www.sqlbi.com

Microsoft Partner
Microsoft
Gold Data Analytics
Gold Data Platform

SSAS MAESTRO

MVP Microsoft Most Valuable Professional

From 0 to DAX

Alberto Ferrari
Senior consultant @SQLBI
alberto.ferrari@sqlbi.com

sqlbi

We write

Books

We teach

Courses

MASTERING DAX Workshop

OPTIMIZING DAX Workshop

DAX PATTERNS

DATA MODELING For Power BI

MASTERING TABULAR VIDEO COURSE

We provide

Consulting

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

We are recognized

BI Experts

Microsoft Partner
Microsoft

Gold Data Analytics
Gold Data Platform

Microsoft CERTIFIED Technology Specialist

Microsoft CERTIFIED IT Professional

Microsoft CERTIFIED Trainer

Microsoft MVP Most Valuable Professional

SSAS MAESTRO

sqlbi

www.sqlbi.com

Check our Articles, Books, Videos, and Courses on

www.sqlbi.com

sqlbi DAX GUIDE DAX.do DAX PATTERNS DAX FORMATTER

The DAX language

- Language of
  - Power BI
  - Analysis Services Tabular
  - Power Pivot
- DAX is simple, but it is not easy
- New programming concepts and patterns

Introduction to the DAX language

Introduction to DAX

What is DAX?

- Programming language
  - Power BI
  - Analysis Services Tabular
  - Power Pivot
- Resembles Excel
  - Because it was born with PowerPivot
  - Important differences
    - No concept of «row» and «column»
    - Different type system
- Many new functions
- Designed for data models and business

Functional language

DAX is a functional language, the execution flows with function calls, here is an example of a DAX formula.

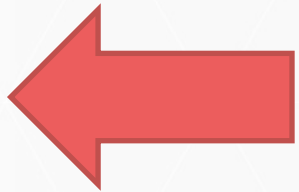
```
=SUMX (
    FILTER (
        VALUES ( 'Date'[Year] ),
        'Date'[Year] < 2005
    ),
    IF (
        'Date'[Year] >= 2000,
        [Sales Amount] * 100,
        [Sales Amount] *  90
    )
)
```

If it is not formatted, it is not DAX

Code formatting is of paramount importance in DAX.

```
=SUMX(FILTER(VALUES('Date'[Year]),'Date'[Year]<2005),IF('Date'[Year]>=2000,
[Sales Amount]*100,[Sales Amount]*90)
```

```
=SUMX (
    FILTER (
        VALUES ( 'Date'[Year] ),
        'Date'[Year] < 2005
    ),
    IF (
        'Date'[Year] >= 2000,
        [Sales Amount] * 100,
        [Sales Amount] *  90

)
```



DAX FORMATTER

www.daxformatter.com

sqlbi

DAX data types

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

Format strings are not data types!

DAX type handling

- Operator Overloading
  - Operators are not strongly typed
  - The result depends on the inputs
- Example:
  - "5" + "4" = 9
  - 5 & 4 = "54"
- Conversion happens when needed
  - Pay attention to undesired conversions

DateTime

- Floating point value
- Integer part
  - Number of days after December, 30, 1899
- Decimal part
  - Seconds: 1 / (24 * 60 * 60)
- DateTime Expressions
  - Date + 1 = The day after
  - Date - 1 = The day before

Calculated columns

- Columns computed using DAX
- Always computed row by row
- Product[Price] means
  - The value of the Price column (explicit)
  - In the Product table (explicit, optional)
  - For the current row (implicit)
  - Different for each row

Column references

- The general format to reference a column
  - 'TableName'[ColumnName]
- Quotes can be omitted
  - If TableName does not contain spaces
  - Do it: omit quotes if there are no spaces in table name
- TableName can be omitted
  - Current table is searched for ColumnName
  - **Don’t do it, harder to understand**

Measures

- Written using DAX
- Do not work row by row
- Instead, use tables and aggregators
- Do not have the «current row» concept
- Examples
  - GrossMargin
    - is a calculated column
    - but can be a measure, too
  - GrossMargin %
    - must be a measure

Naming convention

- Measures should not belong to a table
  - Avoid table name
  - [Margin%] instead of Sales[Margin%]
  - Easier to move to another table
  - Easier to identify as a measure
- Use this syntax when to reference:
  - Columns → Table[Column]
  - Measures → [Measure]

Measures vs calculated columns

- Use a column when
  - Need to slice or filter on the value
- Use a measure
  - Calculate percentages
  - Calculate ratios
  - Need complex aggregations
- Space and CPU usage
  - Columns consume memory in the model
  - Measures consume CPU at query time

Aggregation functions

- Useful to aggregate values
  - SUM
  - AVERAGE
  - MIN
  - MAX
- Aggregate only one column
  - SUM (Orders[Price])
  - **SUM (Orders[Price] * Orders[Quantity])**

The «X» aggregation functions

- Iterators: useful to aggregate formulas
  - SUMX
  - AVERAGEX
  - MINX
  - MAXX
- Iterate over the table and evaluate the expression for each row
- Always receive two parameters
  - Table to iterate
  - Formula to evaluate for each row

Example of SUMX

For each row in the Sales table, evaluates the formula, then sum up all the results.
Inside the formula, there is a «current row».

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

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

1 x 15 = 15
2 x 13 = 26
4 x 11 = 44
8 x 9 = 72
16 x 7 = 112
32 x 5 = 160
64 x 3 = 192
128 x 1 = 128

Result = 749

SUM or SUMX?

Actually, SUM is nothing but syntax sugar for SUMX

```
--
--  This is the compact format for a SUM
--
SUM ( Sales[Quantity] )

--
--  Internally, this is translated into
--
SUMX (
    Sales,
    Sales[Quantity]
)
```

IN operator

- Verify if the result of an expression is included in a list of values:

```
Customer[State] IN { "WA", "NY", "CA" }
```

- It would require multiple OR conditions otherwise:

```
    Customer[State] = "WA"
|| Customer[State] = "NY"
|| Customer[State] = "CA"
```

The DIVIDE function

Divide is useful to avoid using IF inside an expression to check for zero denominators.
It is also faster than using IF.

```
IF (
    Sales[SalesAmount] <> 0,
    Sales[GrossMargin] / Sales[SalesAmount],
    0
)

You can write it better with DIVIDE

DIVIDE (
    Sales[GrossMargin],
    Sales[SalesAmount],
    0
)
```

Using variables

Very useful to avoid repeating subexpressions in your code.

```
VAR
    TotalQuantity = SUM ( Sales[Quantity] )

RETURN

    IF (
        TotalQuantity > 1000,
        TotalQuantity * 0.95,
        TotalQuantity * 1.25
    )
```

Relational functions

- RELATED
  - Follows relationships and returns the value of a column
- RELATEDTABLE
  - Follows relationships and returns all the rows in relationship with the current one
- It doesn’t matter how long the chain of relationships is
  - All the relationships must be in the same direction

Some functions return tables instead of values

Table Functions

Table functions

- Basic functions that work and/or return full tables
  - FILTER
  - ALL
  - VALUES / DISTINCT
  - RELATEDTABLE
  - ADDCOLUMNS / SUMMARIZE
- They are used very often in DAX – their knowledge is required
- Their result is often used in other functions
- They can be combined together to form complex expressions
- We will discover many other table functions later in the course

Filtering a table

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

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

Channel
- Internet
- Store

Color	Large	Small	Total
Green	192		192
Red	112	160	272
Total	**304**	**160**	**464**

The FILTER function

- FILTER
  - Adds a new condition
  - Restricts the number of rows of a table
  - Returns a table
  - Can be iterated by an «X» function
- Needs a table as input
- The input can be another FILTER

Ignoring filters

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

```
SUMX (
    ALL ( Orders ),
    Orders[Quantity] * Orders[Price]
)
```

Channel

- Internet
- Store

Color	Large	Small	**Total**
Green	749	749	749
Red	749	749	**749**
Total	**749**	**749**	**749**

The ALL function

- ALL
  - Returns all the rows of a table
  - Ignores any filter
  - Returns a table
  - That can be iterated by an «X» function
- Needs a table as input
- Can be used with a single column
  - **ALL (Customers[CustomerName])**
  - The result contains a table with one column

ALL with many columns

Returns a table with all the values of all the columns passed as parameters.

```
COUNTROWS (
    ALL (
        Orders[Channel],
        Orders[Color],
        Orders[Size]
    )
)
```

Columns of the same table

Mixing filters

- Table functions can be mixed
- Each one requires a table
- Each one returns a table
- FILTER (ALL (Table), Condition)
  - Puts a filter over the entire table
  - Ignores the current filter context

Mixing filters

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

```
SUMX (
    FILTER (
        ALL( Orders ),
        Orders[Channel]="Internet"
    ),
    Orders[Quantity] * Orders[Price]
)
```

Channel

- Internet
- Store

Color	Large	Small	**Total**
Green	592	592	592
Red	592	592	**592**
Total	**592**	**592**	**592**

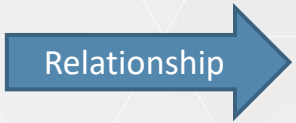
DISTINCT

Returns the unique values of a column, only the ones visible in the current filter context.

```
NumOfProducts :=

COUNTROWS (
    DISTINCT ( Product[ProductCode] )
)
```

How many values for a column?

Amount	ProductId
25.00	1
12.50	2
2.25	3
2.50	3
14.00	4
16.00	5



ProductId	Product
1	Coffee
2	Pasta
3	Tomato
BLANK	BLANK

Tables targets of a relationship might contain an additional blank row, created by DAX to guarantee referential integrity.

VALUES

Returns the unique values of a column, only the ones visible in the current filter context, including the additional blank row if it is visible in the filter context.

```
NumOfProducts :=

COUNTROWS (
    VALUES ( Product[ProductCode] )
)
```

ALLNOBLANKROW

ALL returns the additional blank row, if it exists. ALLNOBLANKROW omits it.

```
--
-- Returns only the existing products
--

= COUNTROWS (
    ALLNOBLANKROW ( Products[ProductKey] )
)
```

Counting different values

Product	CountRowsAll	CountRowsDistinct	CountRowsValues	CountRowsAllNoBlankRow
	4		1	3
Coffee	4	1	1	3
Pasta	4	1	1	3
Tomato	4	1	1	3
Total	**4**	**3**	**4**	**3**

Note the difference among

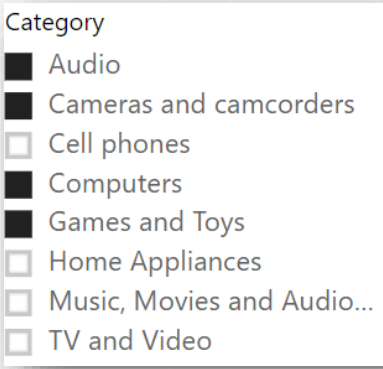
- DISTINCT
- VALUES
- ALL
- ALLNOBLANKROW

ALLSELECTED

ALLSELECTED returns the elements of a table as they are visible outside of the current visual, be either a pivot table in Excel or a visual in Power BI.

```
Pct All =

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

```
Pct AllSel =

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



Category	Sales Amount	Pct All	Pct AllSel
Cameras and camcorders	842,849,210.26	22.15%	47.25%
Computers	842,569,053.80	22.14%	47.24%
Audio	51,553,689.31	1.35%	2.89%
Games and Toys	46,769,456.76	1.23%	2.62%
Total	**1,783,741,410.13**	**46.87%**	**100.00%**

RELATEDTABLE

Returns a table with all the rows related with the current one.

```
NumOfProducts =

COUNTROWS (
    RELATEDTABLE ( Product )
)
```

ADDCOLUMNS

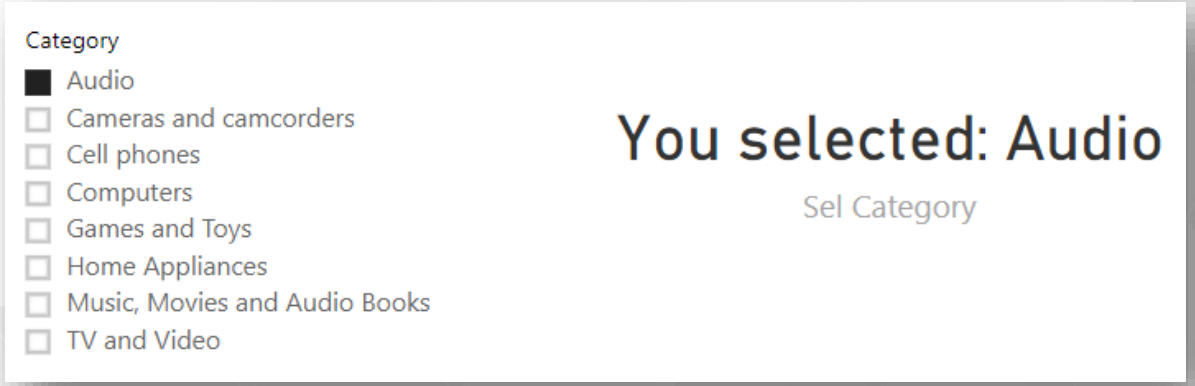
Adds one or more columns to a table expression, keeping all existing columns.
It is an iterator, therefore you can access columns of the iterated table

```
ColorsAndSales

ADDCOLUMNS (
    VALUES ( 'Product'[Color] ),
    "Sales",
        SUMX (
            RELATEDTABLE ( Sales ),
            Sales[Quantity] * Sales[Net Price]
        )
)
```

Tables with one row and one column

When a table contains ONE row and ONE column, you can treat is as a scalar value.

```
Sel Category :=

"You selected: " &
IF (
    HASONEVALUE ( 'Product Category'[Category] ),
    VALUES ( 'Product Category'[Category] ),
    "Multiple values"
)
```



SELECTEDVALUE

SELECTEDVALUE is a convenient function that simplifies retrieving the value of a column, when only one value is visible.

```
SELECTEDVALUE (
    'Product Category'[Category],
    "Multiple values"
)

Equivalent to:

IF (
    HASONEVALUE ( 'Product Category'[Category] ),
    VALUES ( 'Product Category'[Category] ),
    "Multiple values"
)
```

Table variables

A variable can contain either a scalar value or a table.
Using table variables greatly helps in splitting complex expressions.

```
VAR
    SalesGreaterThan10 = FILTER ( Sales, Sales[Quantity] > 10 )

RETURN

    SUMX (
        FILTER (
            SalesGreaterThan10,
            RELATED ( Product[Color] ) = "Red"
        ),
        Sales[Amount]
     )
```

Let us take a look at how DAX works

Evaluation Contexts

Evaluation contexts

- Evaluation contexts are the pillars of DAX
- Simple concepts, hard to learn
- At the beginning, they looks very easy
- Using them, complexity arises
- The devil is in the details

What is an evaluation context?

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

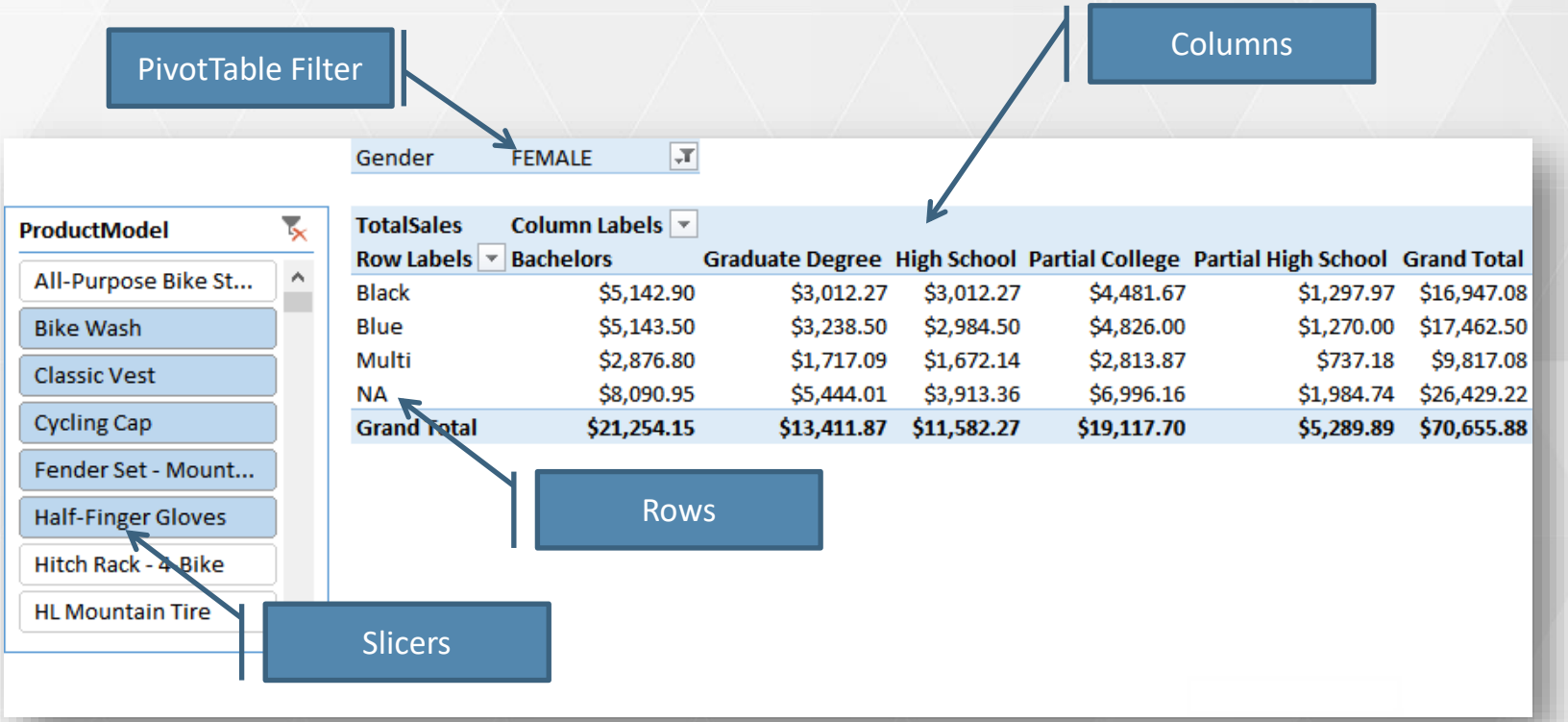
TotalSales
$29,358,677.22

Row Labels	TotalSales
Black	$8,838,411.96
Blue	$2,279,096.28
Multi	$106,470.74
NA	$435,116.69
Red	$7,724,330.52
Silver	$5,113,389.08
White	$5,106.32
Yellow	$4,856,755.63
Grand Total	**$29,358,677.22**

Numbers are sliced by color, i.e. the formula is NOT computing sum of sales, it is computing it for only a subset of the data model

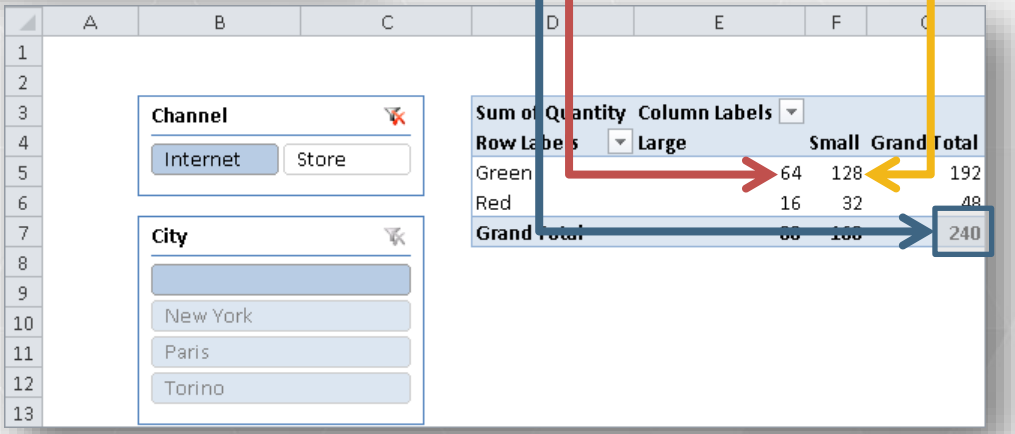
The value of a formula depends on its context

Filter context in a pivot table



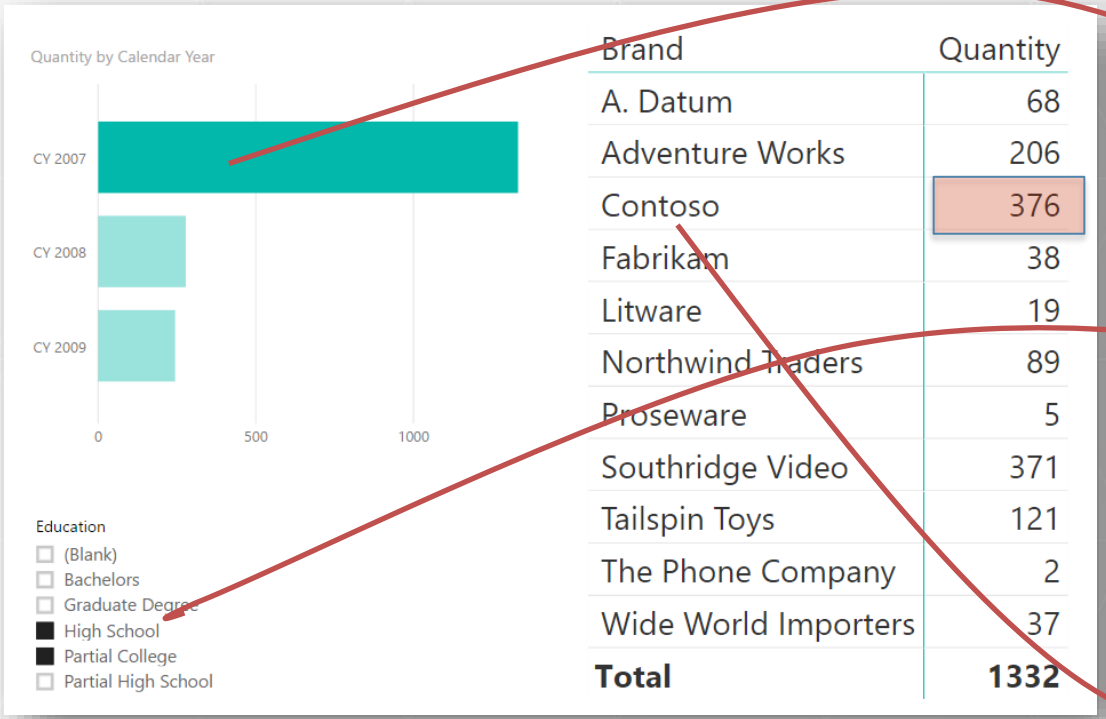
Example of a filter context

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

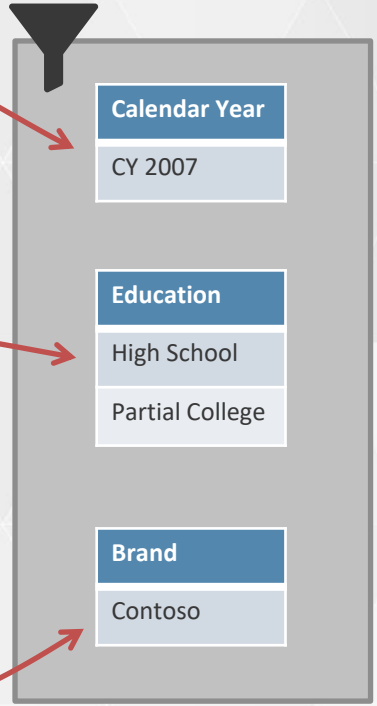


sqlbi

Filter context in Power BI



Brand	Quantity
A. Datum	68
Adventure Works	206
Contoso	376
Fabrikam	38
Litware	19
Northwind Traders	89
Proseware	5
Southridge Video	371
Tailspin Toys	121
The Phone Company	2
Wide World Importers	37
Total	**1332**



sqlbi

Filter context

- Defined by
  - Row Selection
  - Column Selection
  - Report Filters
  - Slicers Selection
- Rows outside of the filter context
  - Are not considered for the computation
- Defined automatically by the client, tool
- Can also be created with specific functions

Row context

- Defined by
  - Calculated column definition
    - Defined automatically for each row
  - Row Iteration functions
    - SUMX, AVERAGEX …
    - All «X» functions and iterators
    - Defined by the user formulas
- Needed to evaluate column values, it is the concept of “current row”

SUMX (Orders, Orders[Quantity]*Orders[Price])

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

16x7=112

32x5=160

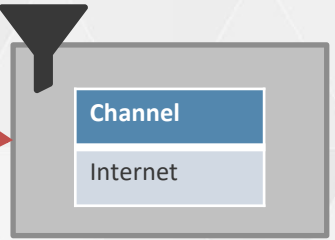
64x3=192

128x1=128

SUM = 592

Channel
- Internet
- Store

Color	Large	Small	**Total**
Green	192	128	**320**
Red	112	160	**272**
Total	**304**	**288**	592



sqlbi

There are always two contexts

- Filter context
  - Filters tables
  - Might be empty
    - All the tables are visible
    - But this never happens in the real world
- Row context
  - Iterates rows
  - For the rows active in the filter context
  - Might be empty
    - There is no iteration running
- Both are «evaluation contexts»

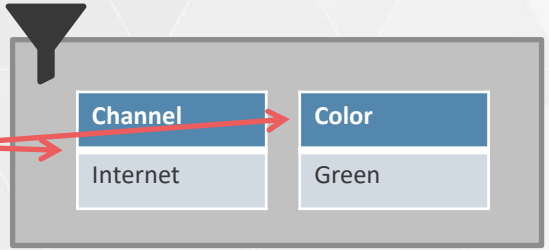
Filtering a table

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

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

Channel
■ Internet
□ Store

Color	Large	Small	**Total**
Green	192		192
Red	112	160	**272**
Total	**304**	**160**	**464**



sqlbi

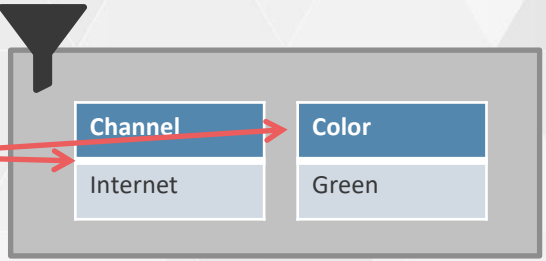
Ignoring filters

City	Channel	Color	Size	Quantity	Price
Paris	Store	Red	Large	1	15
Paris	Store	Red	Small	2	13
Torino	Store	Green	Large	4	11
New York	Store	Green	Small	8	9
	Internet	Red	Large	16	7
	Internet	Red	Small	32	5
	Internet	Green	Large	64	3
	Internet	Green	Small	128	1

```
SUMX (
    ALL ( Orders ),
    Orders[Quantity] * Orders[Price]
)
```

Channel
- Internet
- Store

Color	Large	Small	Total
Green	749	749	749
Red	749	749	**749**
Total	**749**	**749**	**749**

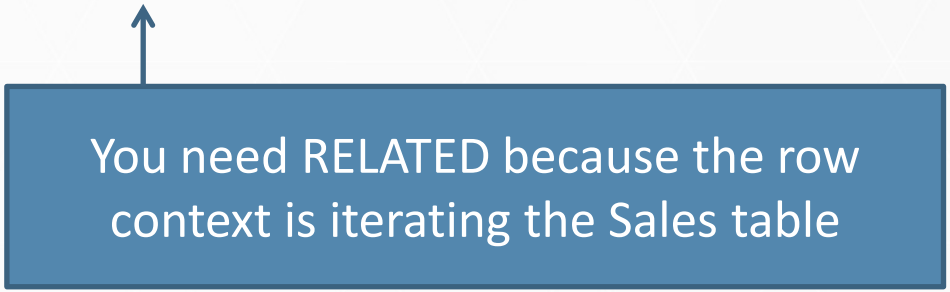


sqlbi

Using RELATED in a row context

Starting from a row context, you can use RELATED to access columns in related tables.

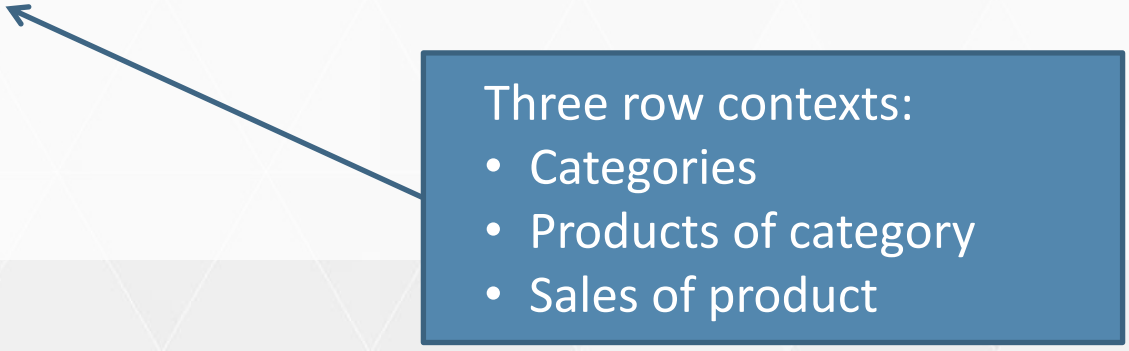
```
SUMX (
    Sales,
    Sales[Quantity]
        * RELATED ( Products[ListPrice] )
        * RELATED ( Categories[Discount] )
)
```



Nesting row contexts

Row contexts can be nested, on the same or on different tables.

```
SUMX (
    Categories,
    SUMX (
        RELATEDTABLE ( Products ),
        SUMX (
            RELATEDTABLE ( Sales )
            ( Sales[Quantity] * Products[ListPrice] ) * Categories[Discount]
        )
    )
)
```



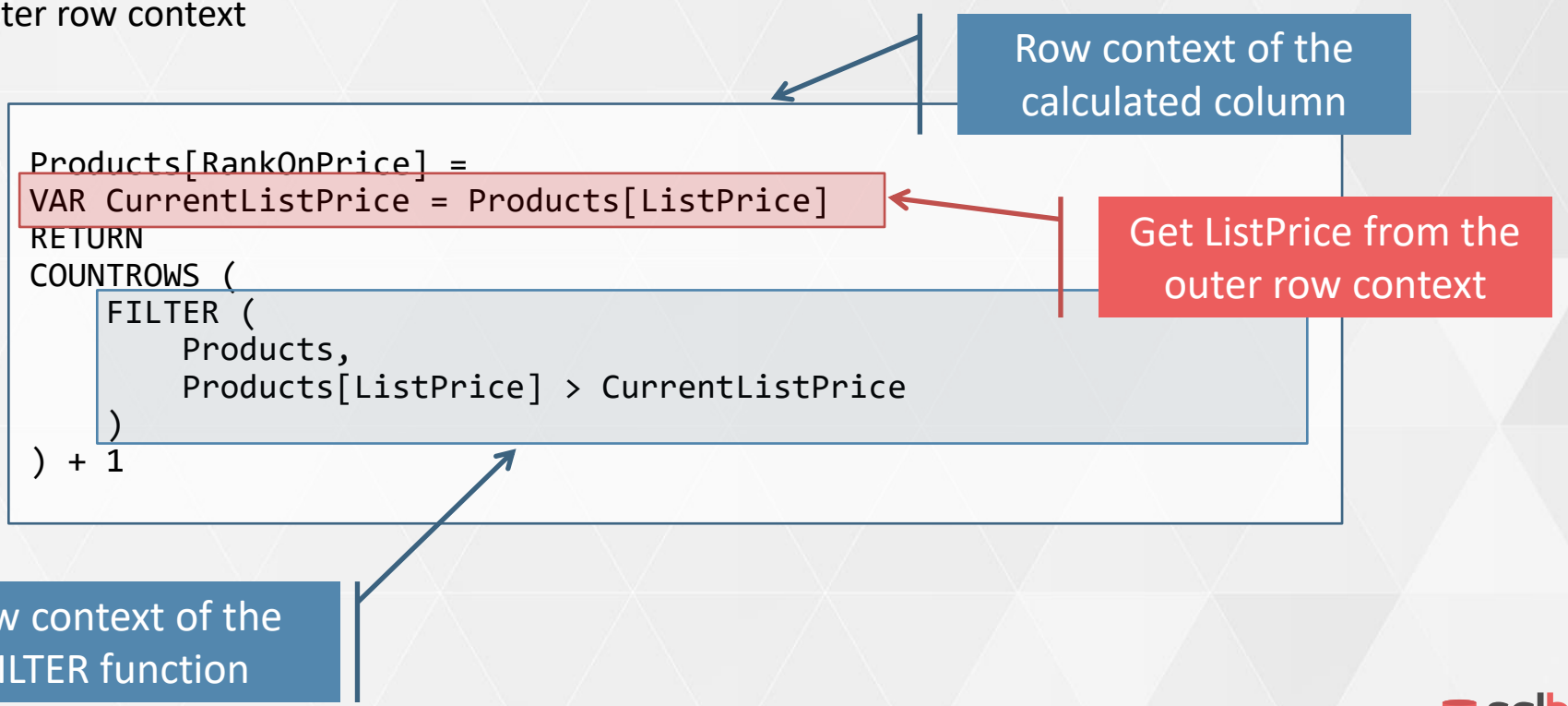
Ranking by price

- Create a calculated column
- Ranks products by list price
- Most expensive product is ranked 1

ProductKey	ProductName	ListPrice	ListPriceRank
314	Road-150 Red, 56	$3,578.27	1
313	Road-150 Red, 52	$3,578.27	1
312	Road-150 Red, 48	$3,578.27	1
311	Road-150 Red, 44	$3,578.27	1
310	Road-150 Red, 62	$3,578.27	1
347	Mountain-100 Si...	$3,399.99	2
346	Mountain-100 Si...	$3,399.99	2
345	Mountain-100 Si...	$3,399.99	2
344	Mountain-100 Si...	$3,399.99	2
351	Mountain-100 B...	$3,374.99	3
350	Mountain-100 B...	$3,374.99	3
349	Mountain-100 B...	$3,374.99	3
348	Mountain-100 B...	$3,374.99	3
380	Road-250 Black, ...	$2,443.35	4
378	Road-250 Black, ...	$2,443.35	4

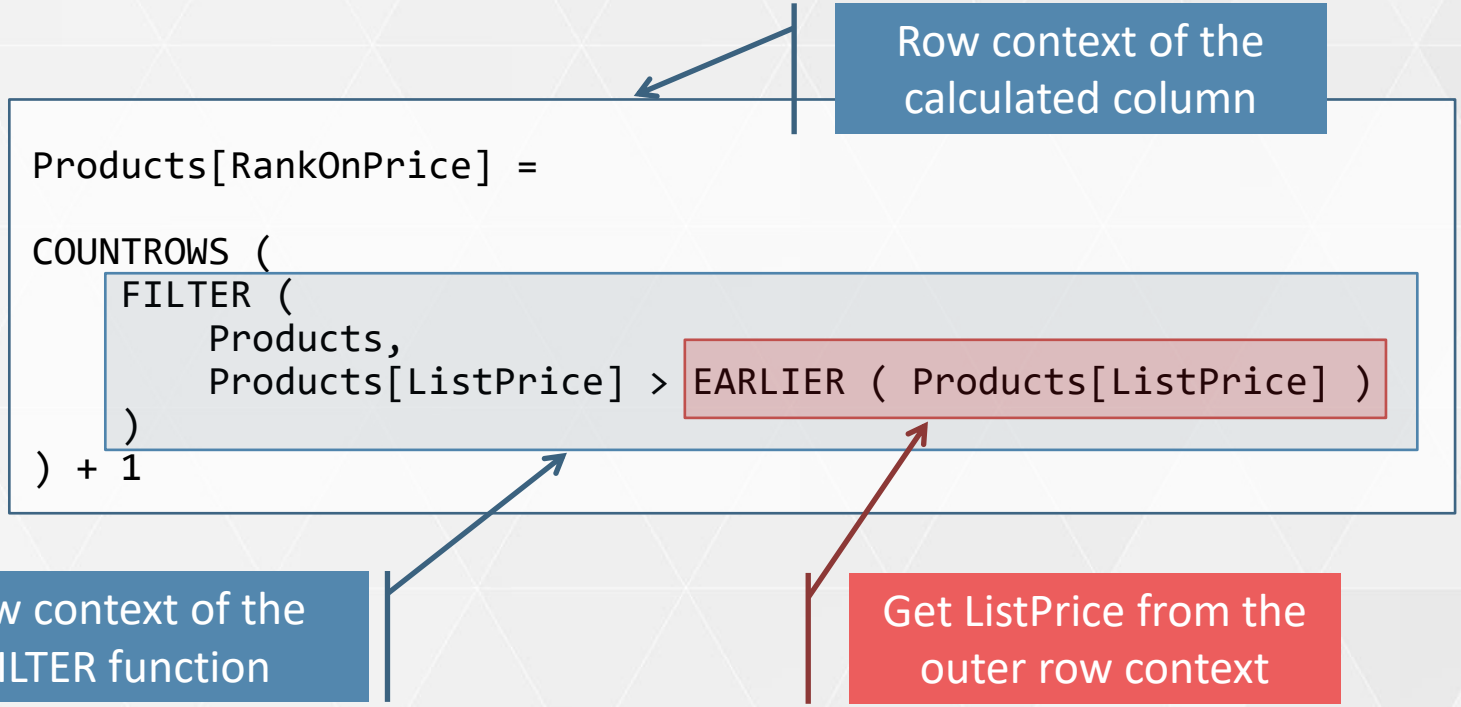
Nesting row contexts

When you nest row contexts on the same table, you can use a variable to save the value of the outer row context



Nesting row contexts

As an alternative, in version of DAX that do not support variables, you can use the EARLIER function



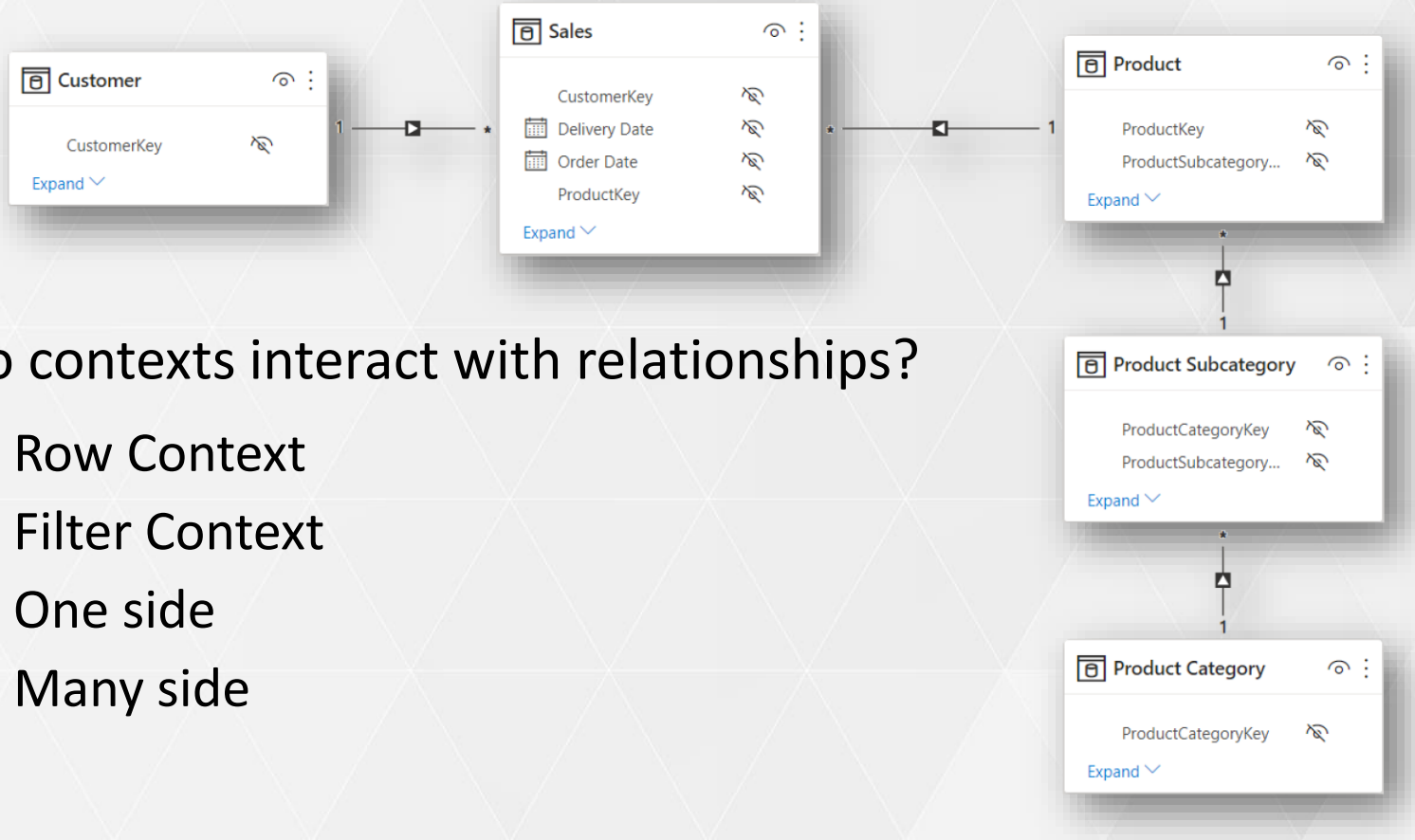
Computing the correct rank

The correct solution requires to rank over the different prices, not the different products.
ALL with a column becomes very handy here.

```
Products[RankOnPrice] =

VAR CurrentListPrice = Products[ListPrice]
VAR AllPrices = ALL ( Products[ListPrice] )

RETURN

COUNTROWS (
    FILTER (
        AllPrices,
        Products[ListPrice] > CurrentListPrice
    )
) + 1
```

Evaluation contexts and relationships

Filters and relationships

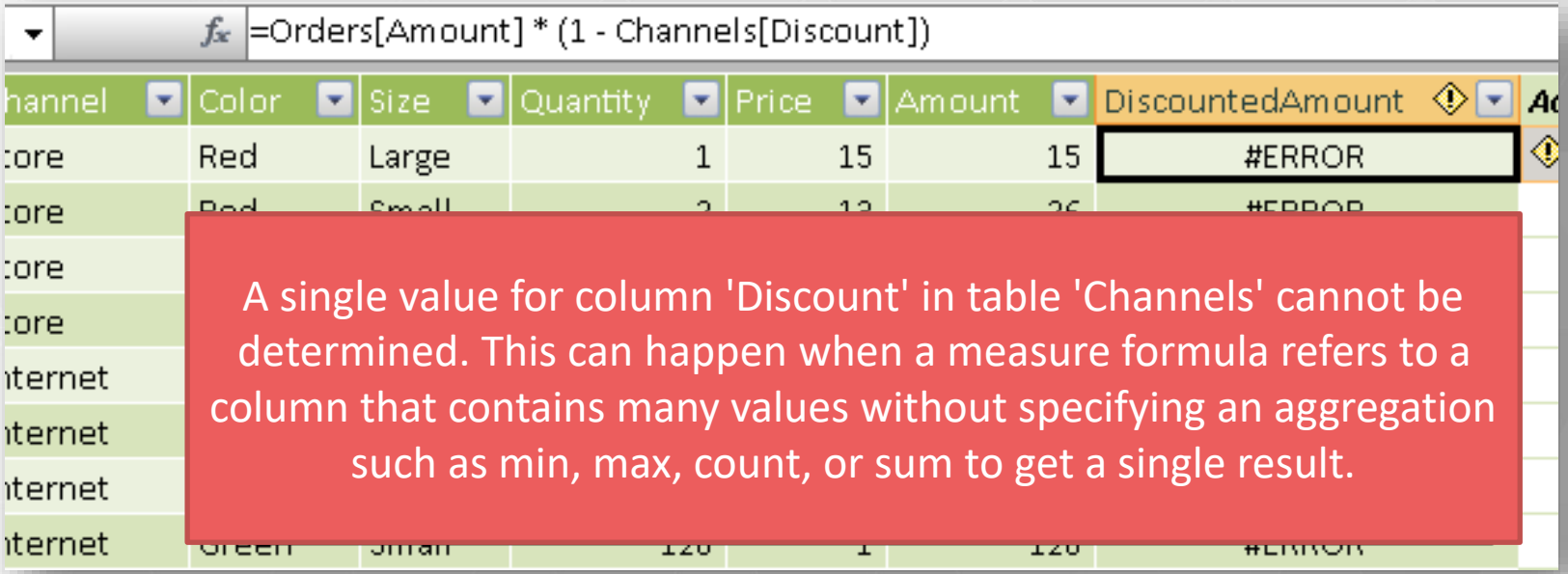


- Do contexts interact with relationships?
  - Row Context
  - Filter Context
  - One side
  - Many side

sqlbi

Row context – multiple tables

Row Context does not propagate over relationships



RELATED

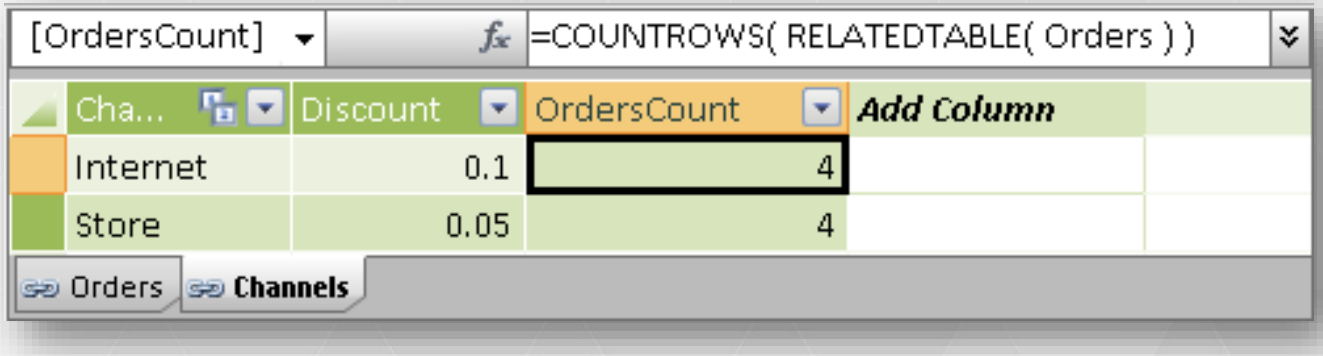
- RELATED (table[column])
  - Opens a new row context on the target table
  - Following relationships

[DiscountedAmo ▾ | fx | =Orders[Amount] * (1 - RELATED(Channels[Discount]))

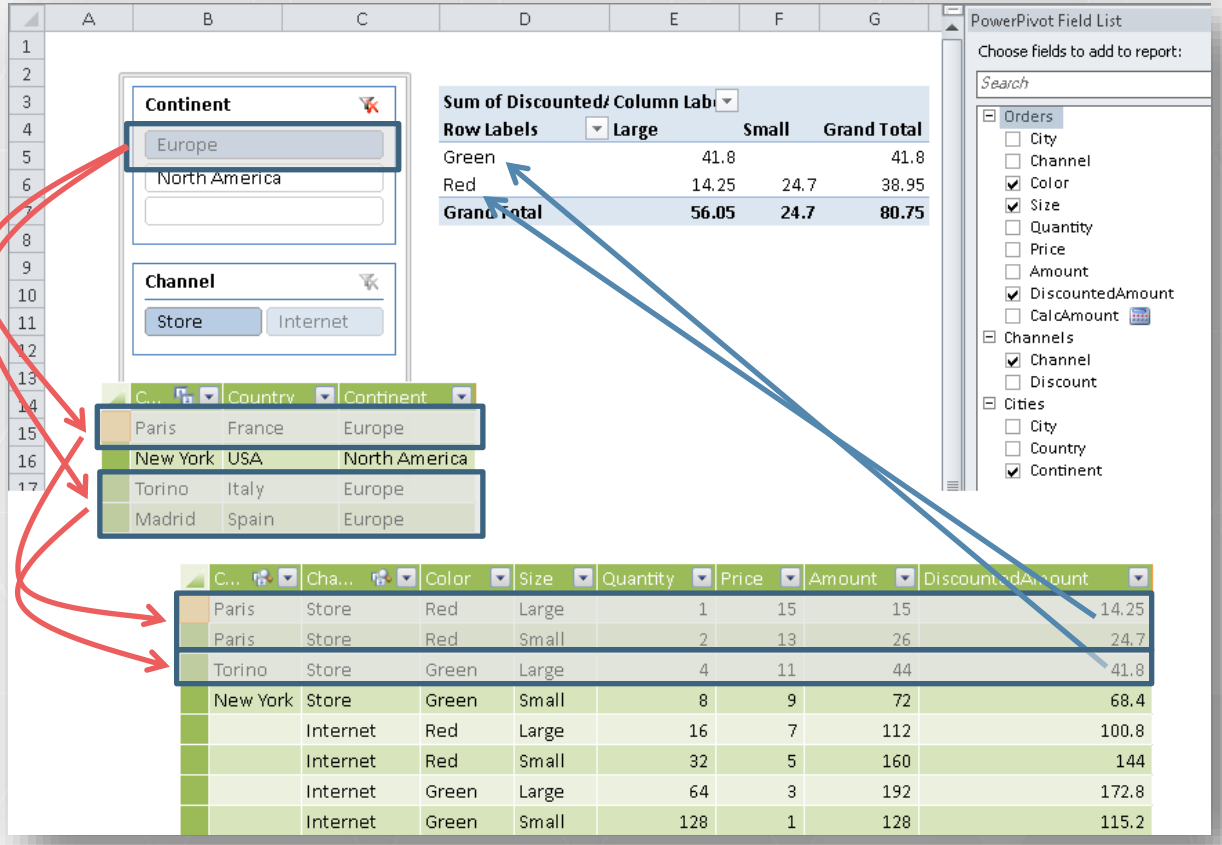
City	Cha...	Color	Size	Quantity	Price	Amount	DiscountedAmount
Paris	Store	Red	Large	1	15	15	14.25
Paris	Store	Red	Small	2	13	26	24.7
Torino	Store	Green	Large	4	11	44	41.8
New York	Store	Green	Small	8	9	72	68.4
	Internet	Red	Large	16	7	112	100.8
	Internet	Red	Small	32	5	160	144
	Internet	Green	Large	64	3	192	172.8
	Internet	Green	Small	128	1	128	115.2

RELATEDTABLE

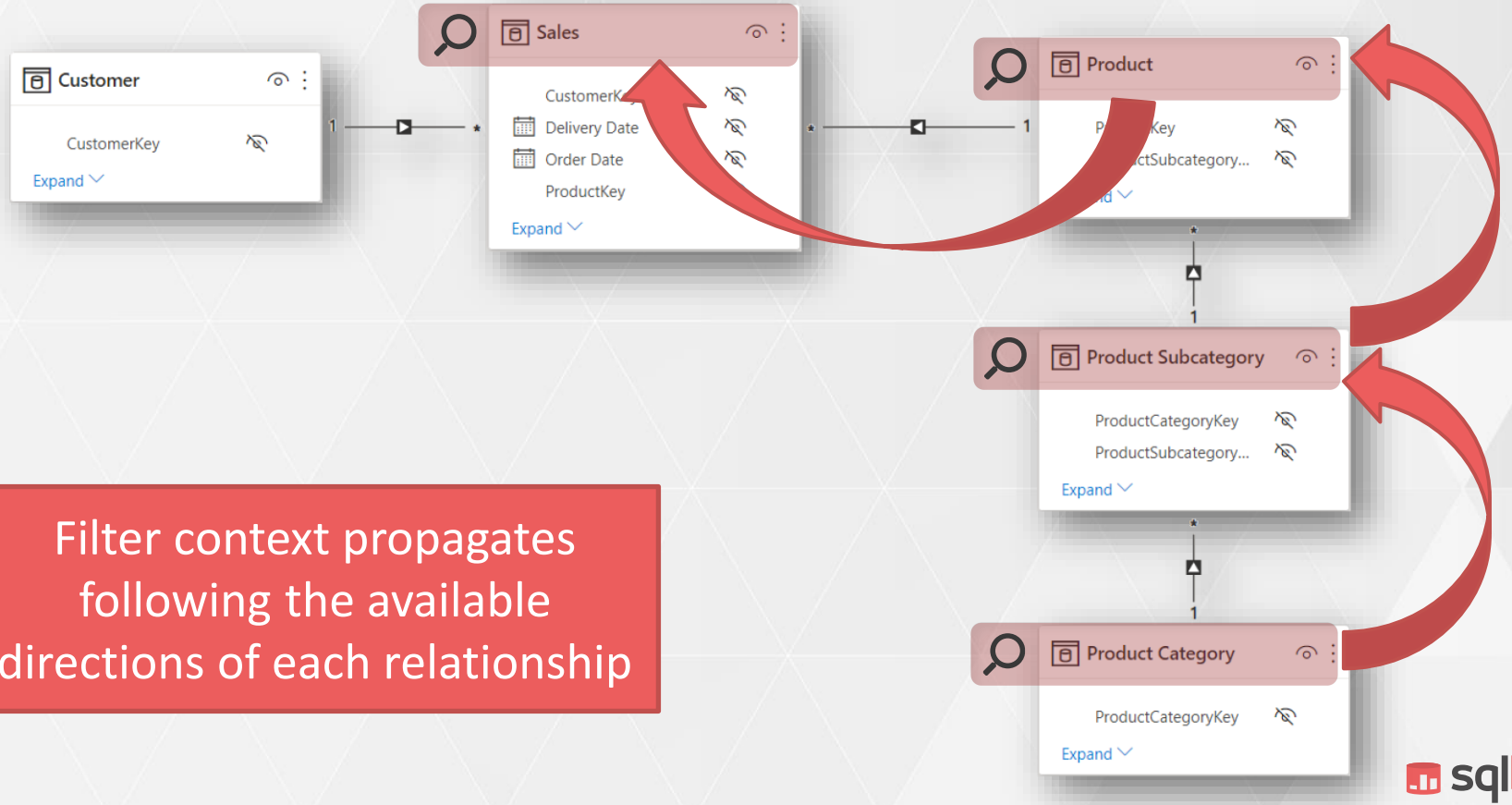
- RELATEDTABLE (table)
  - Filters the parameter table
  - Returns only rows related with the current one
- It is the companion of RELATED



Filter context – many tables



Filters and relationships



Filter context propagates following the available directions of each relationship

Bidirectional cross-filter

- Choose the propagation of the filter context
  - Single: one to many propagation
  - Both: filter context propagates both ways
- Not available in Excel, only Analysis Services and Power BI
- Beware of several details
  - Performance degradation
  - Filtering is active when the “many” side is cross-filtered, numbers might be hard to read for certain measures
  - Ambiguity might appear in the model

The most important DAX function

CALCULATE

CALCULATE syntax

Filters are evaluated in the outer filter context, then combined together in AND, and finally used to build a new filter context into which DAX evaluates the expression.

```
CALCULATE (
    Expression,
    Filter1,
    …
    Filtern
)
```

Repeated many times, as needed

The filter parameters are used to modify the existing filter context. They can add, remove or change existing filter.

CALCULATE examples

Compute the sum of sales where the price is greater than $100.00.

```
NumOfBigSales :=

CALCULATE (
    SUMX ( Sales, Sales[Quantity] * Sales[Net Price] ),
    Sales[Net Price] > 100
)
```

Filter and SUM are on the same table. You can obtain the same result using FILTER.

Filters are tables

Each filter is a table.
Boolean expressions are nothing but shortcuts for table expressions.

```
CALCULATE (
    [Sales Amount],
    Sales[Net Price] > 100
)
```

Is equivalent to

```
CALCULATE (
    [Sales Amount],
    FILTER (
        ALL ( Sales[Net Price] ),
        Sales[Net Price] > 100
    )
)
```

Let’s learn CALCULATE by using some demo

CALCULATE Examples

CALCULATE examples

Compute the sales amount for all of the product colors, regardless of the user selection.

```
SalesAllColors :=

CALCULATE (
    [Sales Amount],
    ALL ( Product[Color] )
)
```

The condition is the list of acceptable values

CALCULATE examples

Compute the sales amount for red products, regardless of user selection for color.
The filter context is applied on the entire model, products filter the Sales table, too.

```
SalesRedProducts :=

CALCULATE (
    [Sales Amount],
    Product[Color] = "Red"
)
```

Filter and SUM are on different tables.
Filter happens because of filter context propagation.

CALCULATE examples

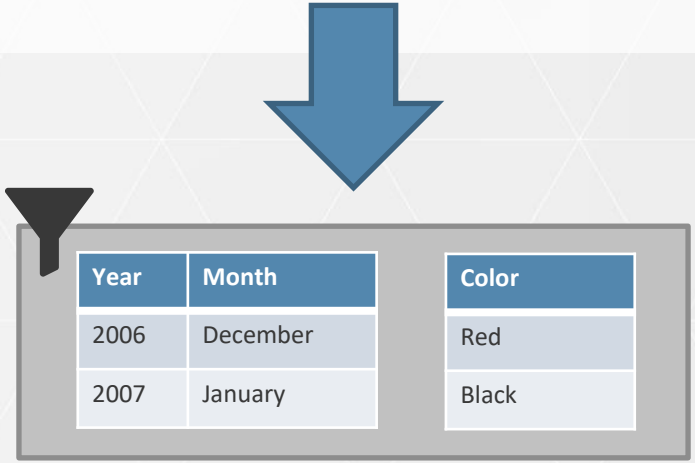
Compute the sales amount for red and blue products, within the user selection for color.

```
SalesTrendyColors :=

CALCULATE (
    [Sales Amount],
    KEEPFILTERS ( Product[Color] IN { "Red", "Blue" } )
)
```

KEEPFILTERS keeps the previous filter, so that the new filter does not override the previous one

What is a filter context?

```
CALCULATE (
    …,
    Product[Color] IN { "Red", "Black" },
    FILTER (
        ALL ( Date[Year], Date[Month] ),
        OR (
            AND ( Date[Year] = 2006, Date[Month] = "December" ),
            AND ( Date[Year] = 2007, Date[Month] = "January" )
        )
    )
)
```

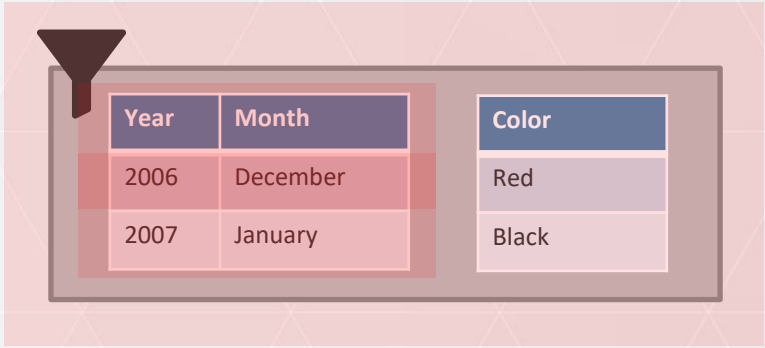


Year	Month
2006	December
2007	January

Color
Red
Black

Filters are tables.
You can use any table function to create a filter in CALCULATE

Filter context definition



Tuple: value for a set of columns

Filter: table of tuples

Filter context: set of filters

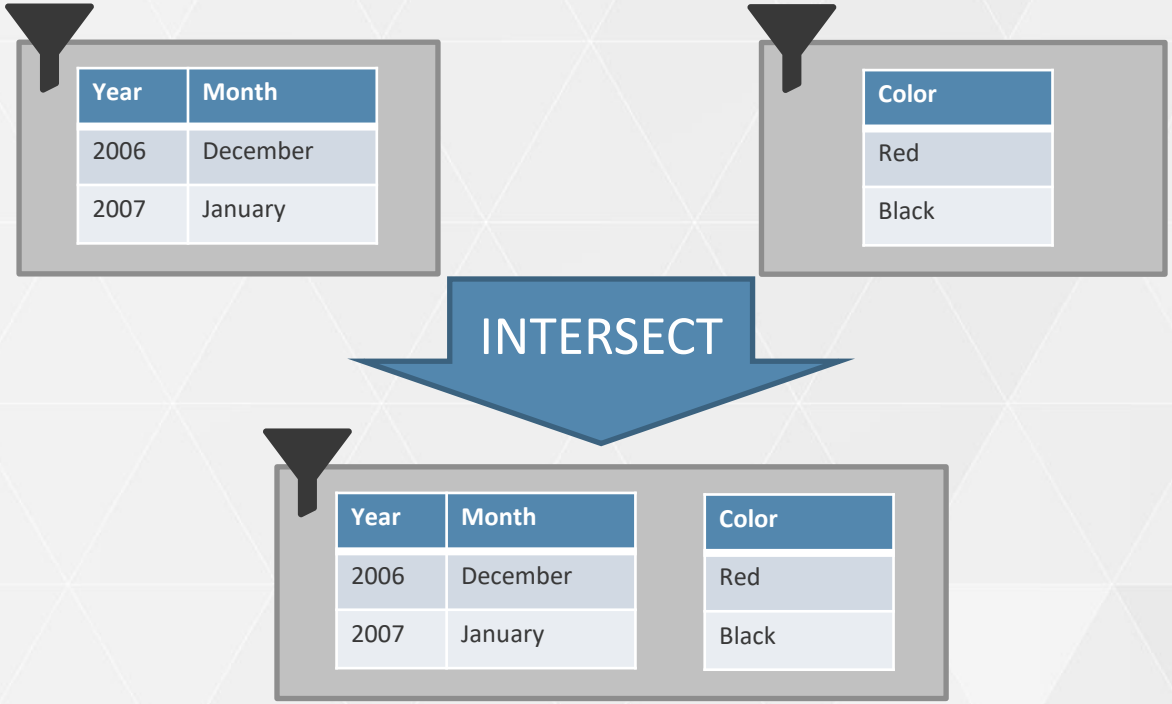
Multiple conditions in CALCULATE

Multiple filter parameters in CALCULATE are intersected, generating a new filter context that uses both filters at the same time.

```
CALCULATE (
    …,
    Product[Color] IN { "Red", "Black" },
    FILTER (
        ALL ( Date[Year], Date[Month] ),
        OR (
            AND ( Date[Year] = 2006, Date[Month] = "December" ),
            AND ( Date[Year] = 2007, Date[Month] = "January" )
        )
    )
)
```

Intersection of filter context

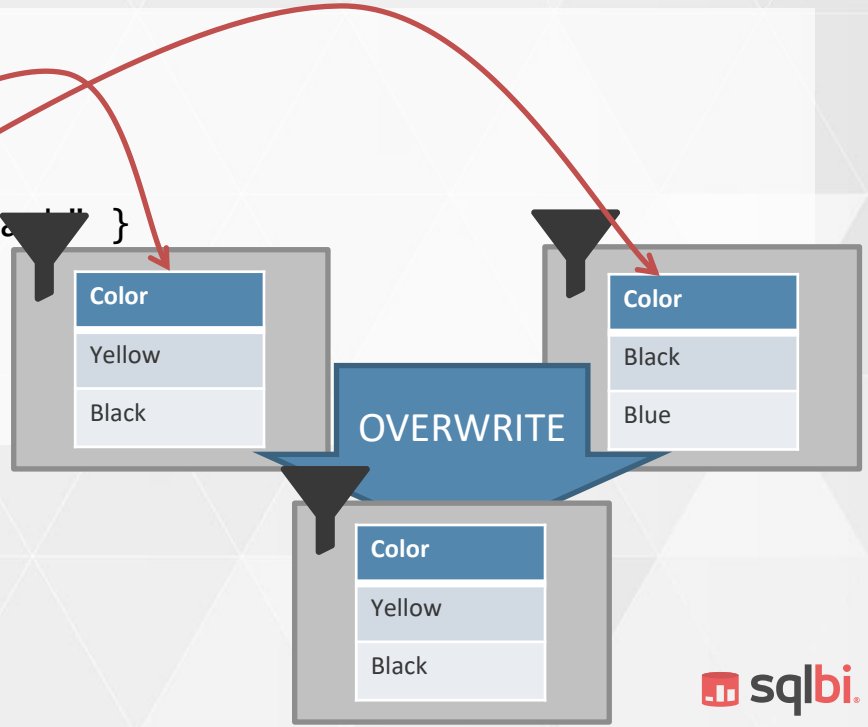
Used by CALCULATE to put filters in AND



Overwriting filter contexts

Nested CALCULATE do not intersect filters, they use another operator, called OVERWRITE.
In fact, the Yellow/Black filter wins against the Black/Blue one, being the innermost.
Yellow/Black overwrites Black/Blue.

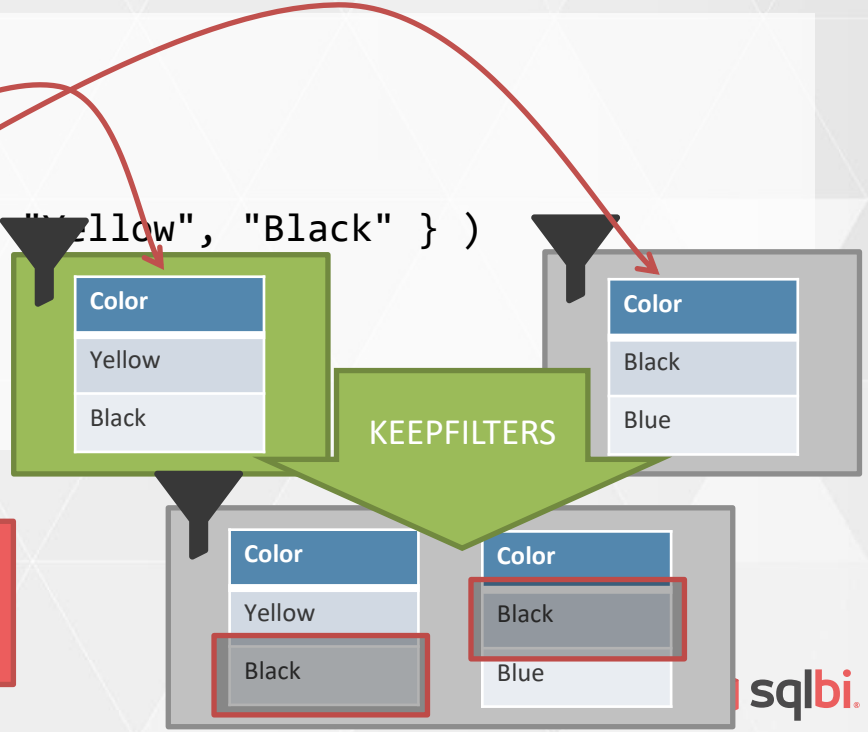
```
CALCULATE (
    CALCULATE (
        …,
        Product[Color] IN { "Yellow", "Black" }
    ),
    Product[Color] IN { "Black", "Blue" }
)
```



KEEPFILTERS

KEEPFILTERS retains the previous filters, instead of replacing them.

```
CALCULATE (
    CALCULATE (
        …,
        KEEPFILTERS ( Product[Color] IN { "Yellow", "Black" } )
    ),
    Product[Color] IN { "Black", "Blue" }
)
```



At the end, only BLACK remains visible

CALCULATE operators

- Overwrite a filter context at the individual columns level
- Remove previously existing filters (ALL)
- Add filters (KEEPFILTERS)
- In DAX you work by manipulating filters with the following internal operators:
  - INTERSECT (multiple filters in CALCULATE)
  - OVERWRITE (nested CALCULATE)
  - REMOVEFILTERS (using ALL)
  - ADDFILTER (using KEEPFILTERS)

Filters on multiple columns

If multiple columns are used in short syntax, then the filter is applied to ALL over the list of columns, that need to be in the same table.

```
--
--  Filtering multiple columns results in filtering ALL over the columns
--
CALCULATE (
    [Sales Amount],
    Sales[Quantity] * Sales[Net Price] > 1000
)

--
--  All columns need to be in the same table
--
CALCULATE (
    [Sales Amount],
    FILTER (
        ALL ( Sales[Quantity], Sales[Net Price] ),
        Sales[Quantity] * Sales[Net Price] > 1000
    )
)
```

KEEPFILTERS might be required, don't filter the table!

KEEPFILTERS is required to keep the same semantics of the table filter. You obtain a similar result by filtering the entire table. Filtering a table is a very bad practice.

```
CALCULATE (
    [Sales Amount],
    KEEPFILTERS (
        FILTER (
            ALL ( Sales[Quantity], Sales[Price] ),
            Sales[Quantity] * Sales[Net Price] > 1000
        )
    )
)
```

```
CALCULATE (
    [Sales Amount],
    FILTER (
        Sales,
        Sales[Quantity] * Sales[Net Price] > 1000
    )
)
```

NEVER filter a table!

Aggregators in compact syntax

Aggregators in compact syntax are evaluated in the outer filter context

```
CALCULATE (
    [Sales Amount],
    Sales[Quantity] < SUM ( Sales[Quantity] ) / 100
)
```

Using variables, however, makes the code more readable

```
VAR TotalQuantity = SUM ( Sales[Quantity] )
RETURN
CALCULATE (
    [Sales Amount],
    Sales[Quantity] < TotalQuantity / 100
)
```

CALCULATE cannot be used in compact syntax

In short syntax filters, measures (and CALCULATE) are not allowed

```
CALCULATE (
    [Sales Amount],
    Sales[Quantity] < [SumOfQuantity] / 100
)
```

Using variables, in this case, is compulsory

```
VAR TotalQuantity = [SumOfQuantity]
RETURN
CALCULATE (
    [Sales Amount],
    Sales[Quantity] < TotalQuantity / 100
)
```

Variables and evaluation contexts

Variables are computed in the evaluation where they are defined, not in the one where they are used. CALCULATE cannot modify the value of a variable because it is already computed.

```
WrongRatio :=

VAR
    Amt = [Sales Amount]
RETURN
    DIVIDE (
        Amt,
        CALCULATE ( Amt, ALL ( Sales ) )
    )
```

Result is always 1

One more feature of CALCULATE

Context transition

Context transition

- Calculate performs another task
- If executed inside a row context
  - It takes the row context
  - Transforms it into an equivalent filter context
  - Applies it to the data model
  - Before computing its expression
- Very important and useful feature
  - Better to learn it writing some code…

Context transition

Sales

Product	Quantity	Net Price
A	1	11.00
B	2	25.00
A	2	10.99

Row Context

```
Test :=
SUMX (
    Sales,
    CALCULATE ( SUM ( Sales[Quantity] ) )
)
```

Filter Context

Product	Quantity	Net Price
A	1	11.00

SUMX Iteration

Row Iterated	Sales[Quantity] Value	Row Result
1	1	1
2	2	2
3	2	2

The result of SUMX is 5

Unexpected results if there are duplicated rows

Sales

Product	Quantity	Net Price
A	1	11.00
B	2	25.00
B	2	25.00

Row Context

```
Test :=
SUMX (
    Sales,
    CALCULATE ( SUM ( Sales[Quantity] ) )
)
```

Filter Context

Product	Quantity	Net Price
B	2	25.00

SUMX Iteration

Row Iterated	Sales[Quantity] Value	Row Result
1	1	1
2	2	4
3	2	4

The result of SUMX is 9

Some notes on context transition

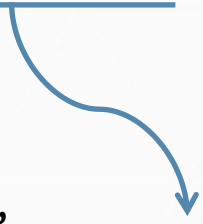
- It is invoked by CALCULATE
- It is expensive: don’t use it iterating large tables
- It does not filter one row, it filters all the identical rows
- It creates a filter context out of a row context
- It happens whenever there is a row context
- It transforms all the row contexts, not only the last one
- Row contexts are no longer available in CALCULATE

Automatic CALCULATE

Whenever a measure is invoked, an automatic CALCULATE is added around the measure.

This is the reason why using [Measure] and Table[Column] as a standard is a best practice.

```
SUMX (
    Orders,
    [Sales Amount]
)
```



```
SUMX (
    Orders,
    CALCULATE ( [Sales Amount] )
)
```

Time to start thinking in DAX

Working with iterators

Computing max daily sales

- What is the maximum amount of sales in one day?
- MAX is not enough
- Consolidate the daily amount and then find the maximum value

Year	Sales Amount	Max Daily Sales
CY 2007	**11,309,946.12**	**126,742.18**
January 2007	**794,248.24**	**92,244.07**
01/02/2007	48,646.02	48,646.02
01/03/2007	92,244.07	92,244.07
01/04/2007	13,950.29	13,950.29
01/05/2007	62,050.83	62,050.83
01/07/2007	23,305.63	23,305.63
01/09/2007	20,543.35	20,543.35
01/10/2007	6,565.56	6,565.56
01/11/2007	22,693.05	22,693.05
01/12/2007	16,251.63	16,251.63
01/13/2007	11,315.05	11,315.05
01/15/2007	58,224.87	58,224.87
01/16/2007	45,595.65	45,595.65

MIN-MAX sales per customer

Iterators can be used to compute values at a different granularity than the one set in the report.

```
MinSalesPerCustomer :=

    MINX ( Customer, [Sales Amount] )


MaxSalesPerCustomer :=

    MAXX ( Customer, [Sales Amount] )
```

Useful iterators

There are many useful iterators, they all behave the same way: iterate on a table, compute an expression and aggregate its value.

```
MAXX
MINX
AVERAGEX
SUMX
PRODUCTX
CONCATENATEX
VARX.P | .S
STDEVX.P | .S
MEDIANX
PERCENTILEX.EXC | .INC
GEOMEANX
```

Probably the most important table in your model

Building a date table

Date table

- Time intelligence needs a date table
  - Built in DAX, Power Query, SQL
  - DAX example: https://www.sqlbi.com/tools/dax-date-template/
- Date table properties
  - All dates should be present – no gaps
  - From 1° of January, to 31° of December
    - Or for the fiscal calendar including full months
  - Otherwise time intelligence DAX functions do not work

Auto date/time and column variations

In Power BI the auto date/time setting automatically creates one date table for each date column in the model.
It is not a best practice, unless you are using a very simple model.
Columns in these auto-created tables can be accessed through “column variations”.

```
Sales Amount YTD :=

TOTALYTD (
    'Sales'[Sales Amount],
    'Sales'[Order Date].[Date]
)
```

Column variation

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

CALENDARAUTO

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

```
--
-- The parameter is the last month
-- of the fiscal year
--
= CALENDARAUTO (
    6
)
```

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

CALENDAR

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

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


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

Mark as date table

- Need to mark the calendar as date table
- Set the column containing the date
- Needed to make time intelligence work if the relationship does not use a Date column
- Multiple tables can be marked as date table
- Used by client tools as metadata information

Set sorting options

- Month names do not sort alphabetically
  - April is not the first month of the year
- Use Sort By Column
- Set all sorting options in the proper way
- Beware of sorting granularity
  - 1:1 between names and sort keys

Using multiple dates

- Date is often a role dimension
  - Many roles for a date
  - Many date tables
- How many date tables?
  - Try to use only one table
  - Use many, only if needed by the model
  - Many date tables lead to confusion
    - And issues when slicing
- Use proper naming convention

Time intelligence functions

Time intelligence in DAX

Time patterns – DAX Patterns

sqlbi

What is time intelligence?

- Many different topics in one name
  - Year To Date
  - Quarter To Date
  - Running Total
  - Same period previous year
  - Working days computation
  - Fiscal Year

Aggregations over time

- Many useful aggregations
  - YTD: Year To Date
  - QTD: Quarter To Date
  - MTD: Month To Date
- They all need a date table
- And some understanding of CALCULATE

Sales 2015 up to 05-15 (v1)

Using CALCULATE you can filter the dates of the period to summarize.

```
SalesAmount20150515 :=

CALCULATE (
    SUM ( Sales[SalesAmount] ),
    FILTER (
        ALL ( 'Date'[Date] ),
        AND (
             'Date'[Date] >= DATE ( 2015, 1, 1 ),
             'Date'[Date] <= DATE ( 2015, 5, 15 )
        )
    )
)
```

Sales 2015 up to 05-15 (v2)

You can replace FILTER with DATESBETWEEN.
The result is always a table with a column.

```
SalesAmount20150515 :=

CALCULATE (
    SUM ( Sales[SalesAmount] ),
    DATESBETWEEN (
        'Date'[Date],
        DATE ( 2015, 1, 1 ),
        DATE ( 2015, 5, 15 )
    )
)
```

Sales Year-To-Date (v1)

Replace the static dates using DAX expressions that retrieve the last day in the current filter.

```
SalesAmountYTD :=

CALCULATE (
    SUM ( Sales[SalesAmount] ),
    DATESBETWEEN (
        'Date'[Date],
        DATE ( YEAR ( MAX ( 'Date'[Date] ) ), 1, 1 ),
        MAX ( 'Date'[Date] )
    )
)
```

Year to date (Time Intelligence)

DATESYTD makes filtering much easier.

```
SalesAmountYTD :=

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

Year to date: the easy way

TOTALYTD: the “DAX for dummies” version.
It hides the presence of CALCULATE, so we suggest not to use it.

```
SalesAmountYTD :=

TOTALYTD (
    SUM ( Sales[SalesAmount] ),
    'Date'[Date]
)
```

Handling fiscal year

The last, optional, parameter is the end of the fiscal year.
Default: 12-31 (or 31/12 – you can use any format regardless of locale settings).

```
SalesAmountYTD :=
TOTALYTD (
    SUM ( Sales[SalesAmount] ),
    'Date'[Date],
    "06-30"
)

SalesAmountYTD :=
CALCULATE (
    SUM ( Sales[SalesAmount] ),
    DATESYTD ( 'Date'[Date], "06-30" )
)
```

DATEADD

Shifts a table back and forth over time, using parameters to define the shift period.
The time period can be DAY, MONTH, QUARTER, YEAR.

```
Sales SPLY :=

CALCULATE (
    SUM( Sales[SalesAmount] ),
    DATEADD ( 'Date'[Date] , -1, YEAR )
)
```

Same period last year

Specialized version of DATEADD that always goes back one year.

```
Sales SPLY :=

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

Moving annual total

DATESINPERIOD returns all the dates in a given number of periods, starting from a reference date. Negative offsets go back in time.

```
CALCULATE (
    SUM ( Sales[SalesAmount] ),
    DATESINPERIOD (
        'Date'[Date],
        MAX ( 'Date'[Date] ),
        -1,
        YEAR
    )
)
```

Running total

Running total requires an explicit filter.
We use a variable to store the last visible date in the current filter context.

```
SalesAmountRT :=

VAR LastVisibleDate = MAX ( 'Date'[Date] )

RETURN

CALCULATE (
    SUM ( Sales[SalesAmount] ),
    FILTER (
        ALL ( 'Date' ),
       'Date'[Date] <= LastVisibleDate
    )
)
```

Semi-additive measures

Semi-additive calculations – DAX Patterns

sqlbi

Semi-additive measures

- Additive Measure
  - SUM over all dimensions
- Semi-additive Measure
  - SUM over some dimensions
  - Different function over other dimensions
  - Time is the standard exception for aggregations
  - Examples
    - Warehouse stocking
    - Current account balance

Current account balance

Name	Country	Occupation	Date	Balance
Katie Jordan	USA	Farmer	1/31/2010	1,687.00
Luis Bonifaz	Argentina	IT Consultant	1/31/2010	1,470.00
Maurizio Macagno	Italy	IT Consultant	1/31/2010	1,500.00
Katie Jordan	USA	Farmer	2/28/2010	2,812.00
Luis Bonifaz	Argentina	IT Consultant	2/28/2010	2,450.00
Maurizio Macagno	Italy	IT Consultant	2/28/2010	
Katie Jordan	USA	Farmer	3/31/2010	
Luis Bonifaz	Argentina	IT Consultant	3/31/2010	
Maurizio Macagno	Italy	IT Consultant	3/31/2010	

Year	Quarter	Month	Katie Jordan	Luis Bonifaz	Maurizio Macagno	**Total**
⊟ CY 2010	⊟ Q1	⊞ January	1,687.00	1,470.00	1,500.00	**4,657.00**
		⊞ February	2,812.00	2,450.00	2,500.00	**7,762.00**
		⊞ March	3,737.00	3,430.00	3,500.00	**10,667.00**
		Total	**8,236.00**	**7,350.00**	**7,500.00**	**23,086.00**
	⊟ Q2	⊞ April	2,250.00	1,960.00	2,000.00	**6,210.00**
		⊞ May	2,025.00	1,764.00	1,800.00	**5,589.00**
		⊞ June	2,700.00	2,352.00	2,400.00	**7,452.00**
		Total	**6,975.00**	**6,076.00**	**6,200.00**	**19,251.00**
	⊟ Q3	⊞ July	3,600.00	3,136.00	3,200.00	**9,936.00**
		⊞ August	5,062.00	4,410.00	4,500.00	**13,972.00**
		⊞ September	2,812.00	2,450.00	2,500.00	**7,762.00**
		Total	**11,474.00**	**9,996.00**	**10,200.00**	**31,670.00**

- Month level correct
- Quarter level wrong
- Year level wrong

Semi-additive measures

- Aggregation depends on the filter
  - Last date, over time
  - SUM for the other dimensions

Year	Quarter	Month	Katie Jordan	Luis Bonifaz	Maurizio Macagno	**Total**
⊟ CY 2010	⊟ Q1	⊞ January	1,687.00	1,470.00	1,500.00	**4,657.00**
		⊞ February	2,812.00	2,450.00	2,500.00	**7,762.00**
		⊞ March	3,737.00	3,430.00	3,500.00	**10,667.00**
		Total	**3,737.00**	**3,430.00**	**3,500.00**	**10,667.00**
	⊟ Q2	⊞ April	2,250.00	1,960.00	2,000.00	**6,210.00**
		⊞ May	2,025.00	1,764.00	1,800.00	**5,589.00**
		⊞ June	2,700.00	2,352.00	2,400.00	**7,452.00**
		Total	**2,700.00**	**2,352.00**	**2,400.00**	**7,452.00**

Semi-additive measures

LASTDATE searches for the last visible date in the current filter context; you can obtain the same result by using a variable to store the last visible day in the filter context.

```
LastBalance :=
CALCULATE (
    SUM ( Balances[Balance] ),
    LASTDATE ( 'Date'[Date] )
)

LastBalance :=
VAR LastDayVisible = MAX ( Date[Date] )
RETURN
    CALCULATE (
        SUM ( Balances[Balance] ),
        'Date'[Date] = LastDayVisible
    )
```

LASTNONBLANK

LASTNONBLANK iterates Date searching the last value for which its second parameter is not a BLANK. Thus, it searches for the last date with any row in the fact table.

```
LastBalanceNonBlank :=

CALCULATE (
    SUM ( Balances[Balance] ),
    LASTNONBLANK (
        'Date'[Date],
        CALCULATE ( COUNTROWS ( Balances ) )
    )
)
```

```
CALCULATE (
   COUNTROWS ( Balances ),
   ALL ( Balances[Name] )
)
```

Semi-additive measures

Most of the times, it is better to search for the last date with transactions, avoiding blank results when there is no data at the end of the period.

```
LastBalance :=

VAR LastDayWithTransactions =
    CALCULATE (
        MAX ( Balances[Date] ),
        ALL ( Balances[Name] )
    )

RETURN
    CALCULATE (
        SUM ( Balances[Balance] ),
        'Date'[Date] = LastDayWithTransactions
    )
```

LASTNONBLANK by customer (1)

Iterating over the customers, it is possible to compute the LASTNONBLANK for each customer, summing at the end the partial results

```
LastBalanceNonBlank :=

SUMX (
    VALUES ( Balances[Name] ),
    CALCULATE (
        SUM ( Balances[Balance] ),
        LASTNONBLANK (
            'Date'[Date],
            CALCULATE ( COUNTROWS ( Balances ) )
        )
    )
)
```

LASTNONBLANK by customer (2)

Using a filter context with multiple columns, you can compute an equivalent LASTNONBLANK filter on a customer-by-customer basis.

```
LastBalanceNonBlankPerCustomer :=

CALCULATE (
    SUM ( Balances[Balance] ),
    TREATAS (
        ADDCOLUMNS (
            VALUES ( Balances[Name] ),
            "LastAvailableDate", CALCULATE ( MAX ( Balances[Date] ) )
        ),
        Balances[Name],
        'Date'[Date]
    )
)
```

This version uses TREATAS and data lineage

Check our Articles, Books, Videos, and Courses on

www.sqlbi.com

sqlbi DAX GUIDE DAX.do DAX PATTERNS DAX FORMATTER