Model Data in Power BI

In this module you will learn:

- How to **design a data model that is intuitive, high-performing, and simple to maintain**
- You will learn about using the **DAX language to create measures**
- Those measures will help you **create a wide variety of analytical solutions.**
- Additionally, **you will learn how to improve performance with your Power Query retrieval tasks**

Introduction

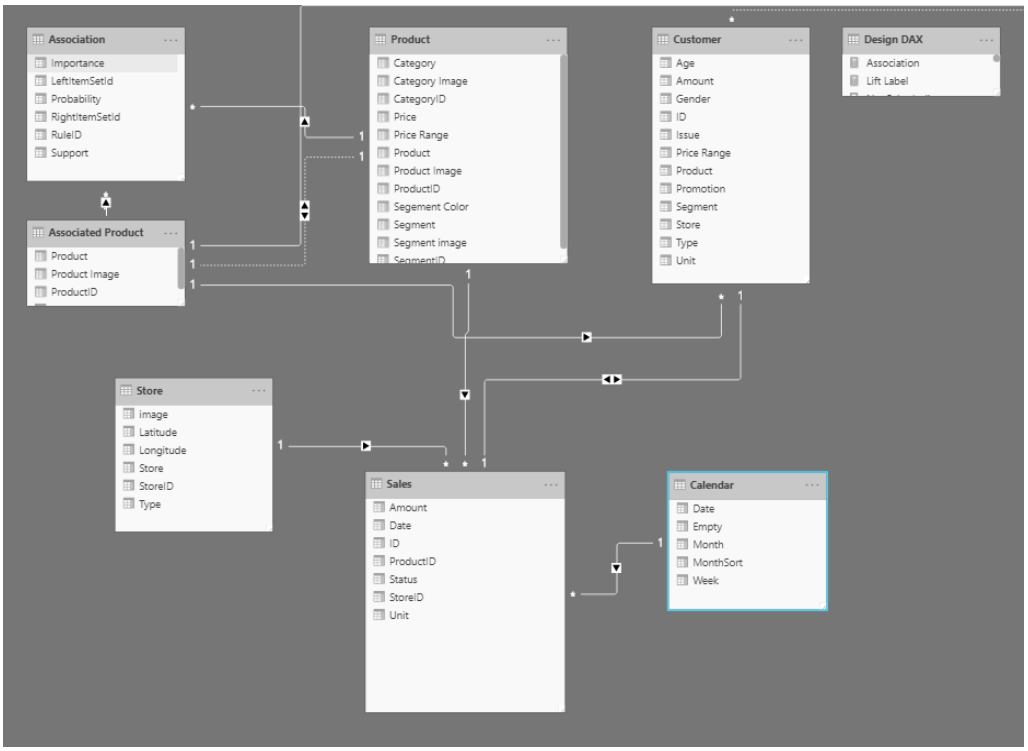
Creating a great data model is **one of the most important tasks that a data analyst can perform in Microsoft Power BI**

A good data model offers the following benefits:

- Data exploration is faster
- Aggregations are simpler to build
- Reports are more accurate
- Writing reports takes less time
- Reports are easier to maintain in the future

Generally, a **smaller data model is better** because **it will perform faster and will be simpler to use**

Aim for simplicity when **designing data models**



The above image is **an example data model**

Boxes contains **tables of data**

each line within the box is a column

lines that **connects the boxes** represent **relationships between the tables**

Relationships are defined between tables through **primary and foreign keys**

Primary Keys

Primary keys are **columns** that **identify each unique, non-null data row**.

A unique value for every single entry in the table

Each row is **assigned a unique value**, which can be referred to by this simple value

Foreign Keys

Values which **references to rows in different tables**

Relationships between tables are formed when you **have primary** and **foreign keys** in **common between different tables**.

Data Schema

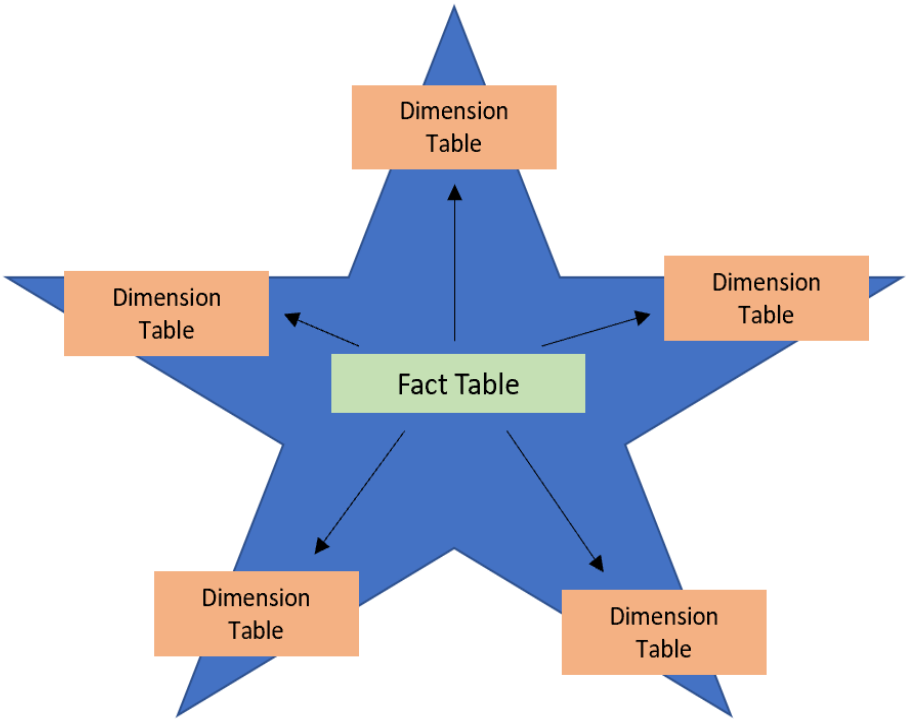
The Basics of a Data Schema

Pulling one table from a relational database (SQL database) and Microsoft Excel file and **creating a relationship between those two tables** and **treating them as a unified dataset**

Star Schemas

designed to simplify your data. **A popular method to simplify your data**

In a star schema, **each table within your dataset is defined as either a dimension or a fact table**



Fact Tables

contain observational or event data values

- sales orders
- product counts
- prices
- transactional dates and times
- quantities

can **contain several repeated values**

e.g., **one product can appear multiple times** in **multiple rows**, for **different customers** on **different dates**

These **values can be aggregated** to **create visuals**

e.g., **a visual of the total sales** is an **aggregation of all sales orders** in the **fact table**

with fact tables it is **common** to see **columns that are filled with numbers and dates**

An example fact table can be **transactions database in an e-commerce website**. Where **it is common to see columns** that are **filled with numbers and sales**. The **numbers** can be **units of measurements**, such as sale amounts, or they can be **keys**, such as **customer ID**. **The dates** represent that is being **recorded, like order data** or **shipped date**.

Fact tables are **usually much larger than dimension tables** because **numerous events occur in fact tables**, such as **individual sales in a transaction database for an e-commerce website**

Dimension Tables

contains the details about the data in the fact tables

- products
- customers
- locations
- employees
- order types

These **tables** are **connected to the fact table** through **KEY COLUMNS**.

Dimensions tables are used to **filter and group the data in fact tables**

The dimension tables, by contrast, **contain unique values**

for instance,

- **one row for each product** in the **Products table**
- **one row for each customer** in the **Customer table**

this data can be used to e.g.,

- **total sales orders visual**, you could group the data so that you see **total sales orders by product**, in which **product is data in the dimension table**.

Dimension tables are **typically smaller because numerous events occur in fact tables**, such as **individual sales**

An Example of A Fact Tables and Dimension Tables

A scenario:
Two tables exist – Employees and Sales

Sales – **a fact table**
contains the sales order values, which **can be aggregated**

Employees – **a dimension table**
contains the specific employee name, which **filters the sales orders**

The **common column between the two tables**, which is the **primary key** in **Employees** table, is **EmployeeID**

therefore we can **establish a relationship between the two tables**

Star schemas and **the underlying data model** are the **foundation of organized reports**.

Work with Tables

Users enjoy **fewer tables to interact with** – this is because **data model and table structures are simplified**

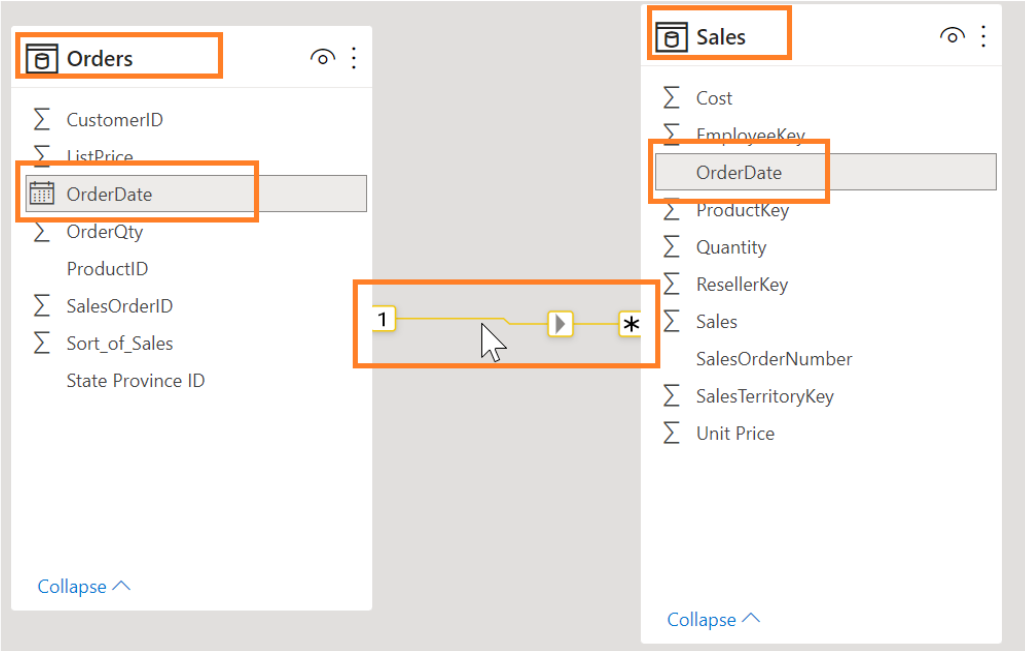
A **simple table structure** will:

- **Be simple to navigate** because of columns and table properties that are specifically and user-friendly
- **Have merged or appended tables** to **simplify the tables** within your data structures
- **Have good-quality relationships** between **tables that make sense**

ONLY ONE KEY RELATIONSHIP CAN EXIST BETWEEN TABLES

Configure Data Model and Build Relationships between Tables

The following image displays how **relationships between Order and Sales table** can **be seen through the OrderDate column**



Assuming that you've already retrieved your data and cleaned it in Power Query,

you can then **go to the Model tab**, where the model is located

You can also **view a complete list of relationships** in the **Manage Relationship** button on the **top ribbon** => a window will pop-up that:

allows you to **create, edit, and delete relationships between tables** and also **autodetects relationship**s that already exist

Autodetect Feature

help you establish relationships between columns that are named similarly

The **Manage Relationship feature** also allows you to **configure relationships between tables,** you can also **configure tables** and **column properties** to **ensure organization** in your table structure

Configure Table and Column Properties

The **Model view** in **Power BI Desktop** also provides options within the **column properties** that you can **view or update**

A **simple method to get to this menu** to update tables and fields is by **Ctrl+clicking** or **Shift+clicking** items on this page.

Under the **General tab,** you can:

- Edit the name and description of the column
- Add synonyms that can be used to identify the column when you are using the Q & A feature
- Add a column into a folder to further organize the table structure
- Hide or show the columns

Under the **Formatting tab**, you can:

- Change the data type
- Format the date

Under the **Advanced tab**, you can:

- Sort by a specific column
- Assign a specific category to the data
- Summarize the data
- Determine if the column or table contains null values

Power BI also allows you to **update these properties on many tables and fields** by **Ctrl+clicking** or **Shift+clicking** items

Create a Date Table

Making calculations based on **date and time** is a common business requirement

Organizations want to know **how their business is doing over months, quarters, fiscal years**, and so on.

The scenario:
Multiple tables, Orders and Sales, **each contain their own date columns**

Develop a table of the total sales and orders by year and month

How can you **build a visual with multiple tables**, each **referencing their own date columns**?

The SOLUTION: Create a common date table that can be used by multiple tables

Create a Common Date Table

Ways to build a common date table:

- Source data
- Data analysis expressions (DAX)
- Power Query

Source Data

Occasionally, **source databases and data warehouses already have their own data tables**

These tables can be used to perform the following tasks:

- Identify company holdings
- Separate calendar and fiscal year
- Identify weekends versus weekdays

Source data tables are mature and ready for immediate use

IF YOU HAVE A TABLE AS SUCH, bring it in into your data model, and **USE IT**

DAX

The **DAX expression functions** CALENDARAUTO() and CALENDAR() can be used to **build a common date table**

CALENDAR()

this function **returns a contiguous range of dates based on a start and end data** that are entered as arguments in the function

An example of the Calendar function to print the dates between January 1st 2020 to December 31st 2020 is:

Calendar = CALENDAR(DATE(2020, 1, 1), DATE(2020, 12, 31))

CALENDARAUTO()

this function **returns a contiguous, complete range of dates that are automatically determined from your dataset**

The **starting date** is the earliest date that exists in the dataset

The **ending date** is the latest date that exists in the dataset **plus data that has been populated to the fiscal month that you can include as an argument** in the CALENDARAUTO() function

An example of the CalendarAuto function being used is:

Dates = CALENDARAUTO()

Dates Columns

Dates column can be quite sparse. You can use the **following DAX functions** to **see:**

All of these functions are entered after **clicking the New Column button** on the **Table Tools section**

- **the columns for just the year**

Year = YEAR(Dates[Date])

- **the month number**

Month Number = MONTH(Dates[Date])

- **the week of the year**

Week Number = WEEKNUM(Dates[Date])

- **the day of the week**

DayOfTheWeek = FORMAT(Dates[Date], “DDDD”)

Mark as the Official Table

In order to mark the table as the **official date table**, find the table in the **Fields pane**
=> **Right-click the name of the table**
=> **Select Mark as date table** in the following figure

By **marking this table as the official date table,** Power BI **performs validation** to **ensure that**:

- **data contains zero null values**
- **data is unique**
- **data contains continuous date values over a period**

Selecting **Mark as date table** will also **remove auto-generated hierarchies from the Date field**

You can also **manually add a hierarchy to the common data table** by **right-clicking the year, month, week, or day columns** in the **Fields pane** and **selecting New hierarchy**

Building a Visual with the Official Date Table

In order to **build a visual between two different tables using the Date table as the common table**
=> You need to first
=> **establish a relationship between this new common date table** and the **Sales and Orders tables**
=> This will **result in you being able to build visuals using the two tables**

An example statistic is **Total Sales and Order Quantity by Time** visual
The **DAX** used for this calculation is:

#Total Sales = SUM(Sales[‘Amount’])

After you have finished, you can **create a table** by returning to the **Visualizations tab** and **selecting the Table visual**

In order to e.g., see the total orders and sales by year and month.
You include:

- Year column
- Month column
- OrderQty column
- #TotalSales measure column

in the visualization

Working with Dimensions

Just a reminder, when building star schemas you will have **fact tables** and **dimension tables**

Fact tables

contains information about events such as sales orders, shipping dates, resellers, and suppliers

Dimension tables

store details about business entities, such as products or time, and are connected back to fact tables through a relationship

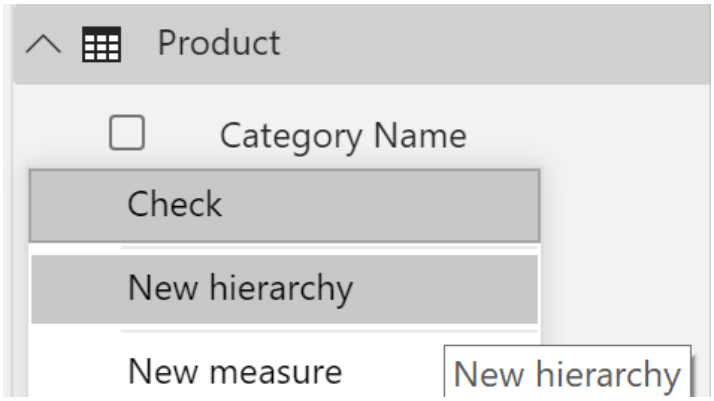
Hierarchies can be used as **one source to help you find detail in dimension tables**.
Form through natural segments in the data

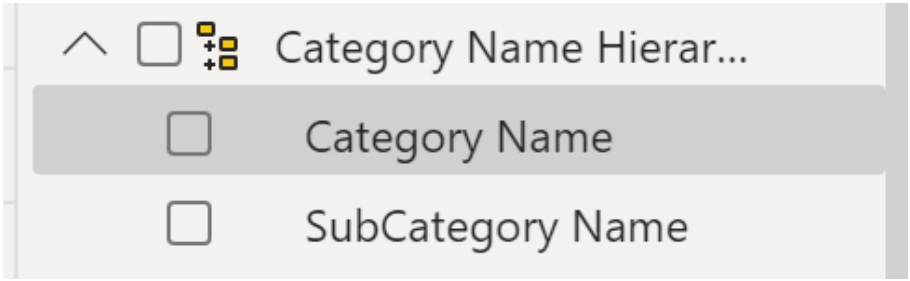
A **hierarchy of dates** can be **segmented into years, months, weeks, and days**

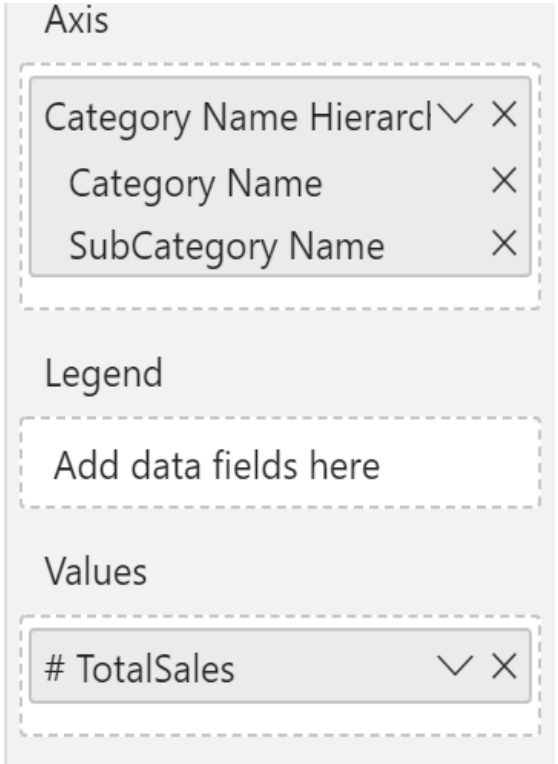
useful, as they **allow you to drill down into specifics of your data** instead of **seeing the data at a high level**

Hierarchies

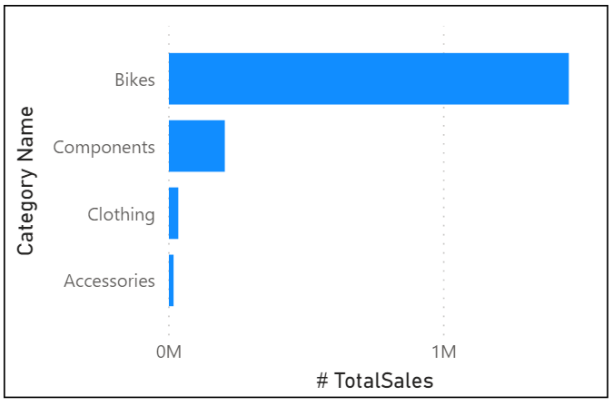
The following pictures are the steps taken to create a hierarchy

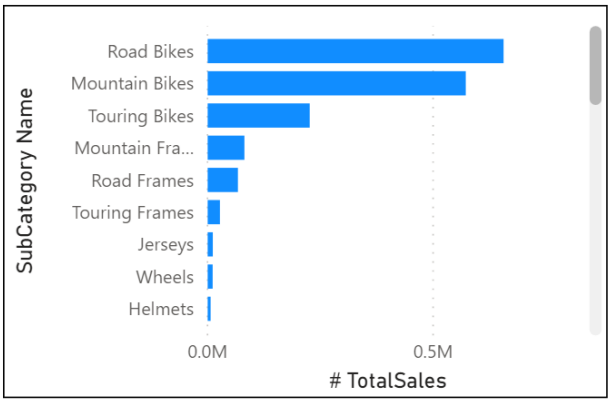






The result is then the **following visualziation**





Hence, **hierarchies allow us to view increasing levels of data on a single view**

Parent-Child Hierarchy

	Employee ID	Employee	Manager ID	Manager
1	1010	Roy F	null	
2	1011	Pam H	1010	Roy F
3	1012	Guy L	1010	Roy F
4	1013	Roger M	1011	Pam H
5	1014	Kaylie S	1011	Pam H
6	1015	Mike O	1012	Guy L
7	1016	Rudy Q	1012	Guy L

The above image is the scenario used to explain how parent-child hierarchy is used.

Roy F is continuously mentioned in the **Manager column**
this **implies that there is a hierarchy between the Employee** column and the **Manager** column

also, it is shown that **multiple employees can have the same manager**
this **implies that the MANAGER is the PARENT** and the **EMPLOYEE is the CHILD** in the **hierarchy**

We need to **flatten the hierarchy** in order to **view more data granularity related to the hierarchy**
basically each step of the hierarchy

Flattening the Parent-Child Hierarchy

The **process of viewing multiple child-levels based on a top-level parent** is known as **flattening the hierarchy**

you are **creating multiple columns in a table to show the hierarchical path of the parent to the child in the same record**

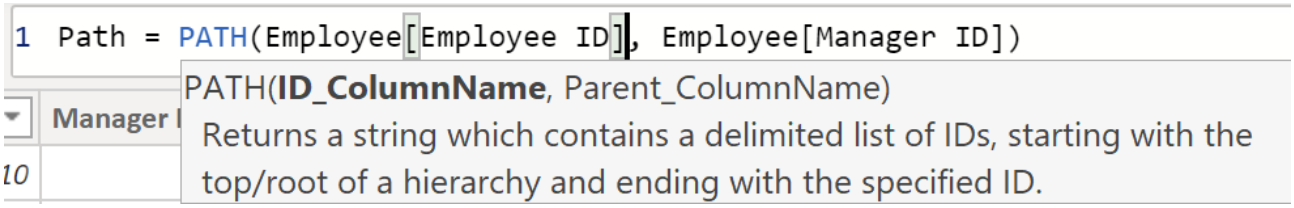
The following **DAX functions** will display the hierarchical path of the parent to the child**:**

- PATH() - a simple DAX function that **returns a text version of the managerial path for each employee**

ManagerEmployeePath = PATH('Employee'[EmployeeID], 'Employe'[ManagerID])

- PATHITEM() - a DAX function that can **separate this path into each level of managerial hierarchy**

An example of executing this DAX function is shown below.



Employee ID	Manager ID	Employee	Manager	Path
1010		Roy F		1010
1011	1010	Pam H	Roy F	1010|1011
1012	1010	Guy L	Roy F	1010|1012
1013	1011	Roger M	Pam H	1010|1011|1013
1014	1011	Kaylie S	Pam H	1010|1011|1014
1015	1012	Mike O	Guy L	1010|1012|1015
1016	1012	Rudy Q	Guy L	1010|1012|1016

The **Path results** are **read from left to right** where
- the **leftmost value** denotes the **parent**
- the **rightmost value** denotes the **child**

In order to **view all three levels of the hierarchy separately**, you **create four columns in the same way you did previously** by entering the following equations.

Using the PATHITEM function to **retrieve the value that resides in the corresponding level of your hierarchy**

- Level 1 = PATHITEM('Employee'[ManagerEmployeePath], 1]
- Level 2 = PATHITEM('Employee'[ManagerEmployeePath], 2]
- Level 3 = PATHITEM('Employee'[ManagerEmployeePath], 3]

The resulting table then becomes:

Employee ID	Name	Manager	Manager ID	Path	Level 1	Level 2	Level 3	Level 4
1000	Quincy Howard			1000	1000			
1001	Mallory Yang	Quincy Howard	1000	1000|1001	1000	1001		
1002	Donovan Maynard	Quincy Howard	1000	1000|1002	1000	1002		
1003	Giselle Mcclain	Mallory Yang	1001	1000|1001|1003	1000	1001	1003	
1004	Melvin Marsh	Mallory Yang	1001	1000|1001|1004	1000	1001	1004	
1005	Ria Snow	Giselle Mcclain	1003	1000|1001|1003|1005	1000	1001	1003	1005
1006	Callie Savage	Giselle Mcclain	1003	1000|1001|1003|1006	1000	1001	1003	1006

The **hierarchy has now been flattened** and we are now **able to view individual levels**

Role-Playing Dimensions

Role-playing dimensions **have multiple valid relationships with fact tables**, meaning that **the same dimension can be used** to **filter multiple columns**

As a result,

You can **filter data differently depending on what information you need to retrieve**

The above image displays how a role-playing dimension is expressed

the **Calendar table can be used to GROUP DATA in BOTH SALES AND ORDER**

If you wanted to **build a visual in which the Calendar table references the Order and Sales table**

the **Calendar table would act as a role-playing dimension**

Define Data Granularity

The **detail that is represented within your data**

meaning that **the more granularity your data has, the greater level of detail within your data**

Defining the **correct data granularity** can **have a big impact on the performance and usability of your Power BI reports and visuals**

An example scenario:
Tracking the temperature of 1,000 temperature controlled semi-trucks

How do you determine the **granularity of this data (level of detail)**?

- **Daily average temperature of each truck? (1,000 rows)**
- **Last recorded temperature (1,000 rows – last value of sensor)**
- **Temperature records that are above and below a normal range of temperatures** (could be 1,000+ or < 1,000 depending on the performance of the temperature control)

Determining **the level of granularity** is **dependent on the business case and scenario**

The **MAIN GOAL**:

- **Bring data** that is **comprehensive and valuable**
  - e.g., settle for **weekly, monthly, or quarterly data**.
  - The fewer the records, the faster the reports and visuals will respond
  - **Faster refresh rate** for the **entire dataset**, allows you to **refresh more frequently**

The **MAIN DRAWBACK**:

- If your **users want to drill into every single transaction**, summarizing the granularity will prevent them from doing that – a negative impact on the user experience

Important to **negotiate the level of data granularity with report users so they understand the implications of these choices**

Data Granularity to Build a Relationship between Two Tables

Differing levels of data granularity could lead to problems:

e.g., **lowest-level of time-based detail** in the **Sales tables** is **by day, e.g., 5/1/2020, 6/18/2020**
however,
the **lowest-level of time-based detail** in the **Budget tables** is **by month, e.g., 5/2020 and 6/2020**

These **different levels of granularities NEED TO BE RECONCILED before you can build a relationship between tables**

A solution could be:
Defining a new table column that matches with the granularity of the selected table

e.g., reformatting **two column headers (Year and Month Number) to a tertiary Date column that combines them** using **M**
an example **M query language could be**:

Column = Table.AddColumn(#”Renamed Columns”, “Custom”, each [Year] & “-” & [Month])

This fix should allow you to create a relationship between the Budget and Calendar tables

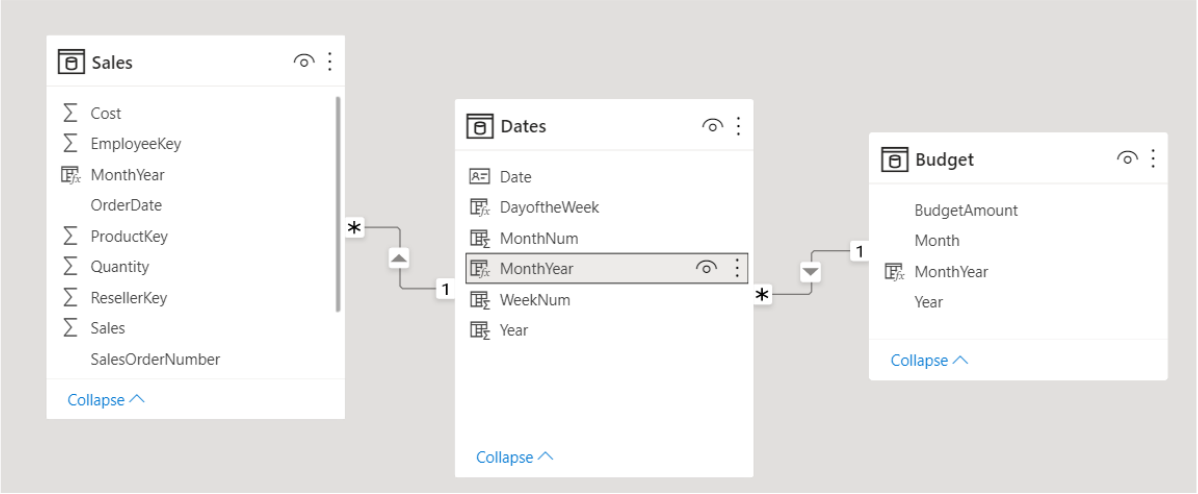
Create a Relationship between Tables

In general, **Power BI automatically detects relationships**, but you can also go to

Manage Relationships => **New**, and create the relationship on a **specified column between multiple tables**.

The **relationship should resemble the following**.

This also **ensures that the granularity is the same between the different tables**



Work With Relationships and Cardinality

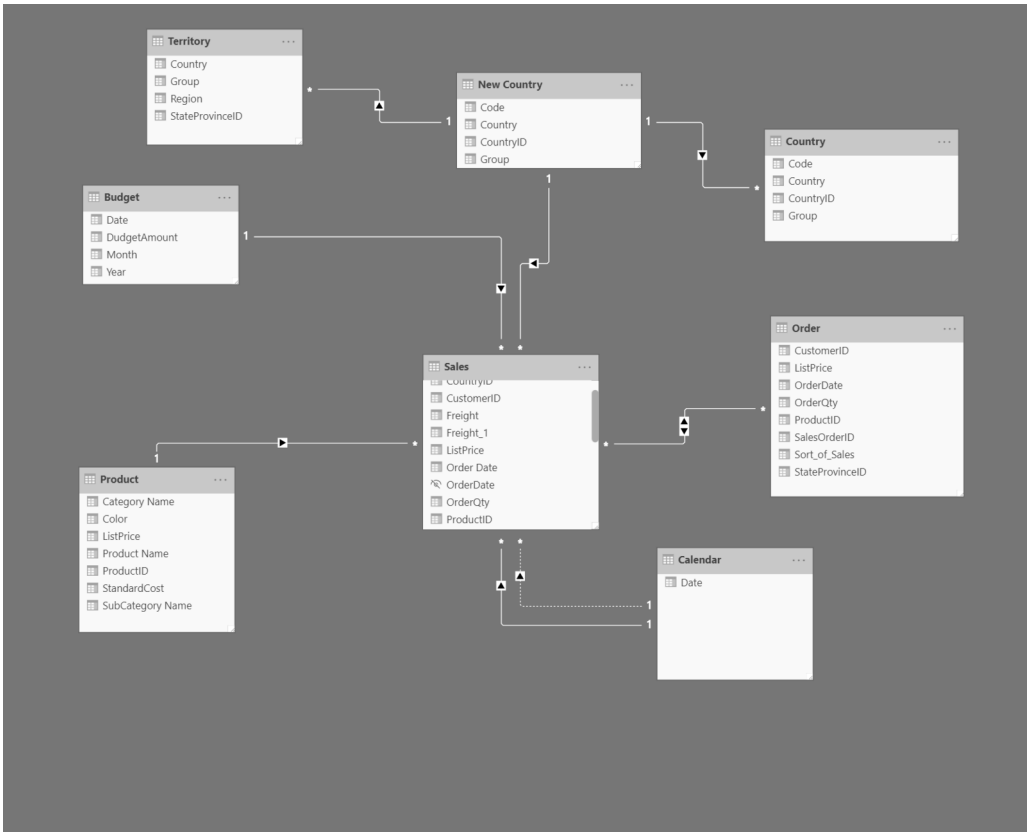
Power BI has the **concept of directionality to a relationship**

Directionality plays an important role in **filtering data between multiple tables**

Power BI **automatically looks for relationships that exist within the data by matching column names**

You can also use **Manage Relationships** to **edit these options manually**

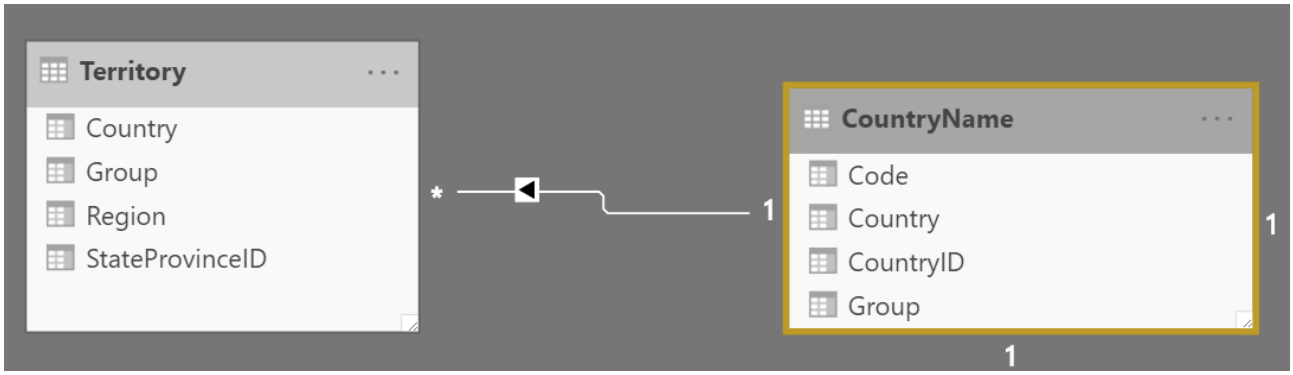
An example **visualizing the** different **auto-detected relationships**



Relationships

The following are different types of relationships in Power BI:

- ## Many-to-one (* : 1) and one-to-many (1 : *) relationships
  - A relationship where you have **many instances of a value in one column that are related to only one unique corresponding instance in another column**
  - Describes the **directionality between fact and dimension tables**
  - Most common type of directionality and is the Power BI default when automatically creating relationships



The above is an example of a one-to-many relationship.

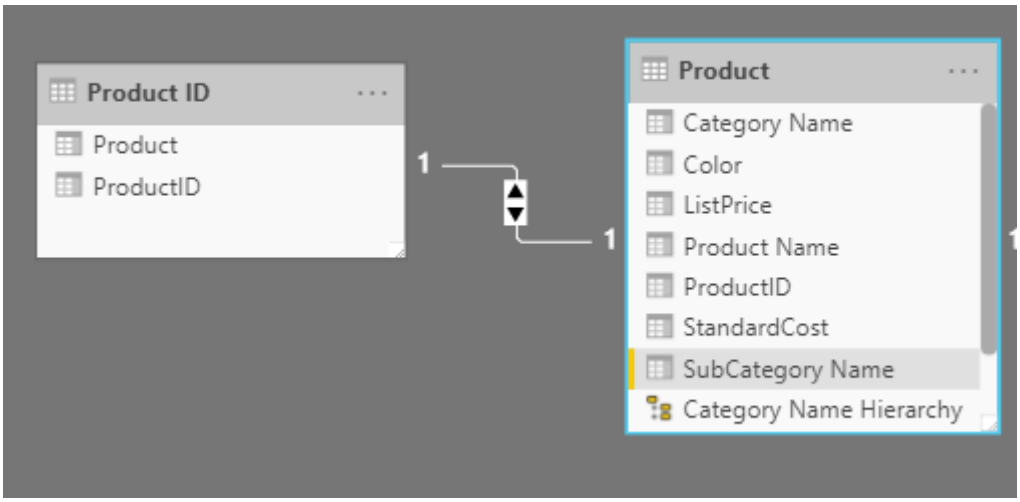
Where you can have **many territories being associated with one country**

- ## One-to-one (1 : 1) relationships

- a relationship where there are **only one instance of a value is common between two tables**
- **Requires unique values in both tables**

This is **NOT RECOMMENDED** because **this relationship stores redundant information** and **is not designed correctly**

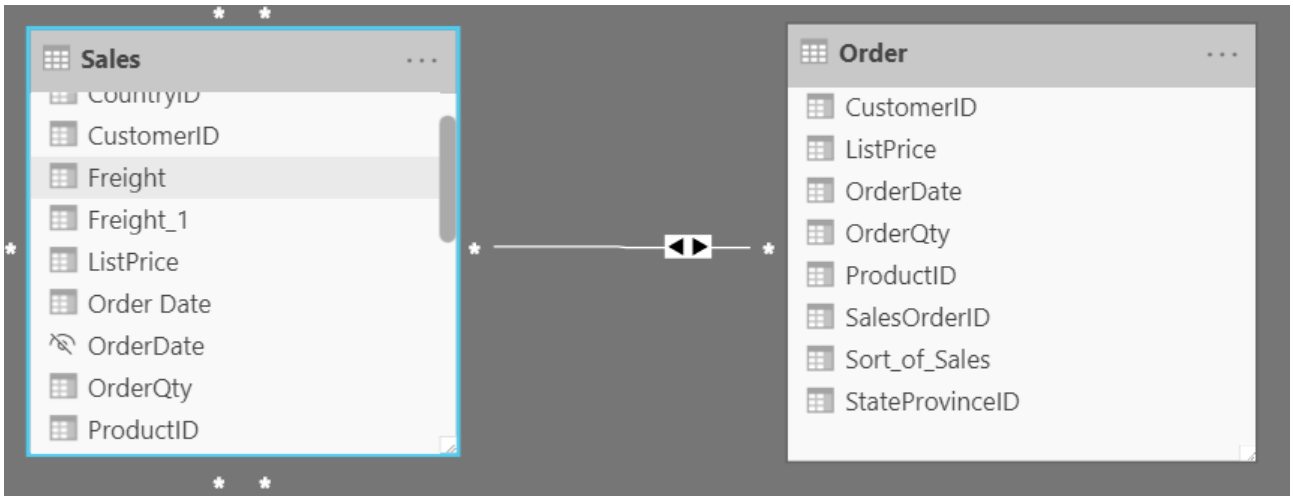
The ideal solution:
Merge/combine the two tables – as **both tables have unique values**



The above is an example of one-to-one relationship.
A product and productID table. This is **redundant** – **the table should just be combined**

- ## Many-to-many (.) relationships
  - A relationship where **many values are in common between two tables**
  - **Does not require unique values in either table**

NOT RECOMMENDED: a **lack of unique values** introduces **ambiguity** – users might not know which column of values is referring to what.



The above displays a many-to-many relationship between the **Sales table and the Order table** on the **OrderDate column**

This is **very ambiguous** as **both tables can have the same order date** – **duplicate / the same orders explored in both tables**

Cross-Filter Direction

Data **can be filtered on one or both sides of a relationship**

With a **single cross-filter direction**:

- Only one table in a relationship can **be used to filter the data**
  - e.g., **Table 1 can be filtered by Table 2**, but **Table 2 cannot be filtered by Table 1**

You **typically want these arrows** to **point to your fact table**

For a **one-to-many** or **many-to-one relationship**, the **cross-filter direction will be from "one" side** meaning that **the filtering will occur in the table that has many values**

Basically the filter propagate **from the "one side" to the "many side"**,
in an example case, filters applied **to the Products table** will **propagate to the Sales table**
but NOT in the opposite direction

Cross-Filter directions or Bi-directional Cross-Filtering

An example:

One table in a relationship can be used to filter the other. A **dimension table can be filtered through the fact table** and the **fact table**

Hard to do, try to avoid this if you are unsure of what to do

Resolve Modelling Challenges

Modelling data is about **establishing and maintaining relationships** so that **you can effectively visualize the data** in the **form that your business requires**

When creating relationships, a common pitfall is circular relationships

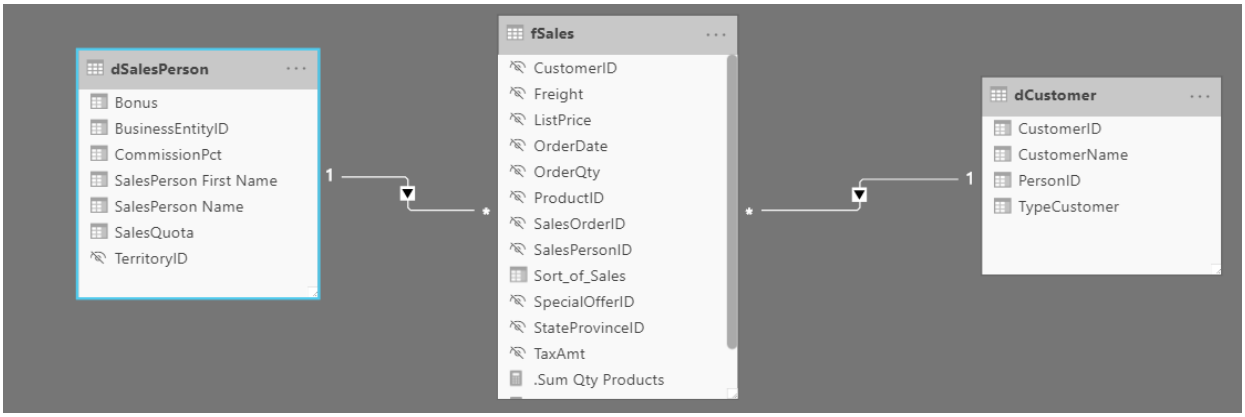
e.g.,
Table 1 has a **many-to-one relationship** with a **column in Table 2**, but **Table 2** has a **one-to-many relationship** with **Table 3** that **ALREADY HAS** its own **relationship with Table 1**

Web of relationship is difficult to manage and becomes a daunting task to build visuals. As it is no longer clear **what relationship exists**.

Identify circular relationships so that **data is usable**

Relationship Dependencies

Dependencies on **either column values or separate tables** will **lead to changes in corresponding queries and calculations** if the **column / table that it was dependent on were to be changed**



Consider the above relationship.
As,

Sales[‘TotalCost’] = Sales[‘Quantity’]*Sales[‘Price’]

TotalCost depends on **Quantity and Price**
hence,
If **a change occurs** in **either quantity or price** then **a change will also occur in TotalCost aswell**

Quick-Measures***

A quick measure creates the calculation formula for you

- **Easy**
- **Fast**

to create **for simple and common calculations**

You can also **create basic measures (calculations using DAX) on columns easily**.

You can do this by, in the **Data view**, **select a table in the Field table**
=> **Right-click the selected table** and **select new quick measure**

You can select a **range of basic calculations/measures** that include:

- Totals
- Time intelligence
- Filters
- Aggregate per category

calculations

ALL OF THE ABOVE CALCULATIONS WITHOUT USING DAX

Bulk Update Properties

You are able to **update multiple columns using single bulk updates**

You can use this to

- **hide columns**
- **format column values**

Hidden Columns

Can be because **they’re used by relationships** or **will be used in row-level security configuration** or **calculation logic**