DAX introduction

avanade

Agenda

Day 1

DAX Basics
Table Functions
Evaluation contexts
Calculate
Evaluation contexts and relationships
Iterators

Day 2

Querying tabular and table functions
Advanced Filter context
Advanced relationship (M2M)
Use Cases
Links (Patterns) / DAX 2016
M language

DAX Basics

What is DAX?

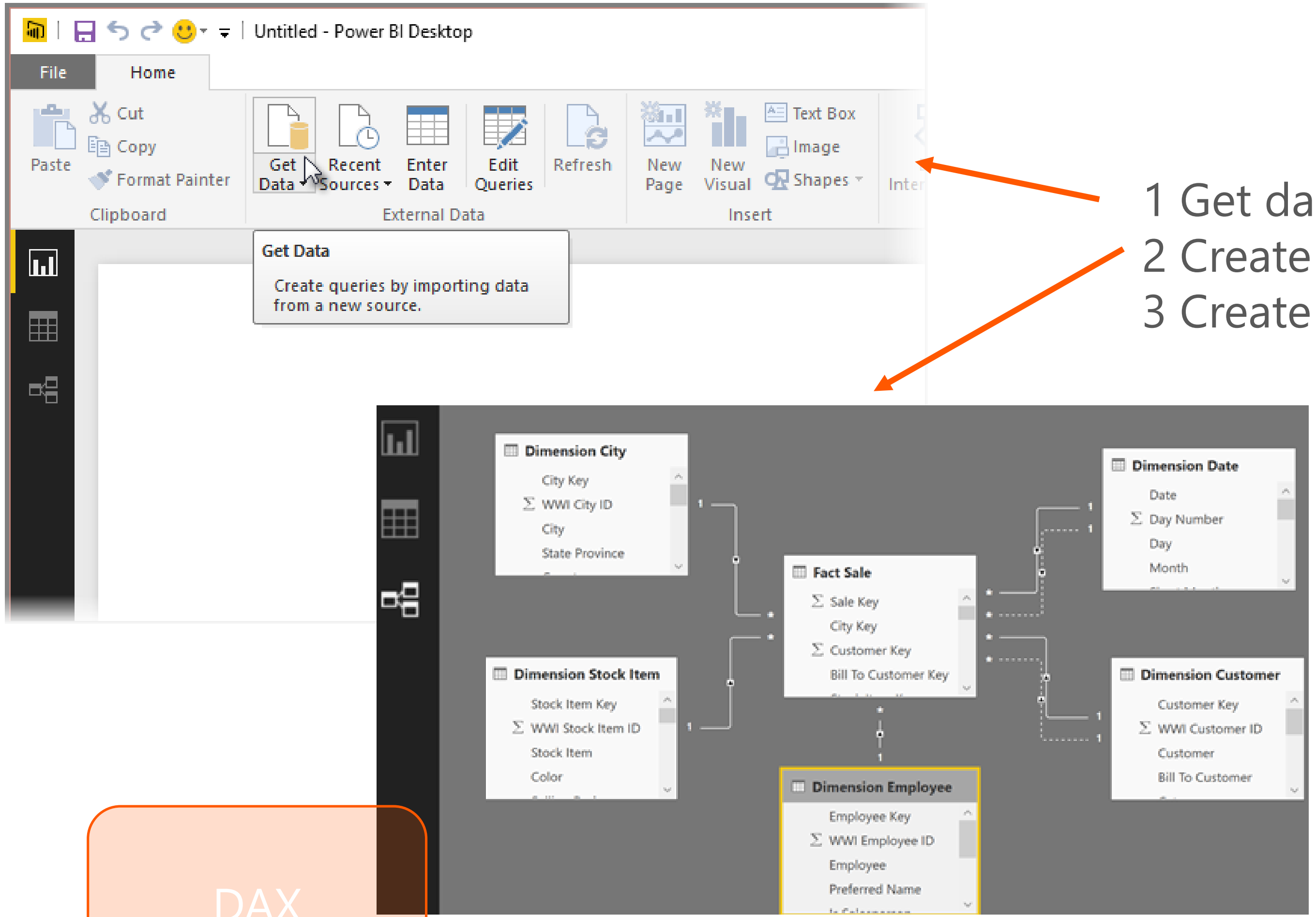
- Data Analysis Expressions (DAX) language is a formula language that allows users to define custom calculations in calculated columns and measures
- DAX is used for tabular models (Power Pivot, SSAS, Power BI)
- Similar to Excel programming
- New concepts : evaluation contexts

Formula Language

- DAX looks like that :

```
SalesOfCar :=
CALCULATE (
        SUM (Sales[Amount]) ,
        Product[Category] = “Car”
)
```

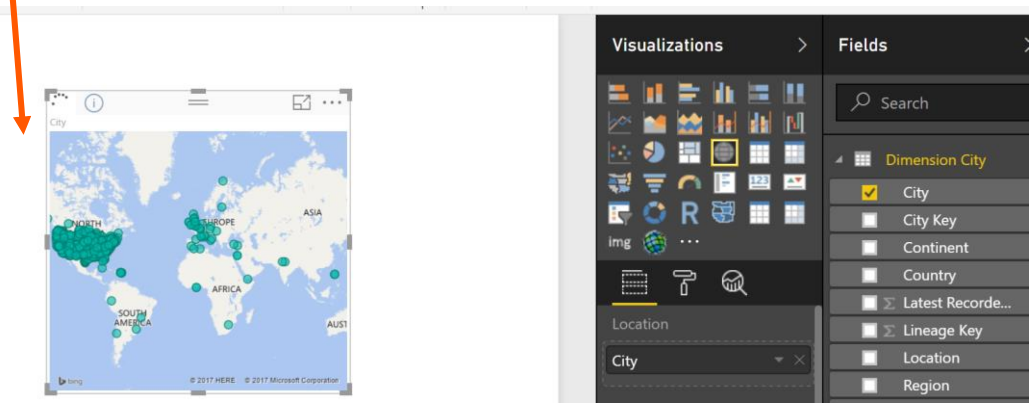
DAX & Power BI



Get Data

Create queries by importing data from a new source.

1 Get data
2 Create a model
3 Create reports



DAX

DAX Types

- Numeric types
  - Integer (64 bit)
  - Decimal (floating point)
  - Currency (money)
  - Date (DateTime)
  - TRUE / FALSE (Boolean)

- Other types
  - String
  - Binary Objects

DAX Syntax

- Table names must be unique within the database.
- Table names must be enclosed in single quotation marks if they contain spaces, other special characters or any non-English alphanumeric characters

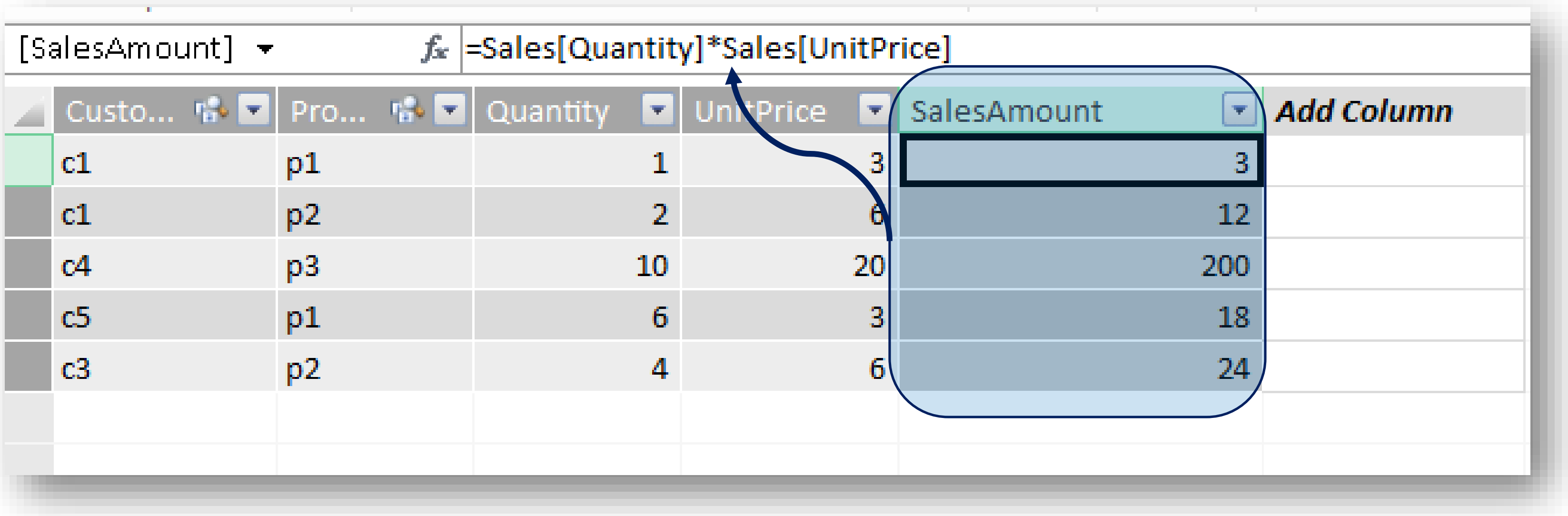
```
Customer[Name]  or ‘Product Category’[Name]
```

- Measure names must always be in brackets
- Measure names can contain spaces
- Each measure name must be unique within a database

```
Sales[Amount] or [Amount]
```

Calculated columns

- Always computed for the current row
- Linked to a specific table

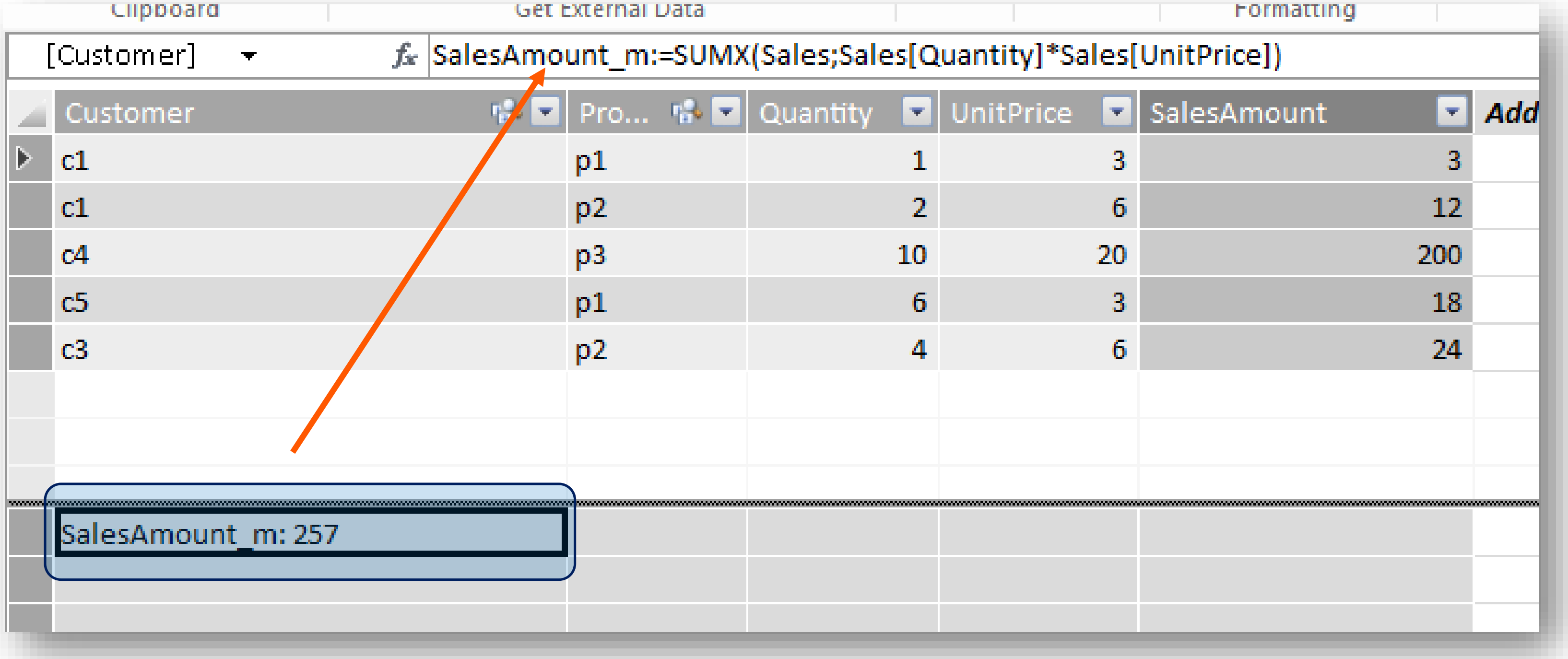


[SalesAmount] ▾ *fx* =Sales[Quantity]*Sales[UnitPrice]

Custo...	Pro...	Quantity	UnitPrice	SalesAmount	*Add Column*
c1	p1	1	3	3	
c1	p2	2	6	12	
c4	p3	10	20	200	
c5	p1	6	3	18	
c3	p2	4	6	24	

Measures

- Are not linked to a specific table
- Use tables and aggregators
- Do not have the «current row» concept



```
SalesAmount_m:=SUMX(Sales;Sales[Quantity]*Sales[UnitPrice])
```

Customer	Pro...	Quantity	UnitPrice	SalesAmount
c1	p1	1	3	3
c1	p2	2	6	12
c4	p3	10	20	200
c5	p1	6	3	18
c3	p2	4	6	24

SalesAmount_m: 257

Measures vs Calculated Columns

- Use a column when
  - Need to slice or filter on the value

- Use a measure
  - Calculate percentages / ratios
  - Need complex aggregations

- Space and CPU usage
  - Columns consume memory = stored in data model
  - Measures consume CPU = calculated on the fly

Aggregation Functions

- Classical aggregation functions
  - SUM
  - AVERAGE
  - MIN
  - MAX

- Iterative equivalent, to calculate row by row :
  - SUMX, AVERAGEX, MINX, MAXX

```
SUMX (
        Sales,
        Sales[UnitPrice] * Sales[Quantity]
)
```

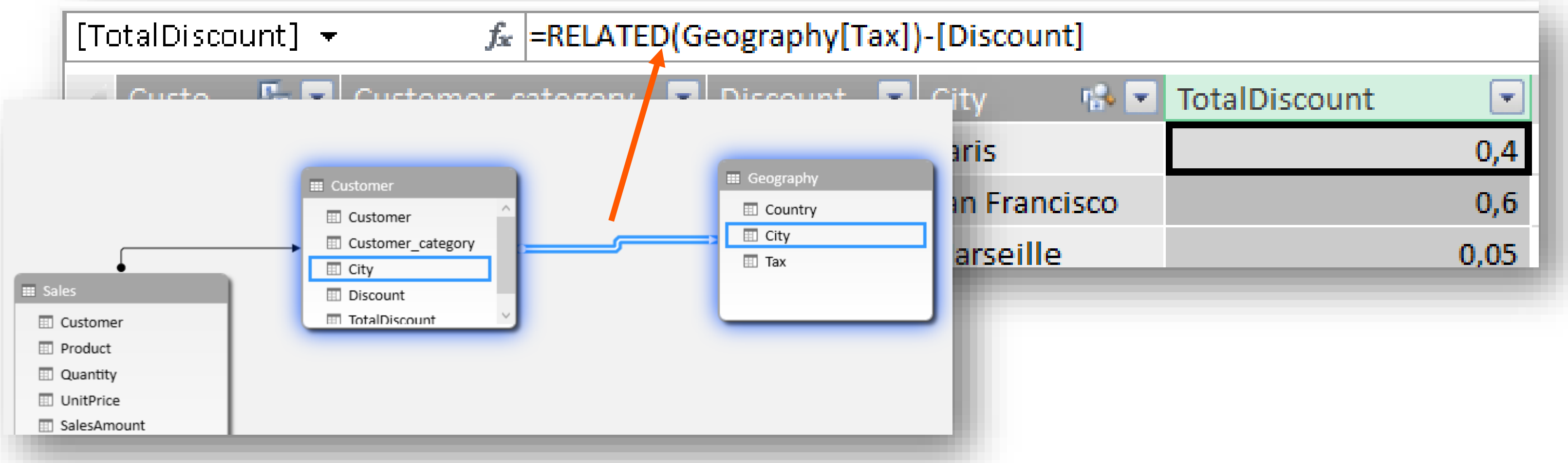
Functions

- Counting Values
- Mathematical Functions
- DIVIDE
- Error handling
- Logical Functions
- …

DAX Function Reference

Relational Functions

This functions work with relationship



```
[TotalDiscount]   =RELATED(Geography[Tax])-[Discount]
```

- RELATED

Returns column from an associated table

- RELATEDTABLE

Returns an associated table

Exercices

30 minutes

1. Data Presentation

2. Model Presentation

3. Calculated Columns and Measures
- Create a SalesAmount calculated column (quantity*price)
- Create a TotalSales Measure (using the previously created SalesAmount calculated column)

4. The «X» Aggregation Functions
- Modify the TotalSales measure avoiding the use of the Sales Amount calculated column

5. Relationnal Functions
- Create a DiscountedUnitPrice calculated column

TIP: as the Discount information is in the Customer table, you should consider using the RELATED function

Table Functions

Table Functions

- Allow you to create subsets of data

- Basic Functions:
  - ALL
  - VALUES
  - FILTER
  - DISTINCT
  - RELATEDTABLE

Each one requires a table
Each one returns a table

- Often used as a parameter in other functions.

  →Table functions can be used in Table functions!

FILTER

- Used to filter a table

- Returns a table with a restricted number of rows according to a condition (Expression)

- Inputs are one table and one expression

```
SUMX (
       FILTER (
              Sales,
              Sales[UnitPrice] > 1
       ),
       Sales[Price] * Sales[Quantity]
)
```

ALL

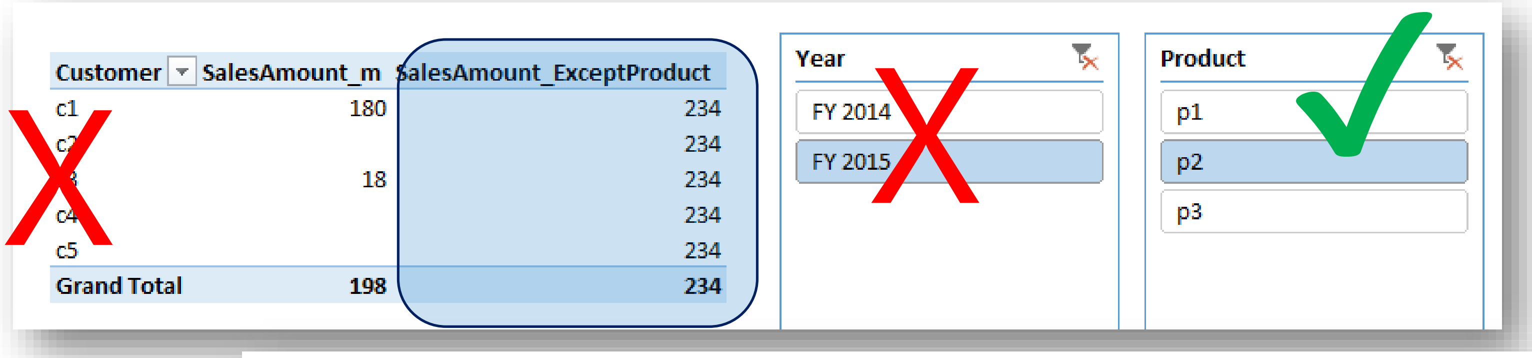
- Returns all rows of the input table
- Ignores the filter context
- ALL is useful for clearing filters and creating calculations on all the rows in a table
- ALL can be used with a single column to return all the values from this column

```
FILTER ( ALL ( Table ), Condition )
```

ALLEXCEPT

- Removes all context filters in the table except filters that have been applied to the specified columns

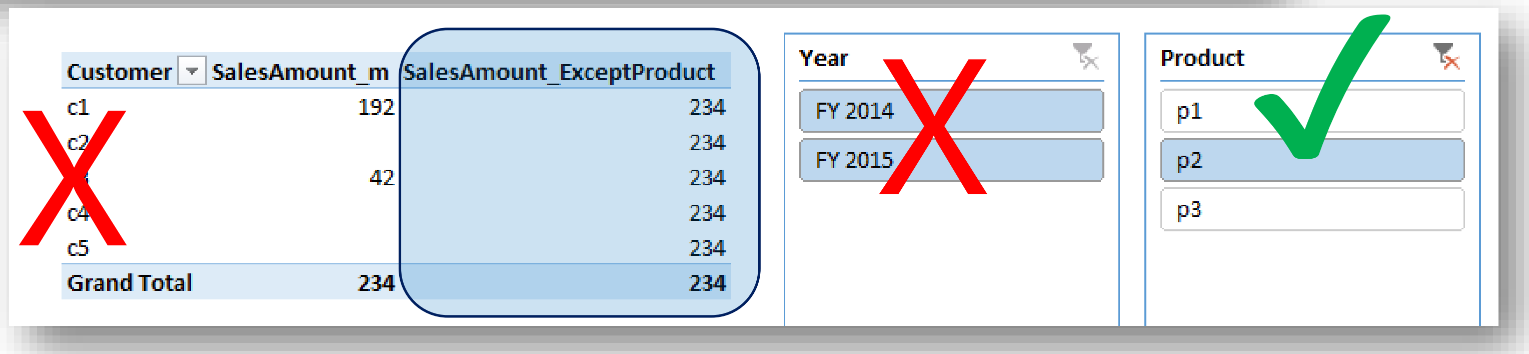
```
ALLEXCEPT ( Sales, Sales[Product] )
```



Customer	SalesAmount_m	SalesAmount_ExceptProduct
c1	180	234
c2		234
c3	18	234
c4		234
c5		234
Grand Total	198	234

Year ✗
- ☐ FY 2014
- ☑ FY 2015

Product ✓
- ☐ p1
- ☑ p2
- ☐ p3

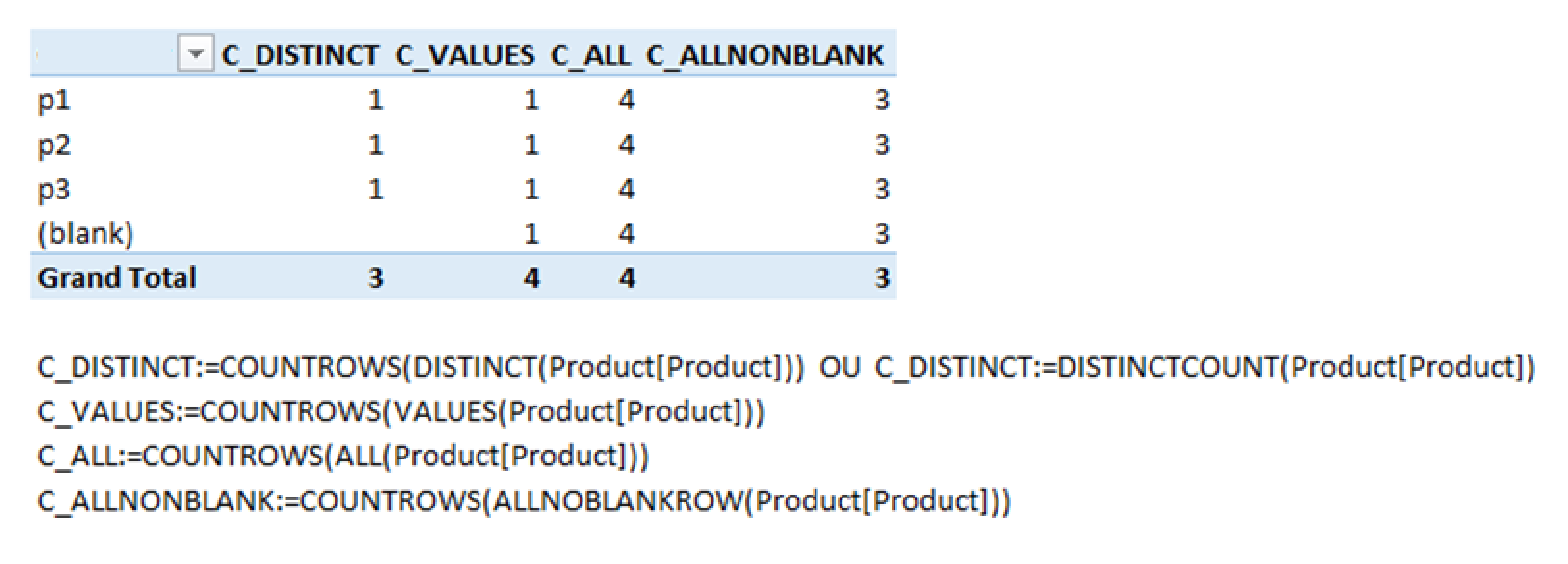


Customer	SalesAmount_m	SalesAmount_ExceptProduct
c1	192	234
c2		234
c3	42	234
c4		234
c5		234
Grand Total	234	234

Year ✗
- ☑ FY 2014
- ☑ FY 2015

Product ✓
- ☐ p1
- ☑ p2
- ☐ p3

Counting different values



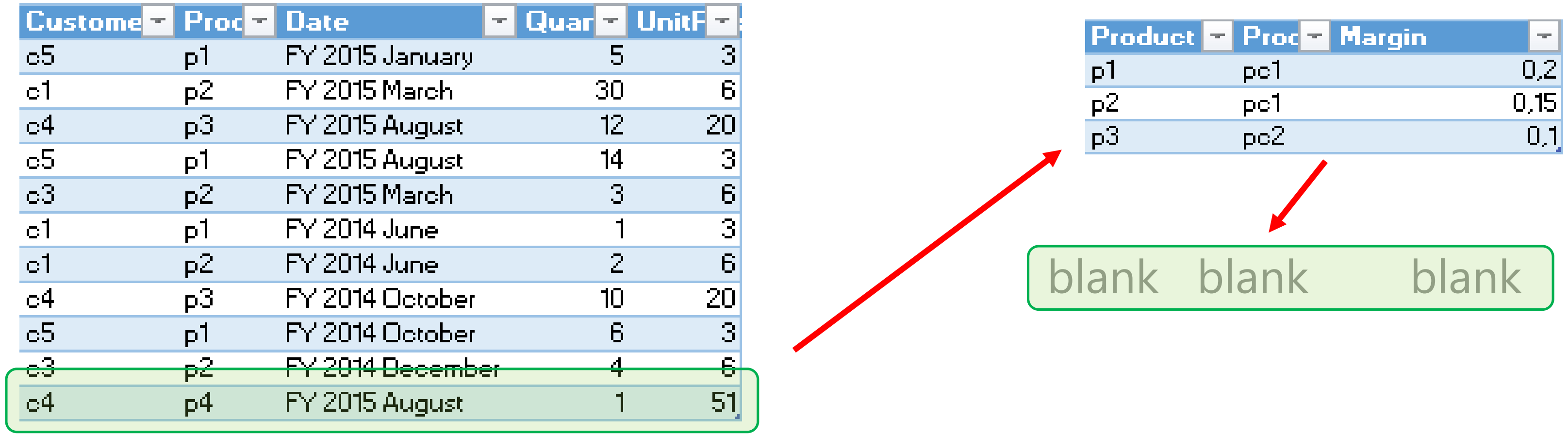
	C_DISTINCT	C_VALUES	C_ALL	C_ALLNONBLANK
p1	1	1	4	3
p2	1	1	4	3
p3	1	1	4	3
(blank)		1	4	3
Grand Total	**3**	**4**	**4**	**3**

```
C_DISTINCT:=COUNTROWS(DISTINCT(Product[Product])) OU C_DISTINCT:=DISTINCTCOUNT(Product[Product])
C_VALUES:=COUNTROWS(VALUES(Product[Product]))
C_ALL:=COUNTROWS(ALL(Product[Product]))
C_ALLNONBLANK:=COUNTROWS(ALLNOBLANKROW(Product[Product]))
```

- Let us show you why…
- (BLANK) ?

Counting different values

- When your Sales table refers to an unknown product in Product table, DAX creates a blank row in the referred table.



Custome	Proc	Date	Quar	UnitP
c5	p1	FY 2015 January	5	3
c1	p2	FY 2015 March	30	6
c4	p3	FY 2015 August	12	20
c5	p1	FY 2015 August	14	3
c3	p2	FY 2015 March	3	6
c1	p1	FY 2014 June	1	3
c1	p2	FY 2014 June	2	6
c4	p3	FY 2014 October	10	20
c5	p1	FY 2014 October	6	3
c3	p2	FY 2014 December	4	6
c4	p4	FY 2015 August	1	51

Product	Proc	Margin
p1	pc1	0,2
p2	pc1	0,15
p3	pc2	0,1
blank	blank	blank

Exercices

30 minutes

1. **COUNTROWS**

- Create a measure that count sales and use it in a PivotTable to report sales by customer

… FILTER & COUNTROWS

- Create a measure that count only sales with quantity>5 and use it in a PivotTable to report sales with quantity>5 by customer

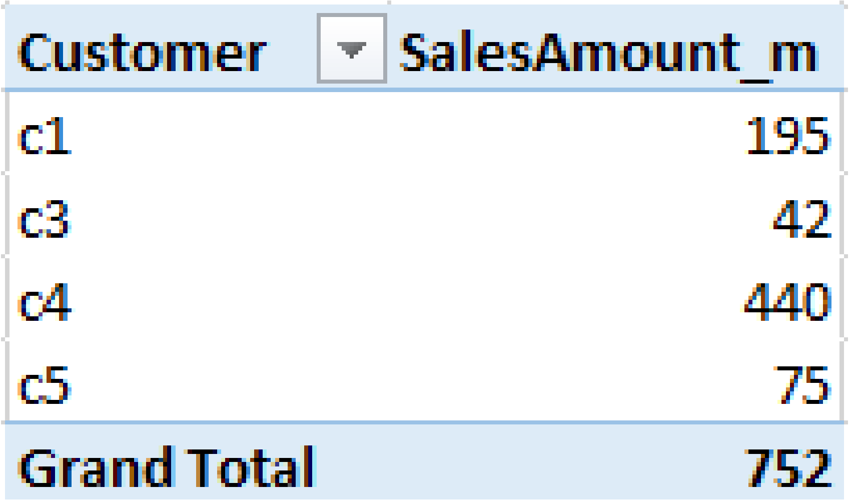
2. RELATEDTABLE

- Create a measure that count the number of product that have never been sold

Evaluation Contexts

Evaluation contexts

```
SalesAmount_m:=SUMX(
        Sales;
        Sales[Quantity]*Sales[UnitPrice]
)
```

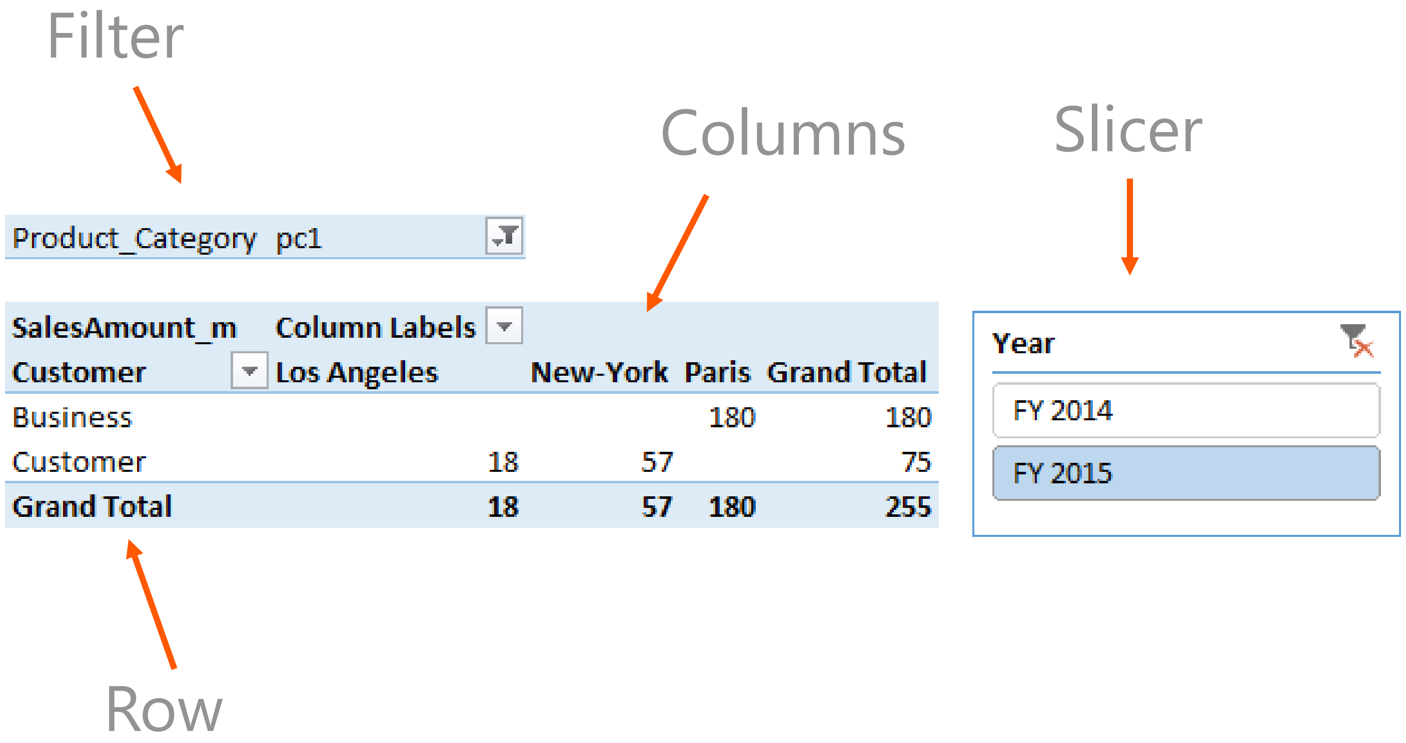


Customer	SalesAmount_m
c1	195
c3	42
c4	440
c5	75
Grand Total	752

The measure is calculated for each Customer.

The value of the measure depends on the context.

Evaluation context sources



Filter

Columns

Slicer

Row

Product_Category pc1

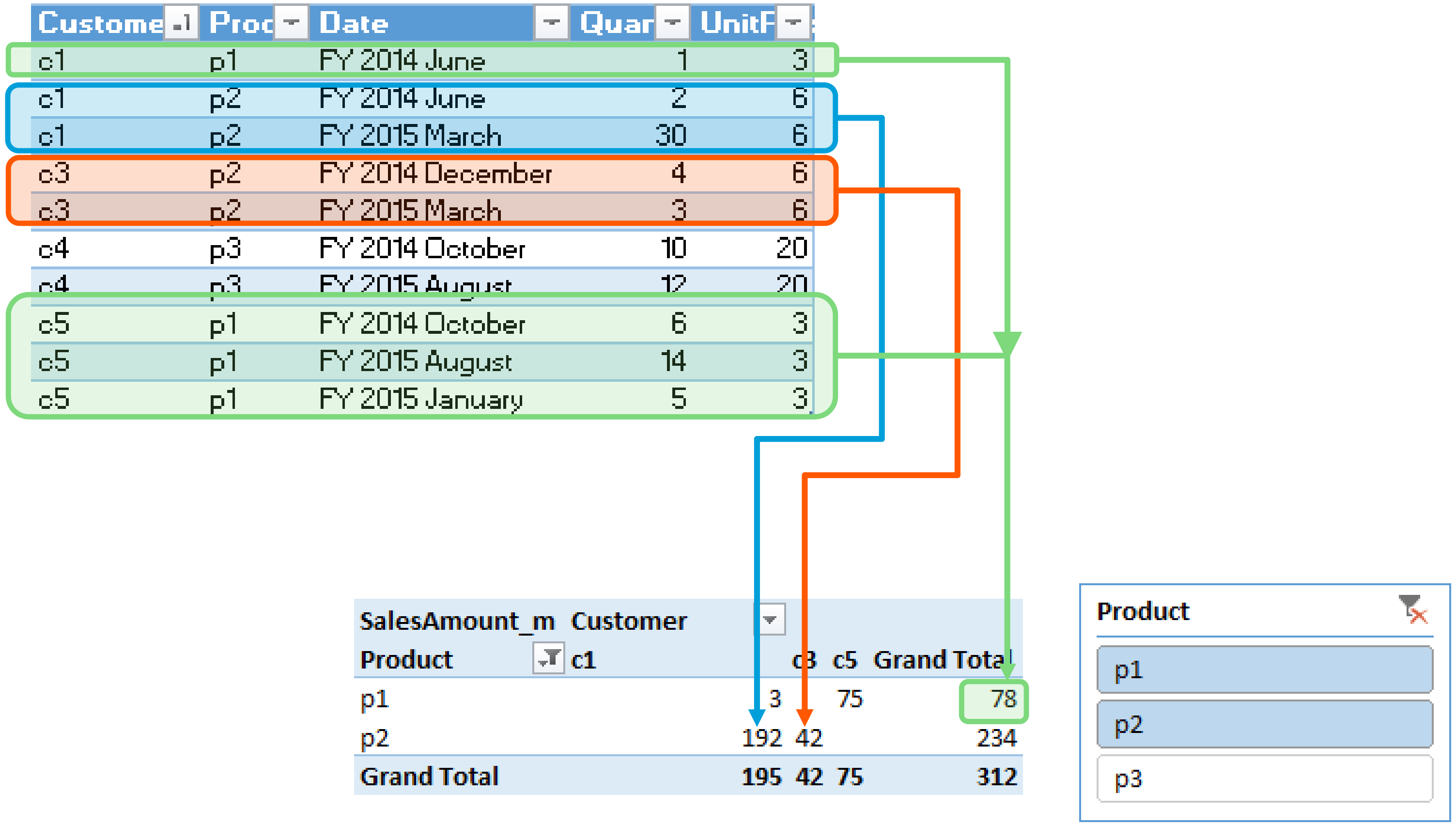
SalesAmount_m	Column Labels			
Customer	Los Angeles	New-York	Paris	Grand Total
Business			180	180
Customer	18	57		75
Grand Total	18	57	180	255

Year

FY 2014

FY 2015

Evaluation context explanation



Custome	Proc	Date	Quar	UnitF
c1	p1	FY 2014 June	1	3
c1	p2	FY 2014 June	2	6
c1	p2	FY 2015 March	30	6
c3	p2	FY 2014 December	4	6
c3	p2	FY 2015 March	3	6
c4	p3	FY 2014 October	10	20
c4	p3	FY 2015 August	12	20
c5	p1	FY 2014 October	6	3
c5	p1	FY 2015 August	14	3
c5	p1	FY 2015 January	5	3

SalesAmount_m	Customer			
Product	c1	c3	c5	Grand Total
p1	3		75	78
p2	192	42		234
Grand Total	**195**	**42**	**75**	**312**

Filter & Row context

Filter context

- Filter context defined by:
  - Row
  - Column
  - Filters
  - Slicers

  With PivotTable or functions !

- Rows outside the filter context are ignored

Row context

- Row context defined by:
  - Calculated column definition
  - Iteration functions («X»)

- Simply the current row !

Classical context error

```
=Sales[Quantity]*Sales[UnitPrice]
```

- In a calculated column: OK because of the row context

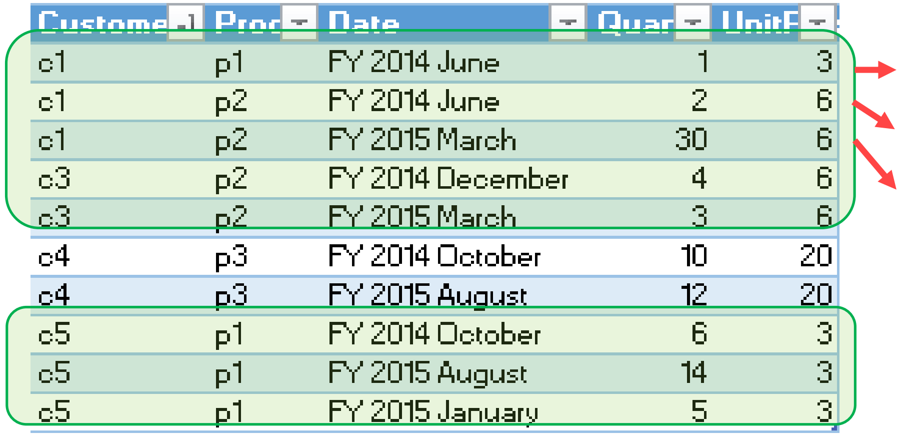
- In a measure: KO !

  →Can’t be determined in the current context

  →You have to use an iteration function (“X”) to create a row context

Evaluation contexts

```
SalesAmount_m:=SUMX(
        Sales;
        Sales[Quantity]*Sales[UnitPrice]
)
```



Custome	Proc	Date	Quar	UnitF
c1	p1	FY 2014 June	1	3
c1	p2	FY 2014 June	2	6
c1	p2	FY 2015 March	30	6
c3	p2	FY 2014 December	4	6
c3	p2	FY 2015 March	3	6
c4	p3	FY 2014 October	10	20
c4	p3	FY 2015 August	12	20
c5	p1	FY 2014 October	6	3
c5	p1	FY 2015 August	14	3
c5	p1	FY 2015 January	5	3

1x3 =3

2x6 =12

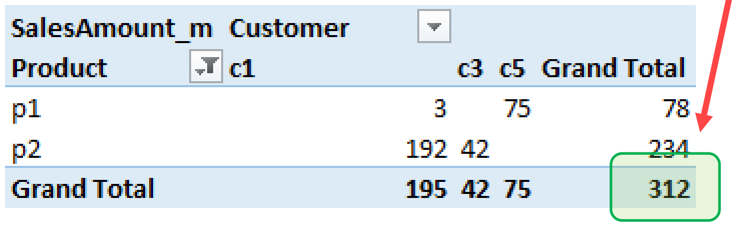
30x6 =180

… …

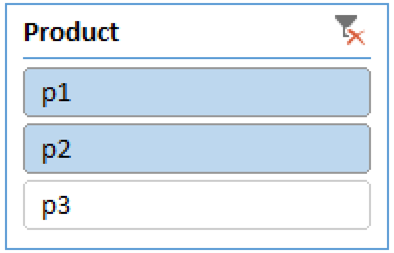
… …

… …

=312



SalesAmount_m	Customer			
Product	c1	c3	c5	Grand Total
p1	3		75	78
p2	192	42		234
Grand Total	195	42	75	312



Product

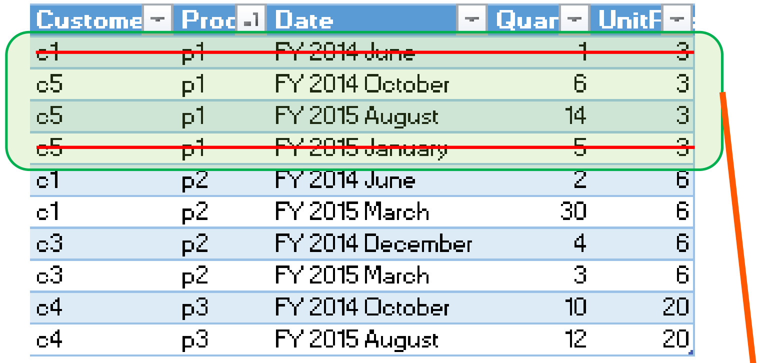
p1

p2

p3

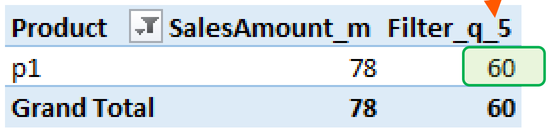
Modifying the evaluation context

- You can modify the evaluation context in two ways:
  - Using the PivotTable (already demonstrated)
  - With formulas:

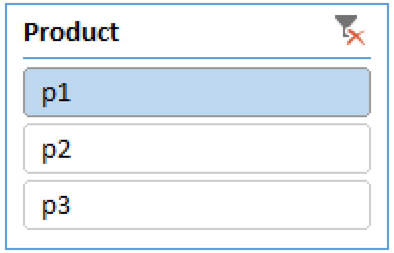


Custome	Proc	Date	Quar	UnitF
~~c1~~	~~p1~~	~~FY 2014 June~~	~~1~~	~~3~~
c5	p1	FY 2014 October	6	3
c5	p1	FY 2015 August	14	3
~~c5~~	~~p1~~	~~FY 2015 January~~	~~5~~	~~3~~
c1	p2	FY 2014 June	2	6
c1	p2	FY 2015 March	30	6
c3	p2	FY 2014 December	4	6
c3	p2	FY 2015 March	3	6
c4	p3	FY 2014 October	10	20
c4	p3	FY 2015 August	12	20

```
Filter_q_5:=SUMX(
        FILTER(
                Sales;
                Sales[Quantity]>5
        );
        Sales[Quantity]*Sales[UnitPrice]
)
```



Product	SalesAmount_m	Filter_q_5
p1	78	60
Grand Total	78	60

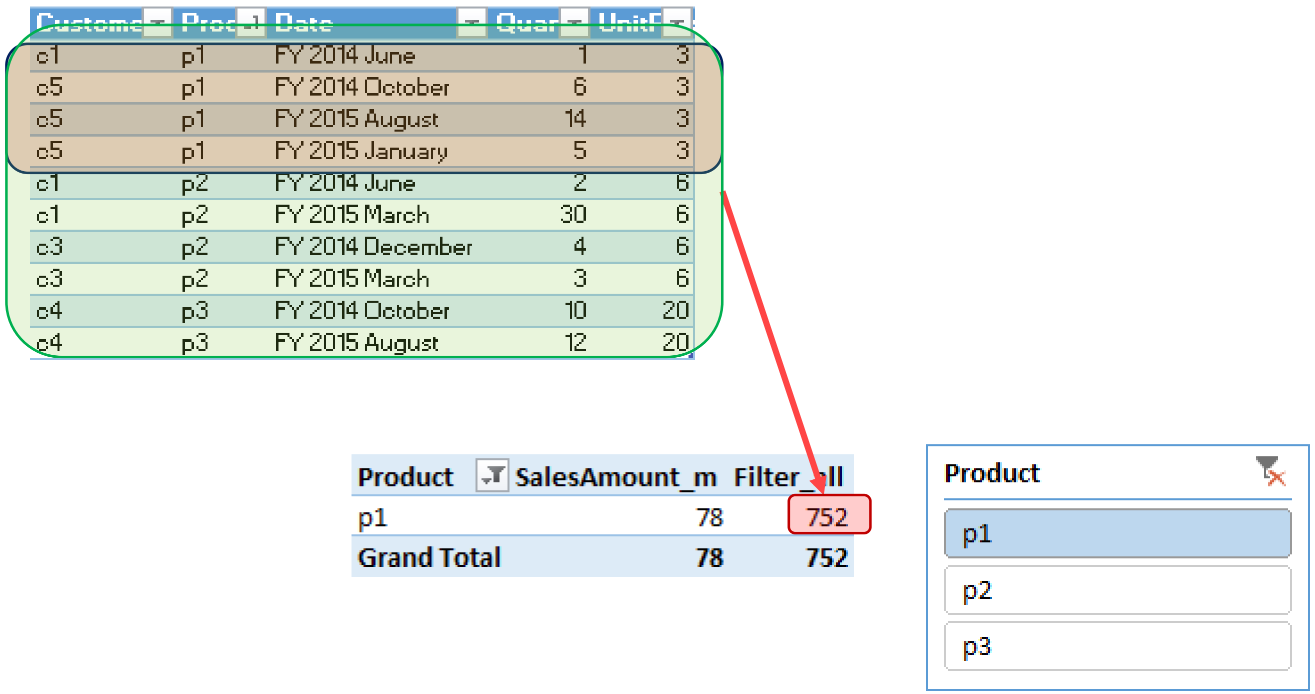


Product

- p1
- p2
- p3

Modifying the evaluation context

```
Filter_all:=SUMX(
        ALL(Sales);
        Sales[Quantity]*Sales[UnitPrice]
)
```



Custome	Proc	Date	Quar	UnitP
c1	p1	FY 2014 June	1	3
c5	p1	FY 2014 October	6	3
c5	p1	FY 2015 August	14	3
c5	p1	FY 2015 January	5	3
c1	p2	FY 2014 June	2	6
c1	p2	FY 2015 March	30	6
c3	p2	FY 2014 December	4	6
c3	p2	FY 2015 March	3	6
c4	p3	FY 2014 October	10	20
c4	p3	FY 2015 August	12	20

Product	SalesAmount_m	Filter_all
p1	78	752
Grand Total	**78**	**752**

Product

- p1
- p2
- p3

More about Evaluation Context

https://www.sqlbi.com/articles/row-context-and-filter-context-in-dax/

CALCULATE Function

CALCULATE

- Evaluates an expression in a context that is modified by the specified filters

```
=CALCULATE(<expression>;<Filter1>;<Filter2>;…)
```

```
SalesAmount_gt10:=
CALCULATE(
    SUM(Sales[SalesAmount]);
    Sales[Quantity]>10
)
```

CALCULATE- Filter

- Filters in CALCULATE are transformed in a complete FILTER function

```
SalesAmount_gt10:=
CALCULATE(
    SUM(Sales[SalesAmount]);
    Sales[Quantity]>10
)
```

=

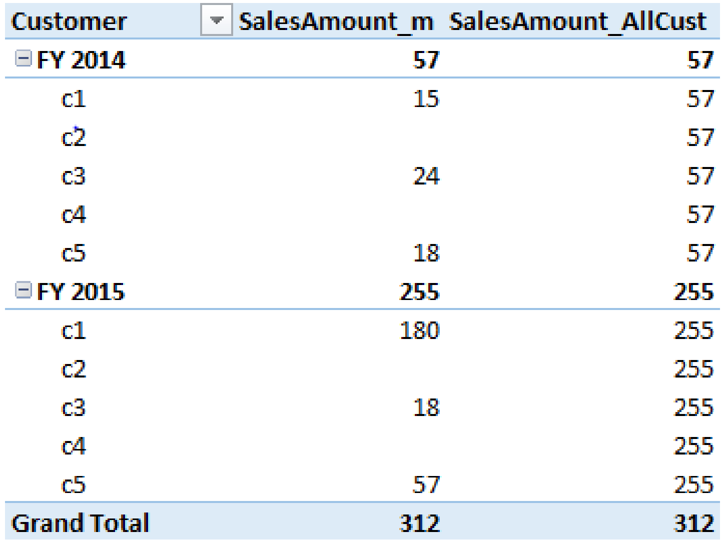
```
SalesAmount_gt10:=
CALCULATE(
    SUM(Sales[SalesAmount]);
    FILTER(ALL(Sales);Sales[Quantity]>10)
)
```

avanade

CALCULATE – Removing filters

- You can remove a filter on Customer to calculate the total for all the customers.

```
SalesAmount_AllCust:=CALCULATE(
    SUM(Sales[SalesAmount]);
    ALL(Customer)
)
```

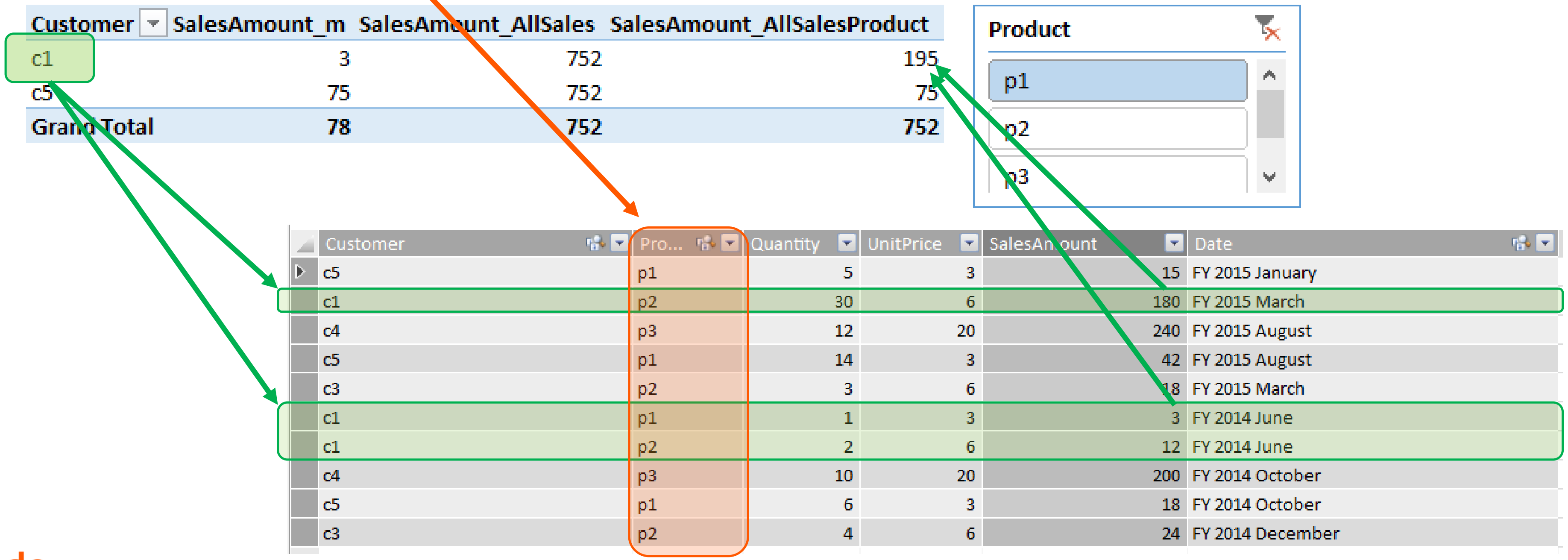


Customer	SalesAmount_m	SalesAmount_AllCust
⊟ FY 2014	**57**	**57**
c1	15	57
c2		57
c3	24	57
c4		57
c5	18	57
⊟ FY 2015	**255**	**255**
c1	180	255
c2		255
c3	18	255
c4		255
c5	57	255
Grand Total	**312**	**312**

CALCULATE – Removing filters

- Specifying a column in the ALL function, you'll remove the filter on this column but keep the other filters.

```
SalesAmount_AllSalesProduct:=CALCULATE(
        SUM(Sales[SalesAmount]);
        ALL(Product[Product])
)
```

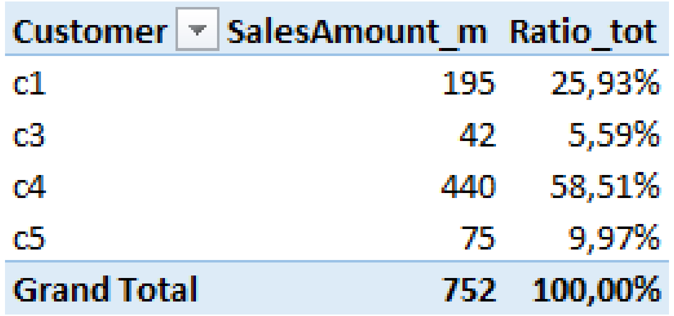


Customer	SalesAmount_m	SalesAmount_AllSales	SalesAmount_AllSalesProduct
c1	3	752	195
c5	75	752	75
Grand Total	**78**	**752**	**752**

Customer	Pro...	Quantity	UnitPrice	SalesAmount	Date
c5	p1	5	3	15	FY 2015 January
c1	p2	30	6	180	FY 2015 March
c4	p3	12	20	240	FY 2015 August
c5	p1	14	3	42	FY 2015 August
c3	p2	3	6	18	FY 2015 March
c1	p1	1	3	3	FY 2014 June
c1	p2	2	6	12	FY 2014 June
c4	p3	10	20	200	FY 2014 October
c5	p1	6	3	18	FY 2014 October
c3	p2	4	6	24	FY 2014 December

CALCULATE – Removing filters use case

- You need to calculate a ratio over the total sales.

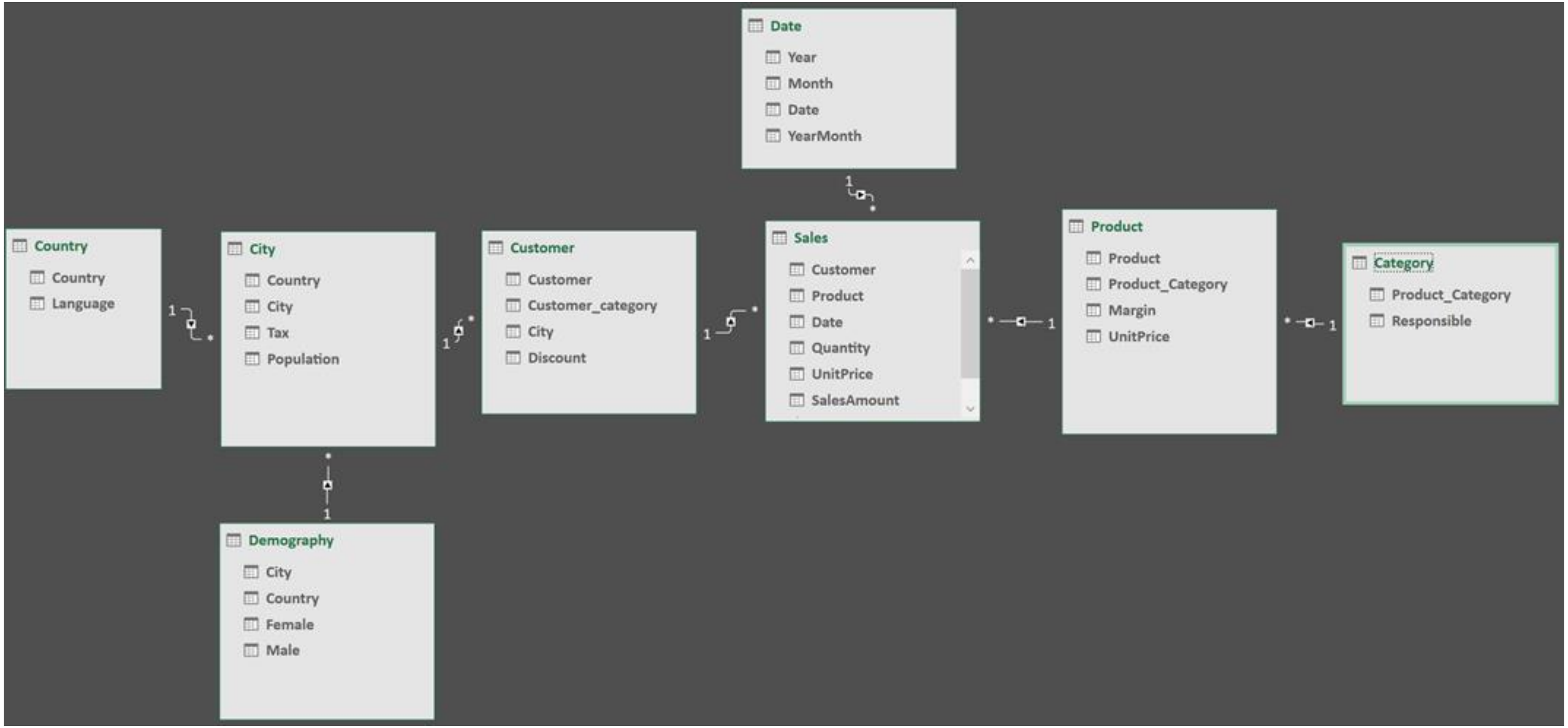


Customer	SalesAmount_m	Ratio_tot
c1	195	25,93%
c3	42	5,59%
c4	440	58,51%
c5	75	9,97%
Grand Total	**752**	**100,00%**

```
Ratio_tot:=
DIVIDE(
    SUM(Sales[SalesAmount]);
    CALCULATE(
        SUM(Sales[SalesAmount]);
        ALL(Sales)
    )
)
```

Evaluation Contexts and Relationships

Starting from this model



Row context

- The row context doesn't propagate over relationships !

✗

```
My_calculate_column=
Sales[UnitPrice]*(1-Customer[Discount])
```

→ RELATED function

✓

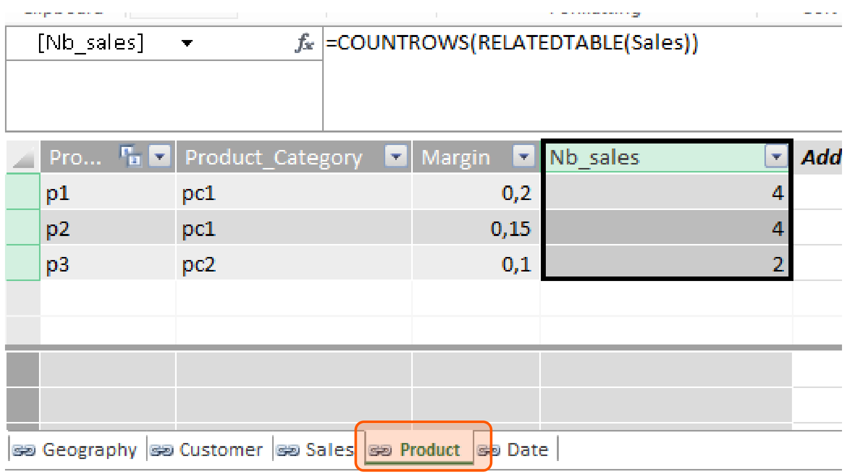
```
My_calculate_column=
Sales[UnitPrice]*(1-RELATED(Customer[Discount]))
```

- RELATED opens a row context on the other table following the relationship.

Row context

- RELATEDTABLE returns only the related rows of the parameter table

```
Nb_sales=
COUNTROWS(RELATEDTABLE(Sales))
```



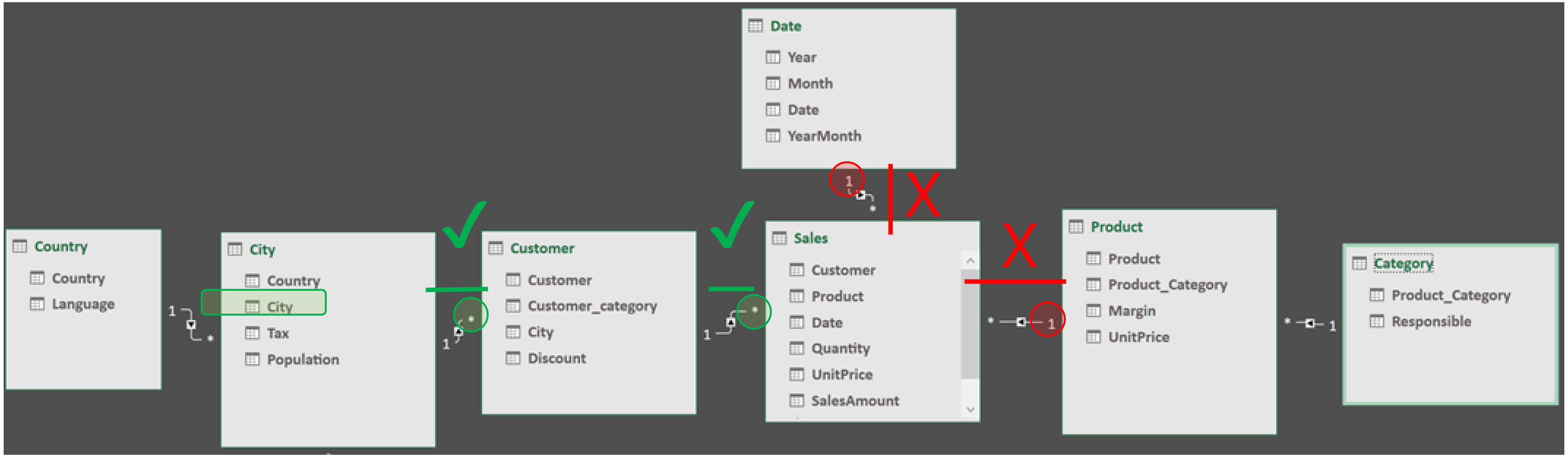
```
[Nb_sales]   =COUNTROWS(RELATEDTABLE(Sales))
```

Pro...	Product_Category	Margin	Nb_sales	Add
p1	pc1	0,2	4	
p2	pc1	0,15	4	
p3	pc2	0,1	2	

→ Number of sales per product

Filter context

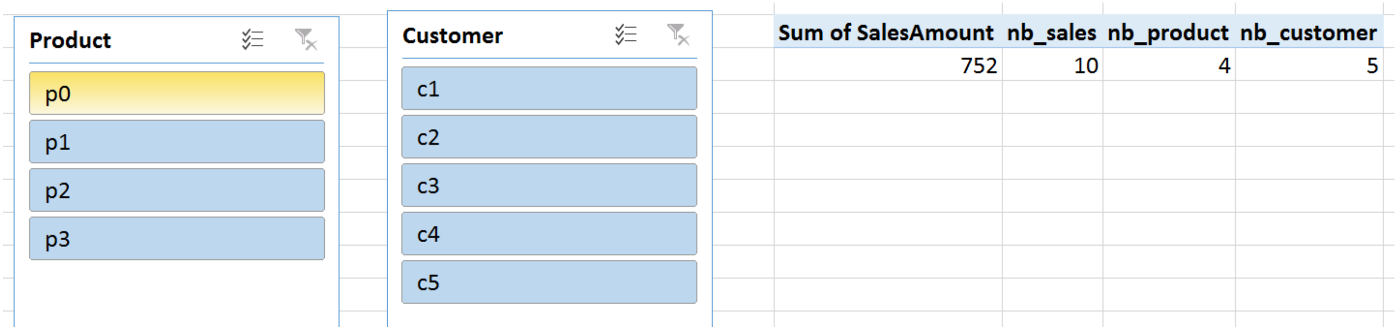
- The filter context is propagated trough relationships



Filter context propagated to Sales, not to Date & Product
→ Unidirectional
→ Possibility to configure multi-direction

Filter context

```
nb_product:=COUNTROWS('Product')
nb_sales:=COUNTROWS(Sales)
nb_customer:=COUNTROWS(Customer)
```



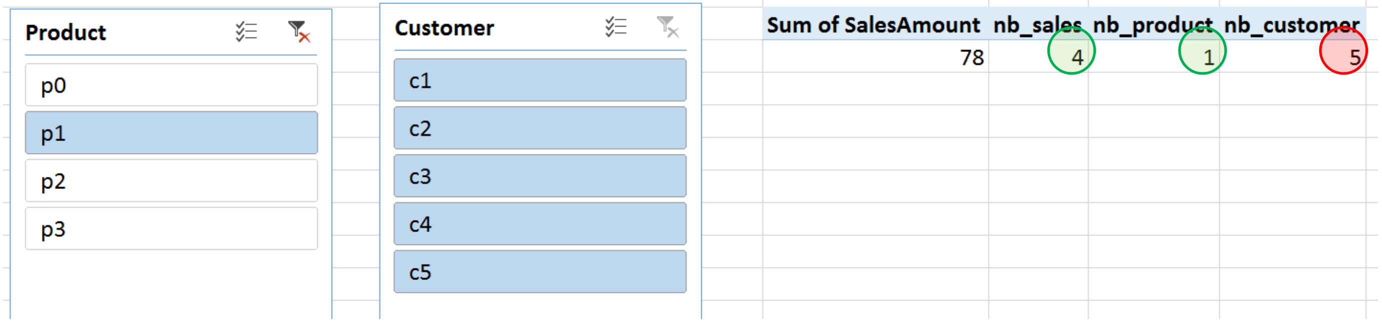
Product

p0
p1
p2
p3

Customer

c1
c2
c3
c4
c5

Sum of SalesAmount	nb_sales	nb_product	nb_customer
752	10	4	5



Product

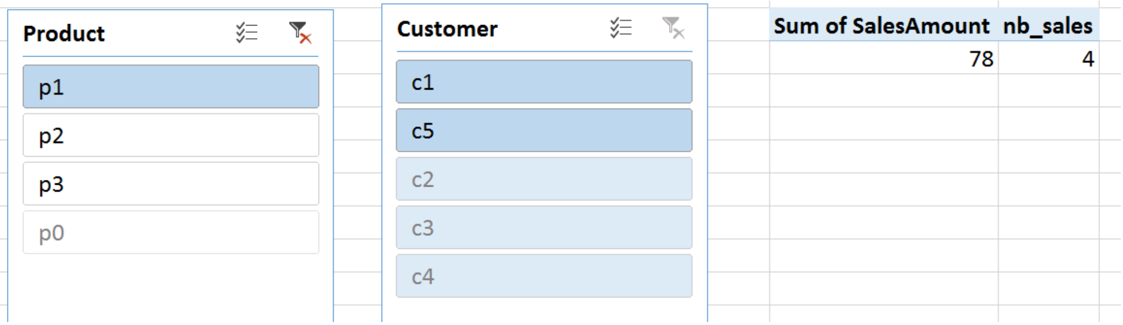
p0
p1
p2
p3

Customer

c1
c2
c3
c4
c5

Sum of SalesAmount	nb_sales	nb_product	nb_customer
78	4	1	5

→ Slicer may vary depending on pivot table !



Product

p1
p2
p3
p0

Customer

c1
c5
c2
c3
c4

Sum of SalesAmount	nb_sales
78	4

Filter context

- CALCULATE – SUMX & RELATED filter condition

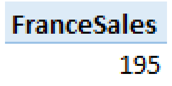
```
P1Sales:=
CALCULATE(
       SUM(Sales[SalesAmount]);
       Product[Product_Category]="Pc1"
)
```

=

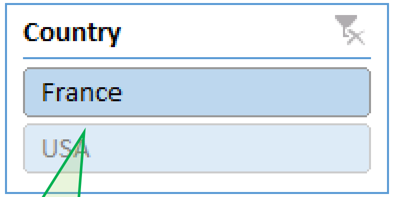
```
P1Sales_long:=
SUMX(
       FILTER(
       Sales;
       RELATED(Product[Product_Category])="Pc1"
   );
       Sales[SalesAmount]
)
```

CALCULATE – filter context modification

```
FranceSales:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        Country[Country]="France"
)
```



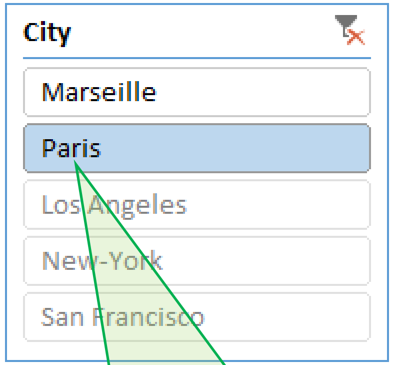
FranceSales
195



Country

France

USA



City

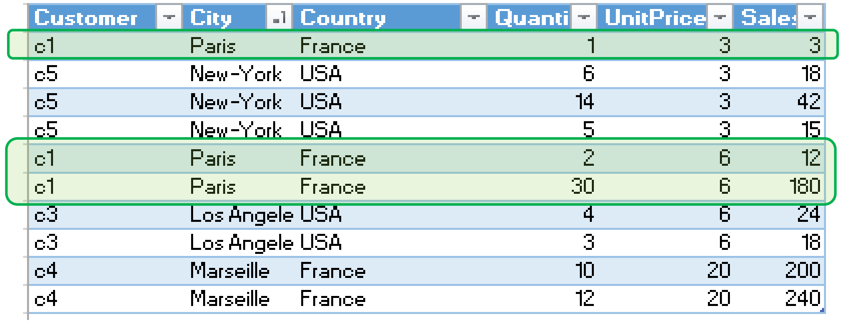
Marseille

Paris

Los Angeles

New-York

San Francisco



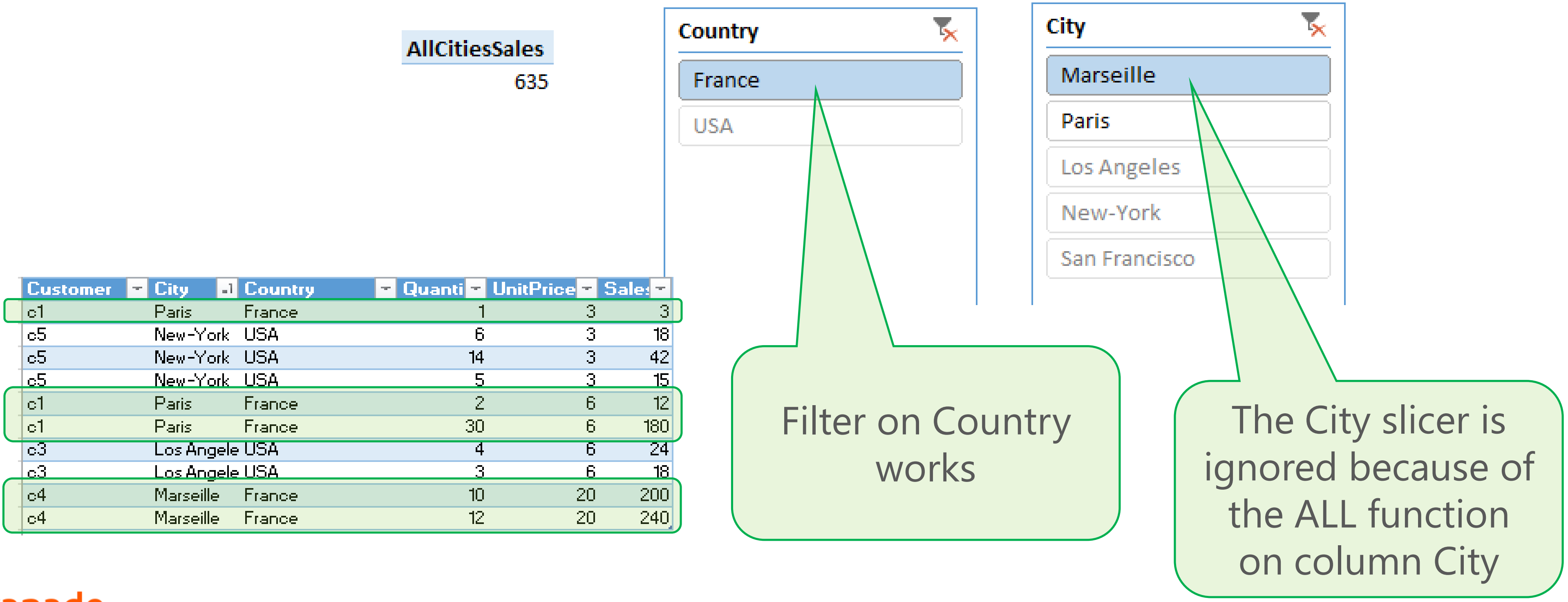
Customer	City	Country	Quanti	UnitPrice	Sale:
c1	Paris	France	1	3	3
c5	New-York	USA	6	3	18
c5	New-York	USA	14	3	42
c5	New-York	USA	5	3	15
c1	Paris	France	2	6	12
c1	Paris	France	30	6	180
c3	Los Angele	USA	4	6	24
c3	Los Angele	USA	3	6	18
c4	Marseille	France	10	20	200
c4	Marseille	France	12	20	240

USA is selected but ignored because France is forced

Filter on Paris works

CALCULATE – filter context modification

```
AllCitiesSales:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ALL(City[City])
)
```



AllCitiesSales
635

Country
France
USA

City
Marseille
Paris
Los Angeles
New-York
San Francisco

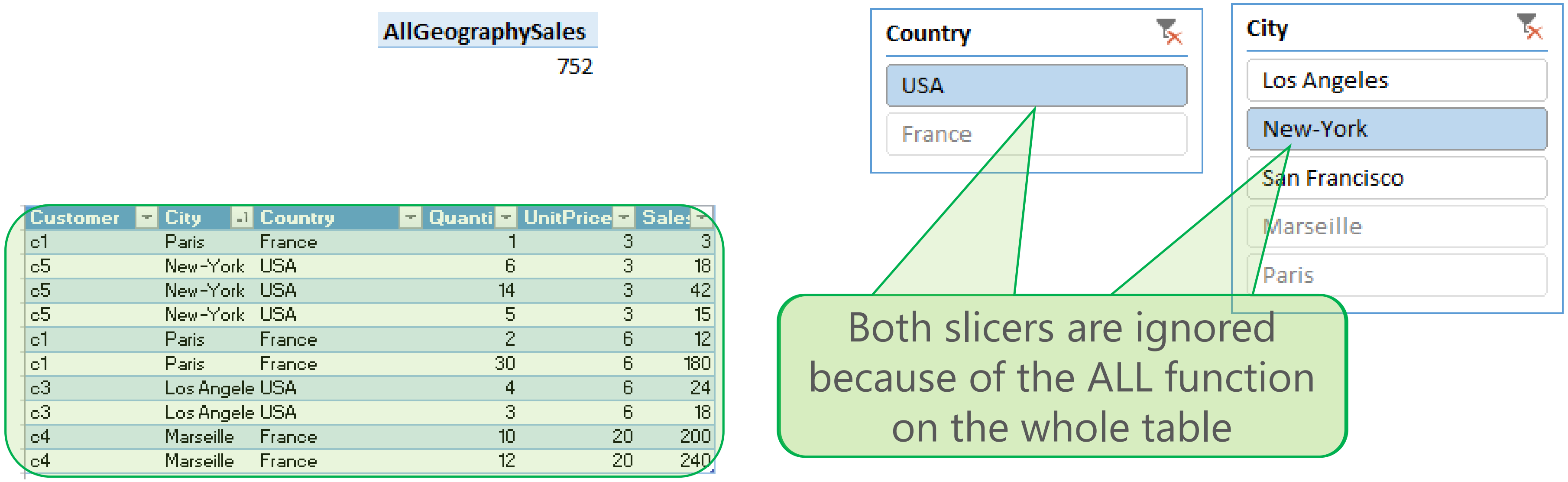
Customer	City	Country	Quanti	UnitPrice	Sale
c1	Paris	France	1	3	3
c5	New-York	USA	6	3	18
c5	New-York	USA	14	3	42
c5	New-York	USA	5	3	15
c1	Paris	France	2	6	12
c1	Paris	France	30	6	180
c3	Los Angele	USA	4	6	24
c3	Los Angele	USA	3	6	18
c4	Marseille	France	10	20	200
c4	Marseille	France	12	20	240

Filter on Country works

The City slicer is ignored because of the ALL function on column City

CALCULATE – filter context modification

```
AllCitySales:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ALL(City)
)
```



AllGeographySales
752

Customer	City	Country	Quanti	UnitPrice	Sale
c1	Paris	France	1	3	3
c5	New-York	USA	6	3	18
c5	New-York	USA	14	3	42
c5	New-York	USA	5	3	15
c1	Paris	France	2	6	12
c1	Paris	France	30	6	180
c3	Los Angele	USA	4	6	24
c3	Los Angele	USA	3	6	18
c4	Marseille	France	10	20	200
c4	Marseille	France	12	20	240

Country
- USA
- France

City
- Los Angeles
- New-York
- San Francisco
- Marseille
- Paris

Both slicers are ignored because of the ALL function on the whole table

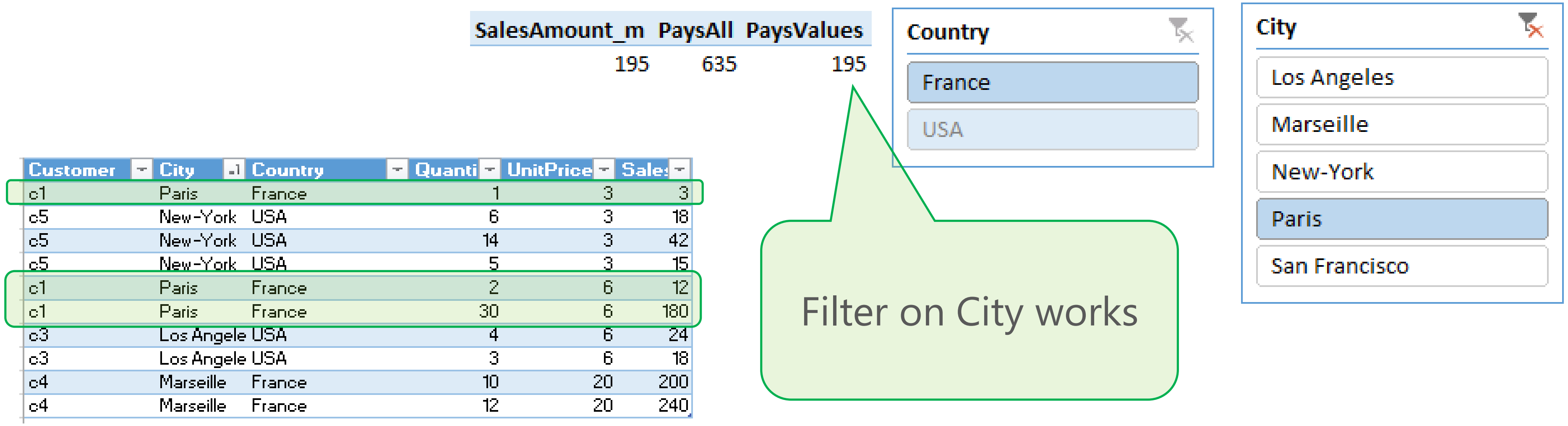
CALCULATE – Filter equivalence

```
CountryAll:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        or(
                City[City]="Paris";
                City[City]="Marseille“
        )
)
```

```
CountryAll:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        FILTER(
                ALL(City[City]);
        OR(
                City[City]="Paris";
                City [City]="Marseille“
        )
    )
)
```

CALCULATE – Filter in current selection

```
CountryValues:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        or(
                City[City]="Paris";
                City[City]="Marseille"
        );
        VALUES(City[City])
)
```



SalesAmount_m	PaysAll	PaysValues
195	635	195

Country
- France
- USA

City
- Los Angeles
- Marseille
- New-York
- Paris
- San Francisco

Customer	City	Country	Quanti	UnitPrice	Sale:
c1	Paris	France	1	3	3
c5	New-York	USA	6	3	18
c5	New-York	USA	14	3	42
c5	New-York	USA	5	3	15
c1	Paris	France	2	6	12
c1	Paris	France	30	6	180
c3	Los Angele	USA	4	6	24
c3	Los Angele	USA	3	6	18
c4	Marseille	France	10	20	200
c4	Marseille	France	12	20	240

Filter on City works

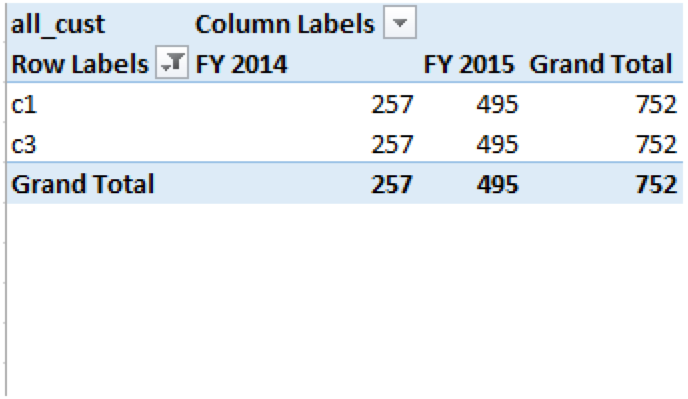
CALCULATE – Filter in current selection

```
PaysValues:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        or(
                City[City]="Paris";
                City[City]="Marseille‟
        );
        VALUES(City[City])
)
```

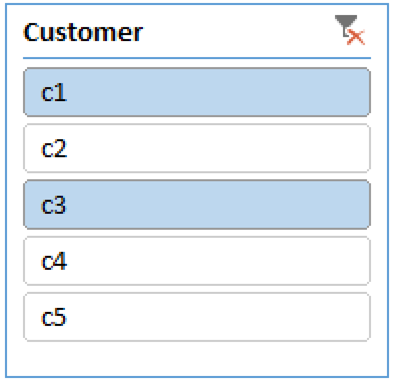
```
PaysValues:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        FILTER(
                VALUES(City[City]);
        OR(
                City[City]="Paris";
                City[City]="Marseille‟
        )
    ))
```

CALCULATE – ALL vs ALLSELECTED

- ALL → Removes the filter
- ALLSELECTED → Keeps the filter



all_cust	Column Labels		
Row Labels	FY 2014	FY 2015	Grand Total
c1	257	495	752
c3	257	495	752
Grand Total	**257**	**495**	**752**

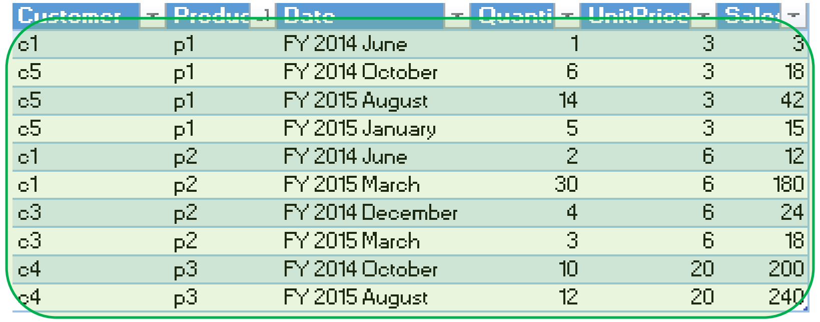


Customer

- c1
- c2
- c3
- c4
- c5

Filter is useless

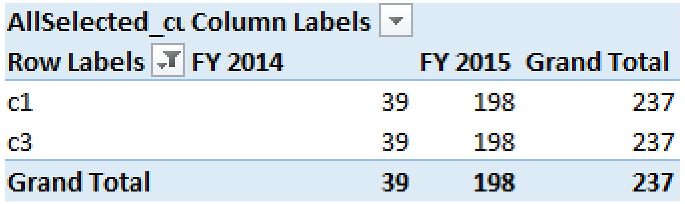
```
All_customer:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ALL(Customer[Customer])
)
```



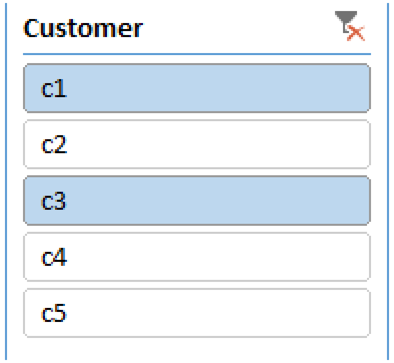
Customer	Produc	Date	Quanti	UnitPrice	Sale
c1	p1	FY 2014 June	1	3	3
c5	p1	FY 2014 October	6	3	18
c5	p1	FY 2015 August	14	3	42
c5	p1	FY 2015 January	5	3	15
c1	p2	FY 2014 June	2	6	12
c1	p2	FY 2015 March	30	6	180
c3	p2	FY 2014 December	4	6	24
c3	p2	FY 2015 March	3	6	18
c4	p3	FY 2014 October	10	20	200
c4	p3	FY 2015 August	12	20	240

CALCULATE – ALL vs ALLSELECTED

- ALL → Removes all the filter
- ALLSELECTED → Keeps the explicit filters (slicer,filter)



AllSelected_cu	Column Labels		
Row Labels	FY 2014	FY 2015	Grand Total
c1	39	198	237
c3	39	198	237
Grand Total	**39**	**198**	**237**

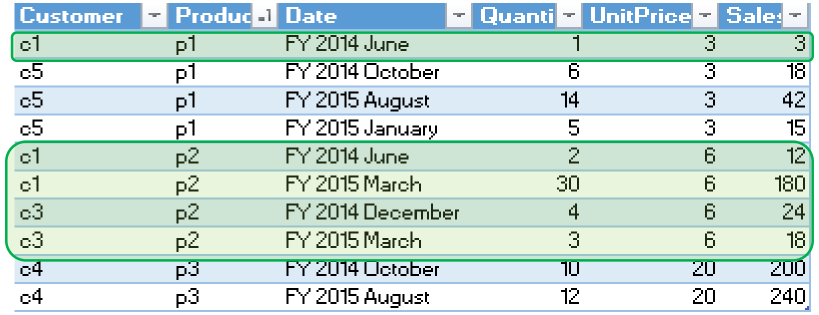


Customer

[✓ c1]
c2
[✓ c3]
c4
c5

Filter is maintained but each row displays the selected total

```
AllSelected_customer:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ALLSELECTED(Customer[Customer])
)
```



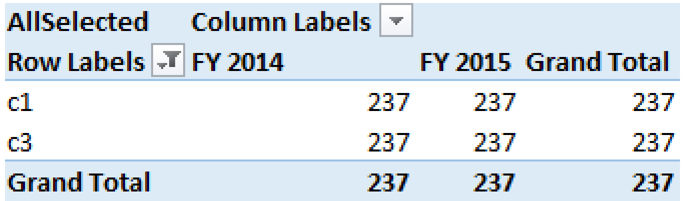
Customer	Produc	Date	Quanti	UnitPrice	Sale
c1	p1	FY 2014 June	1	3	3
c5	p1	FY 2014 October	6	3	18
c5	p1	FY 2015 August	14	3	42
c5	p1	FY 2015 January	5	3	15
c1	p2	FY 2014 June	2	6	12
c1	p2	FY 2015 March	30	6	180
c3	p2	FY 2014 December	4	6	24
c3	p2	FY 2015 March	3	6	18
c4	p3	FY 2014 October	10	20	200
c4	p3	FY 2015 August	12	20	240

Useful to compute ratio over a selection

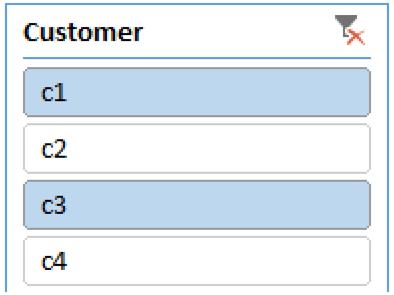
CALCULATE – ALL vs ALLSELECTED

- ALL → Removes the filter
- ALLSELECTED → Keeps the filter

→ Removes rows and columns context



AllSelected	Column Labels		
Row Labels	FY 2014	FY 2015	Grand Total
c1	237	237	237
c3	237	237	237
Grand Total	237	237	237

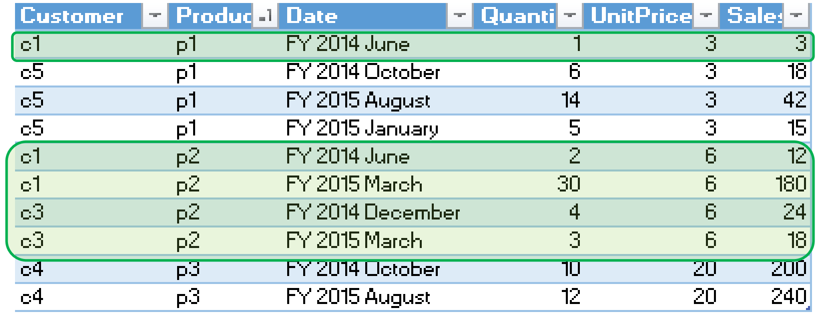


Customer

c1
c2
c3
c4

Filter is maintained but ALLSELECTED is applied on all tables

```
AllSelected:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ALLSELECTED()
)
```



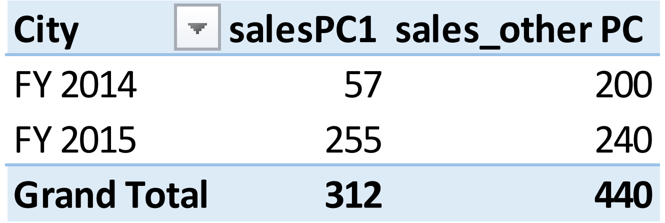
Customer	Produc	Date	Quanti	UnitPrice	Sale
c1	p1	FY 2014 June	1	3	3
c5	p1	FY 2014 October	6	3	18
c5	p1	FY 2015 August	14	3	42
c5	p1	FY 2015 January	5	3	15
c1	p2	FY 2014 June	2	6	12
c1	p2	FY 2015 March	30	6	180
c3	p2	FY 2014 December	4	6	24
c3	p2	FY 2015 March	3	6	18
c4	p3	FY 2014 October	10	20	200
c4	p3	FY 2015 August	12	20	240

Exercices : Evaluation context / Calculate

60 minutes

1. **Calculate sales for PC1 and others (2 measures to create)**

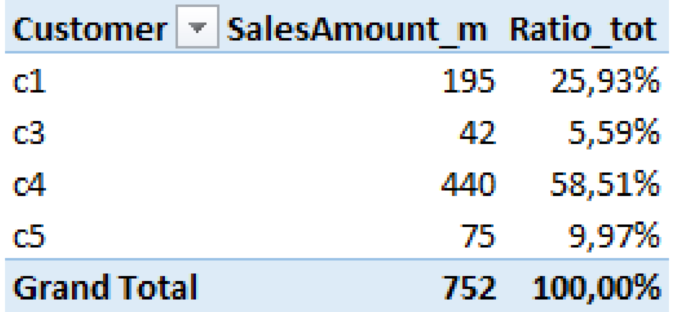
- Using SUMX AND RELATED
- Then Using CALCULATE



City	salesPC1	sales_other PC
FY 2014	57	200
FY 2015	255	240
Grand Total	**312**	**440**

2. CALCULATE & ALL:

- Calculate Sales ratio per customer
- Then do it keeping Slicer values



Customer	SalesAmount_m	Ratio_tot
c1	195	25,93%
c3	42	5,59%
c4	440	58,51%
c5	75	9,97%
Grand Total	752	100,00%

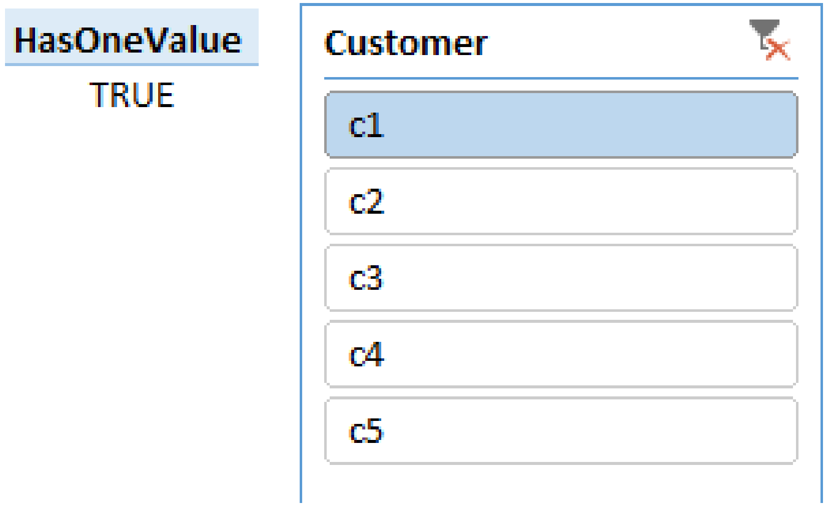
3. Understanding CALCULATE

Sales for product category « pc1 » : what’s wrong?

Using VALUES to solve the problem

HASONEVALUE

- Returns **TRUE** when the context for columnName has been filtered down to one distinct value only. Otherwise is **FALSE**



HasOneValue
TRUE

Customer

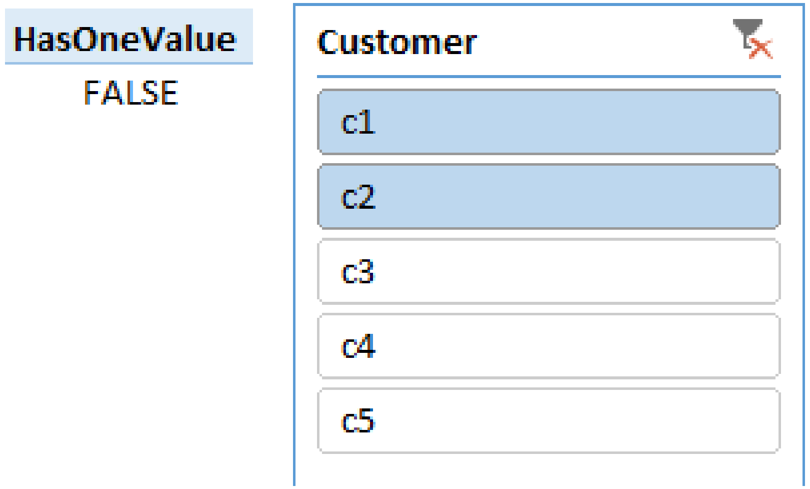
c1

c2

c3

c4

c5



HasOneValue
FALSE

Customer

c1

c2

c3

c4

c5

```
HasOneValue:=
        HasOneValue(
                Customer[Customer]
        )
```

IF

- Checks if a condition provided as the first argument is met. Returns one value if the condition is TRUE, and returns another value if the condition is FALSE.

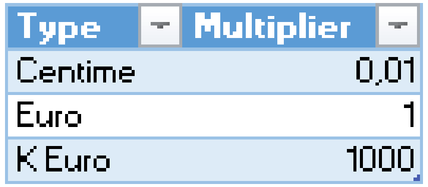
```
IF(
        logical_test>,
        <value_if_true>,
        value_if_false
)
```

```
= if (
        Sales[Product]="p1";
        "p1";
        "OTHER "
)
```

Exercices : Parameter Table

20 minutes

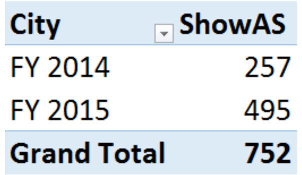
1. **Create a parameter table:**



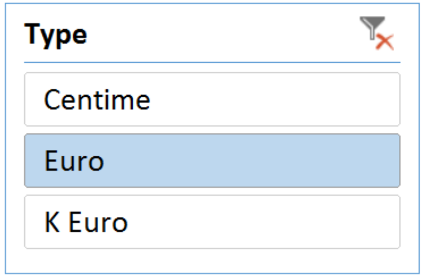
Type	Multiplier
Centime	0,01
Euro	1
K Euro	1000

2. **Add it to the PowerPivot model and rename the table « Multiplier »**

3. Create a mesure named « ShowAs » which allows you to change the unit.



City	ShowAS
FY 2014	257
FY 2015	495
Grand Total	**752**

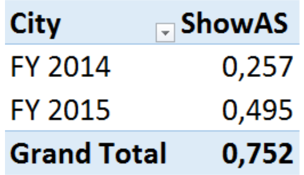


Type

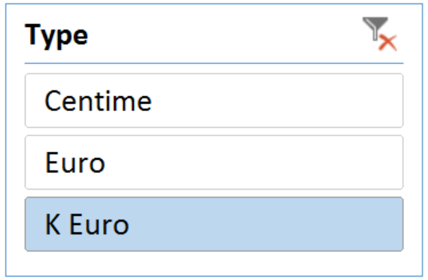
Centime

[✓ Euro]

K Euro



City	ShowAS
FY 2014	0,257
FY 2015	0,495
Grand Total	**0,752**



Type

Centime

Euro

[✓ K Euro]

Iterators

Presentation

- Iterators functions return the result of an **expression** evaluated for each row in a table

```
SalesAmount_m:=SUMX(
      Sales;
      Sales[Quantity]*Sales[UnitPrice]
)
```

- Iterators function
  - MAXX
  - MINX
  - AVERAGEX
  - SUMX

→ And more with DAX 2016

MAXX

```
SalesAmount_m_maxx:=
      MAXX( Sales;Sales[Quantity]*Sales[UnitPrice] )
```

✓

```
SalesAmount_m_max:=
      MAX( Sales[Quantity]*Sales[UnitPrice] )
```

X

→ Semantic error : The MAX function only accepts a column reference as an argument

Average

An example : calculating the average basket

```
Average_basket:=
DIVIDE (
       SUMX( Sales; Sales[Quantity]*Sales[UnitPrice] )       ;
       COUNTROWS( Sales )
)
```

⬇ =

```
Average_basket:= AVERAGEX(Sales;Sales[SalesAmount])
```

Another example : calculating the average sales per product

```
Average_basket_product:=
AVERAGEX(
       Product;
       sumx(
                    RELATEDTABLE(Sales);
                    Sales[UnitPrice]*Sales[Quantity]
       ))
```

RANKX

- Returns the ranking of a number in a list of numbers for each row in the table argument

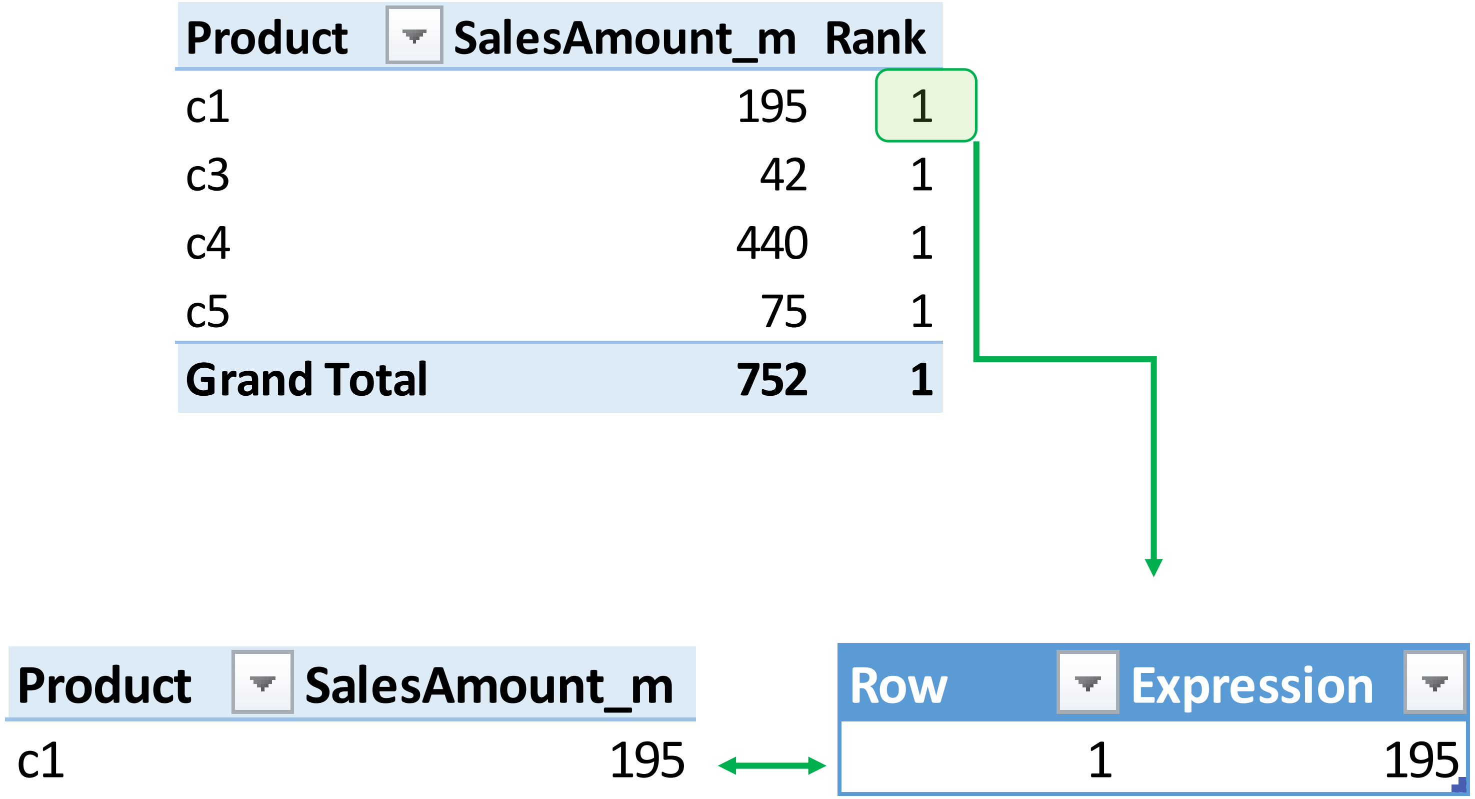
```
RANKX(<table>; <expression>)
```

- Looks like it is easy, but there are some pitfall to avoid…

- RANKX builds a Lookup table with expression computed and return rank by using the position in this table

RANKX

```
Rank:=RANKX(Customer;[SalesAmount_m])
```



Product	SalesAmount_m	Rank
c1	195	1
c3	42	1
c4	440	1
c5	75	1
Grand Total	**752**	**1**

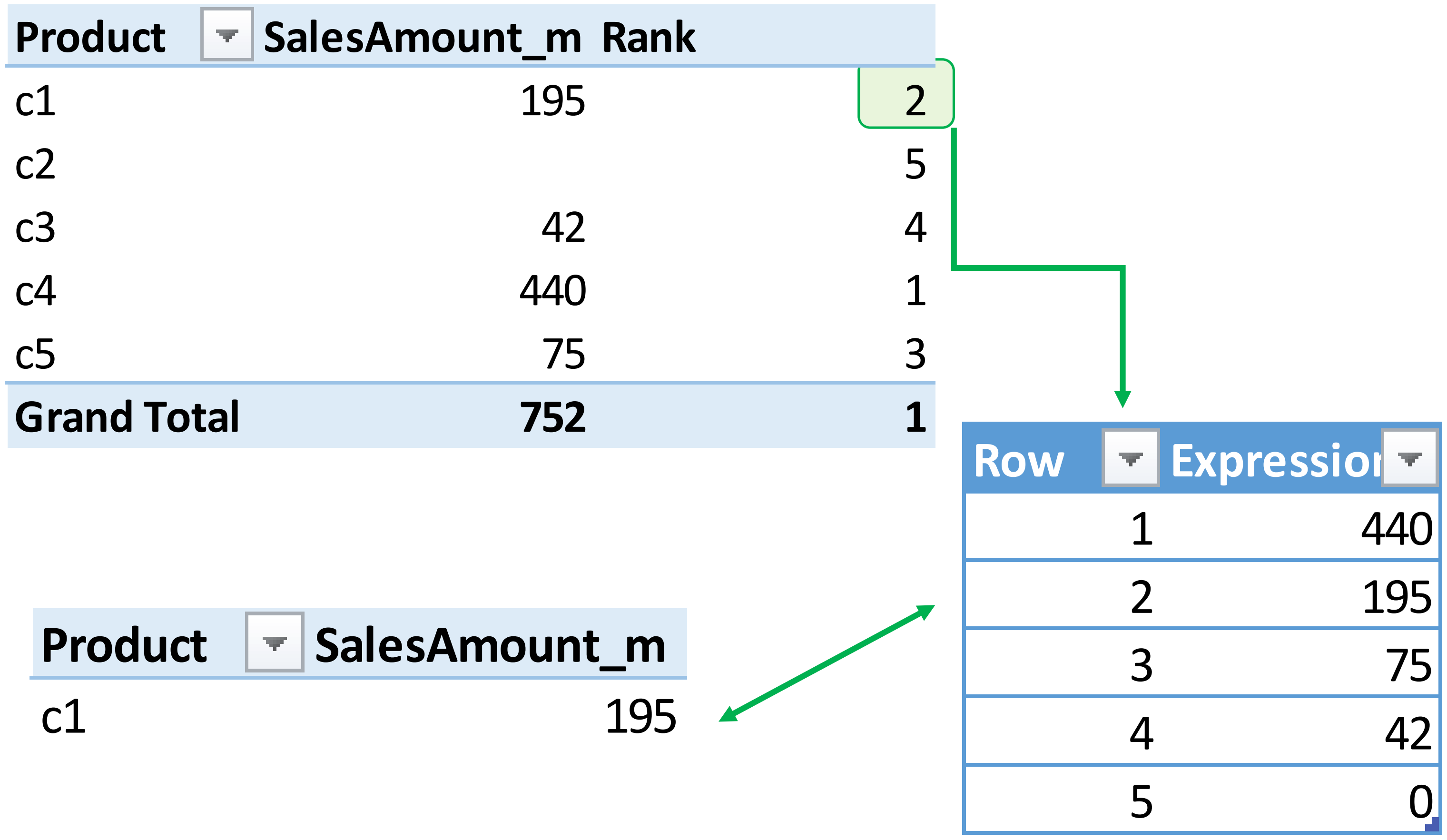
Product	SalesAmount_m
c1	195

Row	Expression
1	195

Lookup table

RANKX

```
Rank:=RANKX(ALL(Customer);[SalesAmount_m])
```



Product	SalesAmount_m	Rank
c1	195	2
c2		5
c3	42	4
c4	440	1
c5	75	3
Grand Total	**752**	**1**

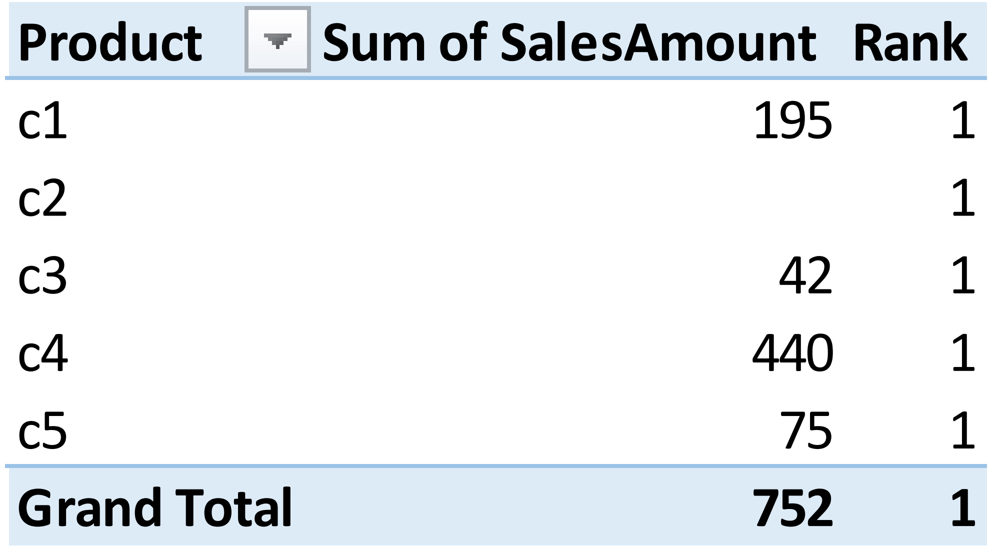
Product	SalesAmount_m
c1	195

Row	Expressior
1	440
2	195
3	75
4	42
5	0

Lookup table

RANKX

```
Rank:=RANKX(ALL(Customer);SUM(Sales[SalesAmount]))
```



Product	Sum of SalesAmount	Rank
c1	195	1
c2		1
c3	42	1
c4	440	1
c5	75	1
Grand Total	**752**	**1**

RANKX

- You need to force context by using CALCULATE

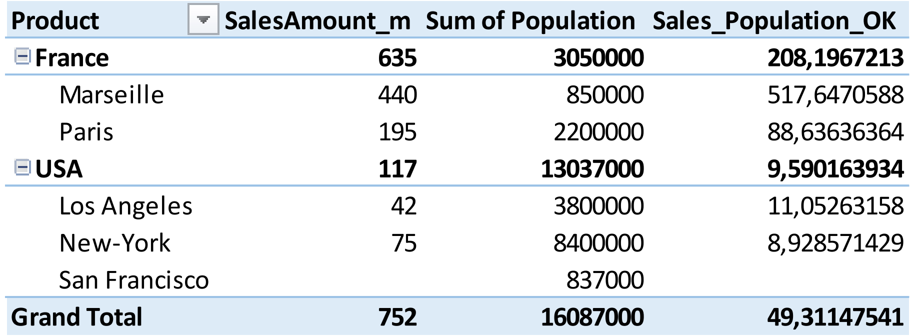
```
Rank:=RANKX(ALL(Customer);CALCULATE(SUM(Sales[SalesAmount])))
```

Exercices : Iterators

45 minutes

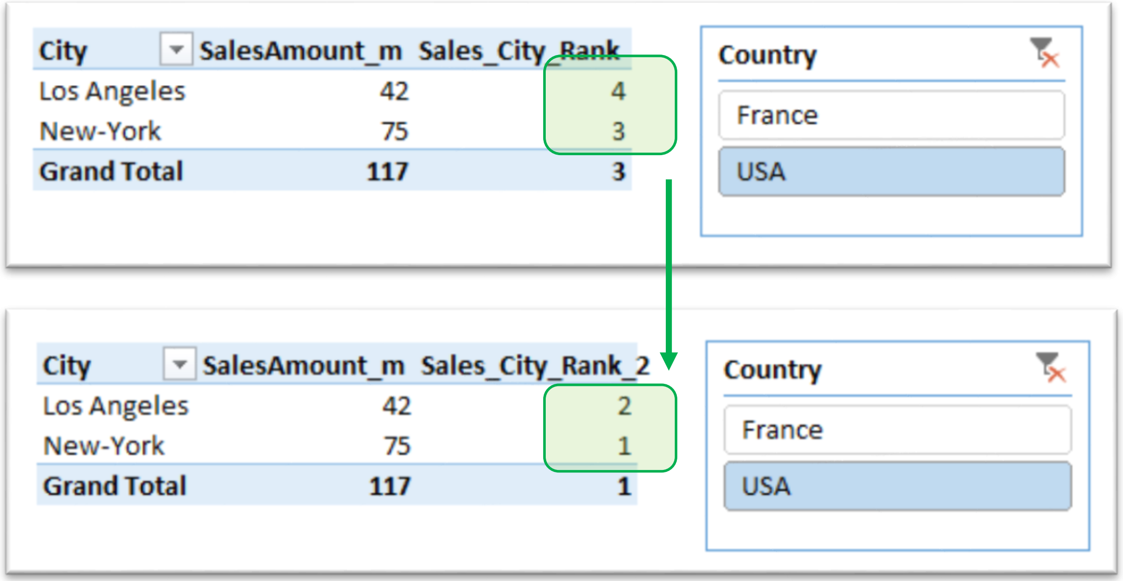
- **SUMX / VALUES**

  Sales per inhabitants



Product	SalesAmount_m	Sum of Population	Sales_Population_OK
⊟ **France**	**635**	**3050000**	**208,1967213**
Marseille	440	850000	517,6470588
Paris	195	2200000	88,63636364
⊟ **USA**	**117**	**13037000**	**9,590163934**
Los Angeles	42	3800000	11,05263158
New-York	75	8400000	8,928571429
San Francisco		837000	
Grand Total	**752**	**16087000**	**49,31147541**

  Ranking Cities



City	SalesAmount_m	Sales_City_Rank
Los Angeles	42	4
New-York	75	3
Grand Total	**117**	**3**

Country
France
USA

City	SalesAmount_m	Sales_City_Rank_2
Los Angeles	42	2
New-York	75	1
Grand Total	**117**	**1**

Country
France
USA

- **AVERAGEX**

  Count number of customers having spent more than the average.

Querying Tabular and Table Functions

Querying in DAX

You might use DAX to query a tabular model when you use SSAS in Tabular mode.

Tools:

- SSMS inside MDX query (no intellisense)
- Excel
- PowerView
- PowerBI Desktop
- DAX Studio (http://daxstudio.codeplex.com/)

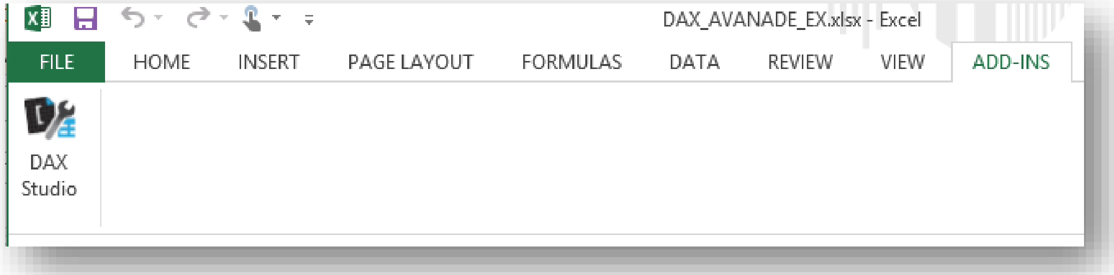
You will use what you learned before…

… but you will need to learn new functions !

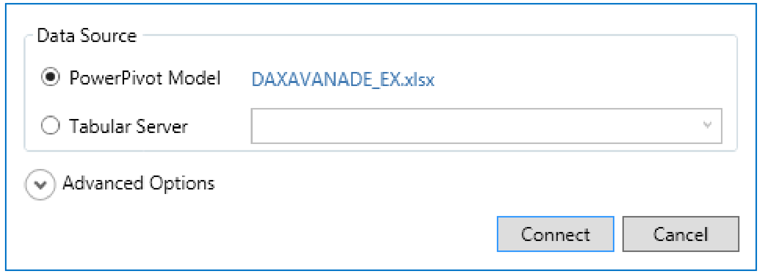
Query your PowerPivot model

To query your PowerPivot model you need to install the **DAX Studio Excel add-in**.

1) Open your Excel file
2) Click on the DAX Studio add-in in the Excel ribbon



3) This will open DAX Studio and give you the possibility to connect to your PowerPivot model



Data Source

● PowerPivot Model DAXAVANADE_EX.xlsx

○ Tabular Server

EVALUATE

- EVALUATE is the statement to write DAX queries

```
DEFINE
        MEASURE <table>[<col>] = <expression>

EVALUATE <Table Expression>

ORDER BY <expression> [ASC | DESC], …

START AT <value>, …
```

```
EVALUATE
        Sales
ORDER BY Sales[Quantity]
```

Filtering data

- You can use a function you already know…

```
EVALUATE
FILTER(
        Product,
        AND(Product[Margin]>0.1,Product[Product_Category]="p")
)
ORDER BY Product[Product]
```

- Or use a faster one

```
EVALUATE
CALCULATETABLE(
        Product,
        Product[Margin]>0.1,
        Product[Product_Category]="p"
)
ORDER BY Product[Product]
```

CALCULATETABLE - equivalence

```
EVALUATE
CALCULATETABLE(
        Product,
        Product[Margin]>0.1
)
ORDER BY Product[Product]
```

=

```
EVALUATE
CALCULATETABLE(
        Product,
        FILTER(
                ALL(Product[Margin]),
                Product[Margin]>0.1
        )
)
ORDER BY Product[Product]
```

Multiple CALCULATETABLE

- Using nested CALCULATETABLE, the evaluation order is not intuitive…

```
EVALUATE
CALCULATETABLE(
        CALCULATETABLE(
                Product,
                ALL(Product[Product_Category])
        ),
        Product[Category]="pc1"
)
```

2

1

Result:
ALL(Product[Category])

- The last executed is the inner.

- Remember this CALCULATETABLE behaviour

SUMMARIZE

- SUMMARIZE performs GROUP BY in DAX

```
EVALUATE
SUMMARIZE(
        Sales,
        Product[Product_Category],
        "Total_Sales",SUM(Sales[SalesAmount])
)
```

- You must provide a label for the calculations

- You should avoid using SUMMARIZE to create calculated columns !
  Use a mix of SUMMARIZE & ADDCOLUMNS instead.

SUMMARIZE & ADDCOLUMNS

- Here is the solution for creating calculated columns

```
EVALUATE
ADDCOLUMNS(
        SUMMARIZE(
                Sales,
                Product[Product_Category]
        ),
        "Total_Sales",CALCULATE(SUM(Sales[SalesAmount]))
)
```

- Note & understand the CALCULATE

- If you don’t use CALCULATE, the result is the total for all Product Categories.

TOPN

- To return only top « x » lines

```
EVALUATE
TOPN(
        10,
        ADDCOLUMNS(
                SUMMARIZE(
                        Sales,
                        Product[Product_Category]
                ),
                "Total_Sales",CALCULATE(SUM(Sales[SalesAmount]))
        ),
        [Total_Sales]
)
```

- [Total_Sales] indicates the « order by » to apply

GENERATE - TOPN

- To return the top « x » lines for each « y »

```
EVALUATE
GENERATE(
        VALUES(Product[Product_Category]),
        TOPN(
                1,
                ADDCOLUMNS(
                        SUMMARIZE(
                                Sales,
                                        Product[Product]
                        ),

        "Total_Sales",CALCULATE(SUM(Sales[SalesAmount]))
                ),
                [Total_Sales],0
        )
)
```

Measures

- It is possible to define measures inside the query
- Simplify a query and make it more readable

```
DEFINE
MEASURE Sales[LowDiscounted]=
CALCULATE(sum(Sales[SalesAmount]),Customer[Discount]<0.1)

EVALUATE
ADDCOLUMNS(
        SUMMARIZE(
                Sales,
                Product[Product]
        )
        "TotSales",CALCULATE(Sales[SalesAmount]),
        "LowDiscounted",Sales[LowDiscounted]
)
```

- Query measure=Model measure → no performance loss

Parameters in queries

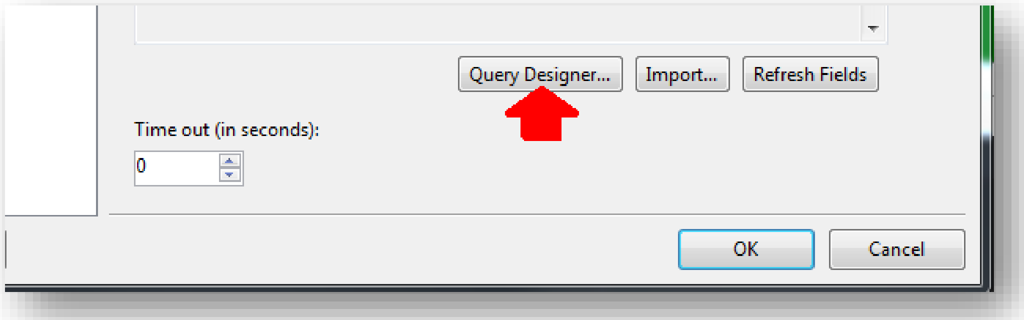
- You can insert parameters in your queries, not supported by all client tools
- Works fine with SSRS !

```
EVALUATE
CALCULATETABLE(
        Customer;
        Customer[City]=@City
)
```

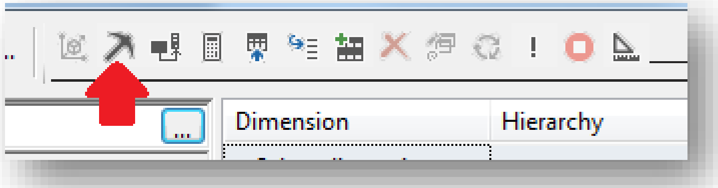
SSRS

- You can query in DAX to fill your datasets
- It is not very user friendly

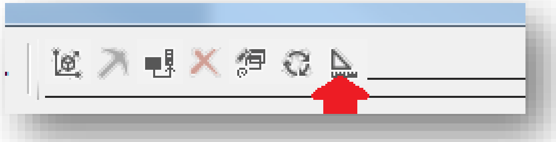
1. Open the Query Designer…



2. Switch to DMX mode

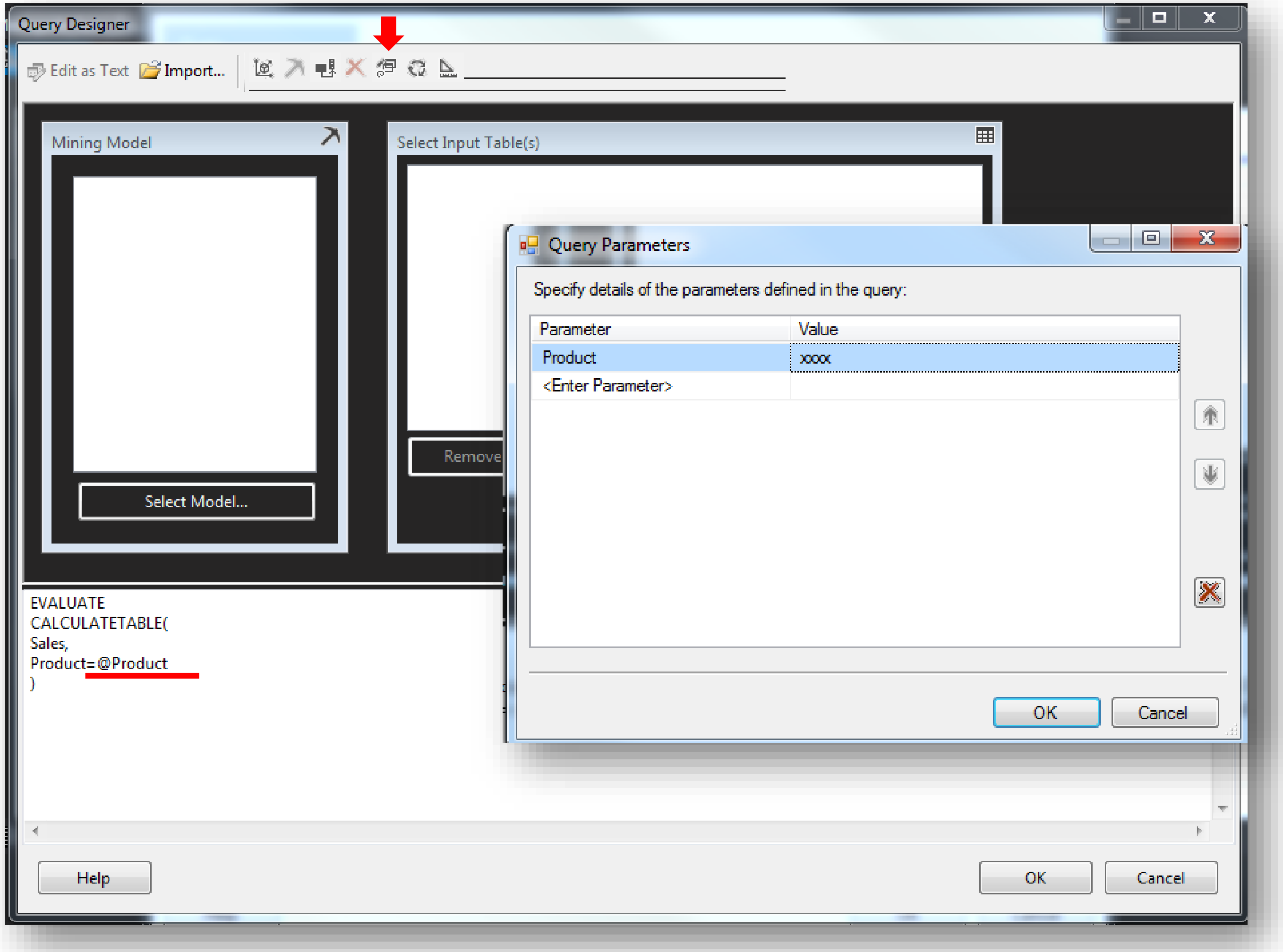


3. Change the Design Mode



SSRS

4. Write your DAX Query and do not forget to specify parameters explicitely



Parameter	Value
Product	xxxx
<Enter Parameter>	

```
EVALUATE
CALCULATETABLE(
Sales,
Product=@Product
)
```

Multi-valued Parameters in SSRS

- You can insert multiple parameters in queries
- But be careful about the performances, they will decrease a lot as the number of parameters increase…

```
EVALUATE
CALCULATETABLE(
        Customer;
        PATHCONTAINS(@City,Customer[City])
)
```

- And the SSRS expression to pass the SSRS multi-valued parameter to the query

```
=Join(Parameters!City.Value, "|")
```

Querying with MDX

- You can query a tabular model with MDX
- Try it in SSMS !

- Missing features in Tabular:
  - Attribute Key/Value/Name → Doesn't exist in Tabular
  - Attribute relationships

- MDX is often slower than DAX on a Tabular model

→ Because MDX queries generates many DAX queries

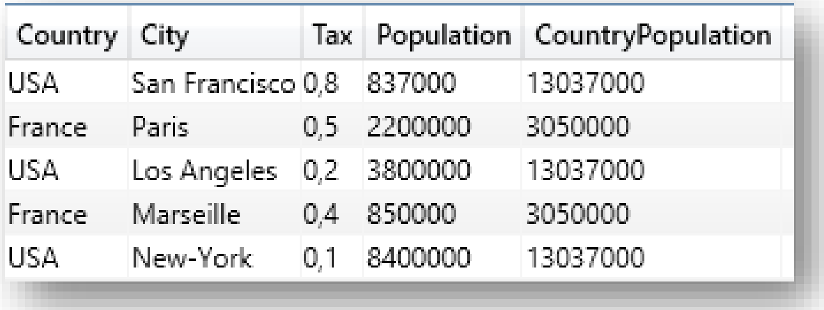
Exercices : Querying Tabular

40 minutes

1. **Simple Query**
    - Make a query to obtain sales by customer
    - Rewrite it with ADDCOLUMNS if you didn't

2. MEASURE

- Try to obtain the following result



Country	City	Tax	Population	CountryPopulation
USA	San Francisco	0,8	837000	13037000
France	Paris	0,5	2200000	3050000
USA	Los Angeles	0,2	3800000	13037000
France	Marseille	0,4	850000	3050000
USA	New-York	0,1	8400000	13037000

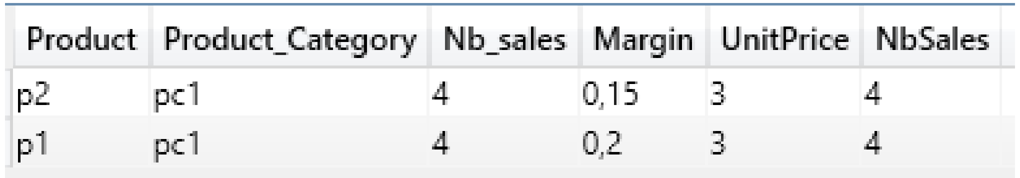
3. Complex Query

- Find the top 2 products (order by sales)

Exercices : Querying Tabular

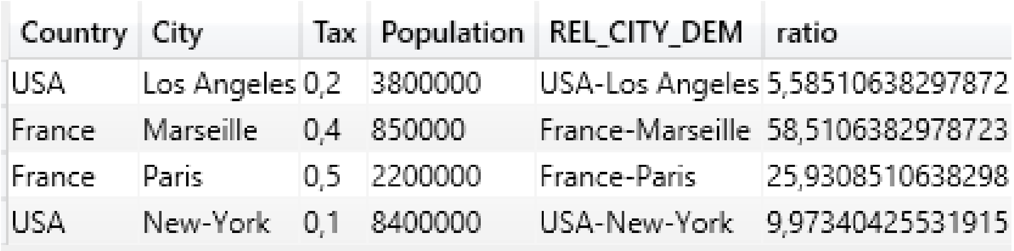
30 minutes

4. Display products sold more than twice



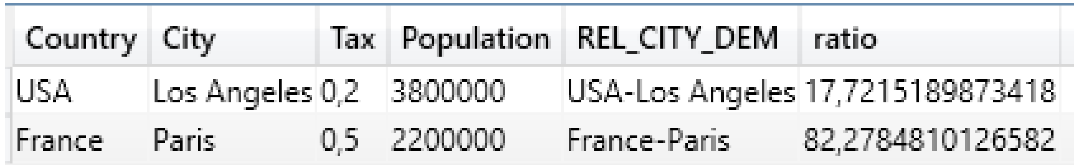
Product	Product_Category	Nb_sales	Margin	UnitPrice	NbSales
p2	pc1	4	0,15	3	4
p1	pc1	4	0,2	3	4

5. Display the cities with the sales ratio (distribution)



Country	City	Tax	Population	REL_CITY_DEM	ratio
USA	Los Angeles	0,2	3800000	USA-Los Angeles	5,58510638297872
France	Marseille	0,4	850000	France-Marseille	58,5106382978723
France	Paris	0,5	2200000	France-Paris	25,9308510638298
USA	New-York	0,1	8400000	USA-New-York	9,97340425531915

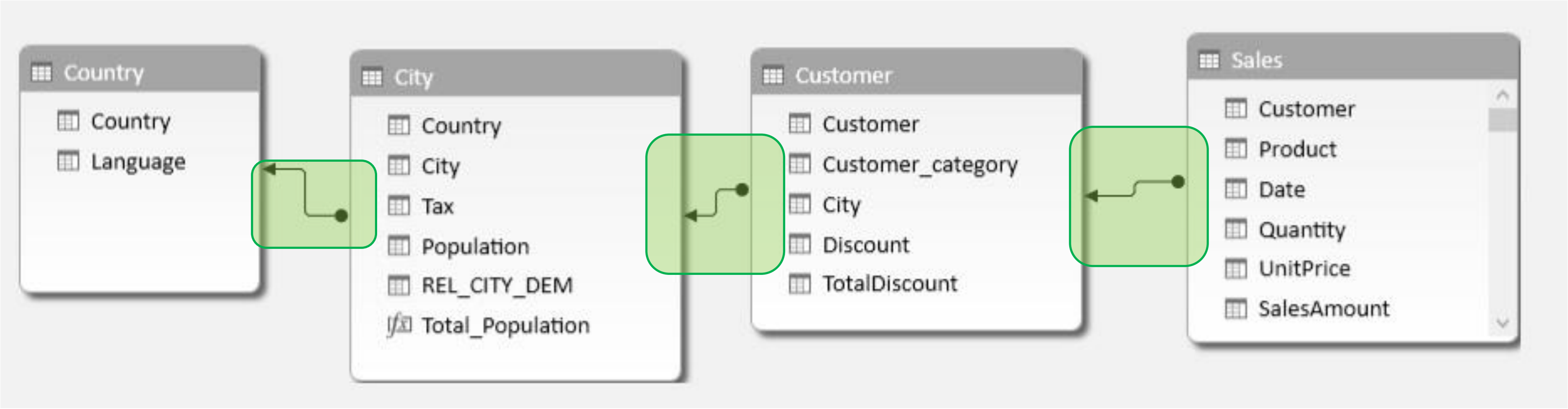
6. Modify the last query to display only Los Angeles and Paris (the ratio must be dynamic)



Country	City	Tax	Population	REL_CITY_DEM	ratio
USA	Los Angeles	0,2	3800000	USA-Los Angeles	17,7215189873418
France	Paris	0,5	2200000	France-Paris	82,2784810126582

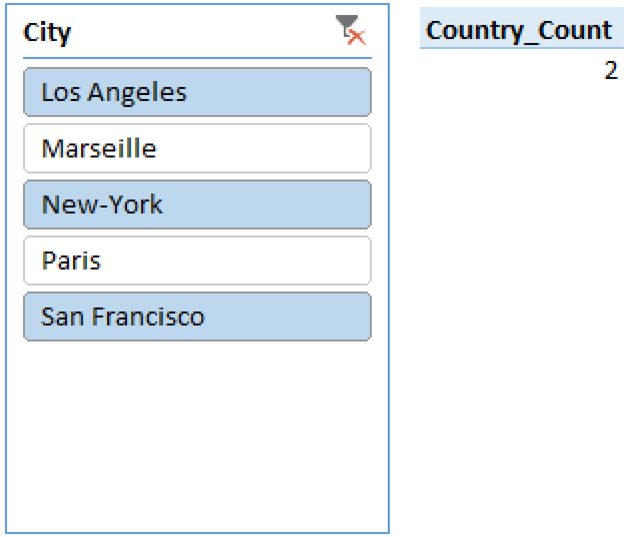
Advanced Filter Context

Filter Propagation



Row context : doesn't follow relationships
Filter context : follows relationships

Filter Propagation



City

Los Angeles

Marseille

New-York

Paris

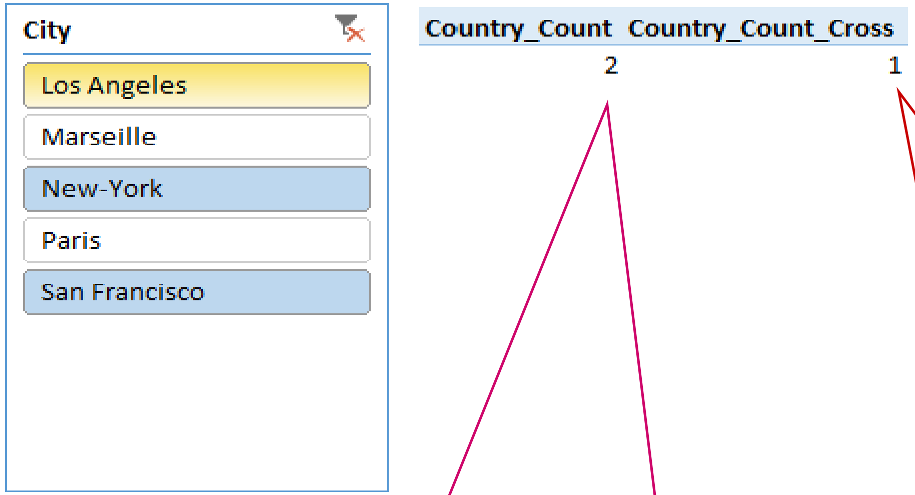
San Francisco

Country_Count
2

```
Country_Count:=COUNTROWS(Country)
```

The count of countries is not filtered by the City slicer !

Filter Propagation



City

- Los Angeles
- Marseille
- New-York
- Paris
- San Francisco

Country_Count	Country_Count_Cross
2	1

```
Country_Count:=
COUNTROWS(
Country
)
```

```
Country_Count_Cross:=
CALCULATE(
   COUNTROWS(
          Country
   )
;City)
```

Crossing the table gives us the good result

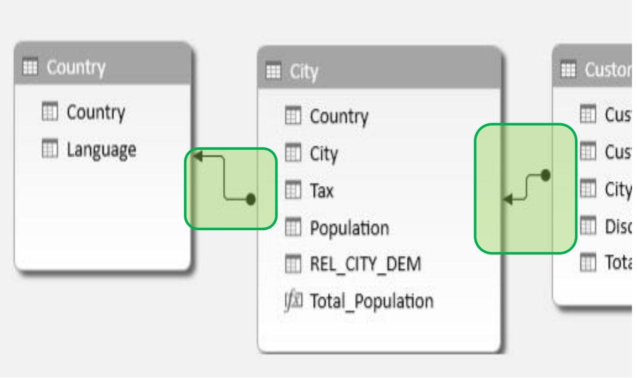
Filter Propagation : table expansion

Customer

Native Columns	Table Extension Vision	
Customer	Customer	
Customer_category	Customer_category	
City	City	
Discount	Discount	
	Customer	Sales
	Product	
	Date	
	Quantity	
	UnitPrice	

City

Native Columns	Table Extension Vision	
Country	Country	
City	City	
Tax	Tax	
Population	Population	
	Customer	Customer
	Customer_category	
	City	
	Discount	
	Customer	Sales
	Product	
	Date	
	Quantity	
	UnitPrice	

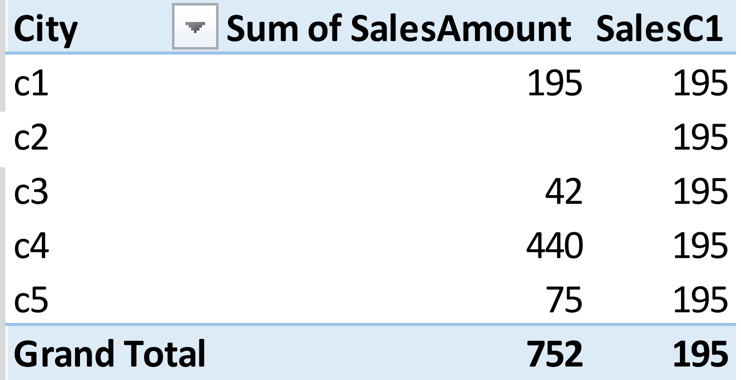


Crossing tables **can be seen** as extending the table

Filter Equivalent

```
SalesC1:= CALCULATE(
        sum(Sales[SalesAmount]);
        Customer[Customer]= "C1"
)
```

```
SalesC1:= CALCULATE(
        sum(Sales[SalesAmount]);
        FILTER(
                ALL(Customer[Customer]);
                Customer[Customer]= "C1"
        )
)
```

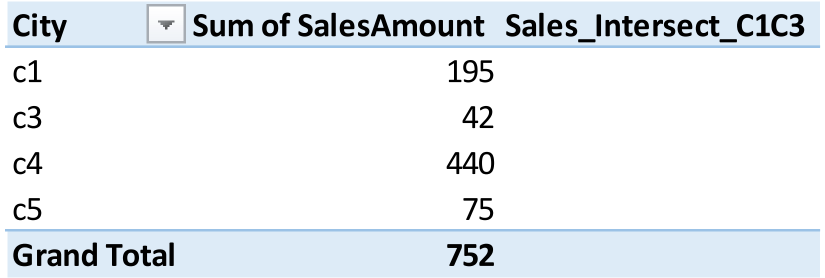


City ▾	Sum of SalesAmount	SalesC1
c1	195	195
c2		195
c3	42	195
c4	440	195
c5	75	195
Grand Total	**752**	**195**

Using Many Filter

Intersection

```
Sales_Intersect_C1C3 :=
CALCULATE(
        sum(Sales[SalesAmount]);
        FILTER(
                ALL(Customer[Customer]);
                Customer[Customer]= "C1"
        );
        FILTER(
                ALL(Customer[Customer]);
                Customer[Customer]= "C3"
        )
)
```

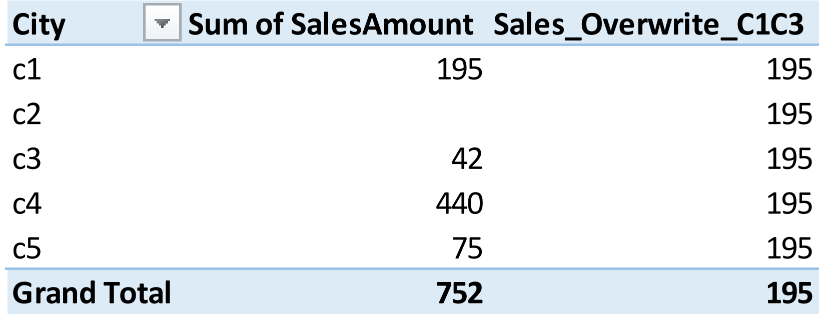


City	Sum of SalesAmount	Sales_Intersect_C1C3
c1	195	
c3	42	
c4	440	
c5	75	
Grand Total	**752**	

Using Many Filter

Overwrite

```
Sales_Overwrite_C1C3:=
CALCULATE(
        CALCULATE(
                sum(Sales[SalesAmount]);
                FILTER(
                        ALL(Customer[Customer]);
                        Customer[Customer]= "C1"
                )
        );
        FILTER(
                ALL(Customer[Customer]);
                Customer[Customer]= "C3"
        )
)
```



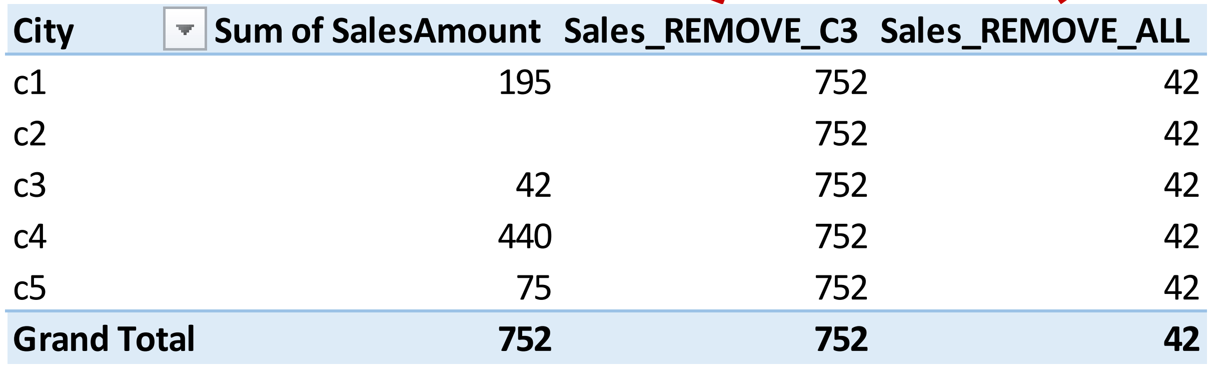
City	Sum of SalesAmount	Sales_Overwrite_C1C3
c1	195	195
c2		195
c3	42	195
c4	440	195
c5	75	195
Grand Total	**752**	**195**

Using Many Filter

REMOVE

```
Sales_REMOVE_C3:=CALCULATE(
      CALCULATE(
            sum(Sales[SalesAmount]);
            ALL(Customer[Customer])
      );
      Customer[Customer]= "C3"
)
```

```
Sales_REMOVE_ALL:=CALCULATE(
      CALCULATE(
            sum(Sales[SalesAmount]);
            Customer[Customer]= "C3"
      );
      ALL(Customer[Customer])
)
```



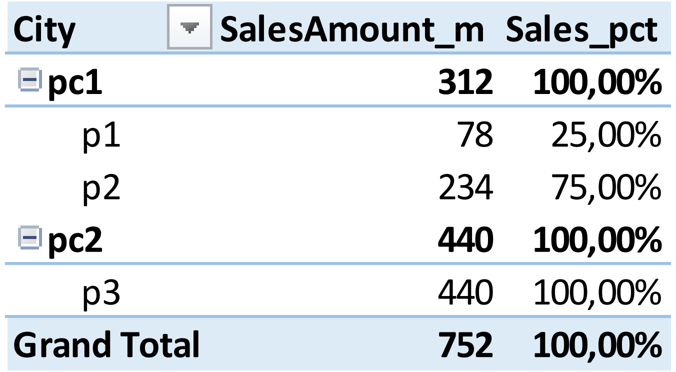
City	Sum of SalesAmount	Sales_REMOVE_C3	Sales_REMOVE_ALL
c1	195	752	42
c2		752	42
c3	42	752	42
c4	440	752	42
c5	75	752	42
Grand Total	**752**	**752**	**42**

Exercices : Advanced Filter Context

30 minutes

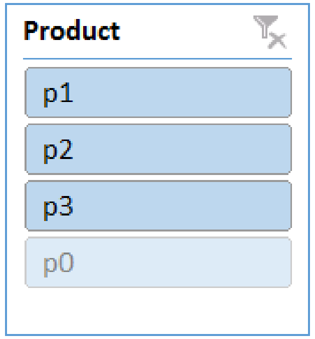
- **Crossing table**

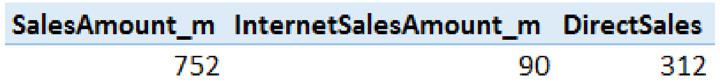
  Sales ratio product / category



City	SalesAmount_m	Sales_pct
pc1	**312**	**100,00%**
p1	78	25,00%
p2	234	75,00%
pc2	**440**	**100,00%**
p3	440	100,00%
Grand Total	**752**	**100,00%**

Direct sales (Sales made with product sold online)



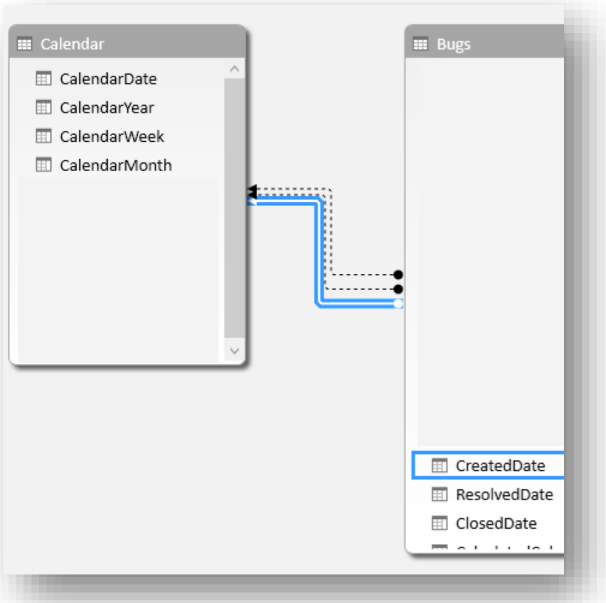


SalesAmount_m	InternetSalesAmount_m	DirectSales
752	90	312

Advanced Relationships

Multiple relationships

- Tabular model doesn’t allow multiple relationships
  … well, actually it does but only one is active.



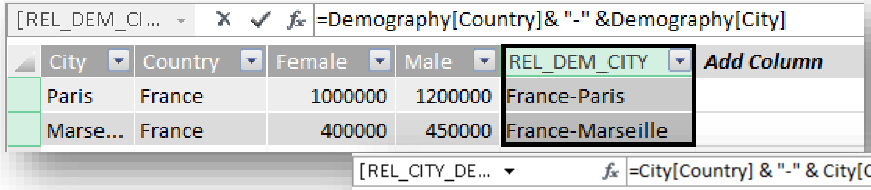
```
count_open:=
CALCULATE(
        COUNTROWS(
                RELATEDTABLE(Bugs)
        );
        USERELATIONSHIP(
                Bugs[CreatedDate];
                Calendar[CalendarDate]
        )
)
```

- When using an inactive one, you have to specify it with the USERELATIONSHIP function

Multi-column relationship

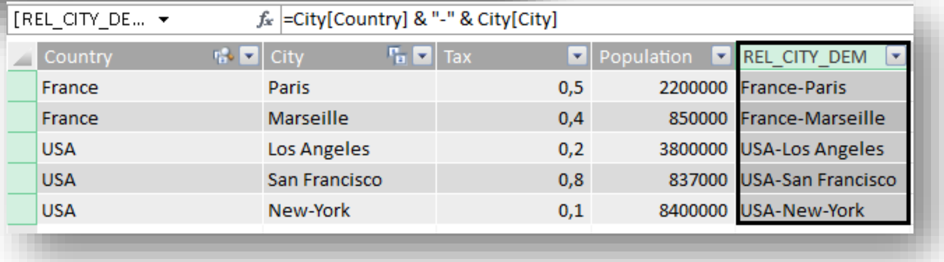
- Tabular model doesn't allow a relationship on more than one column… But you can by-pass this limitation

1. Create a calculated column which concatenates the columns needed to make a relationship in the two tables



[REL_DEM_CI… =Demography[Country]& "-" &Demography[City]

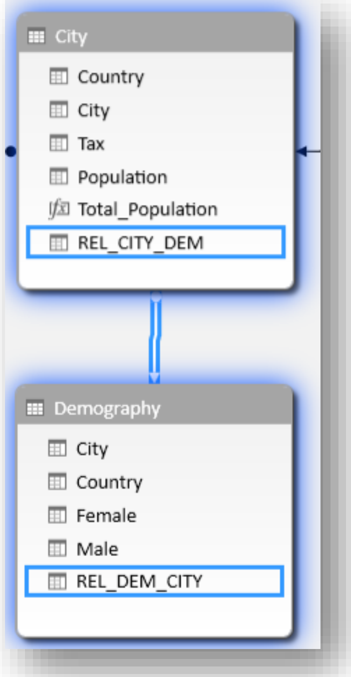
City	Country	Female	Male	REL_DEM_CITY	Add Column
Paris	France	1000000	1200000	France-Paris	
Marse…	France	400000	450000	France-Marseille	



[REL_CITY_DE… =City[Country] & "-" & City[City]

Country	City	Tax	Population	REL_CITY_DEM
France	Paris	0,5	2200000	France-Paris
France	Marseille	0,4	850000	France-Marseille
USA	Los Angeles	0,2	3800000	USA-Los Angeles
USA	San Francisco	0,8	837000	USA-San Francisco
USA	New-York	0,1	8400000	USA-New-York

2. Create the relationship on this column



Different granularity

- Sales at day level
- Sales goals at months level

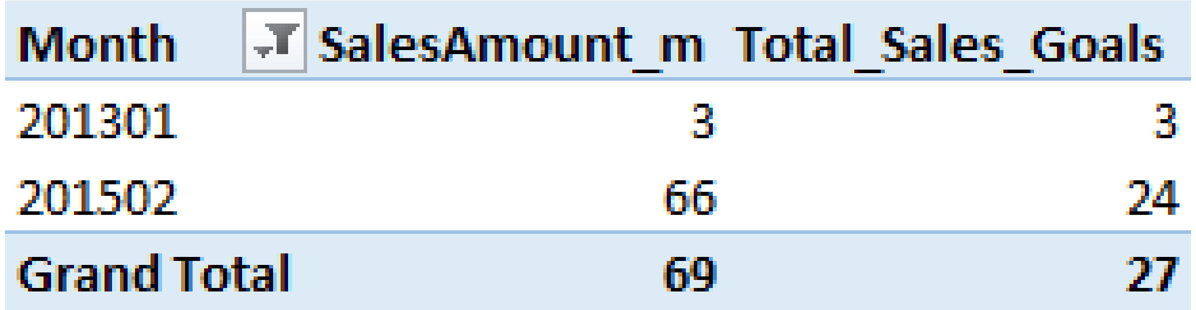
→ You can manage it with DAX Code

```
Total_Sales_Goals:=
CALCULATE(
        SUM(SalesGoals[Amount]);
        FILTER(
                ALL(SalesGoals[YearMonth]);
                CONTAINS(
                        VALUES(Date[YearMonth]);
                        Date[YearMonth];
                        SalesGoals[YearMonth]
)       )       )
```

- *Moves* the filter to another table without a relationship

Different granularities

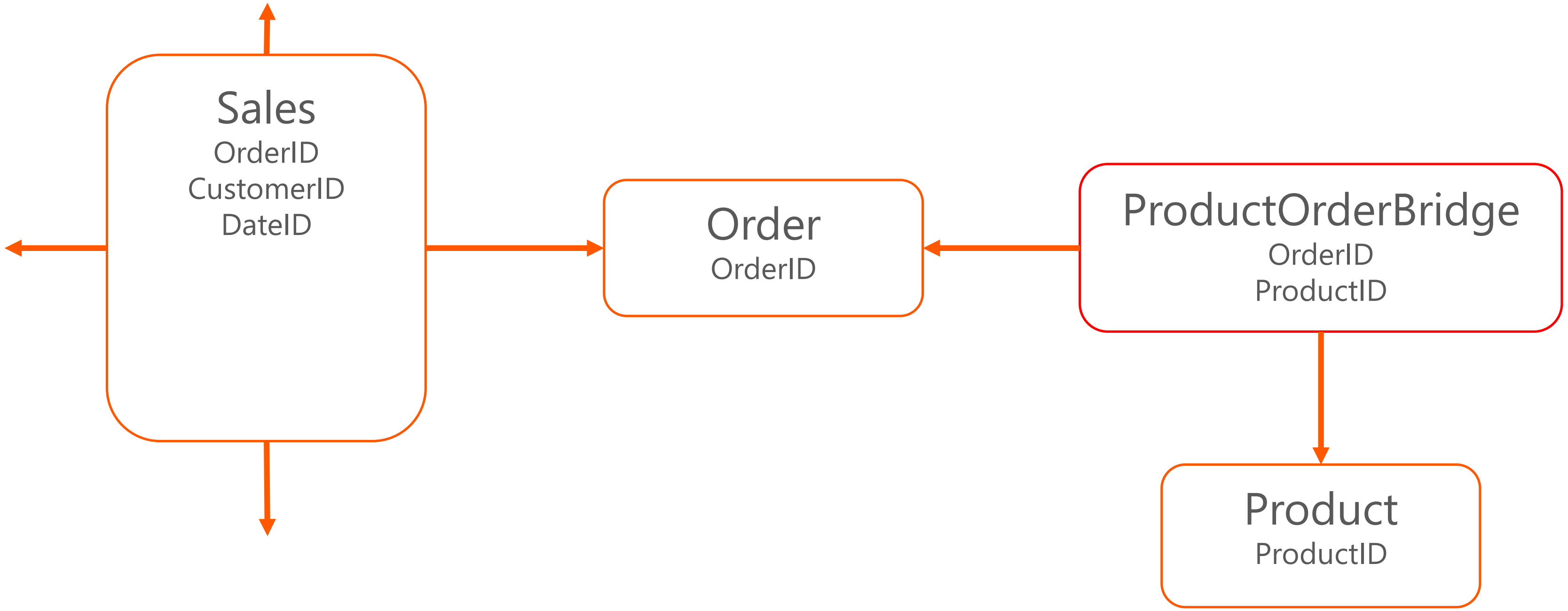
- It allows you to compare sales with sales goals in the same PivotTable using the two measures:



Month	SalesAmount_m	Total_Sales_Goals
201301	3	3
201502	66	24
Grand Total	69	27

Many-To-Many relationship

- M2M relationships are not easy to manage before DAX 2016
- Because of one-way relationship in Tabular model, M2M relationships need additional DAX code

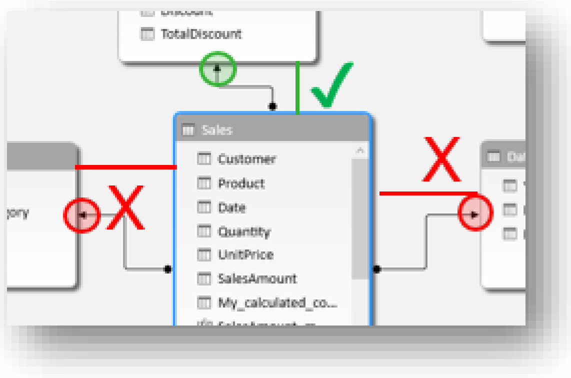


Many-To-Many relationship

- Problems come when you want to filter a result by product

We know that:

1. When we filter the **Products**, the **Bridge** table is filtered too
2. But filter propagation stops there because of the direction of the relationship. Remember…



3. When we filter the **Orders**, the **Sales** table is filtered too

Many-To-Many relationship

- So you have to write DAX code that build a new filter on Orders that takes only the Orders of the selected Products… Easy, no?

```
Amount:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        ProductOrderBridge
)
```

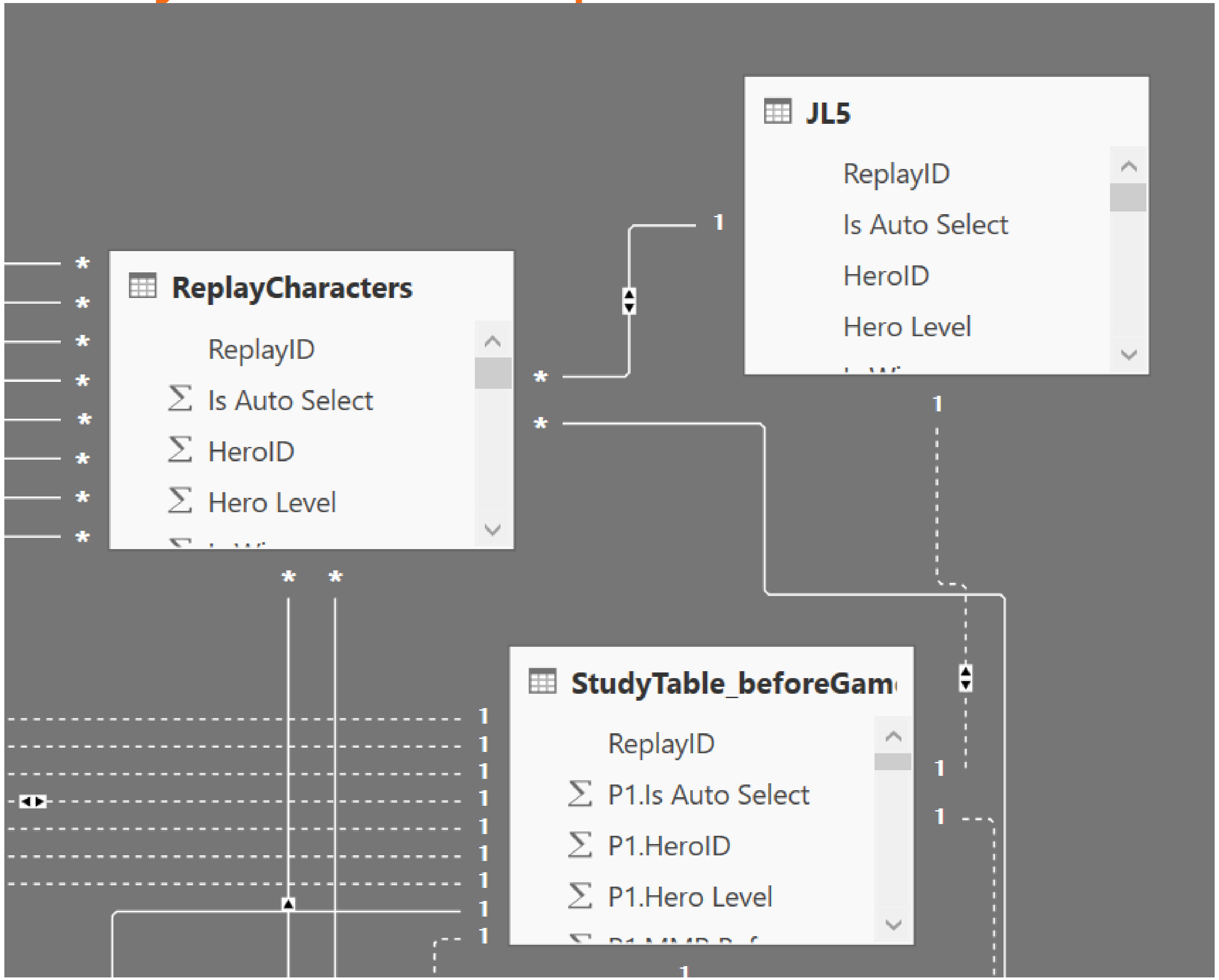
- Putting the bridge table as the second argument of CALCULATE extends the filter context created by **Products** to **Orders**.
- As said before, **Orders** is able to filter the **Sales** table

→ WORKS !

Many-To-Many relationship

- If you don’t get it, we recommend you to read the whitepaper «The Many-to-Many Revolution» written by Marco Russo and Alberto Ferrari (the «Tabular Models» part from page 96) available here: http://www.sqlbi.com/articles/many2many/ .
- Here is another useful article by the same authors: https://www.sqlbi.com/articles/optimize-many-to-many-calculation-in-dax-with-summarize-and-cross-table-filtering/ .
- This is how we got it !

Many-To-Many relationship / DAX 2016



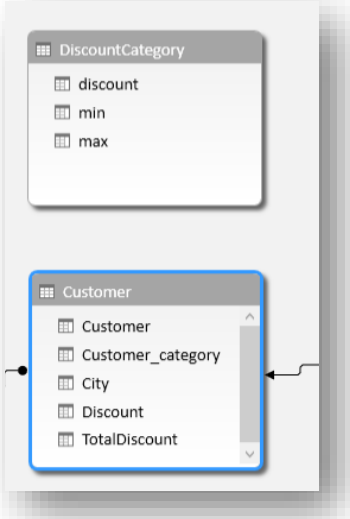
Special Use Cases

Segmentation

- Sometimes it's useful to be able to categorize a quantitative value

DISCOUNT		
LOW	0	0,1
MEDIUM	0,1	0,3
HIGH	0,3	1

- Create a table with NO relationship



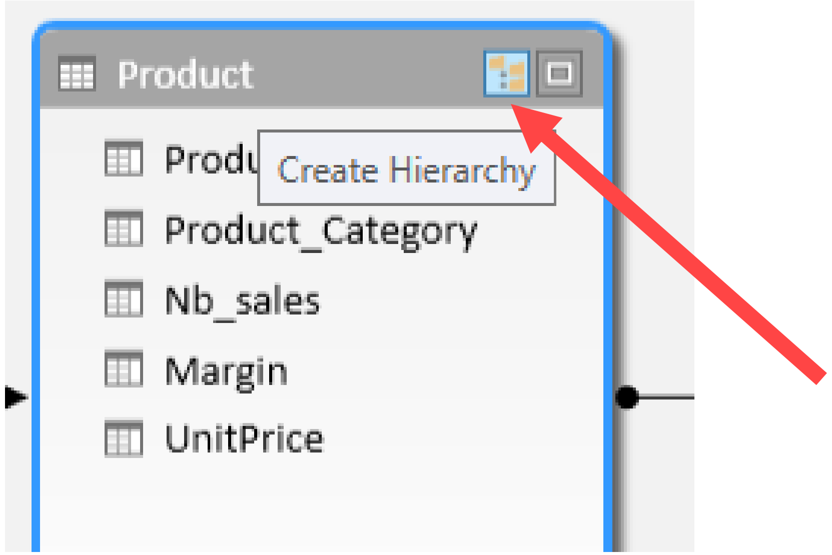
Segmentation

- The relationship is set inside the DAX code in a **calculated column**, in the customer table:

```
DiscountCategory=
CALCULATE(
        VALUES(DiscountCategory[DiscountName]);
        FILTER(
                DiscountCategory;
                AND(
                Customer[Discount]>=DiscountCategory[Min];
                Customer[Discount]< DiscountCategory[Max]
                )
        )
)
```

Hierarchy

- All fields in a hierarchy are from the same table
- Use RELATED to work with several tables

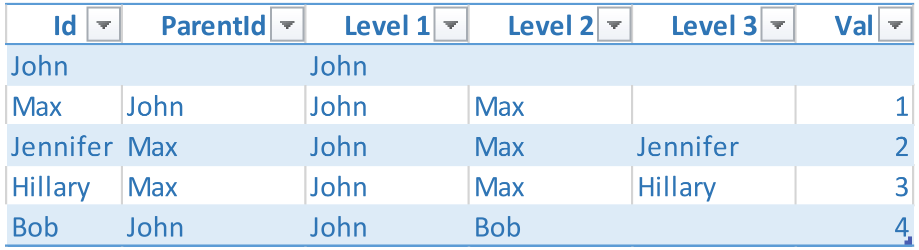


Hierarchy Exercice

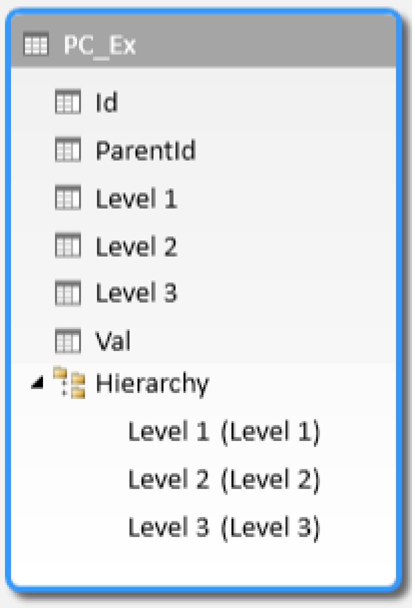
- Let’s create a Parent / Child hierarchy

For this exercice, please use the file DAX_AVANADE_EXE_SpecialUseCases.xlsx

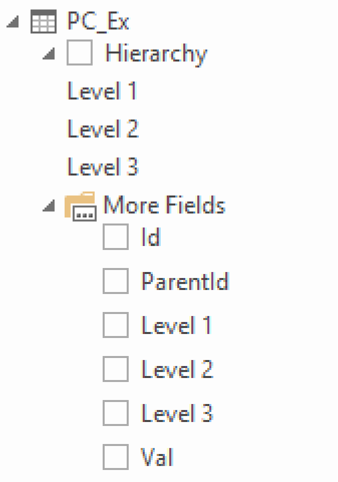
10 minutes



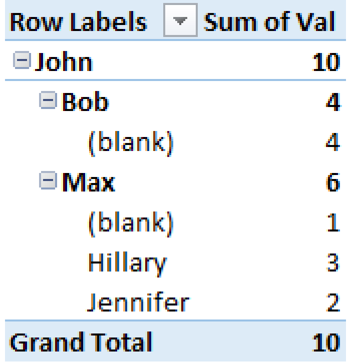
Id	ParentId	Level 1	Level 2	Level 3	Val
John		John			
Max	John	John	Max		1
Jennifer	Max	John	Max	Jennifer	2
Hillary	Max	John	Max	Hillary	3
Bob	John	John	Bob		4



PC_Ex
- Id
- ParentId
- Level 1
- Level 2
- Level 3
- Val
- Hierarchy
  - Level 1 (Level 1)
  - Level 2 (Level 2)
  - Level 3 (Level 3)



PC_Ex
- □ Hierarchy
  - Level 1
  - Level 2
  - Level 3
- More Fields
  - □ Id
  - □ ParentId
  - □ Level 1
  - □ Level 2
  - □ Level 3
  - □ Val



Row Labels	Sum of Val
⊟John	10
⊟Bob	4
(blank)	4
⊟Max	6
(blank)	1
Hillary	3
Jennifer	2
Grand Total	10

Hierarchy Function Exercice

```
PATH(<ID_columnName>; <parent_columnName>)
```

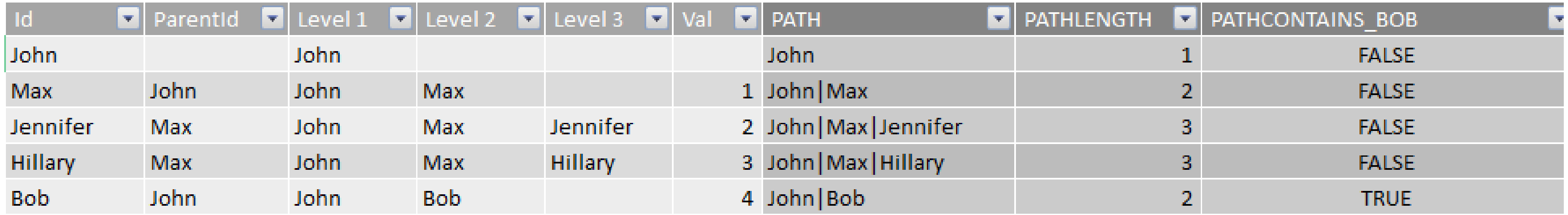
- Returns a delimited text string with the identifiers of all the parents of the current identifier

```
PATHLENGTH(<Path>)
```

- Returns the number of level for a given path

```
PATHCONTAINS(<Path>;<Item>)
```

- Returns true if the specified item exists in the path



Id	ParentId	Level 1	Level 2	Level 3	Val	PATH	PATHLENGTH	PATHCONTAINS_BOB
John		John				John	1	FALSE
Max	John	John	Max		1	John|Max	2	FALSE
Jennifer	Max	John	Max	Jennifer	2	John|Max|Jennifer	3	FALSE
Hillary	Max	John	Max	Hillary	3	John|Max|Hillary	3	FALSE
Bob	John	John	Bob		4	John|Bob	2	TRUE

Time Intelligence – Filtering Dates

- DAX provides some time intelligence functions
- To use it, you must mark your Date table as Date table (Design>Mark as Date Table) and your date column must be unique and typed as date (Advanced>Data Category)

- Here is DATESBETWEEN, useful to select a period

```
SalesAmountQ1:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        DATESBETWEEN(
                ’Date’[Date];
                DATE(2015;9;1);
                DATE(2015;11;30);
        )
)
```

Time Intelligence – Advanced functions

- DATEADD returns a date table shifted in time:

```
SalesAmountLastYear:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        DATEADD(’Date’[Date];-1;YEAR)
)
```

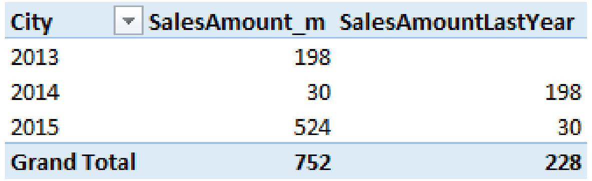
- PARALLELPERIOD returns a date table shifted in time:

```
SalesAmountLastYear:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        PARALLELPERIOD(’Date’[Date];-1;YEAR)
)
```

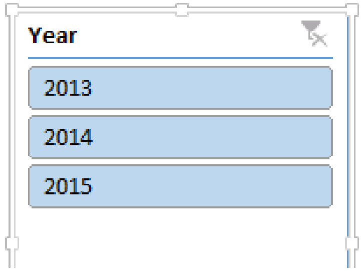
- PARALLELPERIOD returns the result for the whole period

Time Intelligence – Advanced functions

- Here is an example of DATEADD



City	SalesAmount_m	SalesAmountLastYear
2013	198	
2014	30	198
2015	524	30
Grand Total	**752**	**228**



Year

2013

2014

2015

```
SalesAmountLastYear:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        DATEADD('Date'[Date];-1;YEAR)
)
```

=

```
SalesAmountLastYear2:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        SAMEPERIODLASTYEAR('Date'[Date])
)
```

Time Intelligence – Advanced functions

- DAX provides more advanced time functions, as DATESYTD :

```
SalesYTD:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        DATESYTD(’Date’[Date])
)
```

- YTD fiscal year, indicating the end of fiscal year in third position:

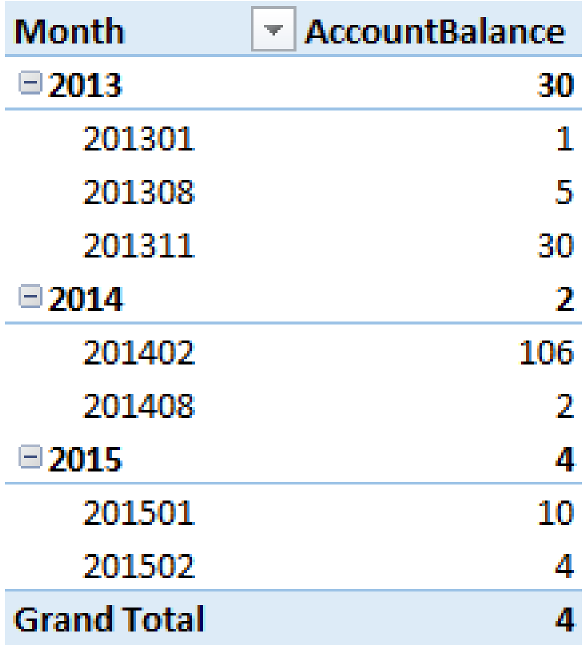
```
SalesFiscal:=
CALCULATE(
        SUM(Sales[SalesAmount]);
        DATESYTD(’Date’[Date], "31/08")
)
```

Time Intelligence – Advanced functions

- Semi-additive measures (can’t be summed over time)
- e.g. Account Balance

- You might want to use the **last value** of the lower granularity when aggregating

```
AccountBalance:=
CALCULATE(
        SUM(Account[Balance]);
        LASTDATE(Sales[Date])
)
```

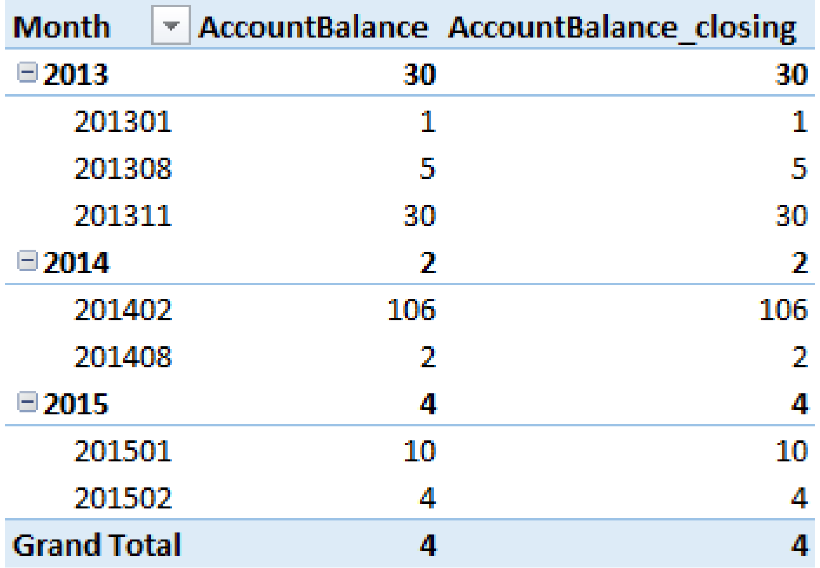


Month	AccountBalance
⊟ 2013	**30**
201301	1
201308	5
201311	30
⊟ 2014	**2**
201402	106
201408	2
⊟ 2015	**4**
201501	10
201502	4
Grand Total	**4**

Time Intelligence – Advanced functions

- Specific functions are available for classical use cases:

```
AccountBalance_closing:=
CLOSINGBALANCEMONTH (
         SUM ( Sales[Quantity]);
         Sales[Date]
)
```



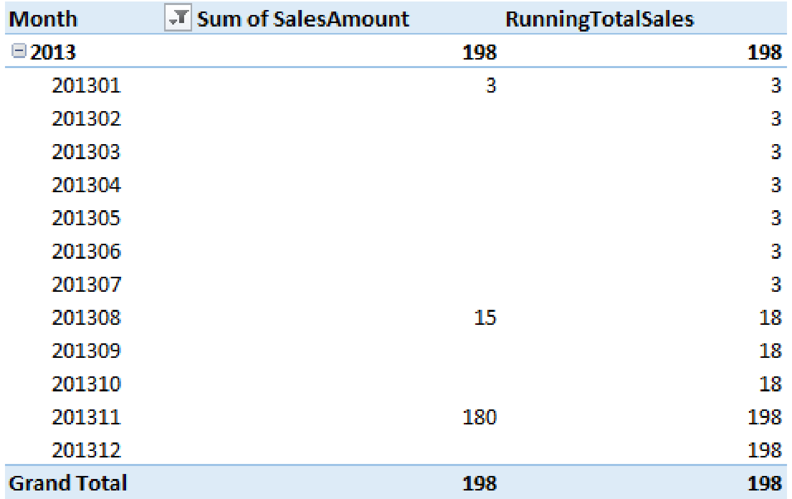
Month	AccountBalance	AccountBalance_closing
⊟ **2013**	**30**	**30**
201301	1	1
201308	5	5
201311	30	30
⊟ **2014**	**2**	**2**
201402	106	106
201408	2	2
⊟ **2015**	**4**	**4**
201501	10	10
201502	4	4
Grand Total	**4**	**4**

- CLOSINGBALANCEMONTH
- CLOSINGBALANCEQUARTER
- CLOSINGBALANCEYEAR

- OPENINGBALANCEMONTH
- OPENINGBALANCEQUARTER
- OPENINGBALANCEYEAR

Time Intelligence – Running Total

```
RunningTotalSales:=
CALCULATE (
        SUM ( Sales[SalesAmount] );
        FILTER(
                ALL ( 'Date' );
                'Date'[Date]<= MAX ( 'Date'[Date])
        )
)
```



Month	Sum of SalesAmount	RunningTotalSales
⊟2013	**198**	**198**
201301	3	3
201302		3
201303		3
201304		3
201305		3
201306		3
201307		3
201308	15	18
201309		18
201310		18
201311	180	198
201312		198
Grand Total	**198**	**198**

Exercices : Special Use Cases

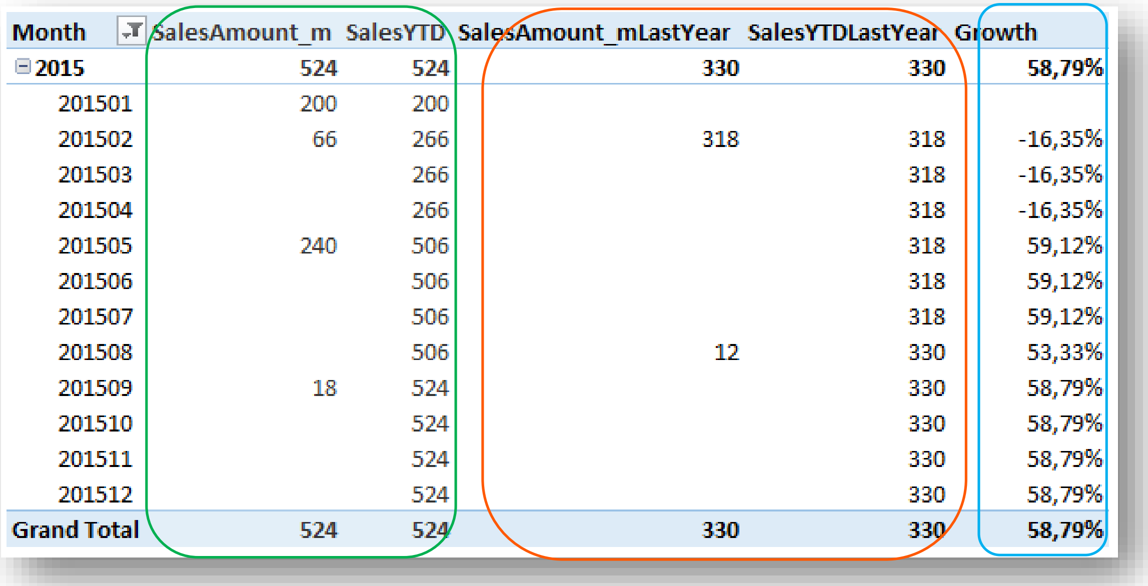
30 minutes

For this exercice, please use the file DAX_AVANADE_EXE_SpecialUseCases.xlsx

1. Create a measure that reports sales YTD for the year 2015

2. Create the same measure to report last year sales YTD

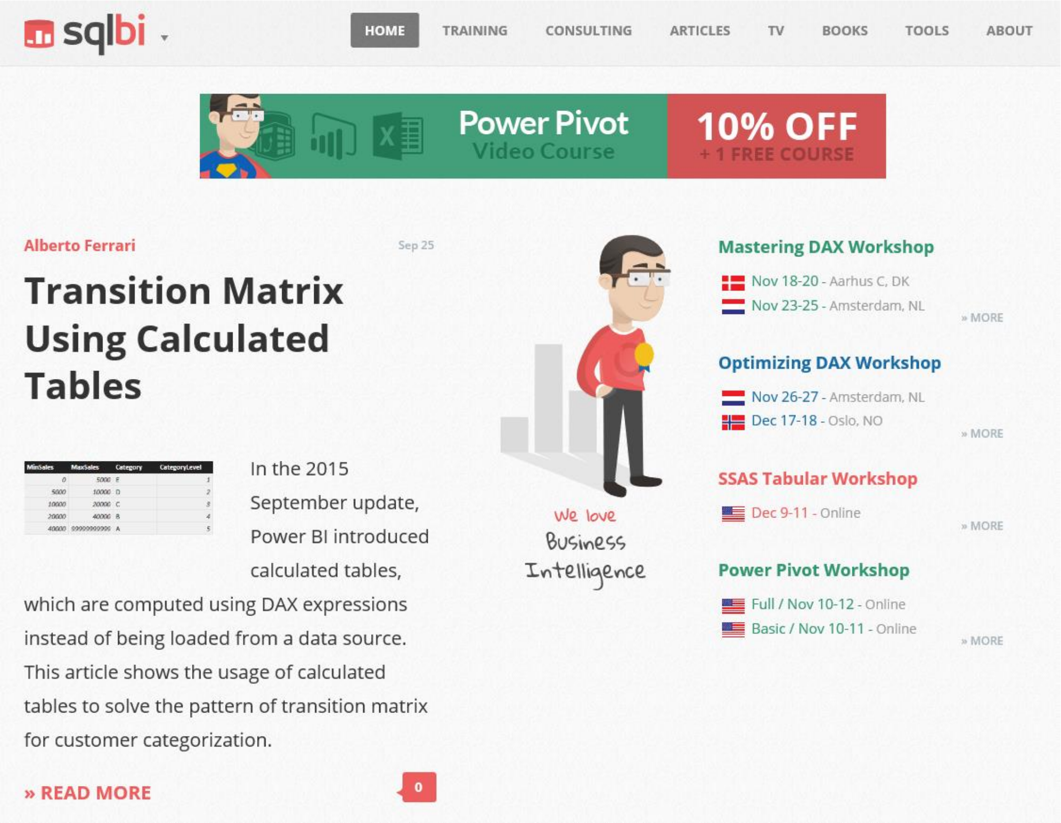
3. Bonus: Calculate YTD growth based on the two measures



Month	SalesAmount_m	SalesYTD	SalesAmount_mLastYear	SalesYTDLastYear	Growth
⊟ 2015	524	524	330	330	58,79%
201501	200	200			
201502	66	266	318	318	-16,35%
201503		266		318	-16,35%
201504		266		318	-16,35%
201505	240	506		318	59,12%
201506		506		318	59,12%
201507		506		318	59,12%
201508		506	12	330	53,33%
201509	18	524		330	58,79%
201510		524		330	58,79%
201511		524		330	58,79%
201512		524		330	58,79%
Grand Total	524	524	330	330	58,79%

Links / DAX 2016

Links



Alberto Ferrari Sep 25

Transition Matrix Using Calculated Tables

MinSales	MaxSales	Category	CategoryLevel
0	5000	E	1
5000	10000	D	2
10000	20000	C	3
20000	40000	B	4
40000	9999999999	A	5

In the 2015 September update, Power BI introduced calculated tables, which are computed using DAX expressions instead of being loaded from a data source. This article shows the usage of calculated tables to solve the pattern of transition matrix for customer categorization.

» READ MORE 0

Mastering DAX Workshop

Nov 18-20 - Aarhus C, DK
Nov 23-25 - Amsterdam, NL
» MORE

Optimizing DAX Workshop

Nov 26-27 - Amsterdam, NL
Dec 17-18 - Oslo, NO
» MORE

SSAS Tabular Workshop

Dec 9-11 - Online
» MORE

Power Pivot Workshop

Full / Nov 10-12 - Online
Basic / Nov 10-11 - Online
» MORE

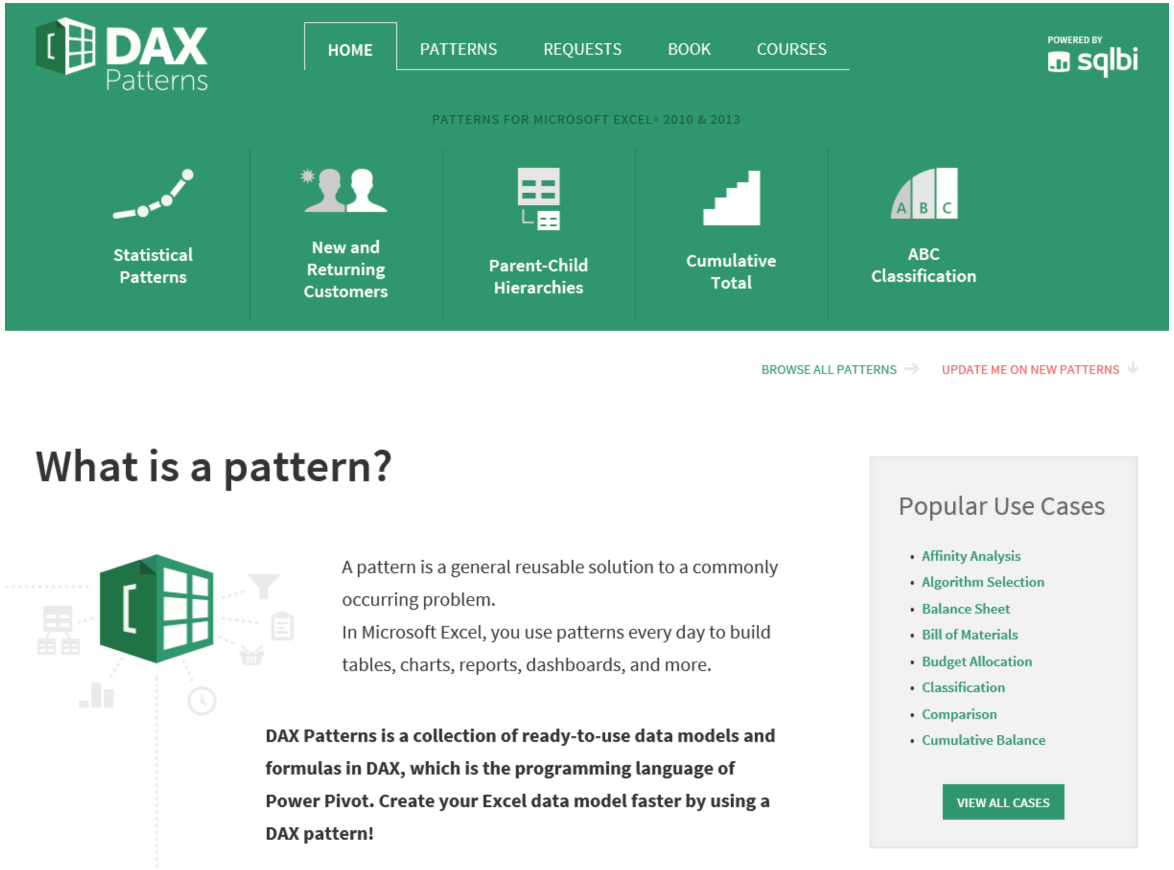
http://www.sqlbi.com/

Articles:

- Use cases
- Optimization

Managed by Marco Russo & Alberto Ferrari

Links



PATTERNS FOR MICROSOFT EXCEL® 2010 & 2013

What is a pattern?

A pattern is a general reusable solution to a commonly occurring problem.
In Microsoft Excel, you use patterns every day to build tables, charts, reports, dashboards, and more.

DAX Patterns is a collection of ready-to-use data models and formulas in DAX, which is the programming language of Power Pivot. Create your Excel data model faster by using a DAX pattern!

Popular Use Cases

- Affinity Analysis
- Algorithm Selection
- Balance Sheet
- Bill of Materials
- Budget Allocation
- Classification
- Comparison
- Cumulative Balance

http://www.daxpatterns.com/

DAX examples:

- Statistical pattern
- Classification
- Handling different granularities

Managed by Marco Russo & Alberto Ferrari

DAX 2016

New functionnalities are coming with SQL Server 2016 / Excel 2016 / Power BI Desktop

Modeling:

- Many 2 many
- BI Directional Cross Filtering

Functions:

- MEDIAN(), UNION(), DATEDIFF(),…

Variables:

- Little Revolution !
- Work as a real variable, evaluated only one time and never change, the filter context doesn’t matter on it
- Will be much easier, especially for querying

Lien : http://www.wiseowl.co.uk/blog/s2501/ssas-tabular-2016-new.htm

DAX 2016 - Variables

If you want the overall population without variables when querying WITHOUT variables:

```
DEFINE
MEASURE City[TotalPop]= calculate(sum(city[population]),ALL(City))
EVALUATE
ADDCOLUMNS(city,"TotalPop",City[TotalPop])
```

=

```
EVALUATE
VAR
        TotalPop=sum(city[population])
RETURN
ADDCOLUMNS(City,"TotalPop",TotalPop)
```

No more headache with evaluation contexts ☺

Références

Articles

http://www.sqlbi.com/

Marco Russo, Alberto Ferrari

Patterns

http://www.daxpatterns.com/

Marco Russo, Alberto Ferrari

Training

Mastering DAX Workshop

http://www.sqlbi.com/training/mastering-dax/

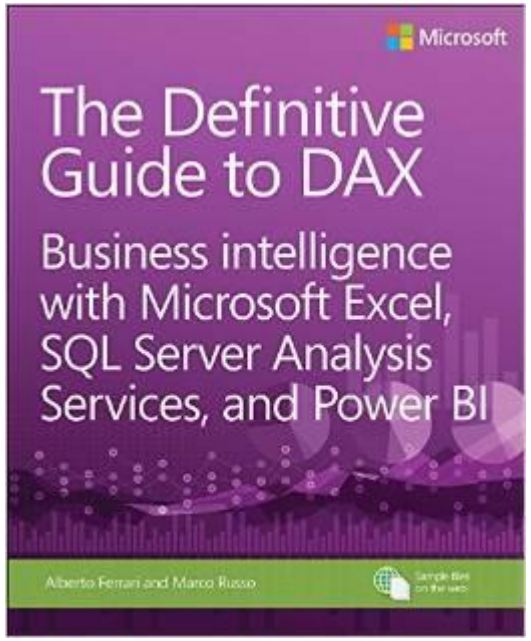
Marco Russo, Alberto Ferrari

Book

The Definitive Guide to DAX

http://www.sqlbi.com/books/the-definitive-guide-to-dax/

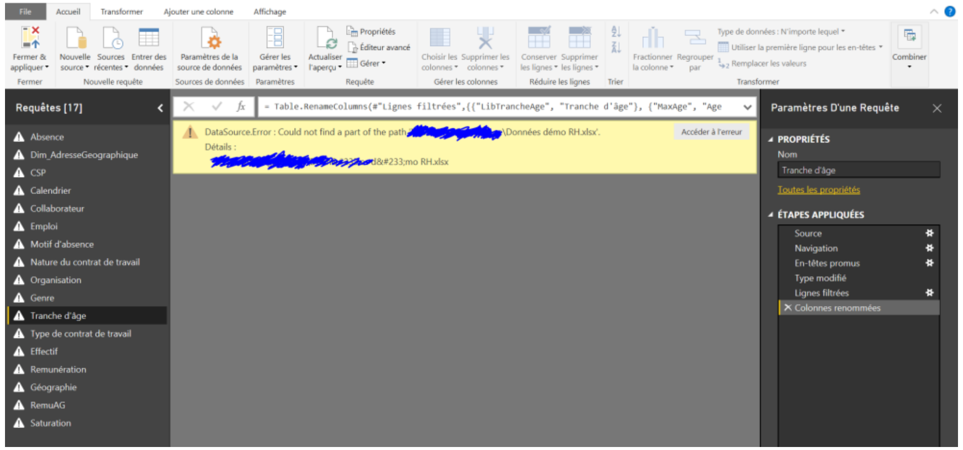
Marco Russo, Alberto Ferrari


Microsoft
The Definitive Guide to DAX
Business intelligence with Microsoft Excel, SQL Server Analysis Services, and Power BI
Alberto Ferrari and Marco Russo
Sample files on the web

M

Power BI – Power Query

- Mid 90s : mini-movement where **business** folks wrote and developed software → **Model Oriented** architecture
- They could not understand the generated code
- Died slowly in the 2000s until the 2015 **Power Query** tool created by Microsoft
- Still not really useful for now, but getting more and more apreciated since mid 2016 and **Power BI**
- Replace VBA in some cases



Requêtes [17]

- Absence
- Dim_AdresseGeographique
- CSP
- Calendrier
- Collaborateur
- Emploi
- Motif d'absence
- Nature du contrat de travail
- Organisation
- Genre
- Tranche d'âge
- Type de contrat de travail
- Effectif
- Remunération
- Géographie
- RemuAG
- Saturation

```
= Table.RenameColumns(#"Lignes filtrées",{{"LibTrancheAge", "Tranche d'âge"}, {"MaxAge", "Age
```

DataSource.Error : Could not find a part of the path [redacted]\Données démo RH.xlsx'.
Détails :
[redacted]d&#233;mo RH.xlsx

Accéder à l'erreur

Paramètres D'une Requête

PROPRIÉTÉS

Nom: Tranche d'âge

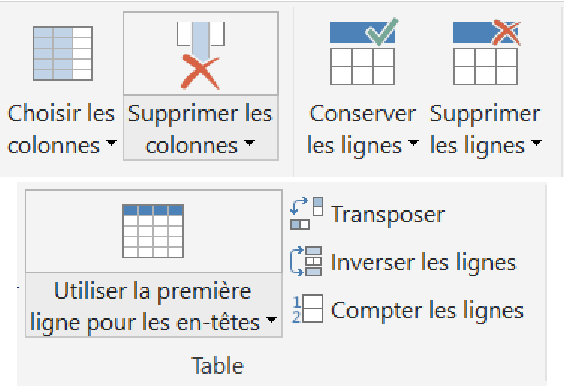
Toutes les propriétés

ÉTAPES APPLIQUÉES

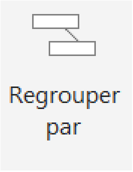
- Source
- Navigation
- En-têtes promus
- Type modifié
- Lignes filtrées
- Colonnes renommées

Few useful functions

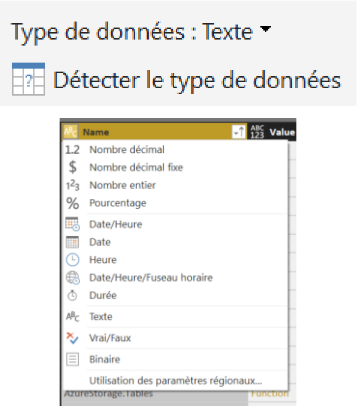
Working on lines/columns



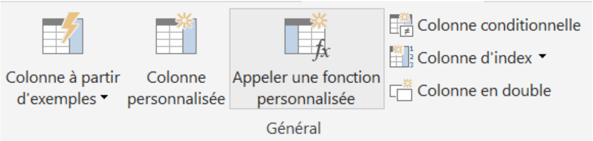
Group by



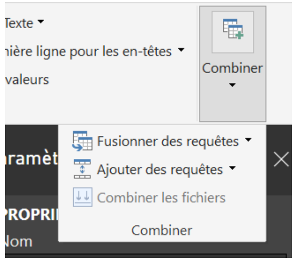
Data type



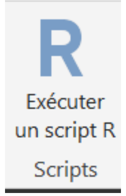
Add columns



Merge / Append

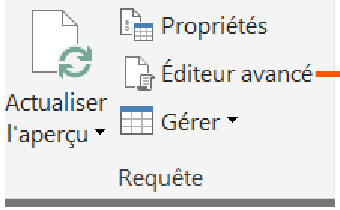


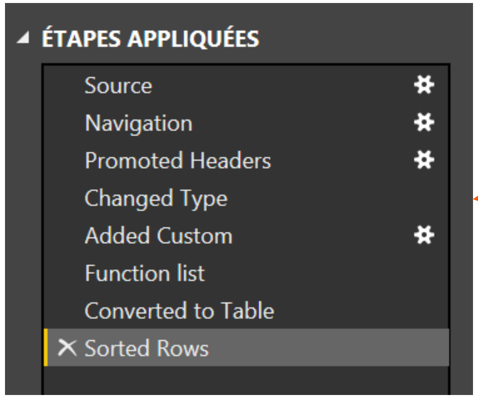
R stuff

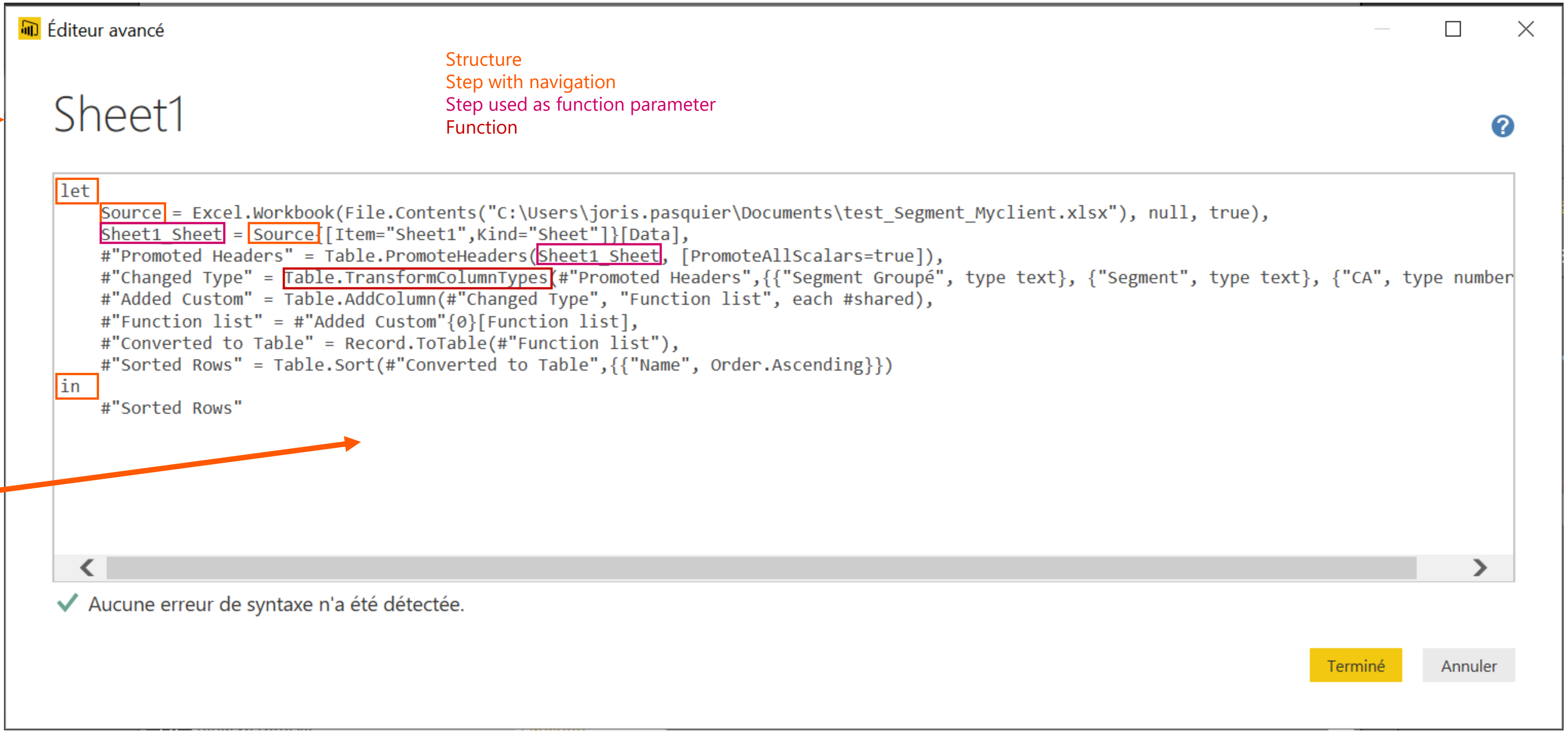


Advanced editor – code reading

Code structure







Structure
Step with navigation
Step used as function parameter
Function

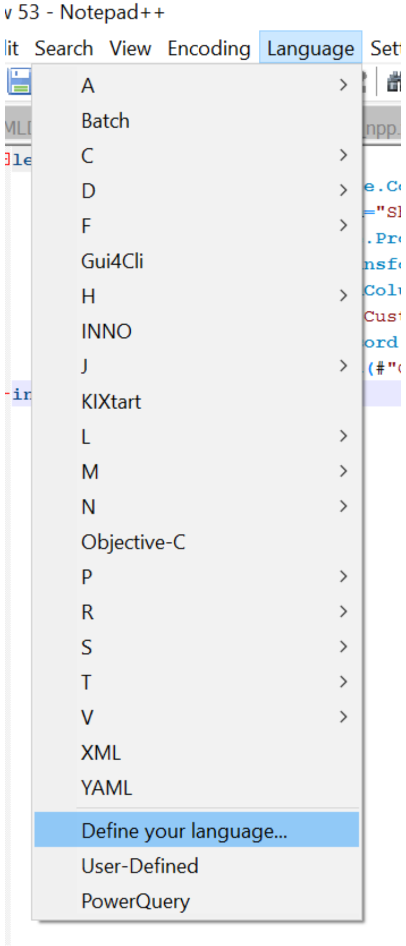
```
let
    Source = Excel.Workbook(File.Contents("C:\Users\joris.pasquier\Documents\test_Segment_Myclient.xlsx"), null, true),
    Sheet1_Sheet = Source{[Item="Sheet1",Kind="Sheet"]}[Data],
    #"Promoted Headers" = Table.PromoteHeaders(Sheet1_Sheet, [PromoteAllScalars=true]),
    #"Changed Type" = Table.TransformColumnTypes(#"Promoted Headers",{{"Segment Groupé", type text}, {"Segment", type text}, {"CA", type number
    #"Added Custom" = Table.AddColumn(#"Changed Type", "Function list", each #shared),
    #"Function list" = #"Added Custom"{0}[Function list],
    #"Converted to Table" = Record.ToTable(#"Function list"),
    #"Sorted Rows" = Table.Sort(#"Converted to Table",{{"Name", Order.Ascending}})
in
    #"Sorted Rows"
```

✓ Aucune erreur de syntaxe n'a été détectée.

avanade

Sexyer editor

- Download the file here
- Open **Notepad++**
- Go to language, define your language
- Import the file, then save as « PowerQuery »
- Link

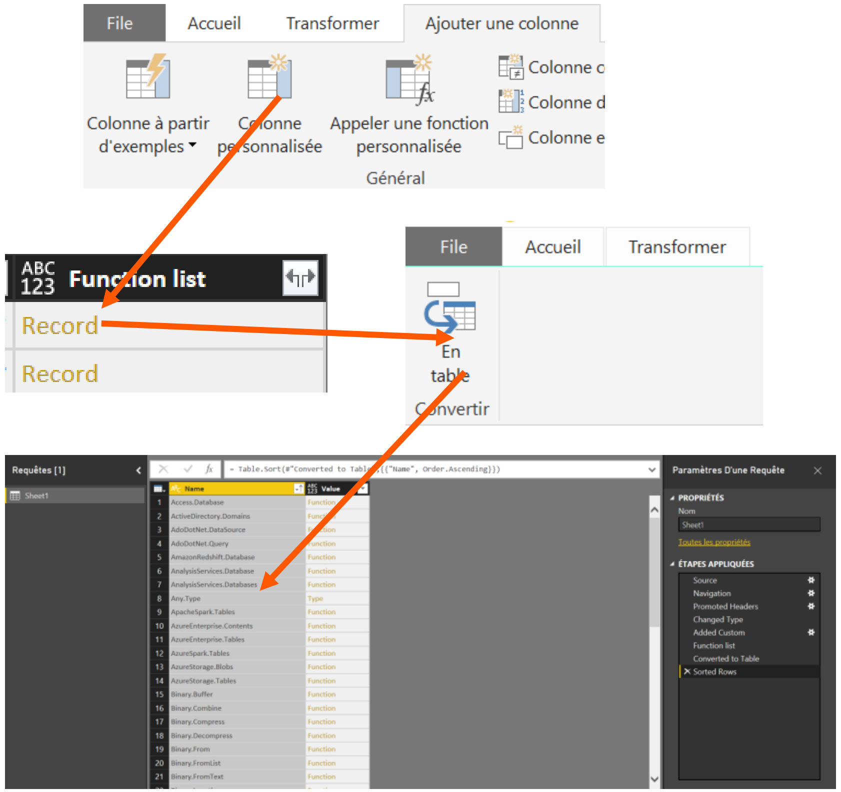


Full list of functions

Full list of function:

- add custom column =#shared,
- Click on the first Record
- then « to table »,
- Can get detail of each function by clicking on « Function »

There are not only functions !



```
= Table.Sort(#"Converted to Table",{{"Name", Order.Ascending}})
```

	Name	Value
1	Access.Database	Function
2	ActiveDirectory.Domains	Function
3	AdoDotNet.DataSource	Function
4	AdoDotNet.Query	Function
5	AmazonRedshift.Database	Function
6	AnalysisServices.Database	Function
7	AnalysisServices.Databases	Function
8	Any.Type	Type
9	ApacheSpark.Tables	Function
10	AzureEnterprise.Contents	Function
11	AzureEnterprise.Tables	Function
12	AzureSpark.Tables	Function
13	AzureStorage.Blobs	Function
14	AzureStorage.Tables	Function
15	Binary.Buffer	Function
16	Binary.Combine	Function
17	Binary.Compress	Function
18	Binary.Decompress	Function
19	Binary.From	Function
20	Binary.FromList	Function
21	Binary.FromText	Function

Paramètres D'une Requête

PROPRIÉTÉS

Nom: Sheet1

Toutes les propriétés

ÉTAPES APPLIQUÉES

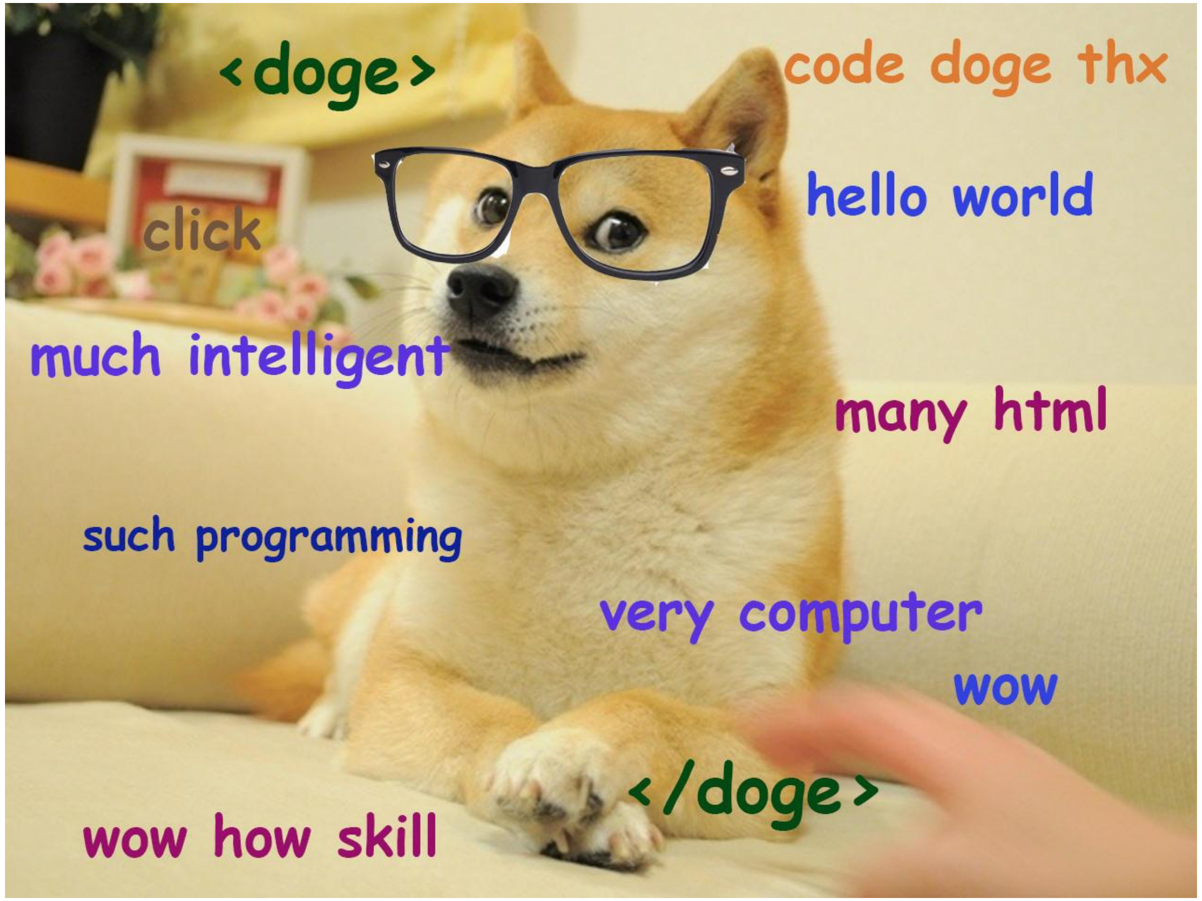
- Source
- Navigation
- Promoted Headers
- Changed Type
- Added Custom
- Function list
- Converted to Table
- Sorted Rows

2	GoogleAnalytics.Accounts	Function

avanade

More functions

https://github.com/tycho01/pquery

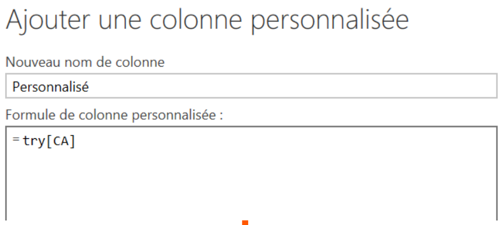


Example list

- Error handling: http://markvsql.com/2015/01/power-query-decathlon-beginner-08-error-handling/
- Examples site : https://blog.crossjoin.co.uk
- TextDelimiters : https://blog.crossjoin.co.uk/2017/04/25/using-text-betweendelimiters-to-extract-urls-from-a-web-page-in-power-bipower-query-m/
- Lists : https://blog.crossjoin.co.uk/2017/01/22/the-list-m-functions-and-the-equationcriteria-argument/
- Date hierarchies : https://blog.crossjoin.co.uk/2016/12/16/power-bi-model-size-bloat-and-auto-datetime-tables/
- SQL parameter : https://blog.crossjoin.co.uk/2016/12/11/passing-parameters-to-sql-queries-with-value-nativequery-in-power-query-and-power-bi/
- Data catalog : https://blog.crossjoin.co.uk/2016/11/29/sharing-power-query-queries-with-azure-data-catalog/
- Flow : https://blog.crossjoin.co.uk/2016/11/13/calling-microsoft-flow-from-power-query-and-power-bi/
- Query to function: https://www.mattmasson.com/2014/11/converting-a-query-to-a-function-in-power-query/
- Stacking : http://excel-inside.pro/blog/2015/11/16/stacking-non-nested-groups-of-repeating-columns-in-power-query/
- Header to data : http://excel-inside.pro/blog/2015/11/12/using-the-header-of-the-report-as-the-data-for-table-columns-in-power-query/
- Extract nested ppt workbook : http://www.excelandpowerbi.com/?p=326
- Replace vba by M example : http://www.cathyastuce.com/powerbi/power-query/692-remplacer-vba-par-powerquery.html

Error Handling : knowledge

Understanding the error



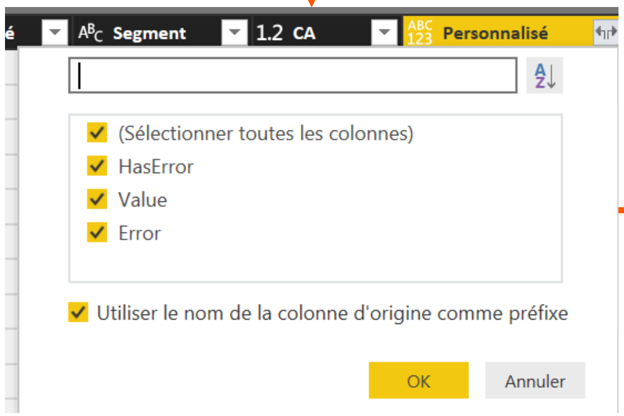
Ajouter une colonne personnalisée

Nouveau nom de colonne

Personnalisé

Formule de colonne personnalisée :

```
= try[CA]
```



☑ (Sélectionner toutes les colonnes)

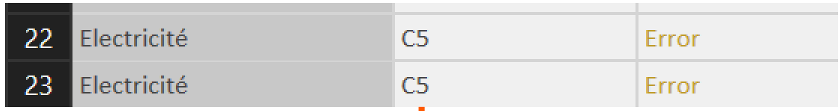
☑ HasError

☑ Value

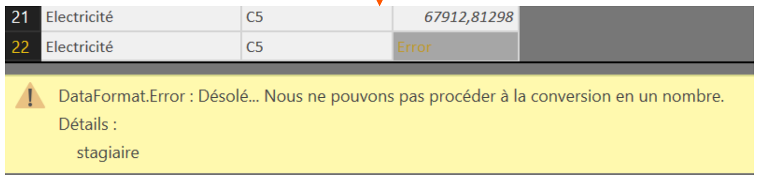
☑ Error

☑ Utiliser le nom de la colonne d'origine comme préfixe

What is an error ?



22	Electricité	C5	Error
23	Electricité	C5	Error

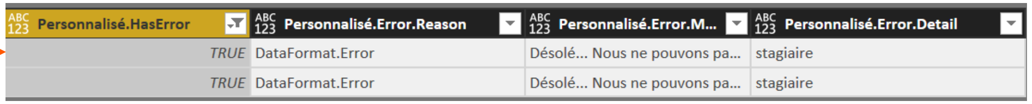


21	Electricité	C5	67912,81298
22	Electricité	C5	Error

DataFormat.Error : Désolé… Nous ne pouvons pas procéder à la conversion en un nombre.

Détails :

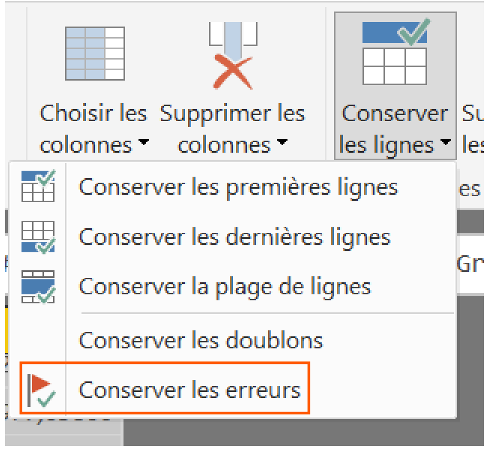
stagiaire

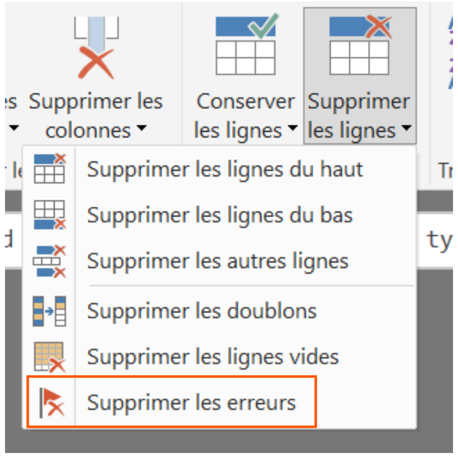


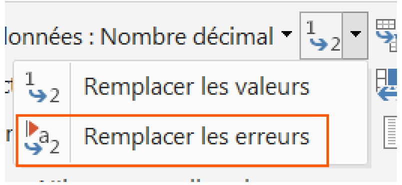
Personnalisé.HasError	Personnalisé.Error.Reason	Personnalisé.Error.M...	Personnalisé.Error.Detail
TRUE	DataFormat.Error	Désolé… Nous ne pouvons pa...	stagiaire
TRUE	DataFormat.Error	Désolé… Nous ne pouvons pa...	stagiaire

Error Handling : actions

Handling the errors

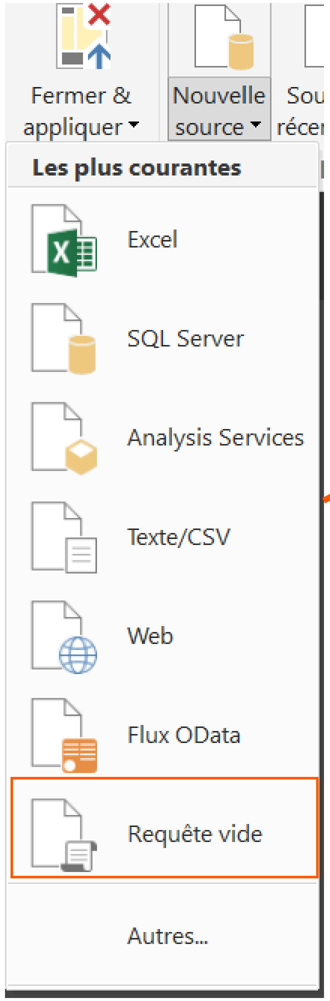



Supprimer les colonnes
Conserver les lignes
Supprimer les lignes
Supprimer les lignes du haut
Supprimer les lignes du bas
Supprimer les autres lignes
Supprimer les doublons
Supprimer les lignes vides
Supprimer les erreurs

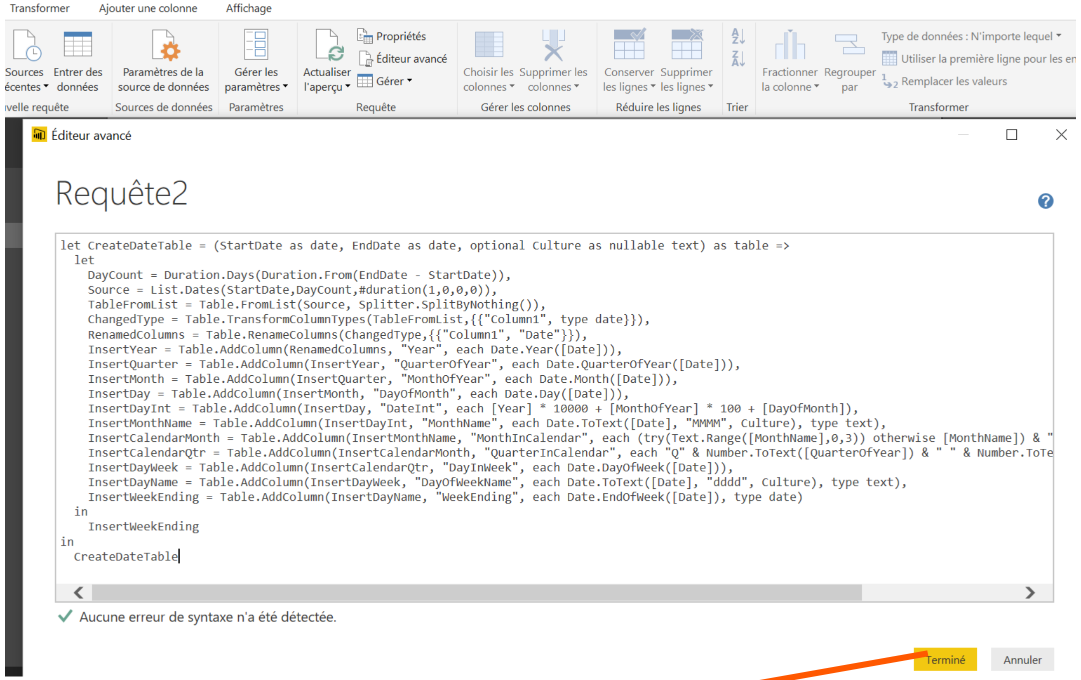


Calendar : M function

Create an empty request



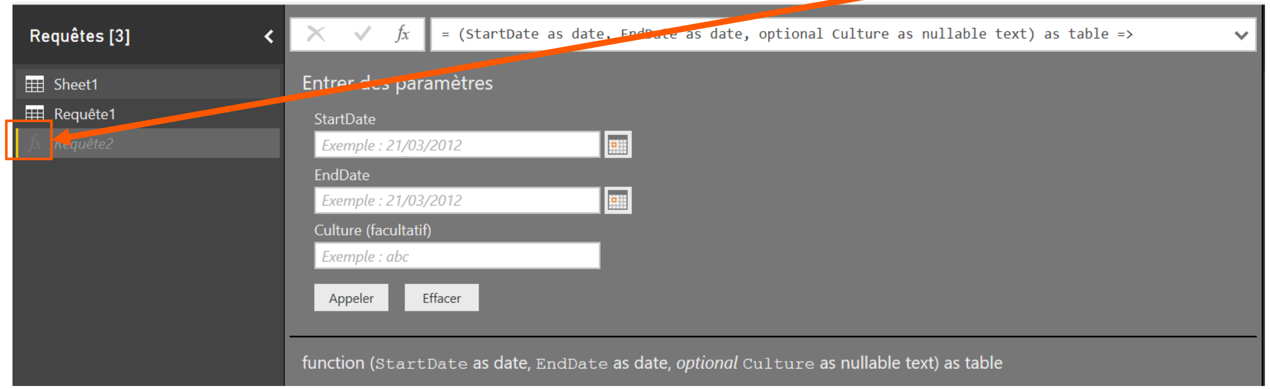
Paste the existing code



Requête2

```
let CreateDateTable = (StartDate as date, EndDate as date, optional Culture as nullable text) as table =>
  let
    DayCount = Duration.Days(Duration.From(EndDate - StartDate)),
    Source = List.Dates(StartDate,DayCount,#duration(1,0,0,0)),
    TableFromList = Table.FromList(Source, Splitter.SplitByNothing()),
    ChangedType = Table.TransformColumnTypes(TableFromList,{{"Column1", type date}}),
    RenamedColumns = Table.RenameColumns(ChangedType,{{"Column1", "Date"}}),
    InsertYear = Table.AddColumn(RenamedColumns, "Year", each Date.Year([Date])),
    InsertQuarter = Table.AddColumn(InsertYear, "QuarterOfYear", each Date.QuarterOfYear([Date])),
    InsertMonth = Table.AddColumn(InsertQuarter, "MonthOfYear", each Date.Month([Date])),
    InsertDay = Table.AddColumn(InsertMonth, "DayOfMonth", each Date.Day([Date])),
    InsertDayInt = Table.AddColumn(InsertDay, "DateInt", each [Year] * 10000 + [MonthOfYear] * 100 + [DayOfMonth]),
    InsertMonthName = Table.AddColumn(InsertDayInt, "MonthName", each Date.ToText([Date], "MMMM", Culture), type text),
    InsertCalendarMonth = Table.AddColumn(InsertMonthName, "MonthInCalendar", each (try(Text.Range([MonthName],0,3)) otherwise [MonthName]) & "
    InsertCalendarQtr = Table.AddColumn(InsertCalendarMonth, "QuarterInCalendar", each "Q" & Number.ToText([QuarterOfYear]) & " " & Number.ToTe
    InsertDayWeek = Table.AddColumn(InsertCalendarQtr, "DayInWeek", each Date.DayOfWeek([Date])),
    InsertDayName = Table.AddColumn(InsertDayWeek, "DayOfWeekName", each Date.ToText([Date], "dddd", Culture), type text),
    InsertWeekEnding = Table.AddColumn(InsertDayName, "WeekEnding", each Date.EndOfWeek([Date]), type date)
  in
    InsertWeekEnding
in
  CreateDateTable
```

✓ Aucune erreur de syntaxe n'a été détectée.



= (StartDate as date, EndDate as date, optional Culture as nullable text) as table =>

Entrer des paramètres

StartDate: Exemple : 21/03/2012

EndDate: Exemple : 21/03/2012

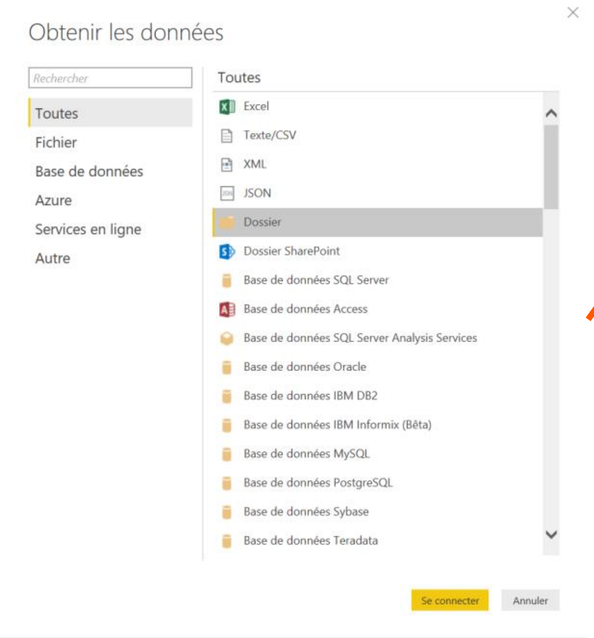
Culture (facultatif): Exemple : abc

function (StartDate as date, EndDate as date, *optional* Culture as nullable text) as table

Use the function to range your own calendar

Managing CSV : combining

Choose folder



Obtenir les données

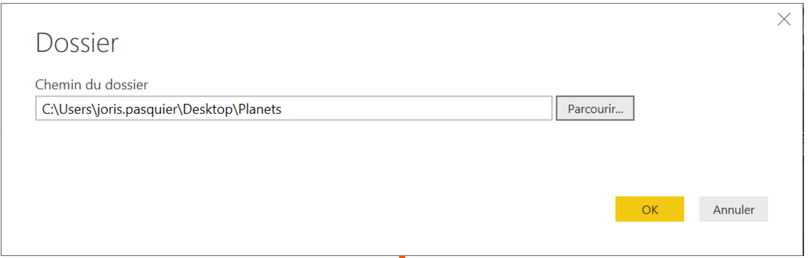
Rechercher

- Toutes
- Fichier
- Base de données
- Azure
- Services en ligne
- Autre

Toutes

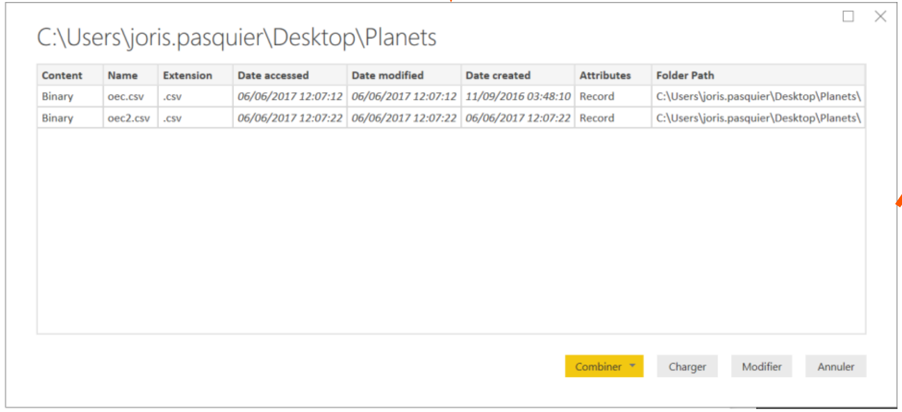
- Excel
- Texte/CSV
- XML
- JSON
- Dossier
- Dossier SharePoint
- Base de données SQL Server
- Base de données Access
- Base de données SQL Server Analysis Services
- Base de données Oracle
- Base de données IBM DB2
- Base de données IBM Informix (Bêta)
- Base de données MySQL
- Base de données PostgreSQL
- Base de données Sybase
- Base de données Teradata

Select your folder, then it's magic !



Dossier

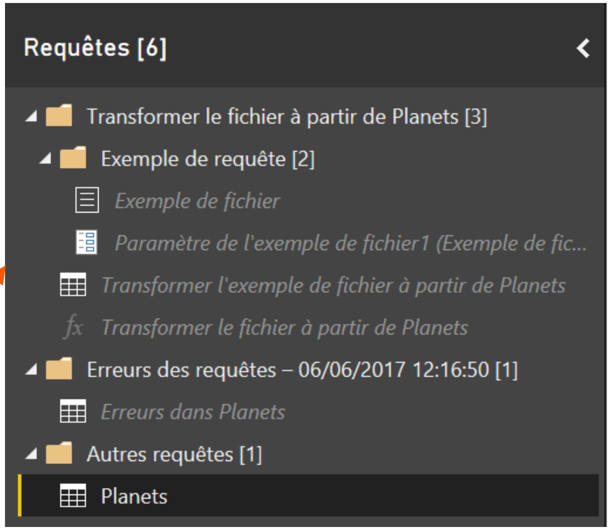
Chemin du dossier: C:\Users\joris.pasquier\Desktop\Planets



C:\Users\joris.pasquier\Desktop\Planets

Content	Name	Extension	Date accessed	Date modified	Date created	Attributes	Folder Path
Binary	oec.csv	.csv	*06/06/2017 12:07:12*	*06/06/2017 12:07:12*	*11/09/2016 03:48:10*	Record	C:\Users\joris.pasquier\Desktop\Planets\
Binary	oec2.csv	.csv	*06/06/2017 12:07:22*	*06/06/2017 12:07:22*	*06/06/2017 12:07:22*	Record	C:\Users\joris.pasquier\Desktop\Planets\

Everything is automatically created

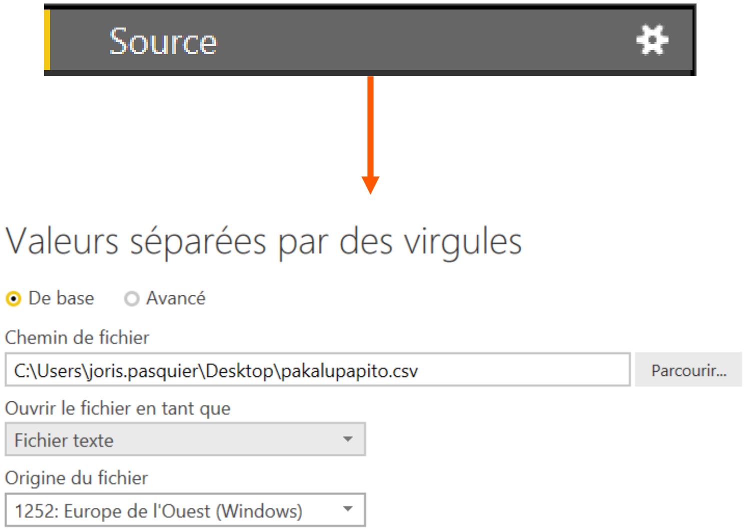


Requêtes [6]

- Transformer le fichier à partir de Planets [3]
  - Exemple de requête [2]
    - *Exemple de fichier*
    - *Paramètre de l'exemple de fichier1 (Exemple de fic...*
  - *Transformer l'exemple de fichier à partir de Planets*
  - *Transformer le fichier à partir de Planets*
- Erreurs des requêtes – 06/06/2017 12:16:50 [1]
  - *Erreurs dans Planets*
- Autres requêtes [1]
  - Planets

Managing CSV : handle format error

Read the file as txt



Valeurs séparées par des virgules

● De base ○ Avancé

Chemin de fichier

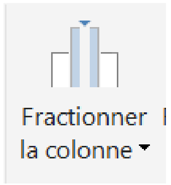
C:\Users\joris.pasquier\Desktop\pakalupapito.csv [Parcourir...]

Ouvrir le fichier en tant que

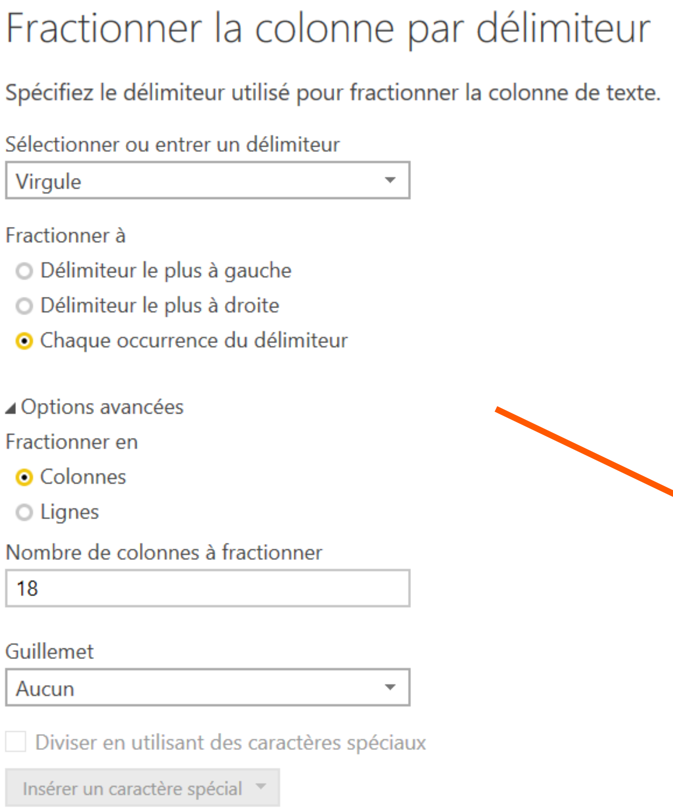
Fichier texte

Origine du fichier

1252: Europe de l'Ouest (Windows)



Read it your own way !



Fractionner la colonne par délimiteur

Spécifiez le délimiteur utilisé pour fractionner la colonne de texte.

Sélectionner ou entrer un délimiteur

Virgule

Fractionner à

○ Délimiteur le plus à gauche

○ Délimiteur le plus à droite

● Chaque occurrence du délimiteur

◢ Options avancées

Fractionner en

● Colonnes

○ Lignes

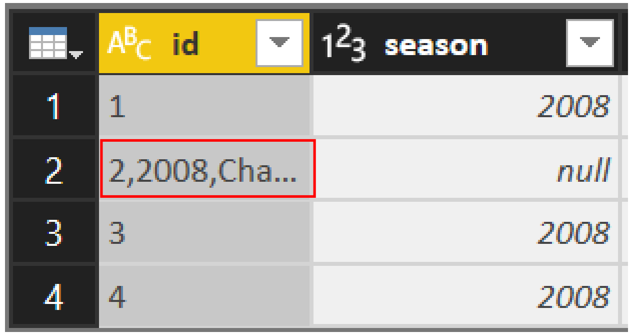
Nombre de colonnes à fractionner

18

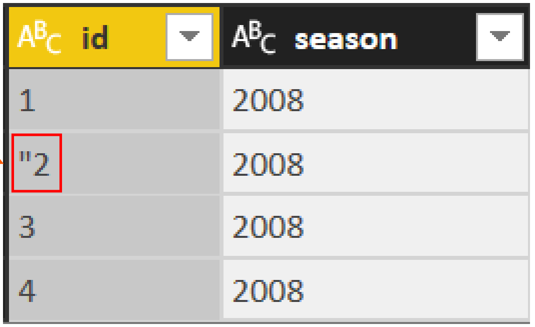
Guillemet

Aucun

□ Diviser en utilisant des caractères spéciaux



	id	season
1	1	*2008*
2	2,2008,Cha...	*null*
3	3	*2008*
4	4	*2008*



id	season
1	2008
"2	2008
3	2008
4	2008

Now just remove the text delimiters…

Exercice : Indian Premier League

30 minutes

1. **Combining the deliveries files**

- Combine the two delivery.csv files
- Describe the code
- Handle the errors using automatic detection

2. **Integrating the matches**

- Import matches.csv
- Detect and handle errors (try try[])

Références

Sites

https://blog.crossjoin.co.uk/

Chris Webb

https://www.mattmasson.com/

Matt Masson

http://www.cathyastuce.com/powerbi/power-query

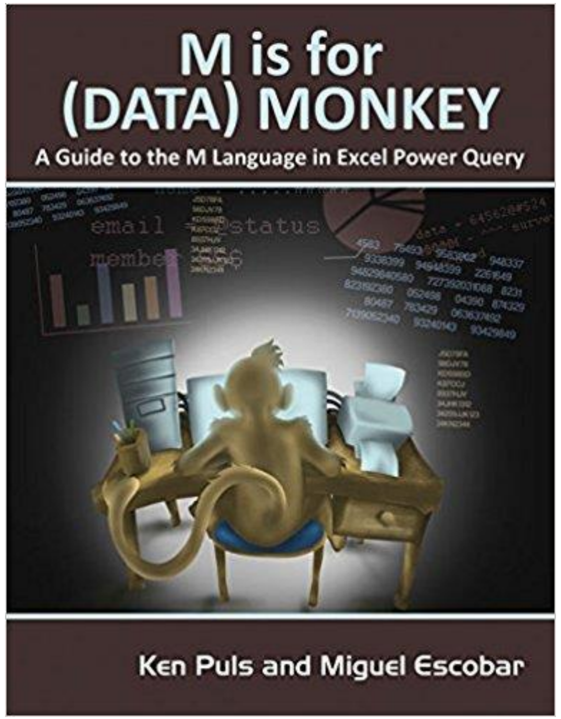
Catherine Monier

Book

M is for Data Monkeys

https://www.powerquery.training/book-files/

Ken Puls, Miguel Escobar


M is for
(DATA) MONKEY
A Guide to the M Language in Excel Power Query
Ken Puls and Miguel Escobar

Contact

Joris Pasquier

joris.pasquier@avanade.com

Billy Phengdy

billy.phengdy@avanade.com

avanade