 # Row Context

refers to the context in which a DAX formula is evaluated for each individual row of a table.

```
1 Amount(calculated Column) = 'Sales by Store'[quantity_sold] * 'Sales by Store'[unit_price]
```

tion_time	store_id	staff_id	customer_id	instore_yn	order	line_item_id	product_id	quantity_sold	unit_price	promo_item_yn	Amount(calculated Column)
06:45:47	8	42	8600	Y	1	1	32	1	3	N	3
09:37:14	8	42	8600	Y	1	1	32	1	3	N	3
09:28:07	8	42	8600	Y	1	1	32	1	3	N	3
07:33:45	8	42	8600	Y	1	1	32	1	3	N	3

```
1 Amount (Measure) = 'Sales by Store'[quantity_sold] * 'Sales by Store'[unit_price]
```

A single value for column 'quantity_sold' in table 'Sales by Store' cannot be determined. This can happen when a measure formula refers to a column that contains many values without specifying an aggregation such as min, max, count, o

Remember: a calculated column is computed row-by-row, a measure is not. A measure is computed at the aggregate level of the report, potentially scanning millions of rows.

If we do sum like this, It is no longer the aggregation of a multiplication. Instead, it became the multiplication of two aggregations.

```
1 Amount sum(Measure) = SUM('Sales by Store'[quantity_sold]) * SUM('Sales by Store'[unit_price])
```

Sum of Amount(calculated Column)	Amount sum(Measure)
42,52,704.88	40,11,06,26,77,540.16

Sum works with columns, but it cannot aggregate expressions. You can obtain the sum of a column, but not the sum of a multiplication.

```
1 Amount sum agg (Measure) = SUM('Sales by Store'[quantity_sold] * 'Sales by Store'[unit_price])
```

The SUM function only accepts a column reference as an argument.

A row context is created by any iterator.An iterator requires a table to iterate over, and that table needs to contain all the columns required for the expression.

```
1 Amount sumx iterator (Measure) = SUMX('Sales by Store','Sales by Store'[quantity_sold] * 'Sales by Store'[unit_price])
```

Sum of Amount(calculated Column)	Amount sumx iterator (Measure)
42,52,704.88	42,52,704.88

Note: A very important aspect of the row context is that the row context iterates through a table row by row. It does not filter the table.

What happens if we provide Sumx in calculated column

The row context is iterating over *Sales* and it is positioned on the current row. But there is another iterator: SUMX. SUMX introduces a new row context that iterates – again – over *Sales*.

```
Amount sumx iterator (Measure) = SUMX('Sales by Store','Sales by Store'[quantity_sold] * 'Sales by Store'[unit_price])
```

customer_id	instore_yn	order	line_item_id	product_id	quantity_sold	unit_price	promo_item_yn	Amount(calculated Column)	Amount sumx iterator (Measure)
8600	Y	1	1	32	1	3	N	3	4252704.88
8600	Y	1	1	32	1	3	N	3	4252704.88
8600	Y	1	1	32	1	3	N	3	4252704.88
8600	Y	1	1	32	1	3	N	3	4252704.88

The previous code could be written as follows

```
Amount cal col sumx = SUMX('Sales by Store','Sales by Store'[Amount(calculated Column)])
```

e_item_id	product_id	quantity_sold	unit_price	promo_item_yn	Amount(calculated Column)	Amount sumx iterator (Measure)	Amount cal col sumx
1	32	1	3	N	3	4252704.88	4252704.88
1	32	1	3	N	3	4252704.88	4252704.88
1	32	1	3	N	3	4252704.88	4252704.88
1	32	1	3	N	3	4252704.88	4252704.88
1	32	1	3	N	3	4252704.88	4252704.88

SUM is nothing but a simplified version of SUMX: we call it syntax sugar. *SUM (Sales[Quantity])* is internally translated into *SUMX (Sales, Sales[Quantity])*. Therefore, a regular aggregator like SUM, AVERAGE, MIN or MAX behaves the same way an iterator does.

This is the reason why a calculated column computing *SUM (Sales[Quantity])* produces the grand total of *Sales[Quantity]* on each row.

```
sum of quantity = SUM('Food Inventory'[quantity_sold])
```

ne_item_id	product_id	quantity_sold	unit_price	promo_item_yn	Amount(calculated Column)	Amount sumx iterator (Measure)	Amount cal col sumx	sum of quantity
1	32	1	3	N	3	4252704.88	4252704.88	141433
1	32	1	3	N	3	4252704.88	4252704.88	141433
1	32	1	3	N	3	4252704.88	4252704.88	141433
1	32	1	3	N	3	4252704.88	4252704.88	141433
1	32	1	3	N	3	4252704.88	4252704.88	141433

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

Filter Context

refers to the set of filters that are applied to data before a DAX expression is evaluated, affecting the result based on selected filters or slicers.

Brand	Sales Amount
A. Datum	147,687.44
Adventure Works	2,761,057.66
Contoso	2,227,244.32
Fabrikam	990,275.08
Litware	506,104.50
Northwind Traders	119,857.67
Proseware	956,335.76
Southridge Video	776,807.78
Tailspin Toys	79,159.15
The Phone Company	1,976,180.03
Wide World Importers	1,796,930.99
Total	**12,337,640.39**

A very common mistake made by newbies is to think that each row in a matrix is evaluated in a row context, because they equate a row in a matrix with the row context. This is not the case. A row context exists within an active iteration. No iteration, no row context. Therefore, the individual rows of the matrix are not computed inside a row context. Each cell of the matrix is evaluated in a filter context that happens to filter a single value for the *Brand* column.

Measure in Sales table

```
Europe Sales :=
CALCULATE (
    [Sales Amount],
    Customer[Continent] = "Europe"
)
```

COPY CONVENTIONS #3 DAX FORMATTER

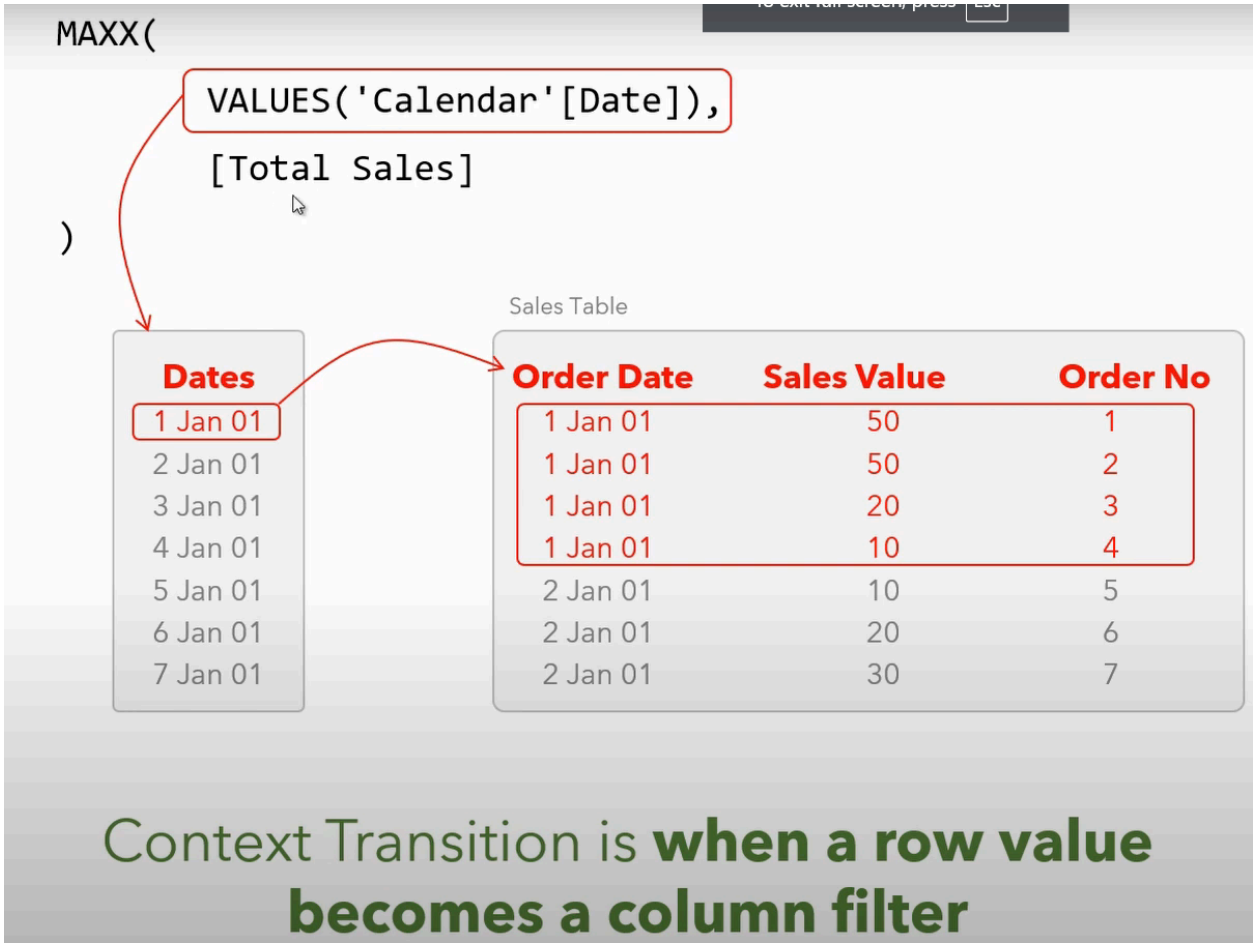
CALCULATE changes the filter context under which *Sales Amount* is being computed, by setting a filter on the *Customer[Continent]* column for it to be Europe.

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

https://www.youtube.com/watch?v=GGH4hh7u6HA

 # Context transition

Context transition in Power BI is the process where row context is converted into filter context when a DAX expression using functions like CALCULATE or CALCULATETABLE is evaluated.



```
MAXX(
    VALUES('Calendar'[Date]),
    SUM(Sales[Sales Value])
)
```

context transition not happening here beacuse
its needs calculate function or diresct measures

Dates
1 Jan 01
2 Jan 01
3 Jan 01
4 Jan 01
5 Jan 01
6 Jan 01
7 Jan 01

Sales Table

Order Date	Sales Value	Order No
1 Jan 01	50	1
1 Jan 01	50	2
1 Jan 01	20	3
1 Jan 01	10	4
2 Jan 01	10	5
2 Jan 01	20	6
2 Jan 01	30	7

```
Best Selling Day =
MAXX(
    VALUES('Calendar'[Date]),
    CALCULATE(SUM(Sales[SalesAmount]))
)
```

2001	32,66,373.66	32,66,373.66
Jul	4,73,388.16	4,73,388.16
Aug	5,06,191.69	5,06,191.69
Sep	4,73,943.03	4,73,943.03
Oct	5,13,329.47	5,13,329.47
Nov	5,43,993.41	5,43,993.41
Dec	7,55,527.89	7,55,527.89
2002	65,30,343.53	65,30,343.53
Jan	5,96,746.56	5,96,746.56
Feb	5,50,816.69	5,50,816.69
Mar	6,44,135.20	6,44,135.20
Apr	6,63,692.29	6,63,692.29
May	6,73,556.20	6,73,556.20
Jun	6,76,763.65	6,76,763.65
Total	2,93,58,677.22	2,93,58,677.22

https://www.sqlbi.com/articles/understanding-context-transition-in-dax/

https://www.youtube.com/watch?v=pTI2ASgecGA

https://www.youtube.com/watch?v=lMuDz6ViU1w&t=16s

Running total

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

Calendar Year	Sales Amount	Sales YTD
CY 2008	**1,189,326,612.81**	**1,189,326,612.81**
January	79,431,234.29	79,431,234.29
February	85,088,461.45	164,519,695.74
March	84,808,709.97	249,328,405.72
April	105,627,816.67	354,956,222.38
May	109,011,089.35	463,967,311.73
June	107,110,706.45	571,078,018.19
July	118,015,094.13	689,093,112.31
August	104,552,290.52	793,645,402.83
September	100,882,614.27	894,528,017.10
October	97,130,506.81	991,658,523.91
November	96,777,975.30	1,088,436,499.22
December	100,890,113.59	1,189,326,612.81
CY 2009	**818,451,151.95**	**818,451,151.95**
January	80,407,442.14	80,407,442.14
February	83,640,730.85	164,048,172.99
March	85,254,818.84	249,302,991.83
April	97,806,971.93	347,109,963.76
May	115,053,236.93	462,163,200.69
June	114,600,442.75	576,763,643.44
July	115,718,443.46	692,482,086.90
Total	**2,007,777,764.76**	**818,451,151.95**

Note: If the goal is to sum values over more than one year, then DATESYTD is no longer useful.

```
Sales RT :=
VAR MaxDate = MAX ( 'Date'[Date] ) -- Saves the last visible date
RETURN
    CALCULATE (
        [Sales Amount],             -- Computes sales amount
        'Date'[Date] <= MaxDate,    -- Where date is before the last visible date
        ALL ( Date )                -- Removes any other filters from Date
    )
```

Calendar Year	Sales Amount	Sales YTD	Sales RT
CY 2008	**1,189,326,612.81**	**1,189,326,612.81**	**1,189,326,612.81**
January	79,431,234.29	79,431,234.29	79,431,234.29
February	85,088,461.45	164,519,695.74	164,519,695.74
March	84,808,709.97	249,328,405.72	249,328,405.72
April	105,627,816.67	354,956,222.38	354,956,222.38
May	109,011,089.35	463,967,311.73	463,967,311.73
June	107,110,706.45	571,078,018.19	571,078,018.19
July	118,015,094.13	689,093,112.31	689,093,112.31
August	104,552,290.52	793,645,402.83	793,645,402.83
September	100,882,614.27	894,528,017.10	894,528,017.10
October	97,130,506.81	991,658,523.91	991,658,523.91
November	96,777,975.30	1,088,436,499.22	1,088,436,499.22
December	100,890,113.59	1,189,326,612.81	1,189,326,612.81
CY 2009	**818,451,151.95**	**818,451,151.95**	**2,007,777,764.76**
January	80,407,442.14	80,407,442.14	1,269,734,054.95
February	83,640,730.85	164,048,172.99	1,353,374,785.80
March	85,254,818.84	249,302,991.83	1,438,629,604.64
April	97,806,971.93	347,109,963.76	1,536,436,576.57

https://www.sqlbi.com/articles/computing-running-totals-in-dax/

Dax
DAX (Data Analysis Expressions) is a formula language that defines custom calculations, aggregations, and data transformations on your data models.