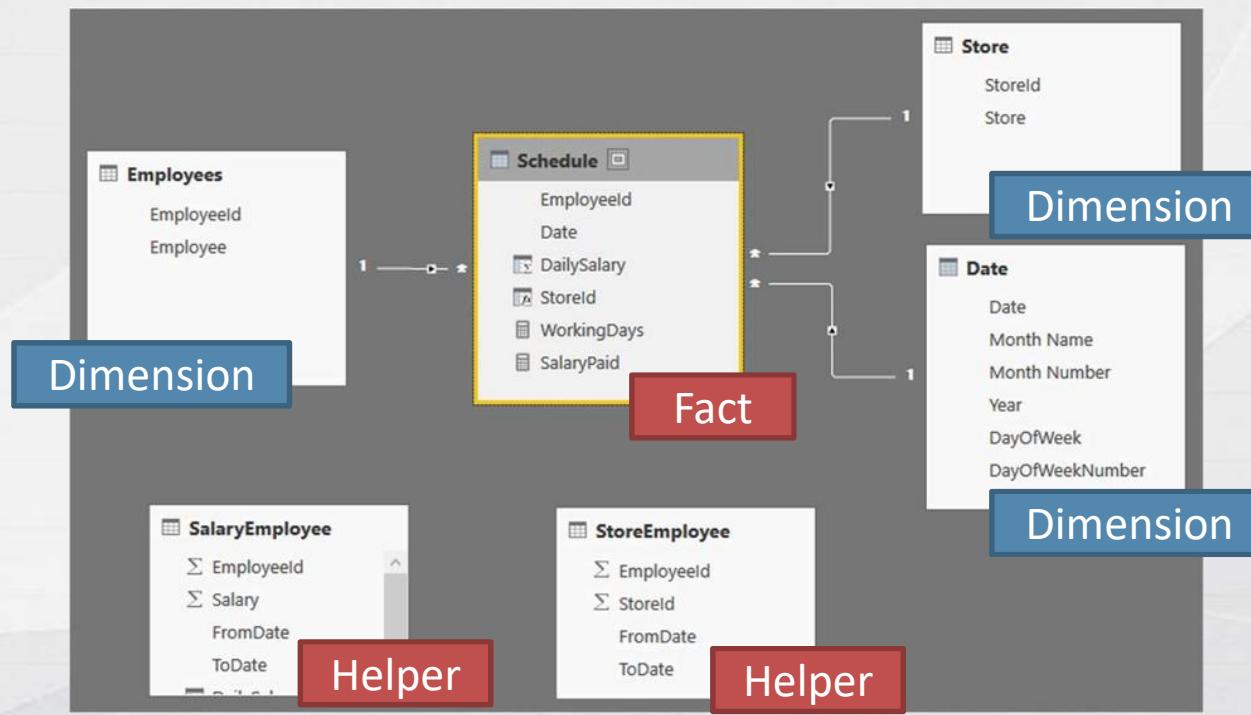
This report is not easy to build: computing the salary paid is not easy because the salary might change every day.

```
SalaryPaid =  
SUMX (  
    FILTER (  
        'Schedule',  
        AND (  
            Schedule[Date] >= MIN ( StoreEmployee[FromDate] ),  
            Schedule[Date] <= MAX ( StoreEmployee[ToDate] )  
        )  
    ),  
    VAR SalaryRows =  
        FILTER (  
            SalaryEmployee,  
            AND (  
                SalaryEmployee[EmployeeId] = Schedule[EmployeeId],  
                AND (  
                    SalaryEmployee[FromDate] <= Schedule[Date],  
                    SalaryEmployee[ToDate] > Schedule[Date]  
                )  
            )  
        )  
    RETURN  
        IF ( COUNTROWS ( SalaryRows ) = 1, MAXX ( SalaryRows, [DailySalary] ) )  
)
```

Employee	Store	Year	SalaryPaid
Michelle	Miami	2015	\$99,928.00
Michelle	Indianapolis	2016	\$51,684.00
Paul	Indianapolis	2015	\$38,523.00
Paul	Miami	2015	\$42,938.00
Paul	Seattle	2016	\$16,851.00
<b>Total</b>			<b>\$249,924.00</b>

# Precompute the values

- Using two calculated columns
- Remove the links with the bridge tables



# DailySalary calculated column



You can use a calculated column for the daily salary, by using a between join with the schedule date and an equijoin with the employee id. Same technique for the Store Id.

```
Schedule[DailySalary] =  
  
VAR CurrentEmployeeId = Schedule[EmployeeId]  
VAR CurrentDate = Schedule[Date]  
  
RETURN  
CALCULATE (  
    VALUES ( SalaryEmployee[DailySalary] ),  
    SalaryEmployee[EmployeeId] = CurrentEmployeeId,  
    SalaryEmployee[FromDate] <= CurrentDate,  
    SalaryEmployee[ToDate] > CurrentDate  
)
```

## Events with different duration

- Whenever possible
  - Move to the least common denominator granularity
  - Precompute the values with calculated columns
  - Reduce the DAX complexity to simple SUM
- The result is a simple star schema
- Complex relationships are gone
- Performance is greatly improved

# Analyzing date and time intervals



Time to practice and work on some models.

In this lab you practice with the shift example, moving from a very simple model to a much more powerful one.

Refer to **lab number 6** on the hands-on manual.

Many-to-many are a powerful tool for the data modeler

## Many-to-many relationships



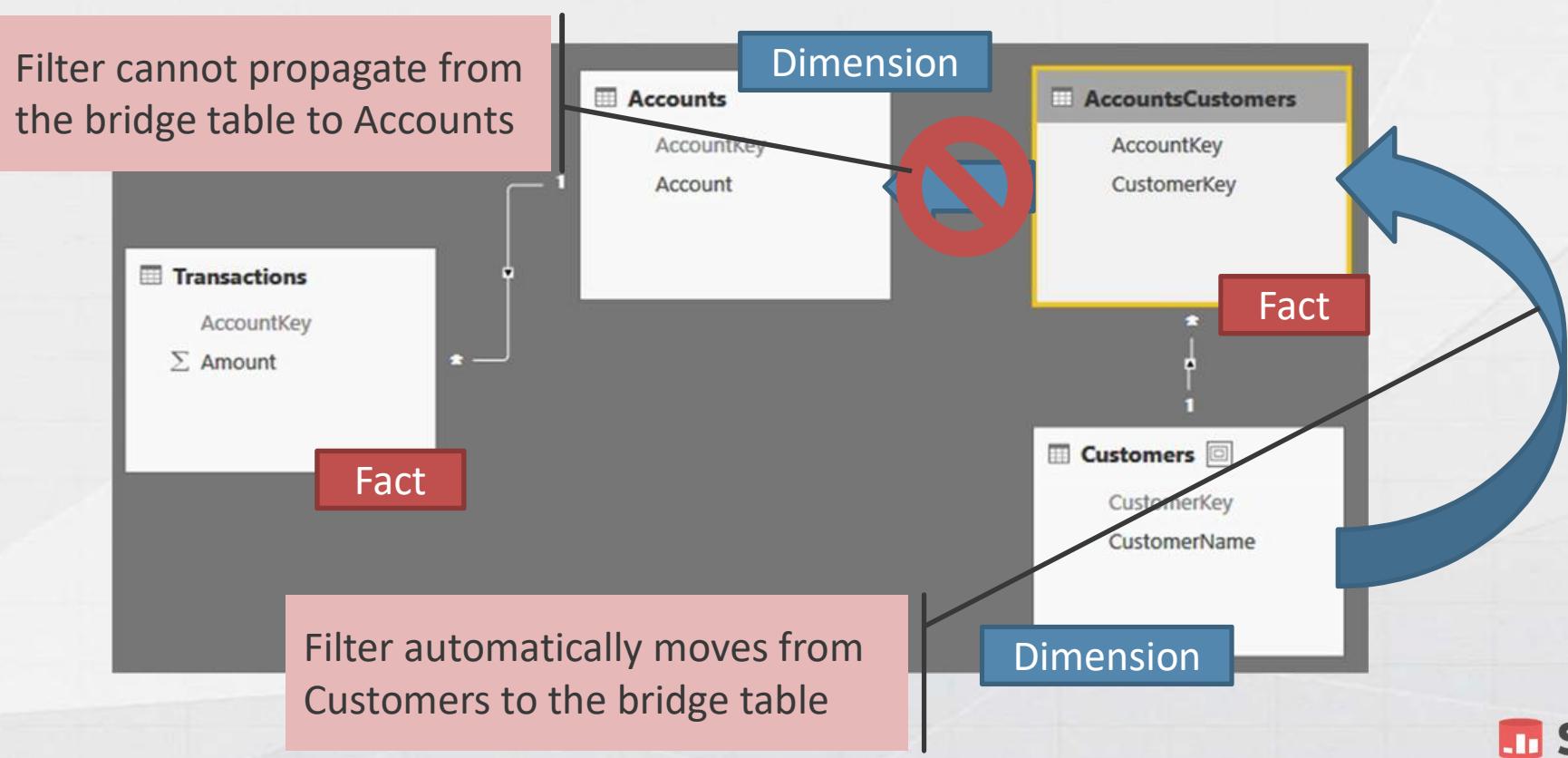
# Many-to-many relationships

- Whenever a relationship is not just one-to-many
- Examples
  - Bank accounts and holders
  - Companies and shareholders
  - Teams, roles and members
  - House and householders
- They are a powerful tool, not an issue at all
- Learning how to use them opens the doors to very powerful scenarios

# Many-to-many challenges

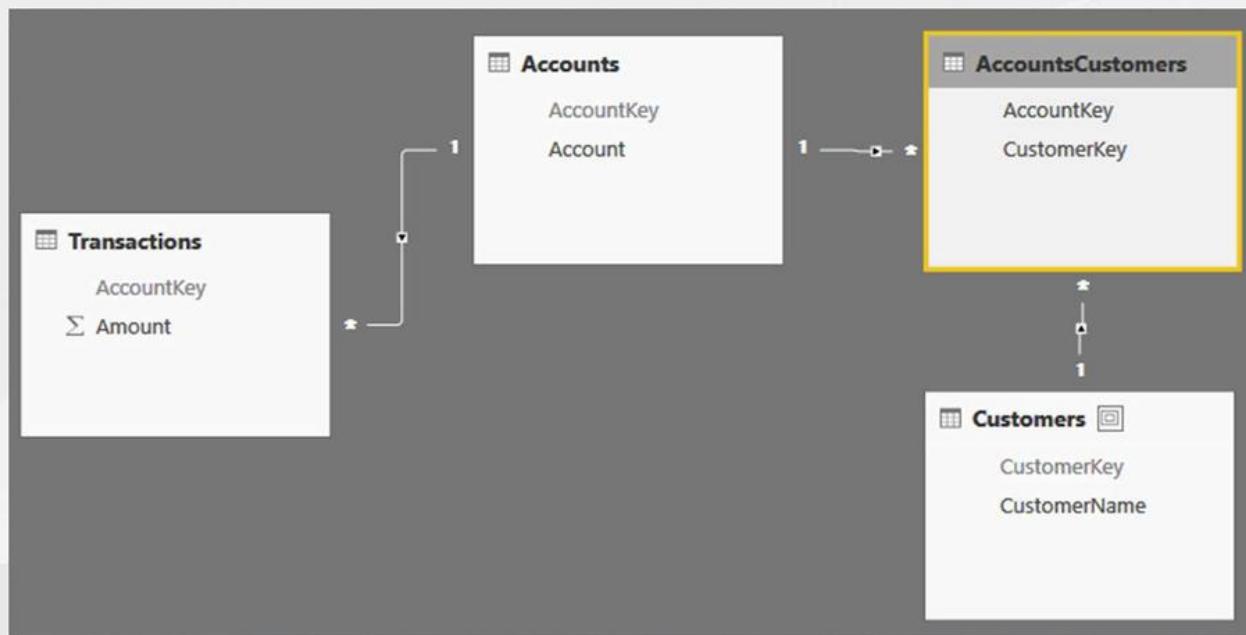
- M2M relationships do not work by default
  - You need to write code or update the model
- M2M relationships generate non-additive calculations
  - Hard to understand at first glance
  - It is in their nature to be non-additive
- Performance might be an issue
  - Not always, pay attention to details

# Bank account example of M2M



# Many-to-many relationships

How can I move the filter from Customers to Accounts?



## Possible solutions to the scenario

- Bidirectional filtering
  - Available in Power BI and SSAS 2016
- Using CROSSFILTER
  - Available in Power BI, SSAS 2016 and Excel 2016
- Expanded table filtering
  - Available in any version of DAX
- Each solution has pros and cons

# Bidirectional Filtering



- Enabled at the relationship level
- The filter context propagates in both ways
- Works with any measure in the model: fewer coding means fewer errors

Edit relationship

Select tables and columns that are related.

AccountsCustomers

AccountKey	CustomerKey
1	1
2	2
3	3

Accounts

AccountKey	Account
1	Mark
2	Paul
3	Robert

Cardinality

Many to one (\*:1)

Cross filter direction

Single

Make this relationship active

Assume referential integrity

OK Cancel

CustomerName	Amount	SumOfAmount
Luke	\$5,000.00	\$800.00
Mark	\$5,000.00	\$2,800.00
Paul	\$5,000.00	\$1,700.00
Robert	\$5,000.00	\$1,700.00
<b>Total</b>	<b>\$5,000.00</b>	<b>\$5,000.00</b>

# Using CROSSFILTER



Changes the direction of a relationship for the duration of a CALCULATE statement  
The pattern need to be repeated for all the measures in the model (the ones using the many-to-many). This is the best practice in complex models to avoid undesired side effects.

```
SumOfAmount CrossFilter =  
  
CALCULATE (  
    SUM ( Transactions[Amount] ),  
    CROSSFILTER (  
        AccountsCustomers[AccountKey],  
        Accounts[AccountKey],  
        BOTH  
    )  
)
```

# Using expanded table filtering



Leveraging table expansion and filter context, you can obtain the same result, without enabling bidirectional filtering on relationships. Useful for older versions of DAX.

```
AmountM2M :=  
CALCULATE (  
    SUM ( Transaction[Amount] ),  
    AccountCustomer  
)  
  
AmountM2M :=  
CALCULATE (  
    SUM ( Transaction[Amount] ),  
    SUMMARIZE ( AccountCustomer, Accounts[AccountKey] )  
)
```

# CROSSFILTER versus expanded tables

- Using CROSSFILTER the filter is propagated only when some filtering is happening
- Using expanded tables, the filter is always active

CustomerName	SumOfAmount	CrossFilter	SumOfAmount	Table Expansion
Luke	\$800.00		\$800.00	
Mark	\$2,800.00		\$2,800.00	
Paul	\$1,700.00		\$1,700.00	
Robert	\$1,700.00		\$1,700.00	
<b>Total</b>	<b>\$10,000.00</b>		<b>\$5,000.00</b>	

AccountKey	Amount
1	\$1,000.00
2	\$1,000.00
3	\$1,000.00
6	\$1,000.00
4	\$1,000.00
5	\$1,000.00
1	(\$200.00)
4	(\$200.00)
3	(\$300.00)
2	(\$300.00)
99	\$5,000.00

# Which technique to choose?

- Bidirectional filtering
  - Set in the model
  - Works with any measure
- Using CROSSFILTER
  - Set in the formula
  - Requires additional coding
- Expanded table filtering
  - Set in the formula
  - Works on any version of DAX
  - Filter is always active, might slow down the model

## Understanding non-additivity

- M2M relationships generate non-additive calculations
- This is not an issue: it is the way M2M works

Account	Luke	Mark	Paul	Robert	Total
Luke	\$800.00				\$800.00
Mark		\$800.00			\$800.00
Mark-Paul		\$1,000.00	\$1,000.00		\$1,000.00
Mark-Robert		\$1,000.00		\$1,000.00	\$1,000.00
Paul			\$700.00		\$700.00
Robert				\$700.00	\$700.00
<b>Total</b>	<b>\$800.00</b>	<b>\$2,800.00</b>	<b>\$1,700.00</b>	<b>\$1,700.00</b>	<b>\$5,000.00</b>

# Non-additivity when coding



You need to be aware of non-additivity when writing code, because the results might be wrong if you do not take into account the additivity behavior of your model

Interest :=

```
[SumOfAmount] * 0.01
```

Interest SUMX :=

```
SUMX (  
    Customers,  
    [SumOfAmount] * 0.01  
)
```

CustomerName	Account	SumOfAmount	Interest	Interest SUMX
Luke	Luke	\$800.00	\$8.00	\$8.00
	<b>Total</b>	<b>\$800.00</b>	<b>\$8.00</b>	<b>\$8.00</b>
	Mark	\$800.00	\$8.00	\$8.00
	Mark-Paul	\$1,000.00	\$10.00	\$10.00
Mark	Mark-Robert	\$1,000.00	\$10.00	\$10.00
	<b>Total</b>	<b>\$2,800.00</b>	<b>\$28.00</b>	<b>\$28.00</b>
	Paul	\$1,000.00	\$10.00	\$10.00
	Paul	\$700.00	\$7.00	\$7.00
Paul	<b>Total</b>	<b>\$1,700.00</b>	<b>\$17.00</b>	<b>\$17.00</b>
	Robert	\$1,000.00	\$10.00	\$10.00
	Robert	\$700.00	\$7.00	\$7.00
	<b>Total</b>	<b>\$1,700.00</b>	<b>\$17.00</b>	<b>\$17.00</b>
<b>Total</b>		<b>\$5,000.00</b>	<b>\$50.00</b>	<b>\$70.00</b>

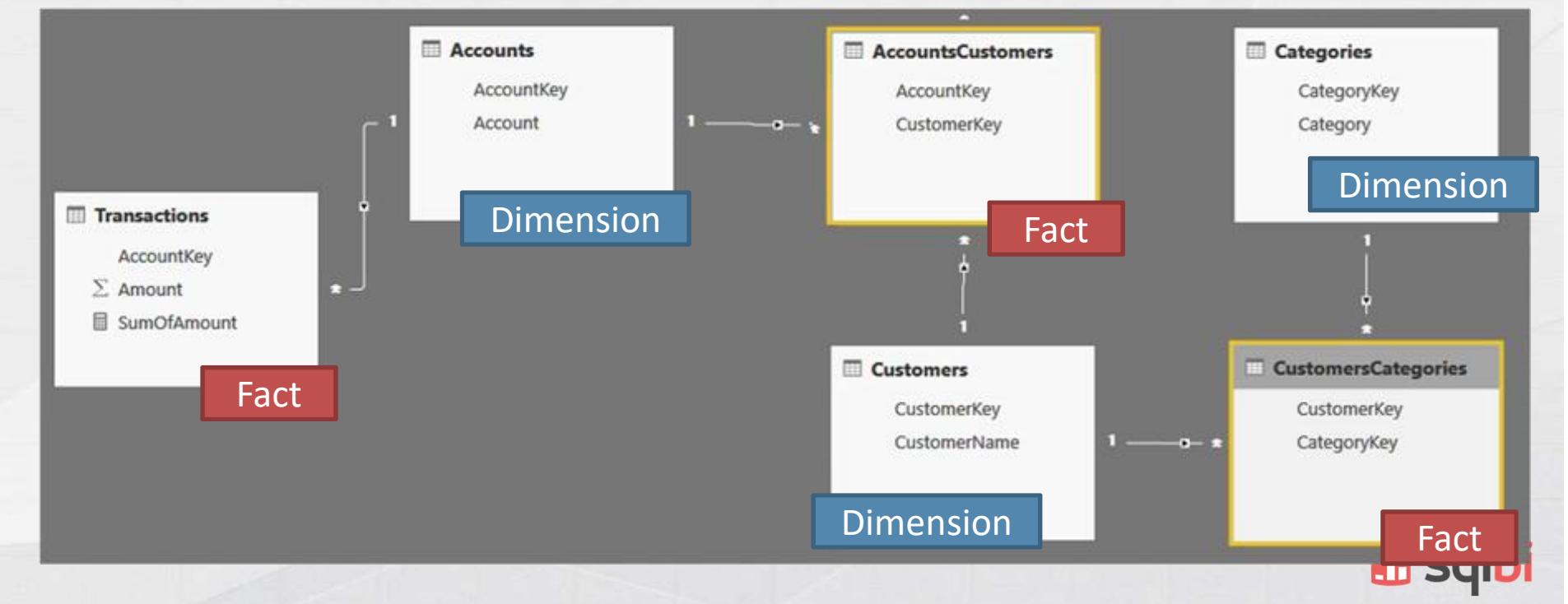
How to handle multiple many-to-many relationships in the same model

## Cascading many-to-many



# Cascading M2M

- Chains of M2M relationships can be harder to manage



# Cascading M2M Using CROSSFILTER



Changes the direction of a relationship for the duration of a CALCULATE statement.

```
SumOfAmount CrossFilter =  
  
CALCULATE (  
    SUM ( Transactions[Amount] ),  
    CROSSFILTER (  
        AccountsCustomers[AccountKey],  
        Accounts[AccountKey],  
        BOTH  
    ),  
    CROSSFILTER (  
        CustomersCategories[CustomerKey],  
        Customers[CustomerKey],  
        BOTH  
    )  
)
```

# Cascading M2M using expanded tables



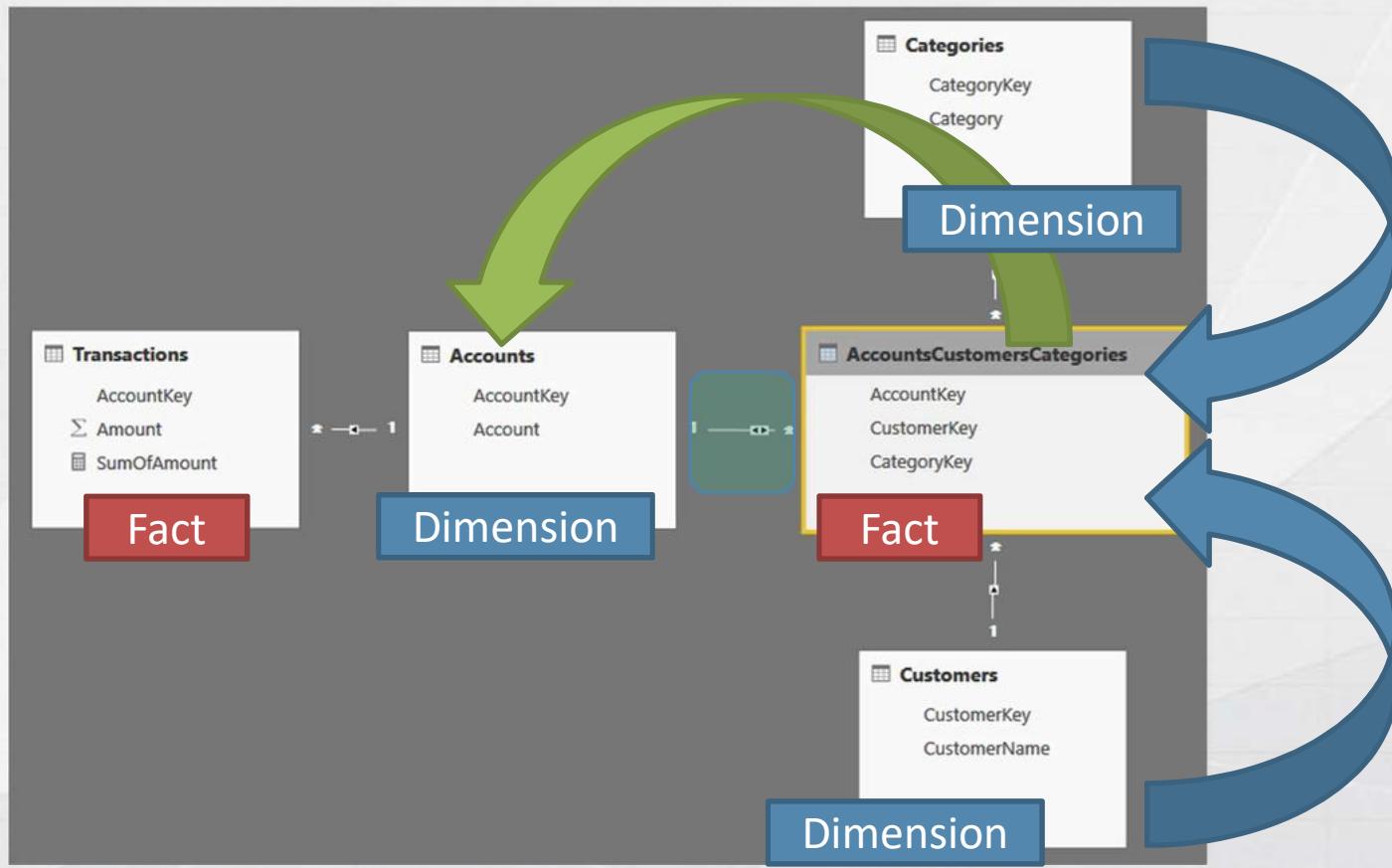
Leveraging table expansion and filter context, you can obtain the same result, without enabling bidirectional filtering on relationships. Pay attention at the order of evaluation of filters.

AmountM2M :=

```
CALCULATE (
    SUM ( Transactions[Amount] ),
    CALCULATETABLE (
        AccountsCustomers,
        CustomersCategories
    )
)
```

The order of the tables is  
very important

# Flattening cascading M2M



# Many-to-many relationships



Time to practice and work on some models.

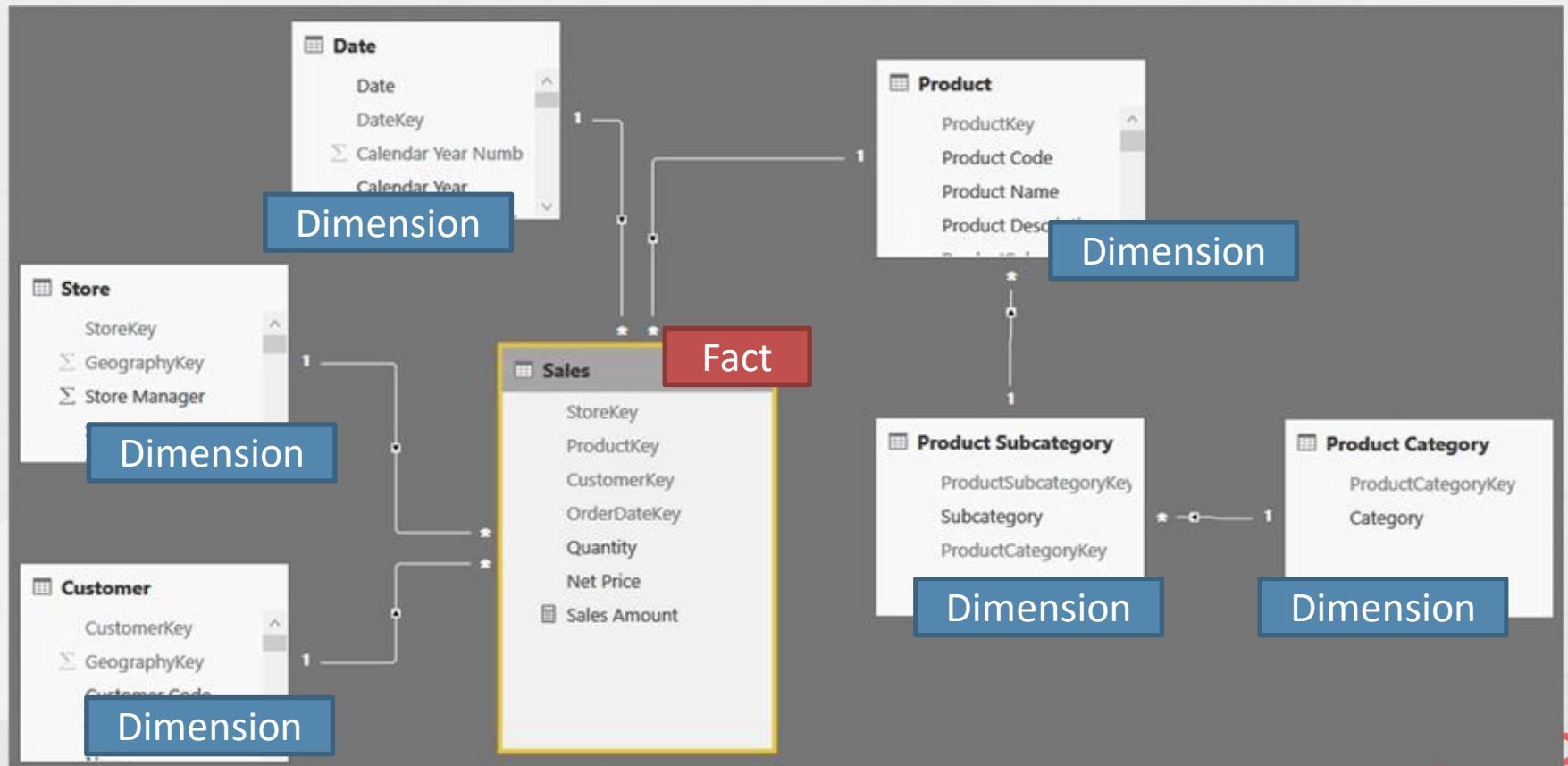
In this lab you practice with regular many-to-many, cascading many-to-many and different kind of calculations over many-to-many.

Refer to **lab number 7** on the hands-on manual.

# Working with different granularities



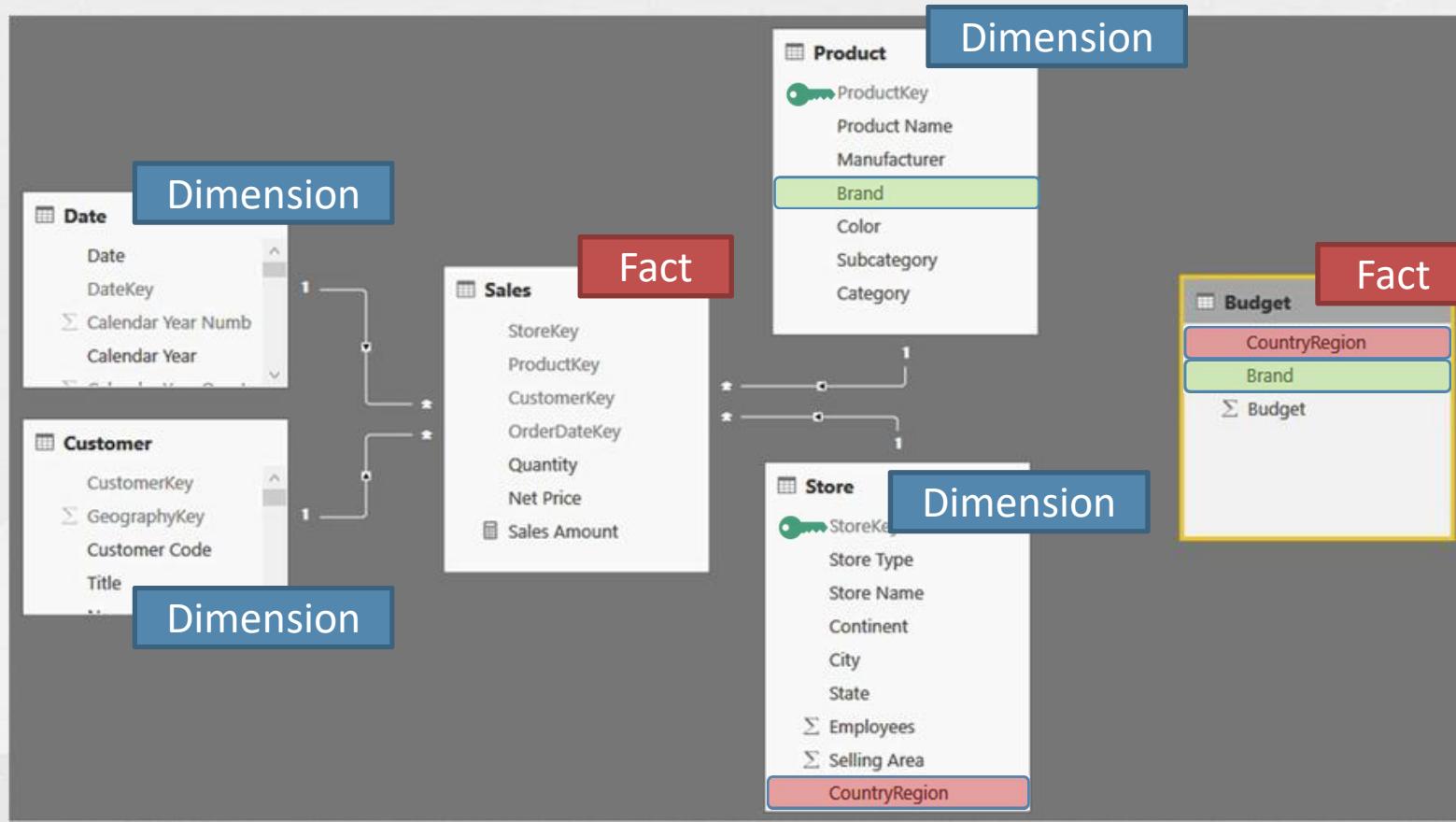
# Dimensions define granularity



# Granularity

- Only dimensions define granularity
  - Duplicated rows can be pre-aggregated
  - Snowflakes do not count for granularity
- Reducing the row count is always a good option
- Pre-aggregate with SUM or the needed aggregation
- Detailed information in the fact table, like the order number, break the rule
  - In such a case, fact table and dimension are mixed

# Analyzing budget data



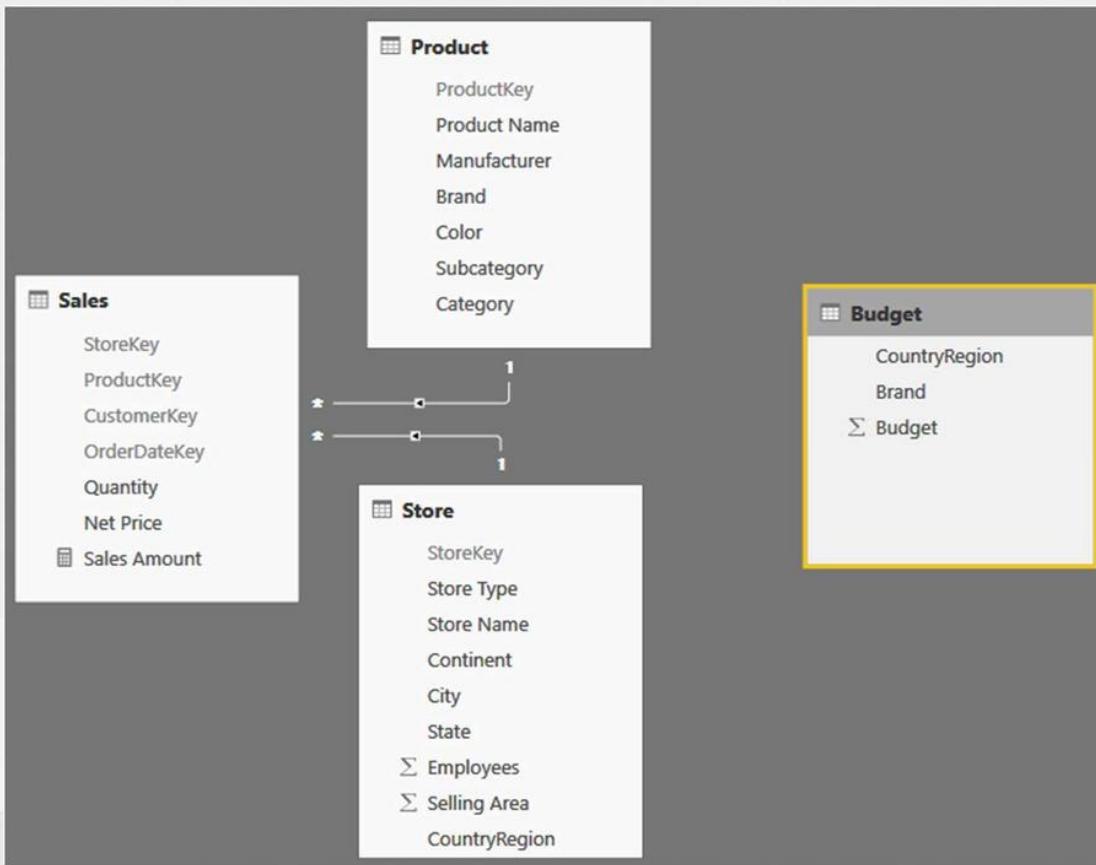
# Missing relationship



- Without the relationship, the model does not work
- The relationship exists, but at a different granularity
- In fact, dimensions have a granularity, too
- Need to build a relationship at a different granularity

Brand	Sales 2009	Budget 2009
A. Datum	\$36,200.51	\$1,141,350.00
Adventure Works	\$32,581.41	\$1,141,350.00
Contoso	\$132,922.31	\$1,141,350.00
Fabrikam	\$110,962.34	\$1,141,350.00
Litware	\$102,200.93	\$1,141,350.00
Northwind Traders	\$2,505.23	\$1,141,350.00
Proseware	\$67,563.50	\$1,141,350.00
Southridge Video	\$35,644.43	\$1,141,350.00
<b>Total</b>	<b>\$629,836.71</b>	<b>\$1,141,350.00</b>

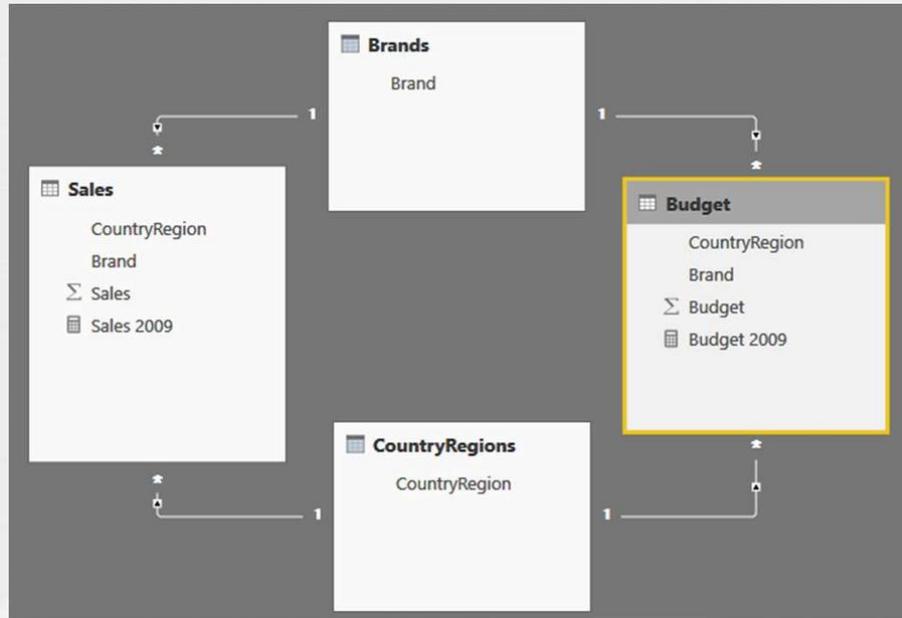
# Different granularities



Budget cannot be related to neither Product nor to Store...



# Reduce granularity on all the tables



Brand	Sales 2009	Budget 2009
A. Datum	\$33,315.51	\$48,500.00
Adventure Works	\$31,766.41	\$67,100.00
Contoso	\$132,785.98	\$239,500.00
Fabrikam	\$99,675.86	\$169,500.00
Litware	\$106,393.85	\$143,000.00
Northwind Traders	\$2,505.23	\$64,500.00
Proseware	\$65,963.93	\$115,500.00
Southridge Video	\$36,392.89	\$61,500.00
<b>Total</b>	<b>\$617,297.77</b>	<b>\$1,141,350.00</b>

# Reducing granularity



- Reduction in the analytical power
- Need a dedicated model
  - Cut the existing dimension granularity
  - ETL required to pre-aggregate the fact tables
- Not the best practice, after all
- A better option is to keep existing granularity
  - Modify the model or use DAX code to set the relationship at the correct granularity

# Using DAX to move the filters



You can use DAX to move the filter from the Product[Brand] column to the Budget[Brand] one, repeating the same operation for the CountryRegion pair of columns.

Budget 2009 :=

```
CALCULATE (
    SUM ( Budget[Budget] ),
    TREATAS (
        VALUES ('Product'[Brand]),
        Budget[Brand]
    ),
    TREATAS (
        VALUES (Store[CountryRegion]),
        Budget[CountryRegion]
    )
)
```

# Using INTERSECT



You can use INTERSECT instead of TREATAS on older versions of SSAS Tabular 2016 and Excel 2016 that do not have TREATAS (released in February 2017).

Budget 2009 :=

```
CALCULATE (
    SUM ( Budget[Budget] ),
    INTERSECT (
        VALUES ( Budget[Brand] ),
        VALUES ( 'Product'[Brand] )
    ),
    INTERSECT (
        VALUES ( Budget[CountryRegion] ),
        VALUES ( Store[CountryRegion] )
    )
)
```

# Using DAX to move the filters (before 2015)

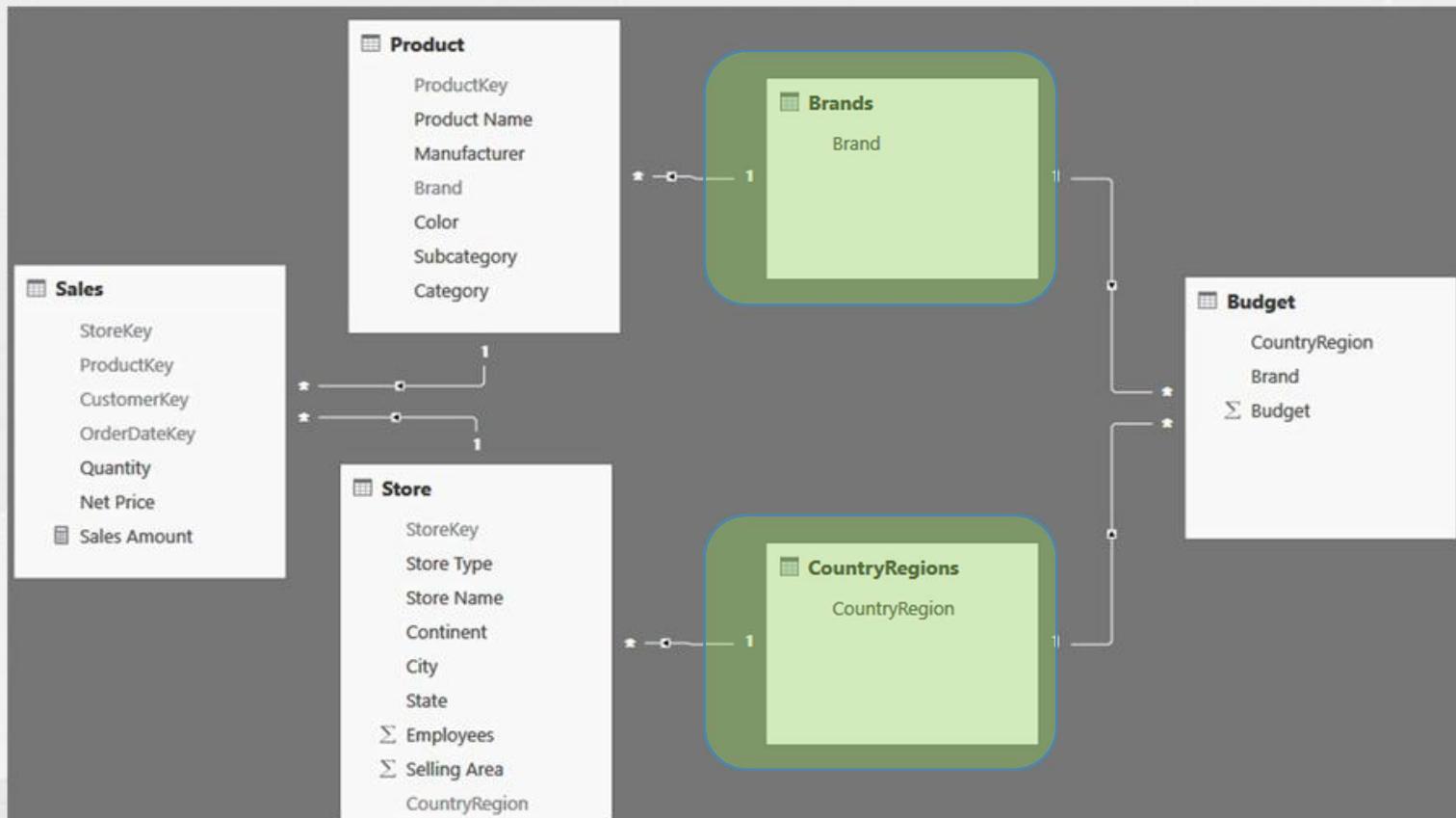
In Excel 2013 and SSAS 2014, TREATAS and INTERSECT functions are not available. You can obtain the same result by using the CONTAINS function along with an iteration on the Budget columns.

```
Budget 2009 Contains =  
CALCULATE (  
    SUM ( Budget[Budget] ),  
    FILTER (  
        VALUES ( Budget[Brand] ),  
        CONTAINS ( VALUES ( 'Product'[Brand] ), 'Product'[Brand], Budget[Brand] )  
    ),  
    FILTER (  
        VALUES ( Budget[CountryRegion] ),  
        CONTAINS (  
            VALUES ( Store[CountryRegion] ),  
            Store[CountryRegion], Budget[CountryRegion]  
        )  
    )  
)
```

# Using DAX to move the filters

- Flexibility
  - You change the filter context in a very dynamic way
  - Full control over the functions used
- Complexity
  - Every measure needs to implement the pattern
  - Error-prone
- Speed
  - Using DAX to move a filter is sub-optimal
  - Leverages the slower part of the DAX engine (FE)

# Filtering through relationships



# Using calculated tables

In Power BI and SSAS 2016 you can build the intermediate tables using DAX calculated tables

```
Brands =
```

```
DISTINCT (
    UNION (
        DISTINCT ( Product[Brand] ),
        DISTINCT ( Budget[Brand] )
    )
)
```

```
CountryRegions =
```

```
DISTINCT (
    UNION (
        ALLNOBLANKROW ( Store[CountryRegion] ),
        ALLNOBLANKROW ( Budget[CountryRegion] )
    )
)
```

Use ALLNOBLANKROW or DISTINCT to avoid circular dependency issues.

Using ALL or VALUES there would be a dependency on the existence of the additional blank row.

# Use the correct column to slice

Using Customer[CountryRegion]

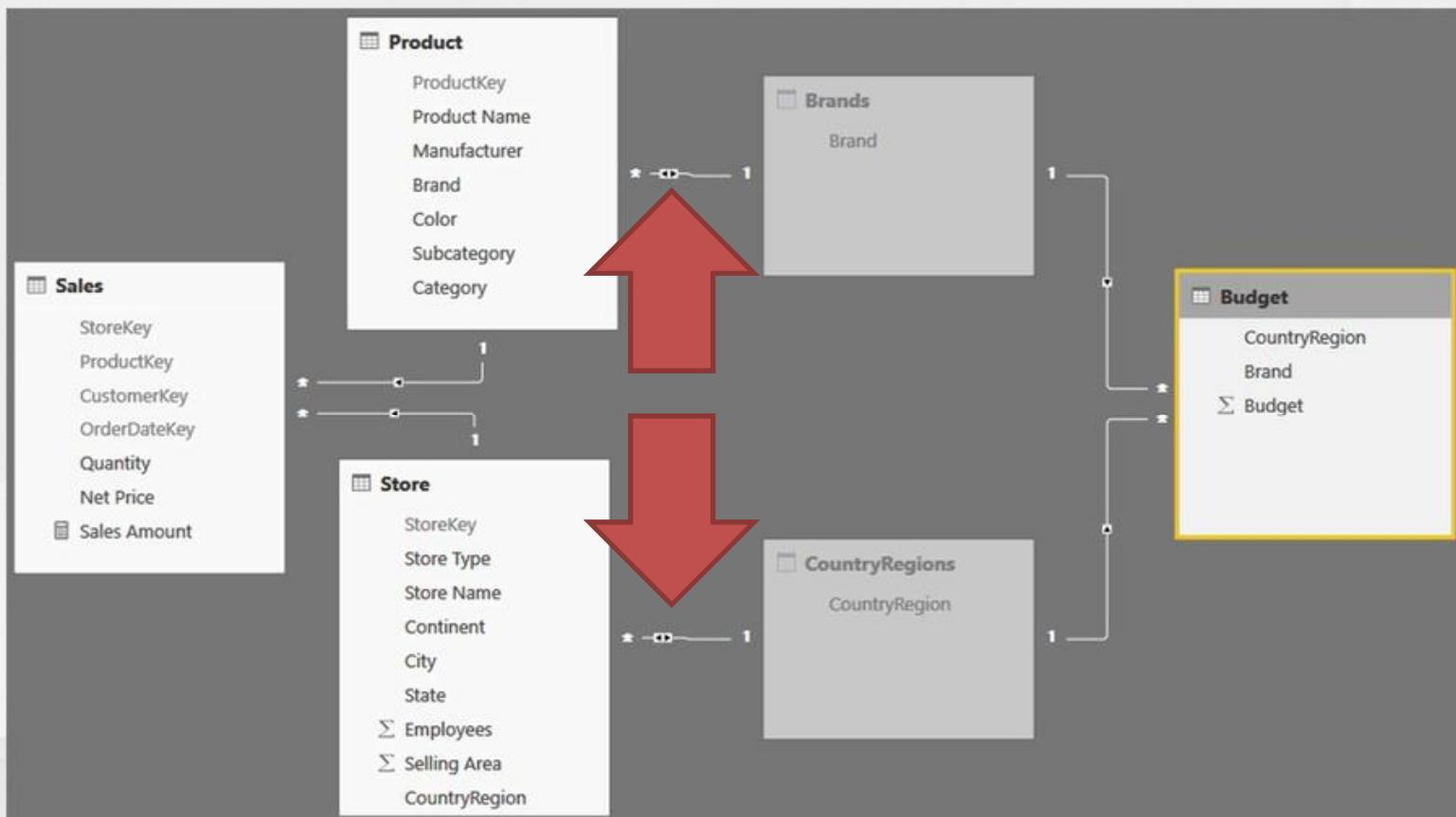
CountryRegion	Sales Amount	Budget
Armenia		1,141,350.00
Australia	\$452,182.63	1,141,350.00
Bhutan		1,141,350.00
Canada	\$45,754.50	1,141,350.00
China	\$36,589.42	1,141,350.00
France	\$42,460.06	1,141,350.00
Germany	\$82,188.98	1,141,350.00
Greece	\$10,288.68	1,141,350.00
India	\$24,071.79	1,141,350.00
Iran		1,141,350.00

Using CountryRegions[CountryRegion]

CountryRegion	Sales Amount	Budget
China	\$598,782.81	418,500.00
Germany	\$469,523.88	327,500.00
United States	\$684,065.08	395,350.00
<b>Total</b>	<b>\$1,752,371.77 1,141,350.00</b>	

Having one table for each attribute might result in a complex data model. Hide them and use filter direction in relationships to improve usability.

# Leveraging bidirectional filtering



## Leveraging relationships

- With DAX code the model is easy but slower
- Using relationships makes the model faster
- Using bidirectional filtering improves usability
  - Available in Power BI and SSAS 2016 (no DAX required)
  - Not available in Excel 2016 (available with CROSSFILTER)
- Still, we have granularity issues
  - When users slice data at a different granularity
  - Calculations follow the many-to-many pattern, leading to results complex to understand

## Granularity is still an issue

- Browsing at a different granularity produces wrong results
- Filtering still happens at the brand level, even if you are browsing by Color
- This is a very dangerous model, it shows incorrect values without any warning

Color	Sales Amount	Budget 2009
Azure	\$6,362.32	\$48,500.00
Black	\$337,734.90	\$1,141,350.00
Blue	\$92,449.76	\$1,062,600.00
Brown	\$115,653.60	\$806,100.00
Gold	\$17,447.49	\$625,750.00
Green	\$64,720.29	\$934,000.00
Grey	\$246,243.30	\$1,076,850.00
Orange	\$33,211.29	\$726,000.00
Pink	\$27,358.17	\$958,750.00
Purple	\$60.60	\$600,500.00
Red	\$48,697.27	\$1,014,100.00
Silver	\$285,768.49	\$1,141,350.00
Silver Grey	\$18,238.76	\$457,500.00
Transparent	\$178.75	\$239,500.00
White	\$453,935.43	\$1,092,850.00
Yellow	\$4,311.34	\$662,000.00
<b>Total</b>	<b>\$1,752,371.77</b>	<b>\$1,141,350.00</b>



# Different granularities

Color	Sales Amount	Budget 2009
Azure	\$6,362.32	\$48,500.00
Black	\$337,734.90	\$1,141,350.00
Blue	\$92,449.76	\$1,062,600.00
Brown	\$115,653.60	\$806,100.00
Gold	\$17,447.49	\$625,750.00
Green	\$64,720.29	\$934,000.00
Grey	\$246,243.30	\$1,076,850.00
Orange	\$33,211.29	\$726,000.00
Pink	\$27,358.17	\$958,750.00
Purple	\$60.60	\$600,500.00
Red	\$48,697.27	\$1,014,100.00
Silver	\$285,768.49	\$1,141,350.00
Silver Grey	\$18,238.76	\$457,500.00
Transparent	\$178.75	\$239,500.00
White	\$453,935.43	\$1,092,850.00
Yellow	\$4,311.34	\$662,000.00
<b>Total</b>	<b>\$1,752,371.77</b>	<b>\$1,141,350.00</b>

How can I hide rows  
when the granularity  
is not the correct one?



# Checking granularity in the report



You can use DAX to understand at which granularity the user is browsing your data, detecting when the granularity of the current context is different from the one of the budget table.

```
ProductsAtBudgetGranularity :=
```

```
    CALCULATE (
        COUNTROWS ( Product ),
        ALL ( Product ),
        VALUES ( Product[Brand] )
    )
```

```
ProductsAtSalesGranularity :=
```

```
    COUNTROWS ( Product )
```

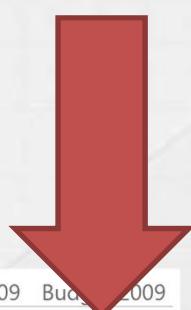
Brand	Color	ProductsAtBudgetGranularity	ProductsAtSalesGranularity
A. Datum	Azure	132	14
	Black	132	18
	Blue	132	4
	Gold	132	4
	Green	132	14
	Grey	132	18
	Orange	132	18
	Pink	132	18
	Silver	132	18
	Silver Grey	132	6
Adventure Works	<b>Total</b>	<b>132</b>	<b>132</b>
	Black	192	54
	Blue	192	12
	Brown	192	15
	Grey	192	14
	Red	192	6
	Silver	192	39
	White	192	52
	<b>Total</b>	<b>192</b>	<b>192</b>

# Hiding the wrong granularity



Using a simple IF statement, you can clear out values that cannot be safely computed.

```
Budget 2009 :=  
IF (  
    AND (  
        [ProductsAtBudgetGranularity] = [ProductsAtSalesGranularity],  
        [StoresAtBudgetGranularity] = [StoresAtSalesGranularity]  
    ),  
    SUM ( Budget[Budget] )  
)
```



	Brand	Color	Sales 2009	Budget 2009
Adventure Works	A. Datum	Azure	\$2,300.20	
		Black	\$9,084.43	
		Gold	\$1,014.00	
		Green	\$5,138.80	
		Grey	\$3,103.05	
		Orange	\$5,306.96	
		Pink	\$1,378.92	
		Silver	\$1,726.35	
		Silver Grey	\$7,147.80	
		<b>Total</b>	<b>\$36,200.51</b>	<b>\$48,500.00</b>
Adventure Works		Black	\$15,303.77	
		Blue	\$520.47	
		Silver	\$8,622.21	
		White	\$8,134.95	
		<b>Total</b>	<b>\$32,581.41</b>	<b>\$67,100.00</b>

bi

## Hiding or reallocating?

- We hid the wrong values using DAX
- How to display correct values?
- A viable option is to define a reallocation factor
  - Using sales in previous year
  - Compute the percentage of the selection vs the total
  - Use the percentage to dynamically reallocate budget
- More complex code, but very dynamic and powerful

# Allocating the budget

Brand	Color	Sales 2008	AllocationFactor	Budget 2009	Allocated Budget
A. Datum	Azure	\$4,062.12	9.33 %		\$4,526.04
	Black	\$681.90	1.57 %		\$759.78
	Blue	\$1,587.60	3.65 %		\$1,768.91
	Green	\$3,450.30	7.93 %		\$3,844.35
	Grey	\$1,800.00	4.14 %		\$2,005.57
	Orange	\$6,424.10	14.76 %		\$7,157.77
	Pink	\$12,566.60	28.87 %		\$14,001.78
	Silver	\$9,557.72	21.96 %		\$10,649.28
	Silver Grey	\$3,398.40	7.81 %		\$3,786.52
	<b>Total</b>	<b>\$43,528.75</b>	<b>100.00 %</b>	<b>\$48,500.00</b>	<b>\$48,500.00</b>
Adventure Works	Black	\$27,481.25	29.36 %		\$19,703.50
	Blue	\$8,603.64	9.19 %		\$6,168.64
	Brown	\$7,549.63	8.07 %		\$5,412.93
	Grey	\$4,644.90	4.96 %		\$3,330.30
	Red	\$5,592.00	5.98 %		\$4,009.35
	Silver	\$18,013.27	19.25 %		\$12,915.15
	White	\$21,702.32	23.19 %		\$15,560.13
	<b>Total</b>	<b>\$93,587.00</b>	<b>100.00 %</b>	<b>\$67,100.00</b>	<b>\$67,100.00</b>

# Allocation factor: the formula



You compute the sales at the correct granularity and then divide the budget by the ratio between sales and sales at budget granularity, using a technique similar to the one used to hide values.

```
Sales2008AtBudgetGranularity :=  
CALCULATE (  
    [Sales 2008],  
    ALL ( Store ),  
    VALUES ( Store[CountryRegion] ),  
    ALL ( Product ),  
    VALUES ( Product[Brand] )  
)
```

```
AllocationFactor :=  
DIVIDE (  
    [Sales 2008],  
    [Sales2008AtBudgetGranularity]  
)
```

```
Allocated Budget := SUM ( Budget[Budget] ) * [AllocationFactor]
```

Beware: this is NOT the  
same as ALLEXCEPT

# Allocation factor: the final formula

The final formula is more complex because the allocation is valid at Brand / CountryRegion granularity.

Allocated Budget :=

```
SUMX (
    KEEPFILTERS (
        CROSSJOIN (
            VALUES ( 'Product'[Brand] ),
            VALUES ( Store[CountryRegion] )
        )
    ),
    [Allocation Factor] * CALCULATE ( SUM ( Budget[Budget] ) )
)
```

# Different granularity



Time to practice and work on some models.

In this lab you work with the budget model, with fact tables at different granularity.

Refer to **lab number 8** on the hands-on manual.

# Segmentation data models



# Segmentation models

- Dividing entities in segments
  - Products by list price
  - Customers by category
  - ABC analysis
- These models are based on non-trivial relationships
- Most of the times, the relationship cannot be created in the model
- Use DAX to build or mimic the relationship

# Static segmentation

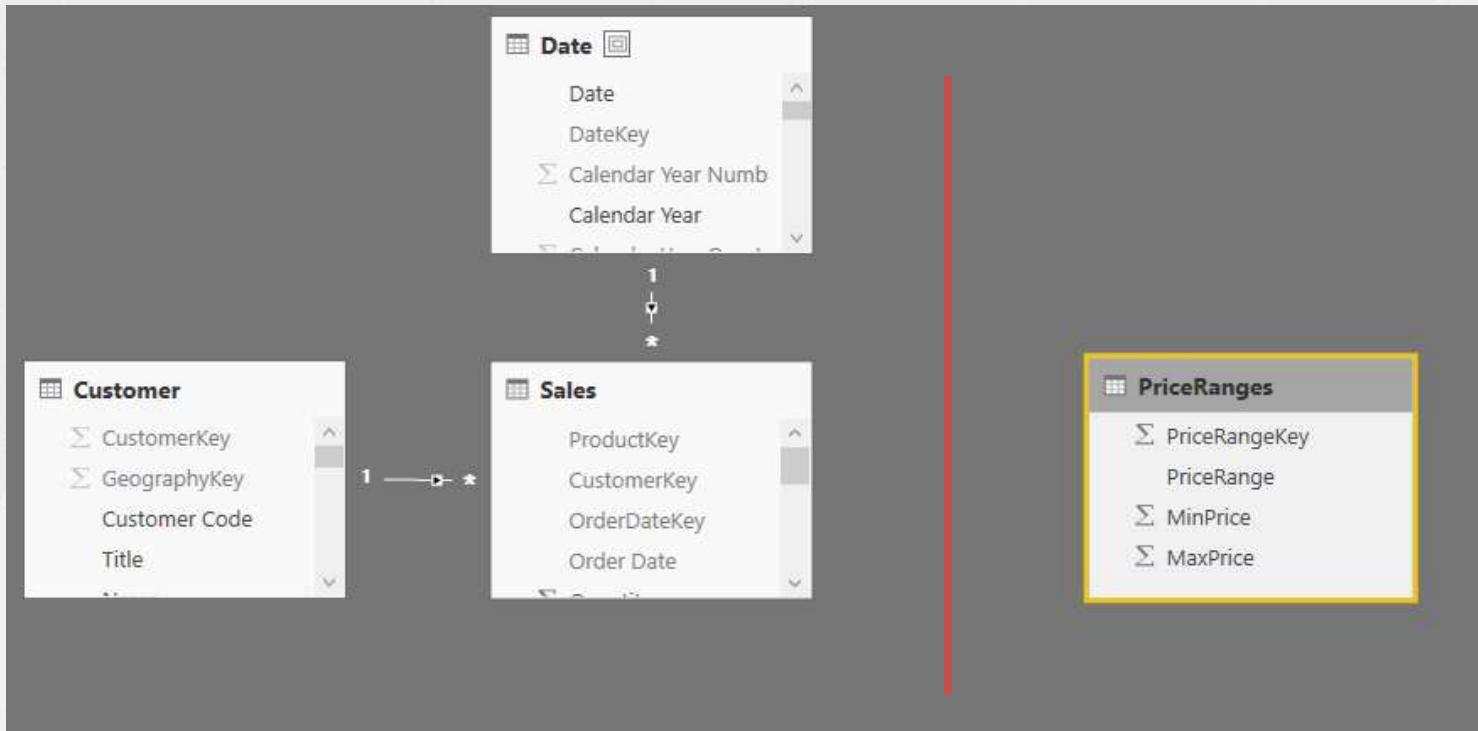


- Analysis of sales based on unit price

PriceRangeKey	PriceRange	MinPrice	MaxPrice
1	VERY LOW	0	10
2	LOW	10	30
3	MEDIUM	30	80
4	HIGH	80	150
5	VERY HIGH	150	99999

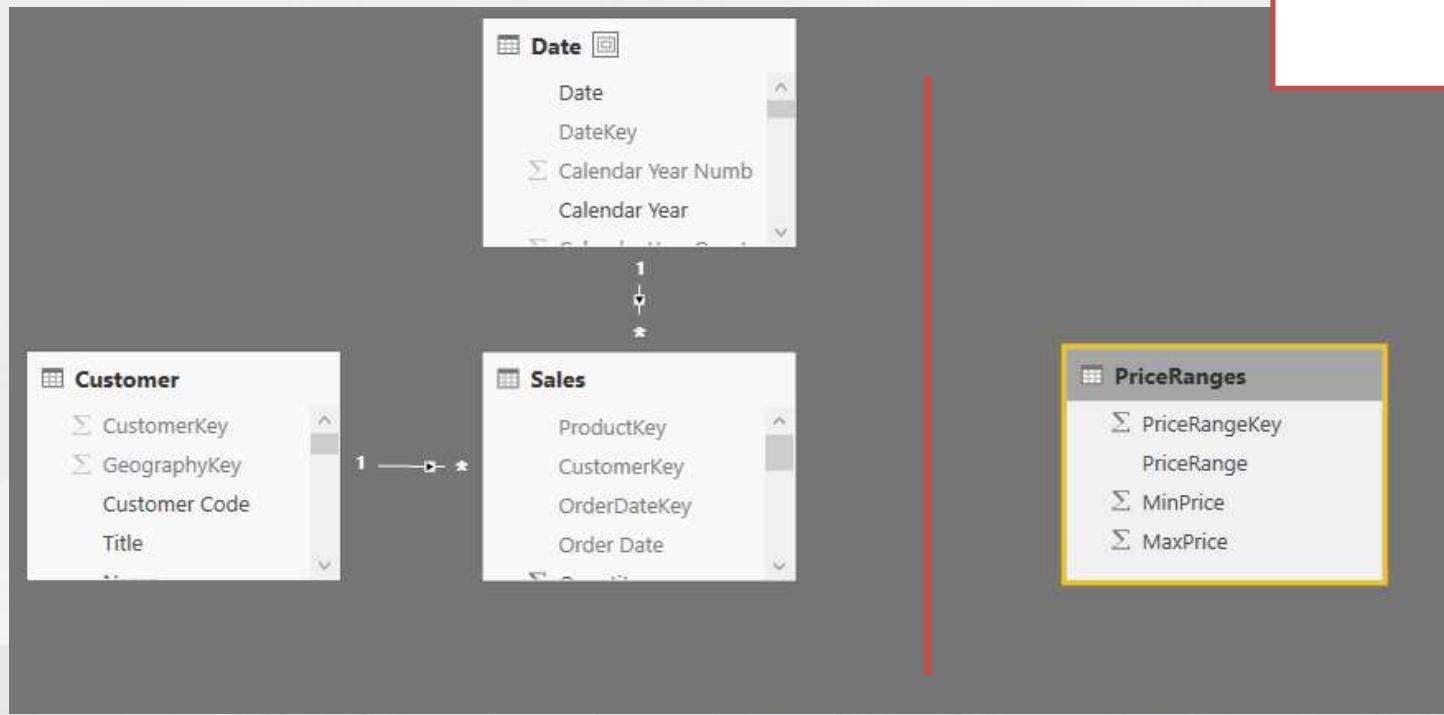
PriceRange	Sales Amount
VERY LOW	\$19,405.17
LOW	\$66,541.73
MEDIUM	\$116,174.19
HIGH	\$287,543.57
VERY HIGH	\$3,730,253.24
<b>Total</b>	<b>\$4,219,917.90</b>

# Static segmentation: the model



# Static segmentation

How can I create the relationship using some sort of “between” join?



# Static segmentation: the formula

A calculated column can denormalize the price segment in the fact table.  
Being a small-cardinality column, the size in RAM is very small.

```
Sales[PriceRange] =  
  
CALCULATE (  
    VALUES ( PriceRanges[PriceRange] ),  
    FILTER (  
        PriceRanges,  
        AND (  
            PriceRanges[MinPrice] <= Sales[Net Price],  
            PriceRanges[MaxPrice] > Sales[Net Price]  
        )  
    )  
)
```

# Denormalizing the key

A similar technique can denormalize the key of the table, so to avoid replicating all of the columns of the segments configuration table.

Pay attention to circular references, this is why we did not use CALCULATE.

```
PriceRangeKey =  
  
CALCULATE (  
    DISTINCT ( PriceRanges[PriceRangeKey] ),  
    FILTER (  
        ALLNOBLANKROW ( PriceRanges ),  
        AND (  
            PriceRanges[MinPrice] <= Sales[Net Price],  
            PriceRanges[MaxPrice] > Sales[Net Price]  
        )  
    ),  
    ALL ( Sales )  
)
```

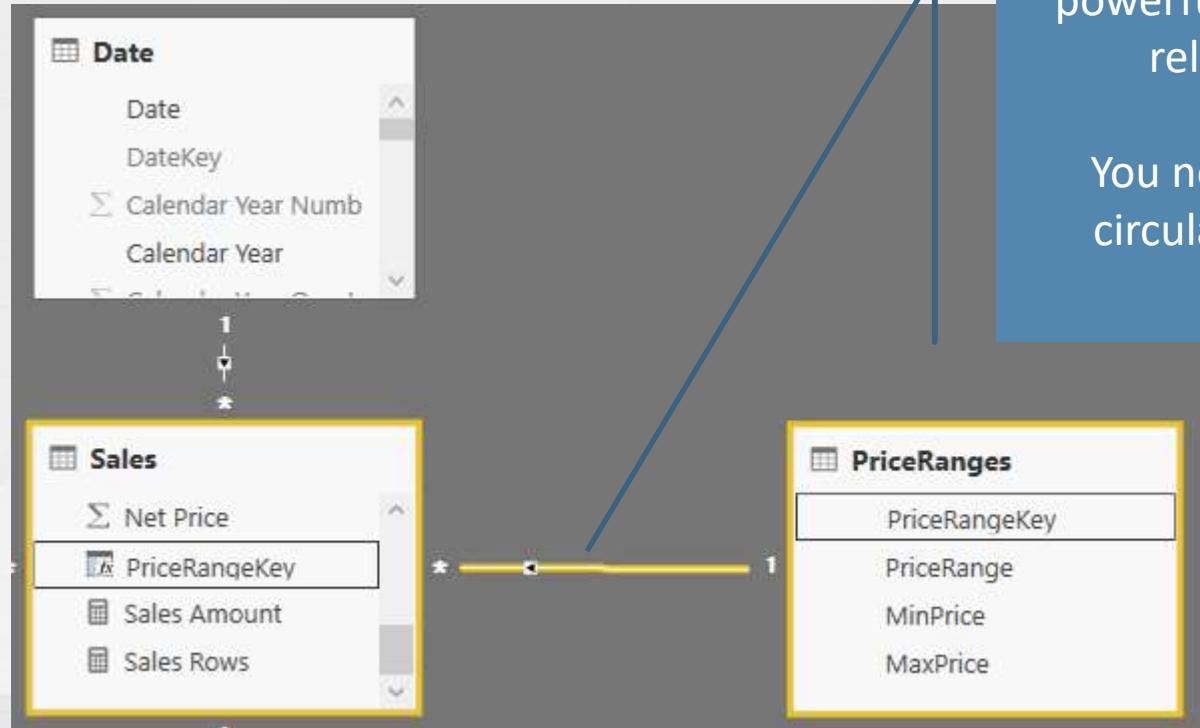
## Denormalizing keys

- Denormalizing the key goes further than you might think at first

*Using DAX, you can build any kind of relationship, no matter how fancy it is, as long as you can compute it in a DAX expression*

- This is an extremely powerful technique, we call it Calculated Relationships

# Calculated relationships



Calculated relationships are very powerful and fast, because they rely on Storage Engine

You need to pay attention at circular dependencies when creating them

# Dynamic segmentation



# Dynamic segmentation



SegmentCode	Segment	MinSale	MaxSale
1	Very Low	0	75
2	Low	75	100
3	Medium	100	500
4	High	500	1000
6	Very High	1000	999999999

The relationship depends on what you put on rows and columns of the report  
In this case, segmentation changes over time.

Segment	CY 2007	CY 2008	CY 2009	Total
Very Low	351	266	255	<b>810</b>
Low	141	14	12	<b>166</b>
Medium	365	76	52	<b>485</b>
High	250	36	35	<b>311</b>
Very High	302	132	160	<b>581</b>
<b>Total</b>	<b>1,409</b>	<b>524</b>	<b>514</b>	<b>2,353</b>

# Dynamic segmentation

SegmentCode	Segment	MinSale	MaxSale
1	Very Low	0	75
2	Low	75	100
3	Medium	100	500
4	High	500	1000
6	Very High	1000	999999999

Segment	CY 2007	CY 2008	CY 2009	Total
Very Low	351	266	255	<b>810</b>
Low	141	14	12	<b>166</b>
Medium	365	76	52	<b>485</b>
High	250	36	35	<b>311</b>
Very High	302	132	160	<b>581</b>
<b>Total</b>	<b>1,409</b>	<b>524</b>	<b>514</b>	<b>2,353</b>

This time, the relationship cannot be created at all...



# Dynamic segmentation: the formula

Dynamic segmentation requires you to build the code in a measure.  
This time, the relationship cannot be created at all.

```
CustInSegment :=
```

```
COUNTROWS (
    FILTER (
        Customer,
        AND (
            [Sales Amount] > MIN ( Segments[MinSale] ),
            [Sales Amount] <= MAX ( Segments[MaxSale] )
        )
    )
)
```

Segment	CY 2007	CY 2008	CY 2009	Total
Very Low	351	266	255	<b>810</b>
Low	141	14	12	<b>166</b>
Medium	365	76	52	<b>485</b>
High	250	36	35	<b>311</b>
Very High	302	132	160	<b>581</b>
<b>Total</b>	<b>1,409</b>	<b>524</b>	<b>514</b>	<b>2,353</b>

## Beware of slicers...

- The previous formula is error-prone
- If the user filters data using a slicer, he can break your code very easily

Segment	Segment	CY 2007	CY 2008	CY 2009	Total
Very Low	Very Low	351	266	255	<b>810</b>
Medium	Very High	302	132	160	<b>581</b>
High	<b>Total</b>	<b>1,409</b>	<b>524</b>	<b>514</b>	<b>2,353</b>
Very High					

# Dynamic segmentation: the formula

Dynamic segmentation requires you to build the code in a measure. This time, the relationship cannot be created at all. Find the correct segment using an iteration.

```
CustInSegment :=
```

```
SUMX (
    Segments,
    COUNTROWS (
        FILTER (
            Customer,
            AND (
                [Sales Amount] > Segments[MinSale],
                [Sales Amount] <= Segments[MaxSale]
            )
        )
    )
)
```

The measure is still  
non-additive over years

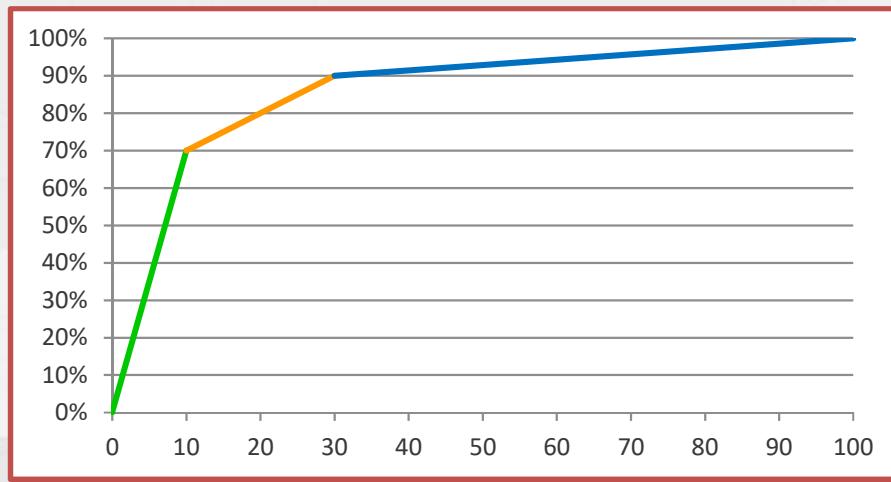
Clustering products based on profitability

## ABC and Pareto analysis



# ABC and Pareto Analysis

- ABC Analysis
  - Class A contains items for 70% of total value
  - Class B contains items for 20% of total value
  - Class C contains items for 10% of total value



# TotalMargin column

It simply computes the total margin per product, the basis of ABC analysis.

```
Product[TotalMargin] =  
SUMX (  
    RELATEDTABLE( Sales ),  
    Sales[Quantity] *  
        (  
            Sales[Net Price] -  
            Sales[Unit Cost]  
        )  
)
```

Product Name	TotalMargin ↑
Adventure Works 26" 720p LCD HDTV M140 Silver	\$81,856.27
Contoso Telephoto Conversion Lens X400 Silver	\$53,464.04
Fabrikam Refrigerator 24.7CuFt X9800 White	\$51,574.26
A. Datum SLR Camera X137 Grey	\$51,459.16
Litware Refrigerator 24.7CuFt X980 Brown	\$29,756.33
Litware Refrigerator 24.7CuFt X980 White	\$28,256.56
NT Washer & Dryer 27in L2700 Blue	\$26,591.59
Proseware Projector 1080p DLP86 Black	\$25,065.45
NT Washer & Dryer 24in M2400 Green	\$24,472.50
SV 16xDVD M360 Black	\$20,989.22
Contoso Projector 1080p X980 White	\$19,648.68

# Margin running total column

Next calculation is the margin running total, that is the sum of all margins higher or equal to the current product.

```
Product[MarginRT] =
```

```
VAR CurrentTotalMargin = 'Product'[TotalMargin]
```

```
RETURN  
    SUMX (  
        FILTER (  
            'Product',  
            'Product'[TotalMargin]  
            >=  
            CurrentTotalMargin  
,  
            'Product'[TotalMargin]  
)
```

Product Name	TotalMargin ↑	MarginRT
Adventure Works 26" 720p LCD HDTV M140 Silver	\$81,856.27	\$81,856.27
Contoso Telephoto Conversion Lens X400 Silver	\$53,464.04	\$135,320.31
Fabrikam Refrigerator 24.7CuFt X9800 White	\$51,574.26	\$186,894.56
A. Datum SLR Camera X137 Grey	\$51,459.16	\$238,353.72
Litware Refrigerator 24.7CuFt X980 Brown	\$29,756.33	\$268,110.06
Litware Refrigerator 24.7CuFt X980 White	\$28,256.56	\$296,366.62
NT Washer & Dryer 27in L2700 Blue	\$26,591.59	\$322,958.20
Proseware Projector 1080p DLP86 Black	\$25,065.45	\$348,023.65
NT Washer & Dryer 24in M2400 Green	\$24,472.50	\$372,496.15
SV 16xDVD M360 Black	\$20,989.22	\$393,485.38
Contoso Projector 1080p X980 White	\$19,648.68	\$413,134.06

# Margin Pct column

Next step is to transform the margin running total into a percentage against the grand total.

```
Product[MarginPct] =  
DIVIDE (  
    'Product'[MarginRT],  
    SUM ( 'Product'[TotalMargin] )  
)
```

Product Name	TotalMargin ↑	MarginRT	MarginPct
Adventure Works 26" 720p LCD HDTV M140 Silver	\$81,856.27	\$81,856.27	4.06 %
Contoso Telephoto Conversion Lens X400 Silver	\$53,464.04	\$135,320.31	6.70 %
Fabrikam Refrigerator 24.7CuFt X9800 White	\$51,574.26	\$186,894.56	9.26 %
A. Datum SLR Camera X137 Grey	\$51,459.16	\$238,353.72	11.81 %
Litware Refrigerator 24.7CuFt X980 Brown	\$29,756.33	\$268,110.06	13.28 %
Litware Refrigerator 24.7CuFt X980 White	\$28,256.56	\$296,366.62	14.68 %
NT Washer & Dryer 27in L2700 Blue	\$26,591.59	\$322,958.20	16.00 %
Proseware Projector 1080p DLP86 Black	\$25,065.45	\$348,023.65	17.24 %
NT Washer & Dryer 24in M2400 Green	\$24,472.50	\$372,496.15	18.45 %
SV 16xDVD M360 Black	\$20,989.22	\$393,485.38	19.49 %
Contoso Projector 1080p X980 White	\$19,648.68	\$413,134.06	20.47 %

# ABC Class column

As a final step, transform the percentage into the class (A, B or C).

Product[ABC Class] =

```
IF (
    'Product'[MarginPct] <= 0.7,
    "A",
    IF (
        'Product'[MarginPct] <= 0.9,
        "B",
        "C"
    )
)
```

Product Name	TotalMargin ↑	MarginRT	MarginPct	ABC Class
Adventure Works 26" 720p LCD HDTV M140 Silver	\$81,856.27	\$81,856.27	4.06 %	A
Contoso Telephoto Conversion Lens X400 Silver	\$53,464.04	\$135,320.31	6.70 %	A
Fabrikam Refrigerator 24.7CuFt X9800 White	\$51,574.26	\$186,894.56	9.26 %	A
A. Datum SLR Camera X137 Grey	\$51,459.16	\$238,353.72	11.81 %	A
Litware Refrigerator 24.7CuFt X980 Brown	\$29,756.33	\$268,110.06	13.28 %	A
Litware Refrigerator 24.7CuFt X980 White	\$28,256.56	\$296,366.62	14.68 %	A
NT Washer & Dryer 27in L2700 Blue	\$26,591.59	\$322,958.20	16.00 %	A
Proseware Projector 1080p DLP86 Black	\$25,065.45	\$348,023.65	17.24 %	A
NT Washer & Dryer 24in M2400 Green	\$24,472.50	\$372,496.15	18.45 %	A
SV 16xDVD M360 Black	\$20,989.22	\$393,485.38	19.49 %	A
Contoso Projector 1080p X980 White	\$19,648.68	\$413,134.06	20.47 %	A

# Using Variables

By leveraging variables, ABC can be computed with a single calculated column.

```
ABC Class =  
  
VAR CurrentTotalSales = [Margin]  
  
VAR AllSales = CALCULATE ( [Margin], ALL ( 'Product' ) )  
  
VAR AllProductMargins = ADDCOLUMNS ( 'Product', "ProductMargin", [Margin] )  
  
VAR CurrentMarginRt =  
    SUMX (   
        FILTER ( AllProductMargins, [ProductMargin] >= CurrentTotalSales ),  
        [ProductMargin]   
    )  
  
VAR CurrentPct = DIVIDE ( CurrentMarginRt, AllSales )  
  
VAR CurrentABC = IF ( CurrentPct <= 0.7, "A", IF ( CurrentPct <= 0.9, "B", "C" ) )  
  
RETURN  
    CurrentABC
```

Handling multiple currencies is a very challenging experience

## Working with multiple currencies



## Using multiple currencies

- Buy with multiple currencies
- Sell with multiple currencies
- Own money in different currencies
- Report profit and loss with different currencies
- Currency exchange is updated every day
- In other words, numbers change due to too many different parameters and reporting becomes hard

## We cover three scenarios

- Multiple source currencies, single reporting currency
  - Sell with multiple currencies
  - Report with a single currency
- Single source currency, multiple reporting currencies
  - Sell with a single currency
  - Report with multiple currencies
- Multiple source and reporting currencies
  - Sell with multiple currencies
  - Report with multiple currencies

# Beware of simple calculations



A simple SUMX works, if sliced by currency, but it produces a meaningless result at the grand total.

At the grand total there are different currencies summed together, which is useless, if not wrong.

Sales Amount :=

```
SUMX (  
    Sales,  
    Sales[Quantity]  
    * Sales[Net Price]  
)
```

Currency	CY 2007	CY 2008	CY 2009	Total
Armenian Dram	181,160.03	98,721.99	133,696.43	<b>413,578.45</b>
Australian Dollar	181,974.84	128,466.53	155,045.00	<b>465,486.37</b>
Canadian Dollar	136,916.66	159,722.90	159,992.10	<b>456,631.66</b>
Danish Krone	151,100.25	84,134.31	130,819.13	<b>366,053.69</b>
EURO	170,560.82	124,161.46	158,567.36	<b>453,289.64</b>
Hong Kong Dollar	120,129.99	130,518.12	161,045.14	<b>411,693.25</b>
Indian Rupee	174,326.71	123,890.69	139,402.73	<b>437,620.13</b>
Thai Baht	123,623.76	159,501.98	106,073.85	<b>389,199.58</b>
US Dollar	219,422.89	113,417.08	97,892.87	<b>430,732.85</b>
<b>Total</b>	<b>1,459,215.95</b>	<b>1,122,535.05</b>	<b>1,242,534.61</b>	<b>3,824,285.61</b>

# Removing the total is a simple task



To remove the total, you need to check if a single currency is selected and act accordingly, by blanking the measure if multiple currencies are visible.

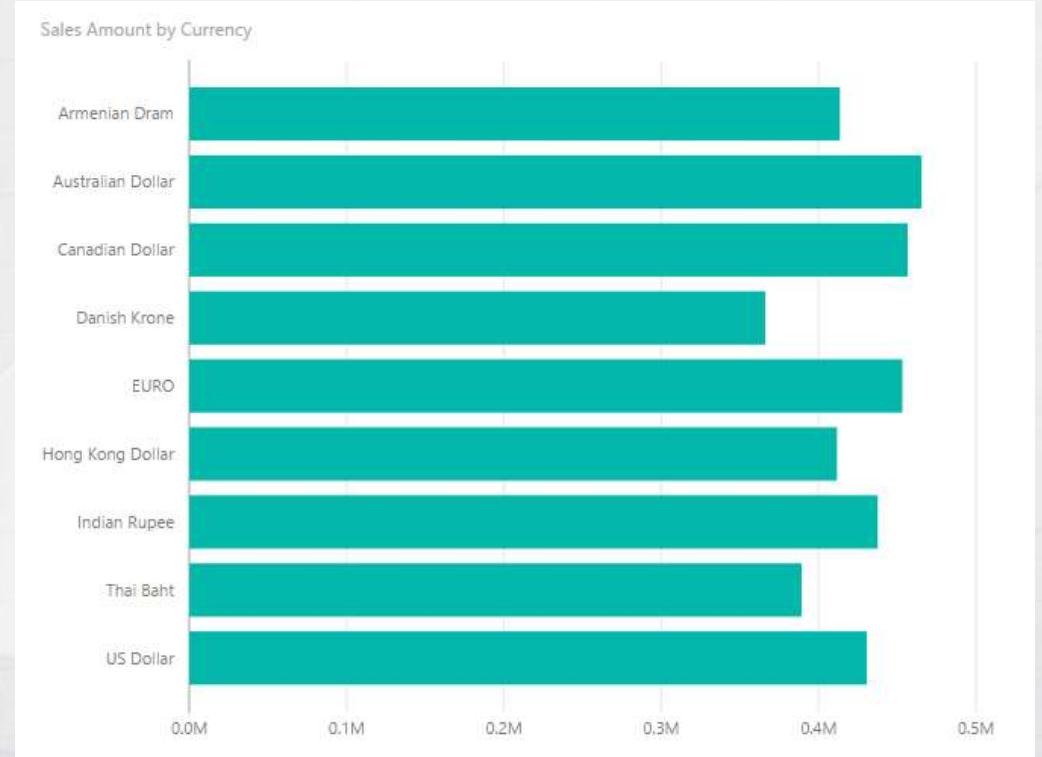
Sales Amount :=

```
IF (
    HASONEVALUE ( Currency[Currency] )
    SUMX (
        Sales,
        Sales[Quantity]
        * Sales[Net Price]
    )
)
```

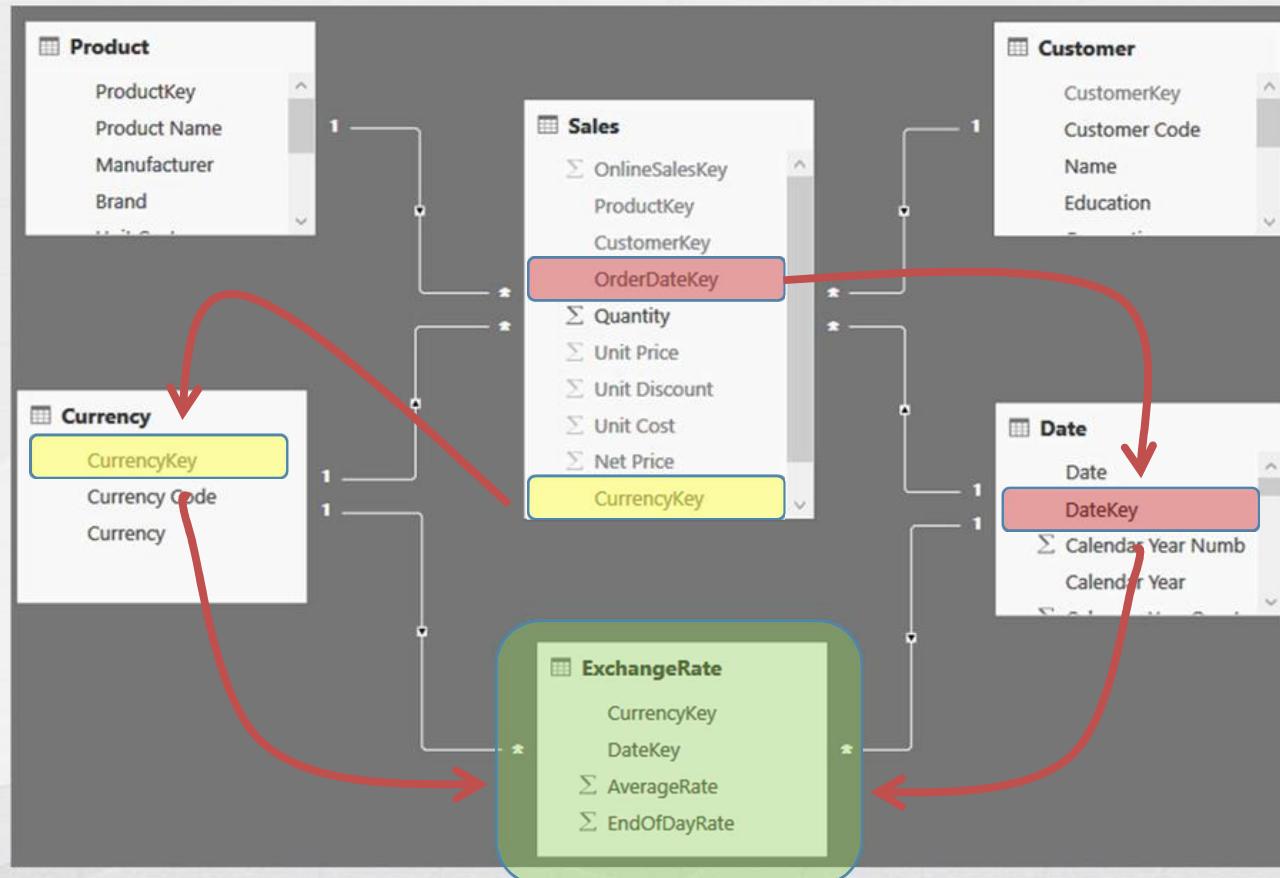
	Currency	CY 2007	CY 2008	CY 2009	Total
	Armenian Dram	181,160.03	98,721.99	133,696.43	<b>413,578.45</b>
	Australian Dollar	181,974.84	128,466.53	155,045.00	<b>465,486.37</b>
	Canadian Dollar	136,916.66	159,722.90	159,992.10	<b>456,631.66</b>
	Danish Krone	151,100.25	84,134.31	130,819.13	<b>366,053.69</b>
	EURO	170,560.82	124,161.46	158,567.36	<b>453,289.64</b>
	Hong Kong Dollar	120,129.99	130,518.12	161,045.14	<b>411,693.25</b>
	Indian Rupee	174,326.71	123,890.69	139,402.73	<b>437,620.13</b>
	Thai Baht	123,623.76	159,501.98	106,073.85	<b>389,199.58</b>
	US Dollar	219,422.89	113,417.08	97,892.87	<b>430,732.85</b>
	<b>Total</b>				

## Nevertheless, a chart is misleading

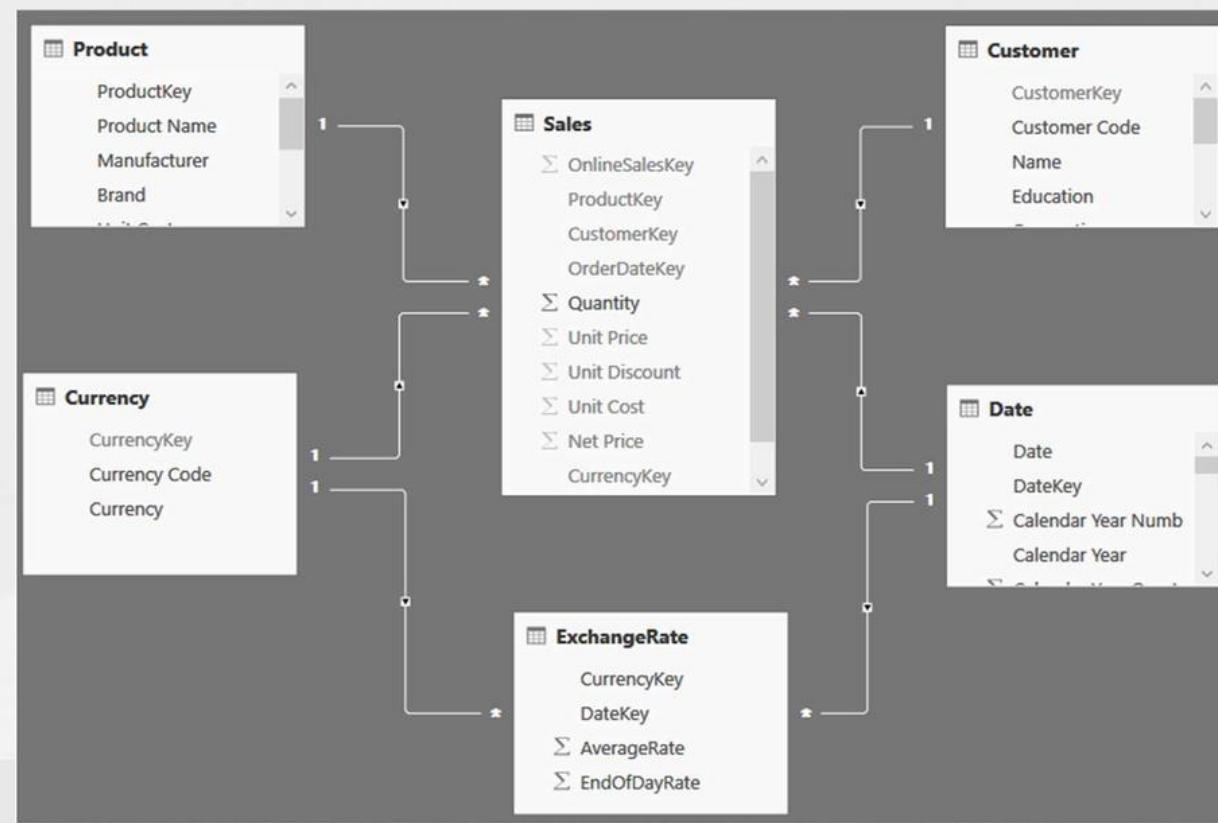
- There are no numeric values
- The size of the bars suggest some sort of comparison
  - Unfortunately, it would be wrong
- Charts with multiple currencies are misleading



# Multiple sources, one reporting currency



# Multiple sources, one reporting currencies



What is the easiest  
way to convert all the  
different currencies in  
a single one?



# Conversion with a calculated column



If the conversion is fixed at the date of the event (as it is likely), then a calculated column can denormalize the exchange rate straight in the fact table.

```
Sales[RateToUsd] =  
LOOKUPVALUE(  
    ExchangeRate[AverageRate],  
    ExchangeRate[CurrencyKey], Sales[CurrencyKey],  
    ExchangeRate[DateKey], RELATED ('Date'[Date])  
)
```

```
Sales Amount USD =  
SUMX (  
    Sales,  
    Sales[Quantity] * DIVIDE ( Sales[Net Price], Sales[RateToUsd] )  
)
```

# Check for missing exchange rates



Whenever handling these conversions, pay attention to missing information. What if the exchange rate is not available for some days? Get the latest available, then.

RateToUsd =

```
VAR CurrentDate = RELATED ( 'Date'[Date] )
VAR CurrentCurrency = Sales[CurrencyKey]
```

RETURN

```
LOOKUPVALUE (
    ExchangeRate[AverageRate],
    ExchangeRate[CurrencyKey], Sales[CurrencyKey],
    ExchangeRate[Date], CALCULATE (
        MAX ( 'ExchangeRate'[Date] ),
        'ExchangeRate'[Date] <= CurrentDate,
        ExchangeRate[CurrencyKey] = CurrentCurrency,
        ALL ( ExchangeRate )
    )
)
```

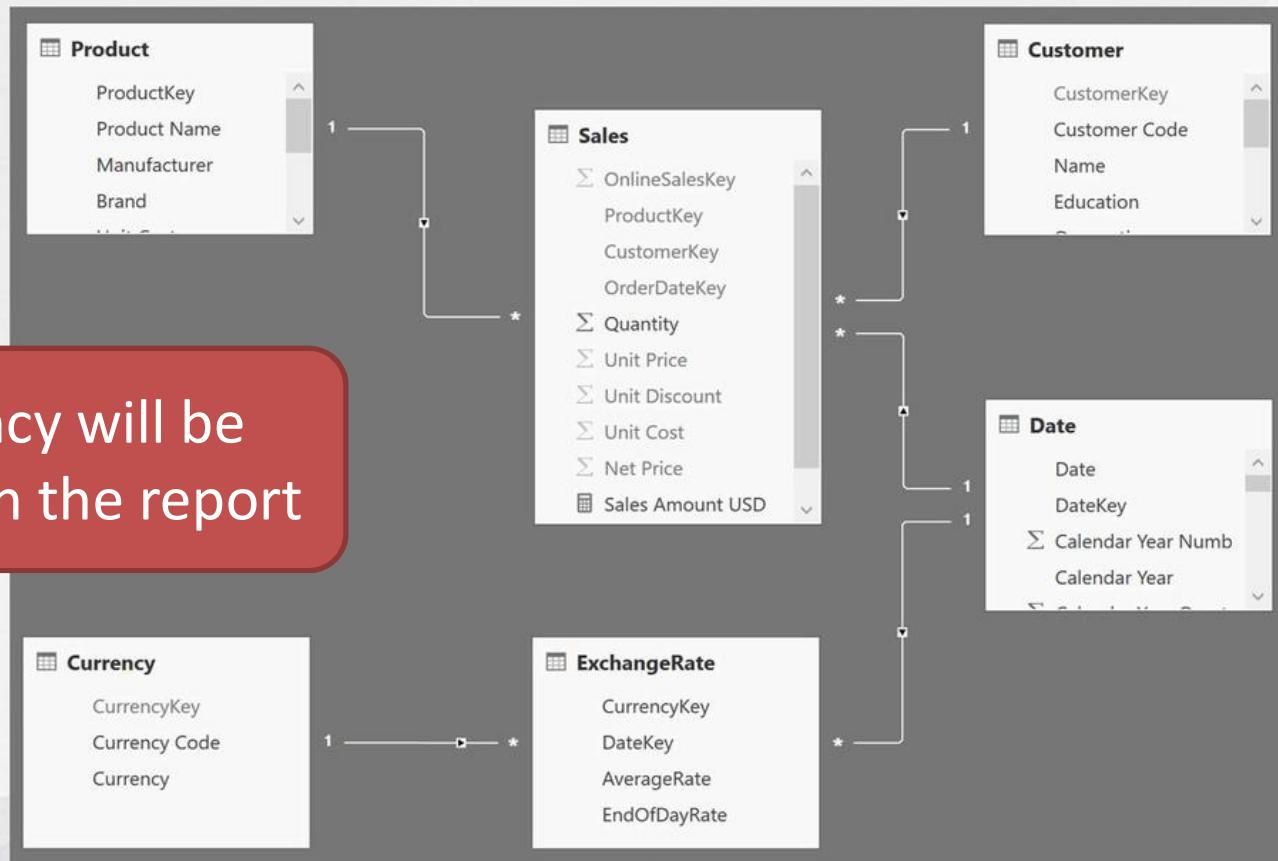
Currency	CY 2007	CY 2008	CY 2009	Total
Armenian Dram	\$546.23	\$322.85	\$375.86	<b>\$1,244.94</b>
Australian Dollar	\$151,722.84	\$112,736.93	\$118,496.89	<b>\$382,956.66</b>
Canadian Dollar	\$127,000.26	\$152,197.14	\$137,926.76	<b>\$417,124.16</b>
Danish Krone	\$27,499.38	\$17,389.56	\$24,282.17	<b>\$69,171.11</b>
EURO	\$235,090.17	\$187,431.98	\$219,589.81	<b>\$642,111.97</b>
Hong Kong Dollar	\$15,386.65	\$17,131.00	\$20,773.18	<b>\$53,290.83</b>
Indian Rupee	\$4,303.89	\$2,944.15	\$2,882.09	<b>\$10,130.13</b>
Thai Baht	\$3,829.75	\$5,015.39	\$3,077.48	<b>\$11,922.62</b>
US Dollar	\$219,422.89	\$113,417.08	\$97,892.87	<b>\$430,732.85</b>
<b>Total</b>	<b>\$784,802.08</b>	<b>\$608,586.08</b>	<b>\$625,297.11</b>	<b>\$2,018,685.26</b>

## Multiple reporting currencies

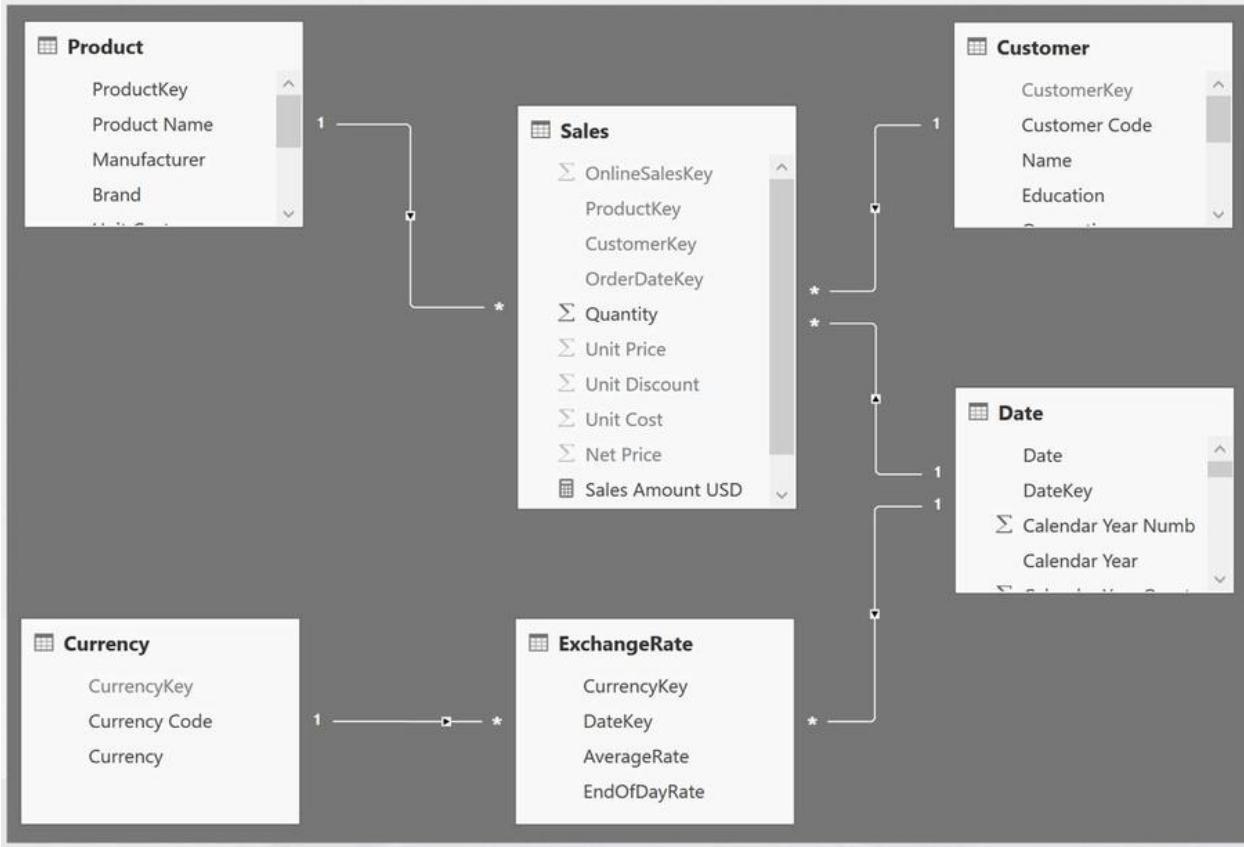
- Storing exchange rate in a calculated column
  - Is straightforward, but it works only if you have a single currency for reporting
- If you need to report in multiple currencies, you can:
  - Create one calculated column for each currency
  - Use measures instead
- Measures are more flexible

# Single source, multiple reporting currencies

Currency will be  
filtered in the report



# Single source, multiple reporting currencies



Now a calculated column no longer works, as the reporting currency is unknown



## What the formula should perform

- Check that a single currency is selected
- For each date:
  - Compute the value of sales (single currency)
  - Determine the conversion ratio of that day in the selected currency
  - If no conversion is available, search for the previous available one
  - Perform the conversion
- Aggregate with (SUM) the values computed so far

# Multiple reporting currencies: the formula

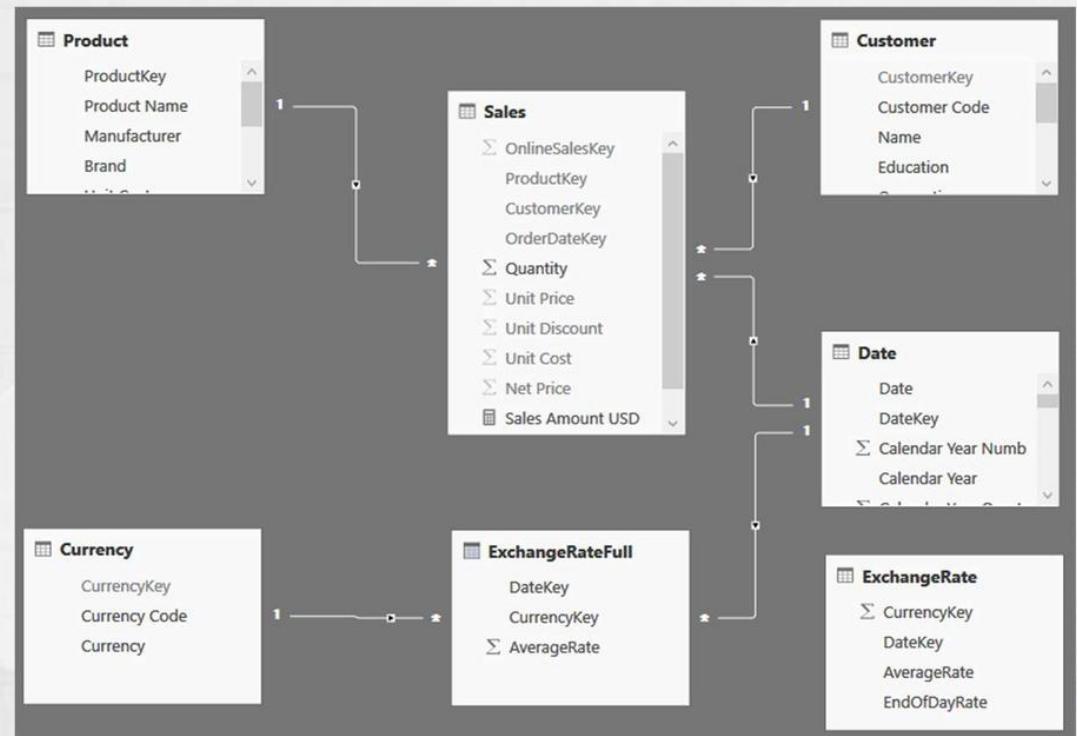
```
Sales Converted =  
  
IF (  
    HASONEVALUE ( 'Currency'[Currency] ),  
    SUMX (   
        VALUES ( 'Date'[Date] ),  
        VAR CurrentDate = 'Date'[Date]  
        VAR LastDateAvailable =  
            CALCULATE (   
                MAX ( 'ExchangeRate'[DateKey] ),  
                'ExchangeRate'[DateKey] <= CurrentDate,  
                ALL ( 'Date' )  
            )  
        VAR Rate =  
            CALCULATE (   
                VALUES ( ExchangeRate[AverageRate] ),  
                ExchangeRate[DateKey] = LastDateAvailable,  
                ALL ( 'Date' )  
            )  
        RETURN  
            [Sales Amount USD] * Rate  
    )  
)
```

## Report in multiple currencies

- The total is still missing, because the numbers cannot be added together
- Rows represent the same value, in different currencies

# Precomputing full exchange rates

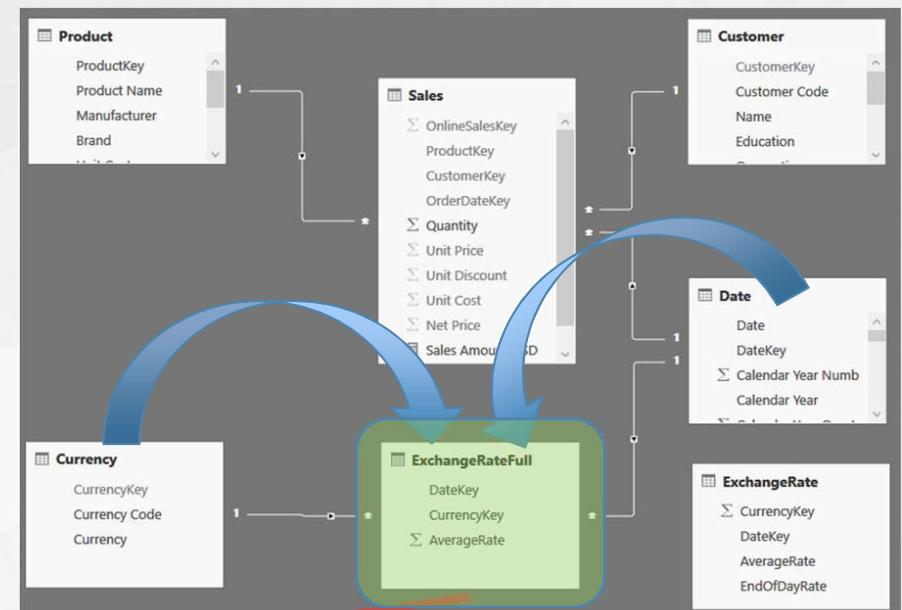
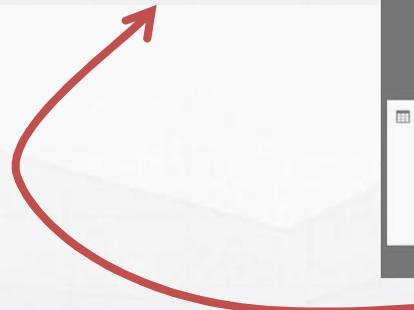
- The most complex part of the code is the detection of missing exchange rates
- Precompute it in a table so to make the code easier and faster
- A calculated table is the best option
- Move the complexity in the calculated table



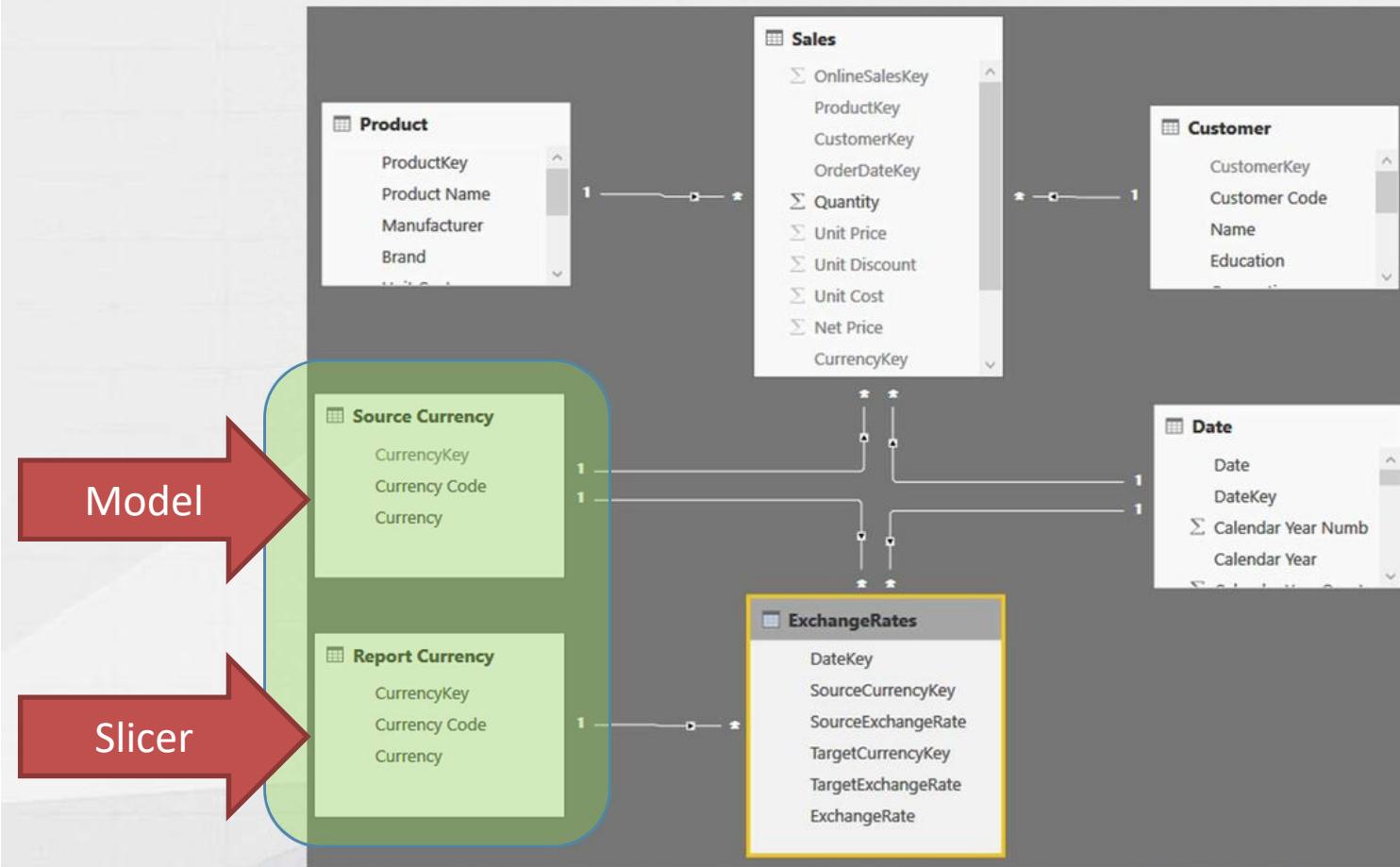
# The formula becomes much simpler

Avoiding the need to check for each row the existence of an exchange rate, the code is now much simpler (remember that this code might be needed in multiple measures).

```
Sales Converted =  
IF (  
    HASONEVALUE ( 'Currency'[Currency] ),  
    SUMX (   
        VALUES ( 'Date'[Date] ),  
        [Sales Amount USD] *  
        CALCULATE (   
            VALUES ( ExchangeRateFull[AverageRate] )  
        )  
    )  
)
```



# Multiple sources, multiple reporting currencies



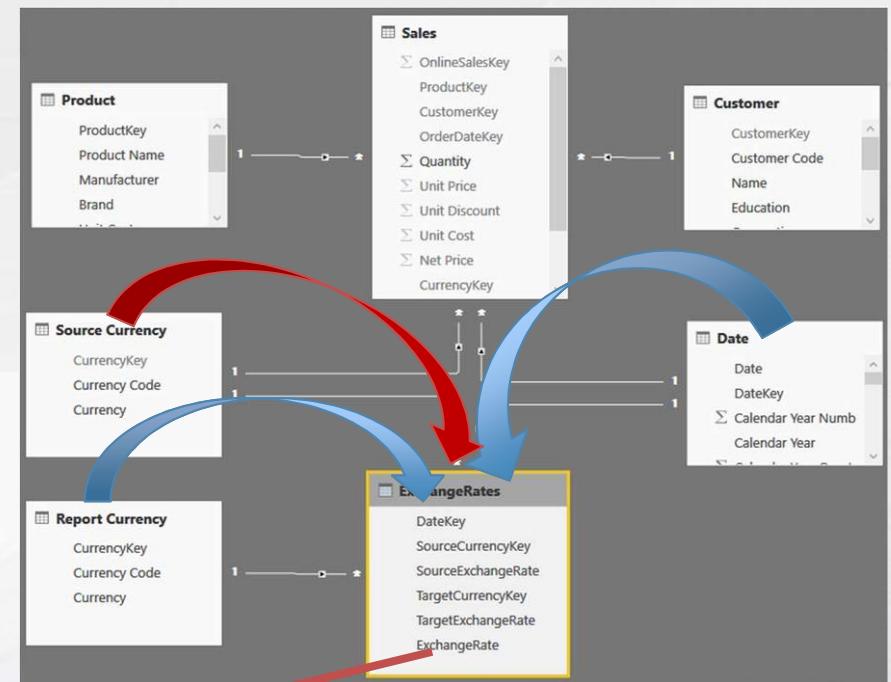
## Comments on the model

- The exchange rate table now contains two currencies
  - Source
  - Target
- The value is the exchange rate between the two

# The formula is similar to the previous one

The ExchangeRates table receives three different filters: one coming from the slicer, and two coming from context transition

```
Sales Amount Converted =  
IF (  
    HASONEVALUE ( 'Report Currency'[Currency] ),  
    SUMX (   
        SUMMARIZE (   
            Sales,  
            'Date'[Date],  
            'Source Currency'[Currency]  
        ),  
        [Sales Amount] *  
        CALCULATE (   
            VALUES ( ExchangeRates[ExchangeRate] )  
        )  
    )  
)
```



# Sample report with multiple currencies

Report  
Currency

Sales  
Currency

	Currency	Currency	CY 2007	CY 2008	CY 2009	Total
EURO	Armenian Dram	43,373,742.38	19,173,119.06	34,759,303.63	<b>97,306,165.08</b>	
	Australian Dollar	159,842.98	99,235.98	148,748.47	<b>407,827.44</b>	
	Canadian Dollar	109,409.84	115,736.45	135,707.60	<b>360,853.89</b>	
	Danish Krone	613,644.73	276,219.86	511,179.97	<b>1,401,044.57</b>	
	EURO	170,560.82	124,161.46	158,567.36	<b>453,289.64</b>	
	Hong Kong Dollar	686,597.21	672,497.85	906,915.35	<b>2,266,010.41</b>	
	Indian Rupee	5,173,111.76	3,568,148.79	4,880,686.52	<b>13,621,947.08</b>	
	Thai Baht	2,927,959.72	3,421,109.47	2,657,193.07	<b>9,006,262.26</b>	
	US Dollar	158,649.44	75,565.55	71,791.82	<b>306,006.81</b>	
	<b>Total</b>	<b>53,373,518.89</b>	<b>27,525,794.47</b>	<b>44,230,093.81</b>	<b>125,129,407.17</b>	
US Dollar	Armenian Dram	60,370,506.29	30,189,302.89	47,865,914.70	<b>138,425,723.89</b>	
	Australian Dollar	218,812.35	148,993.71	204,242.14	<b>572,048.20</b>	
	Canadian Dollar	148,139.37	169,625.07	185,952.40	<b>503,716.84</b>	
	Danish Krone	830,847.78	413,258.38	705,403.35	<b>1,949,509.52</b>	
	EURO	123,896.88	82,818.64	114,555.62	<b>321,271.15</b>	
	Hong Kong Dollar	937,908.50	996,945.47	1,248,510.82	<b>3,183,364.80</b>	
	Indian Rupee	7,079,309.87	5,281,242.15	6,743,294.80	<b>19,103,846.83</b>	
	Thai Baht	3,997,511.72	5,117,644.99	3,657,183.94	<b>12,772,340.64</b>	
	US Dollar	219,422.89	113,417.08	97,892.87	<b>430,732.85</b>	
	<b>Total</b>	<b>73,926,355.67</b>	<b>42,513,248.38</b>	<b>60,822,950.66</b>	<b>177,262,554.72</b>	
<b>Total</b>	Currency	CY 2007	CY 2008	CY 2009	Total	
	EURO	43,373,742.38	19,173,119.06	34,759,303.63	<b>97,306,165.08</b>	
	Armenian Dram	159,842.98	99,235.98	148,748.47	<b>407,827.44</b>	
	Australian Dollar	109,409.84	115,736.45	135,707.60	<b>360,853.89</b>	
	Danish Krone	613,644.73	276,219.86	511,179.97	<b>1,401,044.57</b>	
	EURO	170,560.82	124,161.46	158,567.36	<b>453,289.64</b>	
	Hong Kong Dollar	686,597.21	672,497.85	906,915.35	<b>2,266,010.41</b>	
	Indian Rupee	5,173,111.76	3,568,148.79	4,880,686.52	<b>13,621,947.08</b>	
	Thai Baht	2,927,959.72	3,421,109.47	2,657,193.07	<b>9,006,262.26</b>	
	US Dollar	158,649.44	75,565.55	71,791.82	<b>306,006.81</b>	
US Dollar	<b>Total</b>	<b>53,373,518.89</b>	<b>27,525,794.47</b>	<b>44,230,093.81</b>	<b>125,129,407.17</b>	
	Armenian Dram	60,370,506.29	30,189,302.89	47,865,914.70	<b>138,425,723.89</b>	
	Australian Dollar	218,812.35	148,993.71	204,242.14	<b>572,048.20</b>	
	Canadian Dollar	148,139.37	169,625.07	185,952.40	<b>503,716.84</b>	
	Danish Krone	830,847.78	413,258.38	705,403.35	<b>1,949,509.52</b>	
	EURO	123,896.88	82,818.64	114,555.62	<b>321,271.15</b>	
	Hong Kong Dollar	937,908.50	996,945.47	1,248,510.82	<b>3,183,364.80</b>	
	Indian Rupee	7,079,309.87	5,281,242.15	6,743,294.80	<b>19,103,846.83</b>	
	Thai Baht	3,997,511.72	5,117,644.99	3,657,183.94	<b>12,772,340.64</b>	
	US Dollar	219,422.89	113,417.08	97,892.87	<b>430,732.85</b>	
	<b>Total</b>	<b>73,926,355.67</b>	<b>42,513,248.38</b>	<b>60,822,950.66</b>	<b>177,262,554.72</b>	

# Thank you!



Check our articles, whitepapers and courses on  
[www.sqlbi.com](http://www.sqlbi.com)