Visual Calculations

Making DAX easier!

Thank you sponsors!

INSPARK

redgate

PRIVINITY®
DATA PRIVACY SOLUTIONS

sogeti
Part of Capgemini

infoSupport
Solid Innovator

ilionx

Rate Data Saturday Holland

SUPPORT CASPER

1 review = 1 €

Towards beating pancreatic cancer

Marc Lelijveld

Technical Evangelist & Architect
At Macaw

MVP Microsoft® Most Valuable Professional

sessionize MOST ACTIVE SPEAKER 2022

sessionize MOST ACTIVE SPEAKER 2023

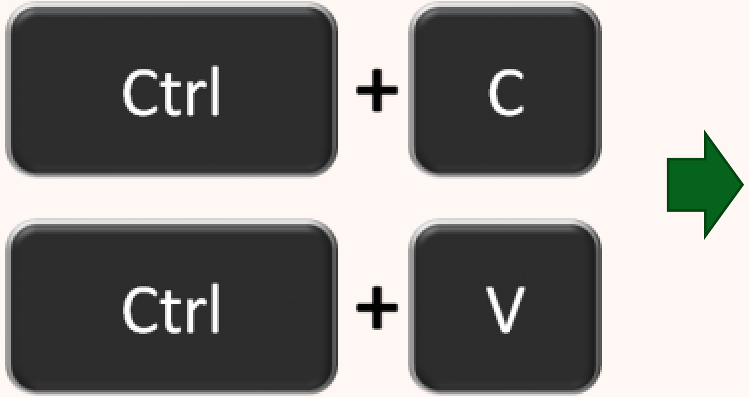
@MarcLelijveld

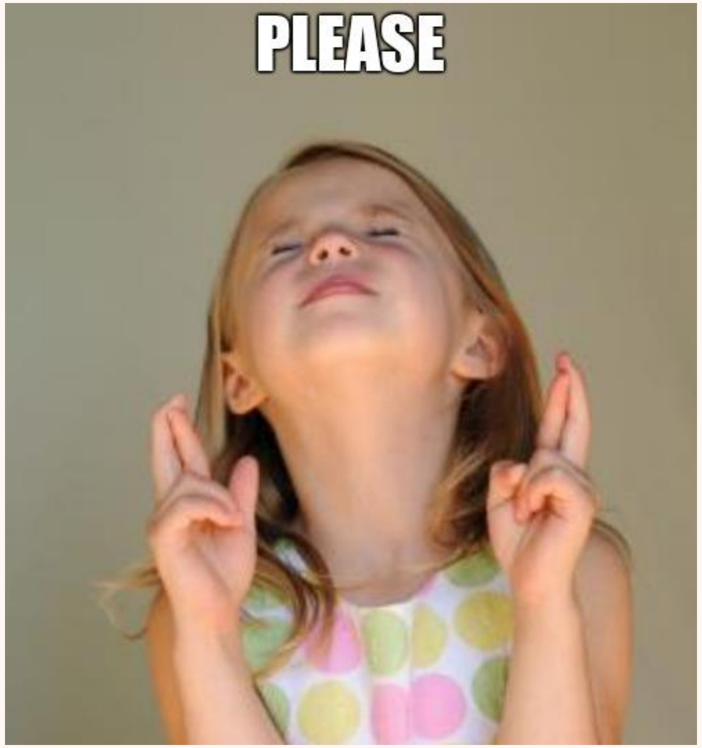
linkedin.com/in/MarcLelijveld

Data-Marc.com

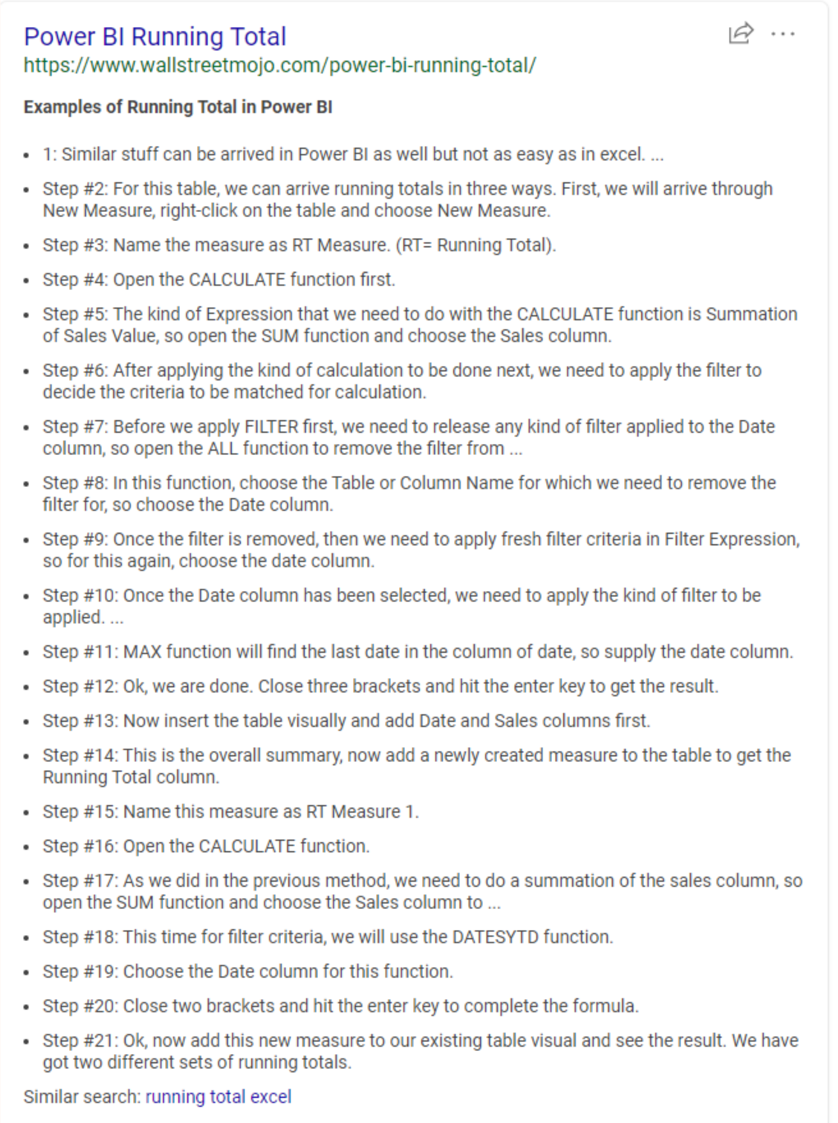
DutchFabricUsergroup.com

How most of us write DAX today





Ever tried to do a running total in DAX?



Power BI Running Total
https://www.wallstreetmojo.com/power-bi-running-total/

Examples of Running Total in Power BI

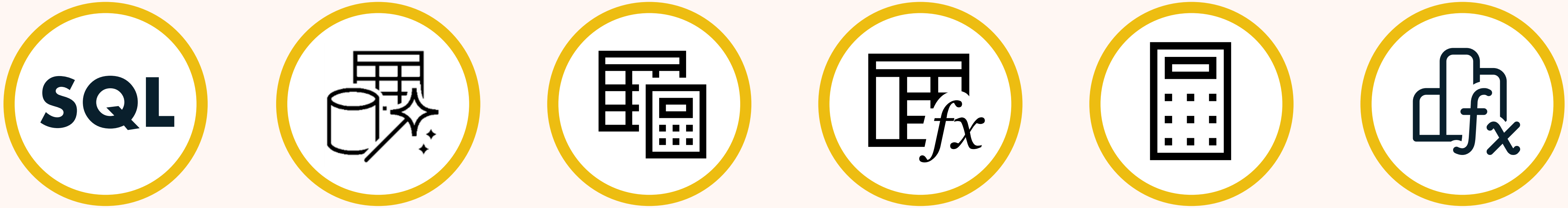
- 1: Similar stuff can be arrived in Power BI as well but not as easy as in excel. ...
- Step #2: For this table, we can arrive running totals in three ways. First, we will arrive through New Measure, right-click on the table and choose New Measure.
- Step #3: Name the measure as RT Measure. (RT= Running Total).
- Step #4: Open the CALCULATE function first.
- Step #5: The kind of Expression that we need to do with the CALCULATE function is Summation of Sales Value, so open the SUM function and choose the Sales column.
- Step #6: After applying the kind of calculation to be done next, we need to apply the filter to decide the criteria to be matched for calculation.
- Step #7: Before we apply FILTER first, we need to release any kind of filter applied to the Date column, so open the ALL function to remove the filter from ...
- Step #8: In this function, choose the Table or Column Name for which we need to remove the filter for, so choose the Date column.
- Step #9: Once the filter is removed, then we need to apply fresh filter criteria in Filter Expression, so for this again, choose the date column.
- Step #10: Once the Date column has been selected, we need to apply the kind of filter to be applied. ...
- Step #11: MAX function will find the last date in the column of date, so supply the date column.
- Step #12: Ok, we are done. Close three brackets and hit the enter key to get the result.
- Step #13: Now insert the table visually and add Date and Sales columns first.
- Step #14: This is the overall summary, now add a newly created measure to the table to get the Running Total column.
- Step #15: Name this measure as RT Measure 1.
- Step #16: Open the CALCULATE function.
- Step #17: As we did in the previous method, we need to do a summation of the sales column, so open the SUM function and choose the Sales column to ...
- Step #18: This time for filter criteria, we will use the DATESYTD function.
- Step #19: Choose the Date column for this function.
- Step #20: Close two brackets and hit the enter key to complete the formula.
- Step #21: Ok, now add this new measure to our existing table visual and see the result. We have got two different sets of running totals.

Similar search: running total excel

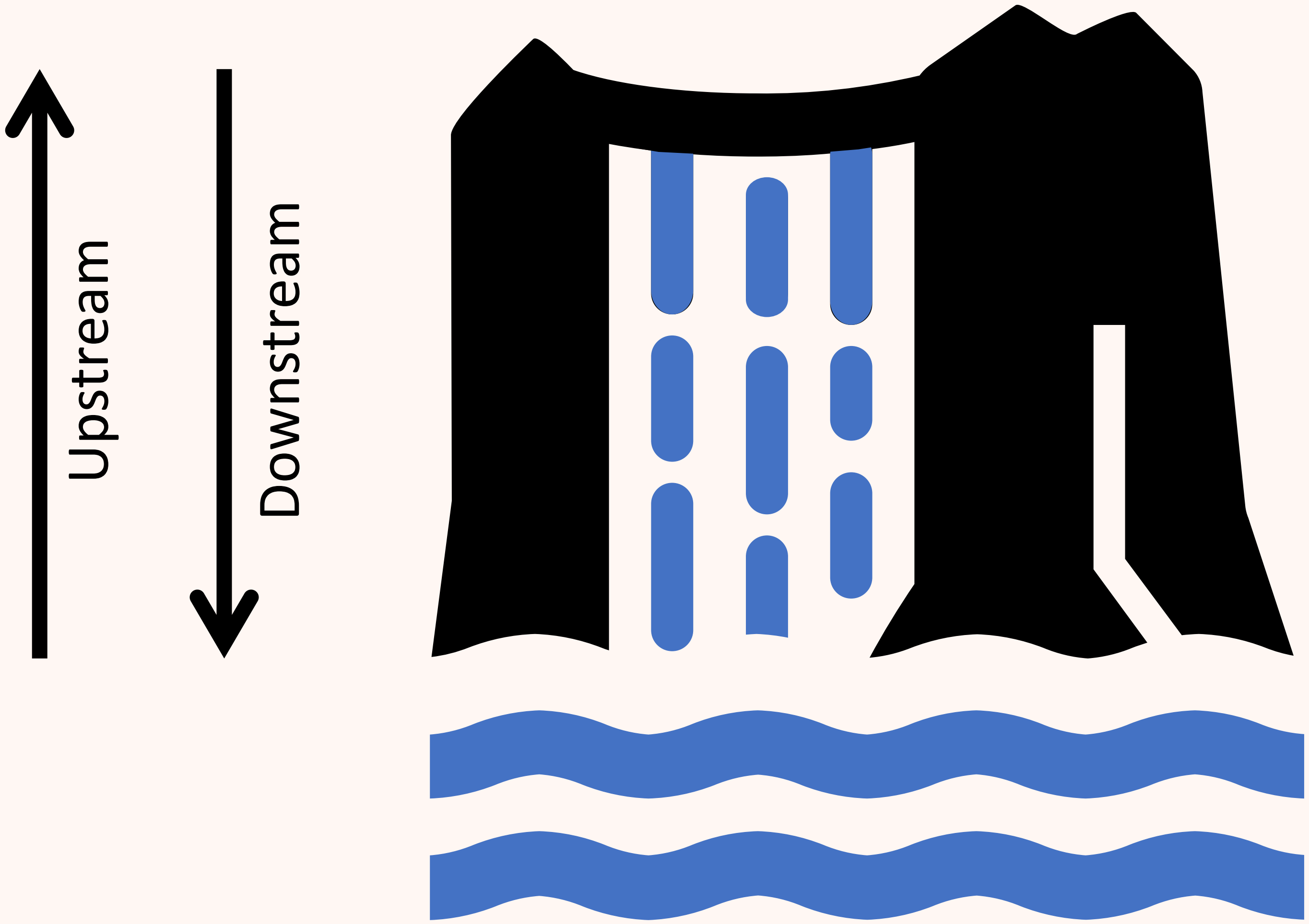
21 steps!

How would you want to write DAX?

What calculation options do we have?



Which options fits in where?



SQL Query

Power Query

Calculated Table

Calculated Column

Measure

Visual Calculation

Let’s face it

- DAX is hard. (to be exact, filter context is hard)
- DAX calculations (measures) are scoped to the model and independent of each other
- Typical business type calculations are surprisingly hard to do.
  - Quick measures help – but the DAX it generates is hard to comprehend / edit

What if you could…

- define a calculation on exactly what’s on your visual without having to worry about what makes DAX hard?
- *easily write and read* the DAX statement needed for your business calculation?
- define a calculation with no to minimal typing, but instead use point-and-click if you wanted to?

Enter: Visual Calculations
(Visual Calcs for short)

- DAX calculations defined on a visual
- Can refer to any field that’s on the visual, including other visual calcs
- Are executed in scope of the visual only, not the model
- Can refer to the visual structure as well, instead of just referencing fields
- Most of the time you don’t have to worry about the complexity of filter context

Visual calcs vs measures vs calc columns

Calculated column

- Defined on a table
- Works on a row-by-row basis (row context)
- Computed at dataset refresh (for import tables) or query refresh (for DirectQuery tables)
- Result persisted (for import tables)

Measure

- Defined in the data model
- Works on sets of rows (filter context)
- Computed at query execution

Visual calculations

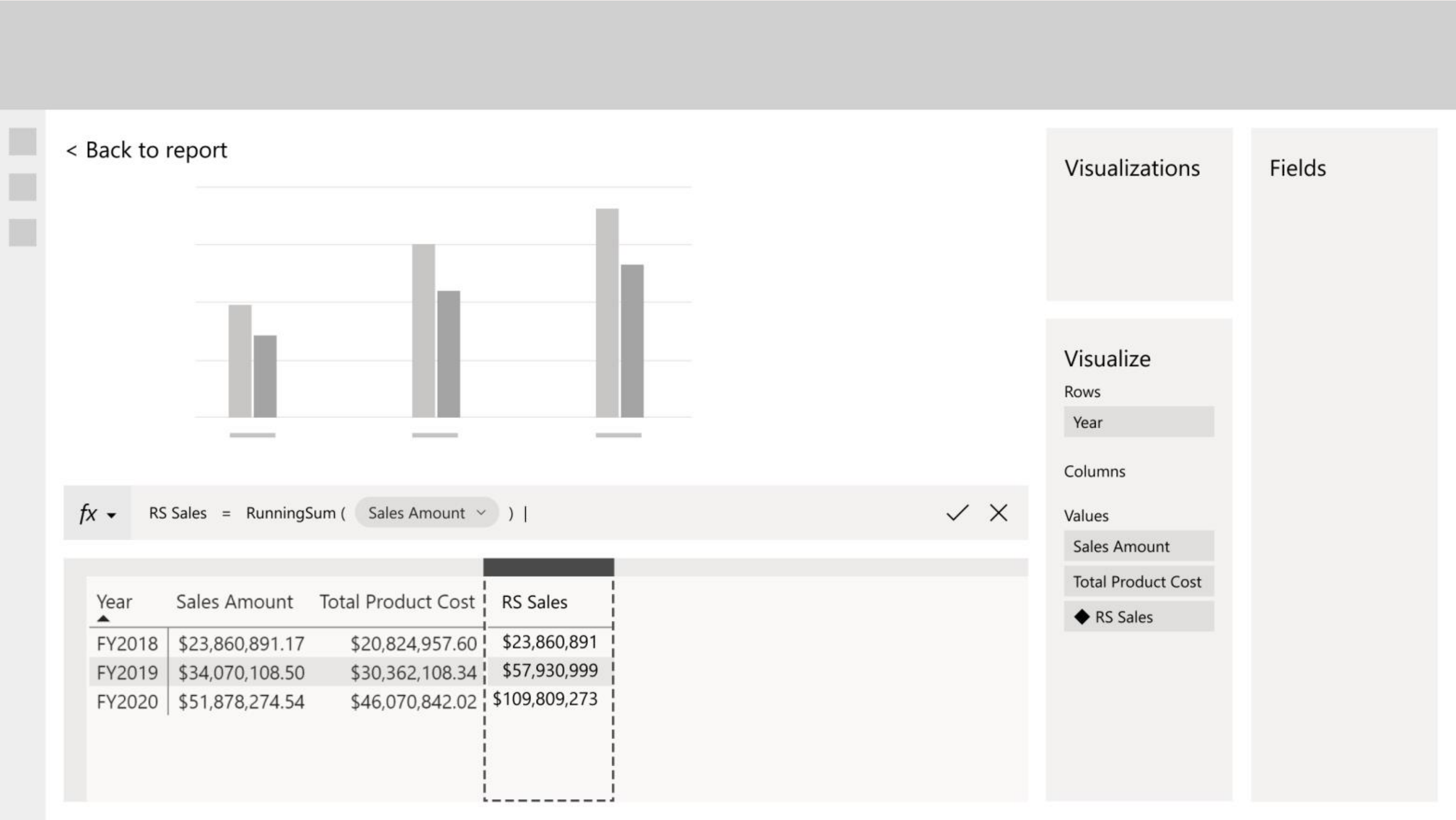
- Defined on a visual
- “Visible context”
- Computed at query execution
- Can refer to visual structure

Visual calculations are easy and flexible

- WYSIWYG
- Point-and-click provided, if you want
- Just “visible context”
- High-level functions for common business calculations
- Refer to visual structure

Mental model

- Any visual can be represented as a matrix
- A visual calculation adds a column on that matrix
- A visual calculation is added to the matrix but can be hidden from the visual itself
- A visual calculation can refer to any field / visual calculation on the visual



RS Sales = RunningSum (Sales Amount) |

Year	Sales Amount	Total Product Cost	RS Sales
FY2018	$23,860,891.17	$20,824,957.60	$23,860,891
FY2019	$34,070,108.50	$30,362,108.34	$57,930,999
FY2020	$51,878,274.54	$46,070,842.02	$109,809,273

Visualize

Rows
Year

Columns

Values
Sales Amount
Total Product Cost
◆ RS Sales

Demo

#Mindblown #PowerBI

What are Visual Calcs?

DAX calculations in the scope of a visual.

In support thereof:

- new functions
- new concepts

Visual calc expressions

- It’s just DAX. Most DAX works.

```
OrderCategory =
  SWITCH (
        TRUE,
       [Sales Amount] <= 100, ’Small’,
       [Sales Amount] <= 500, ’Medium’,
        ‘Large’
  )
```

- Basic arithmetic: `Profit = [Sales] – [Cost]`
- New visual calc functions

Visual Calcs vs ‘regular’ DAX (1/2)

- Goal: implement a running sum calculation that sums Sales Amount over Year.
- Solution:
  - Current DAX:

```
CALCULATE(
    SUM('Sales'[Sales Amount]),
    FILTER(
        ALLSELECTED('Date'[Year]),
        ISONORAFTER('Date'[Year], MAX('Date'[Year]), DESC)
    )
)
```

  - Visual Calcs (assuming Year is on Rows):

```
RUNNINGSUM([Sales Amount])
```

Visual Calcs vs ‘regular’ DAX (2/2)

- Goal: given a list of states and number of restaurants per state, for each state calculate the difference in number of restaurant in that state vs the one above.

Current DAX:

```
DiffNumRestaurants =
VAR _temp =
    SUMMARIZE (
        ALLSELECTED ( Pizza_data ),
        Pizza_data[state],
        "NumRestaurants", [# restaurants]
    )
VAR _currentState =
    SELECTEDVALUE ( Pizza_data[state] )
VAR _previousState =
    TOPN (
        1,
        FILTER ( _temp, Pizza_data[state] < _currentState ),
        Pizza_data[state], DESC
    )
RETURN
    [# restaurants] - SUMX ( _previousState, [NumRestaurants] )
```

Visual Calcs (technically Window functions):

```
Diff =
VAR PreviousStateRestaurants =
    CALCULATE (
        [# restaurants],
     OFFSET ( -1, ALLSELECTED ( Pizza_data[state] ),
     ORDERBY ( Pizza_data[state] ) )
    )
RETURN
    [# restaurants] - PreviousStateRestaurants
```

Visual Calcs:

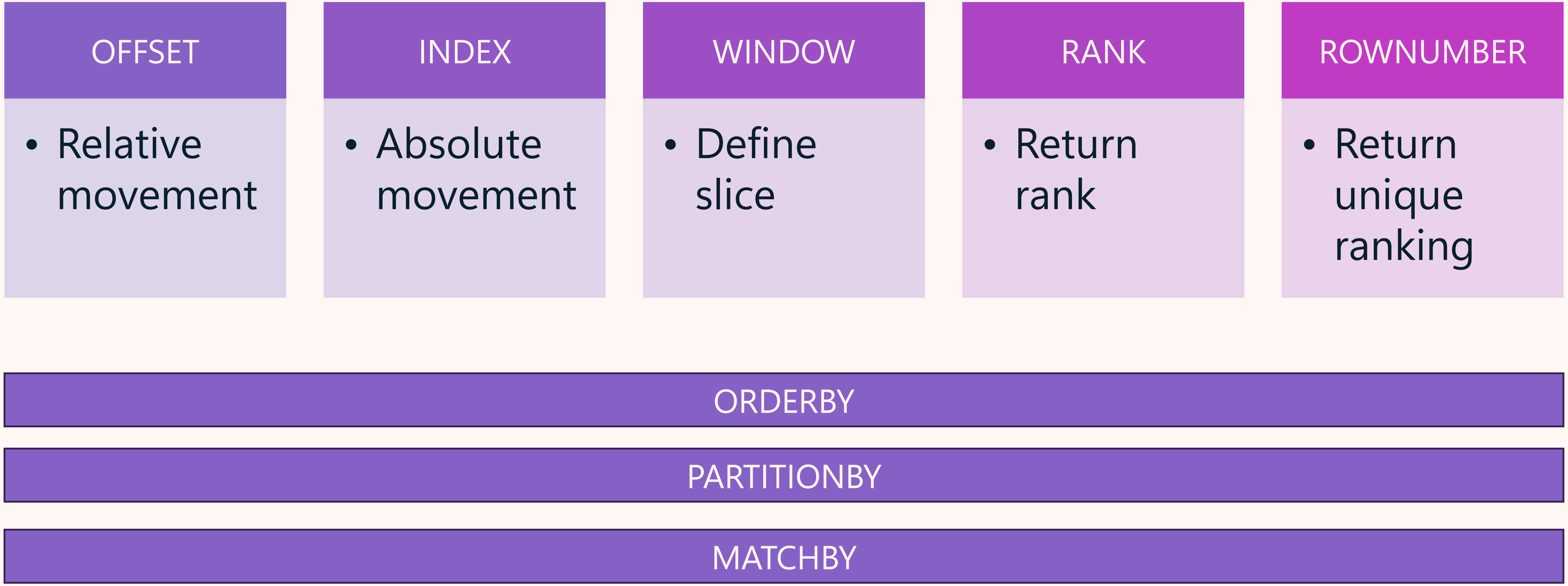
```
Diff = [# restaurants] – PREVIOUS( [# restaurants] )
```

New functions

- We are introducing multiple new functions, divided in levels.
- Higher-level functions are easier-to-use shortcuts to lower-level functions

Level	Functionality offered	Included functions	Flexibility	Complexity
Low	Returning a single item, a set of items or an index number	Window functions	High	High
Medium	Returning a single item, a set of items	Movement functions Hierarchical functions	Medium	Medium-High
High	Business-type calculations	Calculation functions	Medium	Low

Low level functions
(aka window functions)



OFFSET

- Relative movement

INDEX

- Absolute movement

WINDOW

- Define slice

RANK

- Return rank

ROWNUMBER

- Return unique ranking

ORDERBY

PARTITIONBY

MATCHBY

Medium level functions

Movement functions:

Previous: move up / left in *direction*

Next: move down/right in *direction*

First: move to beginning in *direction*

Last: move to end in *direction*

Inspection function:

ISATLEVEL: check if a column is on the current level

Lattice navigations functions:

Expand: add detail level

ExpandAll: add all detail levels

Collapse: remove detail level

CollapseAll: remove all detail levels

Selection function:

Range: define slice

High level functions

RunningSum: add running sum in direction

MovingAverage: add moving average in direction

New concepts

Visual matrix: the matrix that represents the data in a visual

Axis: defines how a calculation traverse the *visual matrix* on which it's being executed.

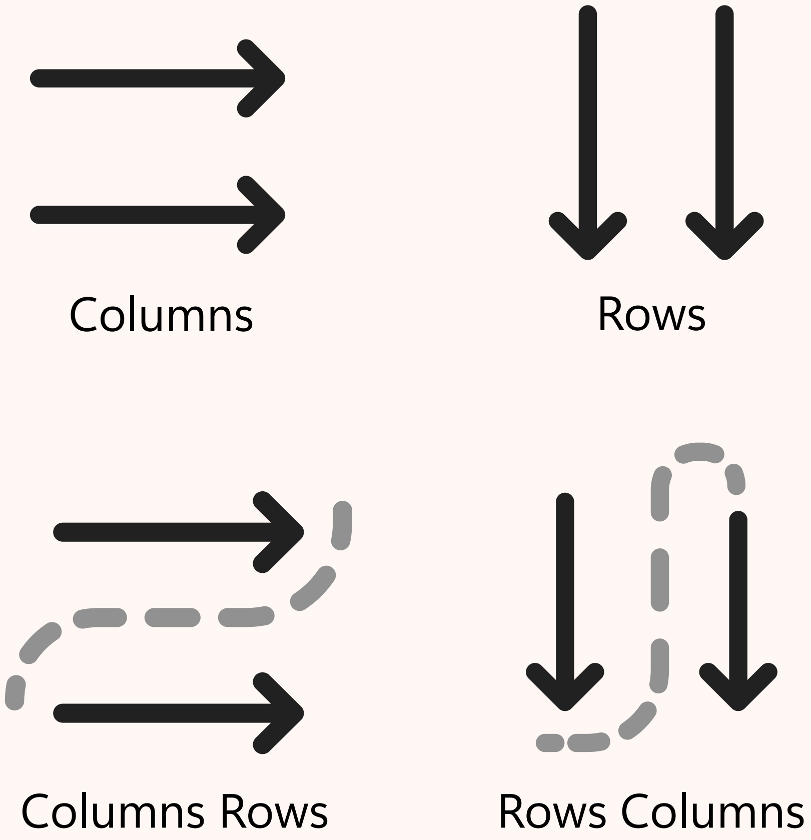
Reset: defines when a calculation restarts while traversing the *axis*

Direction: combines *axis* and *reset*

Hidden: a field that is on the *visual matrix* but not shown on the visual

Concept: axis

- Think of axis as how the calculation traverses the visual matrix on which it’s being executed
- Default: `Rows`



Columns

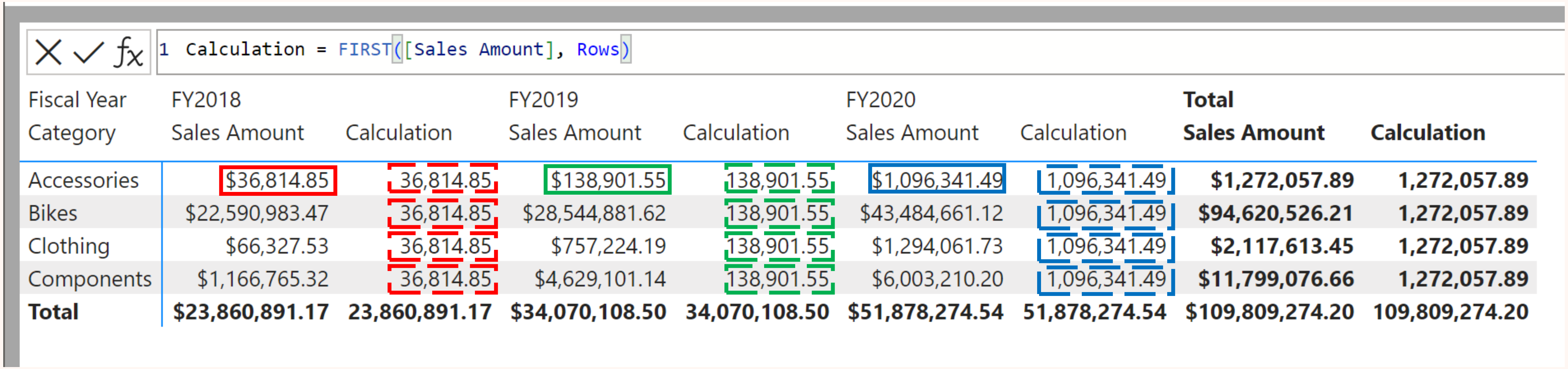
Rows

Columns Rows

Rows Columns

Example: Axis

`FIRST([Sales Amount], Rows)` →
for each *Row*, retrieve the first Sales Amount from the first *Row*)

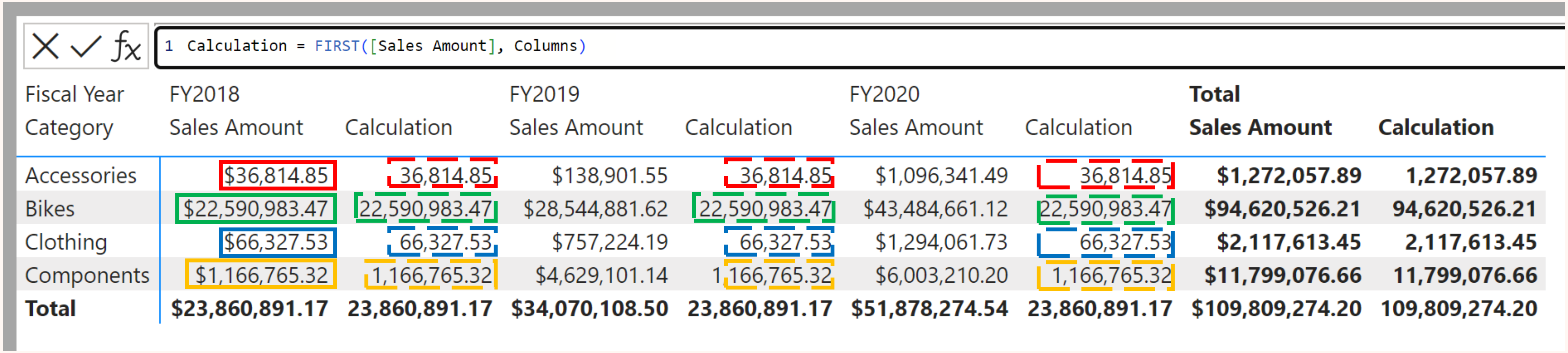


```
Calculation = FIRST([Sales Amount], Rows)
```

Fiscal Year	FY2018		FY2019		FY2020		**Total**	
Category	Sales Amount	Calculation	Sales Amount	Calculation	Sales Amount	Calculation	**Sales Amount**	**Calculation**
Accessories	$36,814.85	36,814.85	$138,901.55	138,901.55	$1,096,341.49	1,096,341.49	**$1,272,057.89**	**1,272,057.89**
Bikes	$22,590,983.47	36,814.85	$28,544,881.62	138,901.55	$43,484,661.12	1,096,341.49	**$94,620,526.21**	**1,272,057.89**
Clothing	$66,327.53	36,814.85	$757,224.19	138,901.55	$1,294,061.73	1,096,341.49	**$2,117,613.45**	**1,272,057.89**
Components	$1,166,765.32	36,814.85	$4,629,101.14	138,901.55	$6,003,210.20	1,096,341.49	**$11,799,076.66**	**1,272,057.89**
Total	**$23,860,891.17**	**23,860,891.17**	**$34,070,108.50**	**34,070,108.50**	**$51,878,274.54**	**51,878,274.54**	**$109,809,274.20**	**109,809,274.20**

Example: Axis

`FIRST([Sales Amount], Columns)` →
for each *Column*, retrieve the first Sales Amount from the first *Column*)

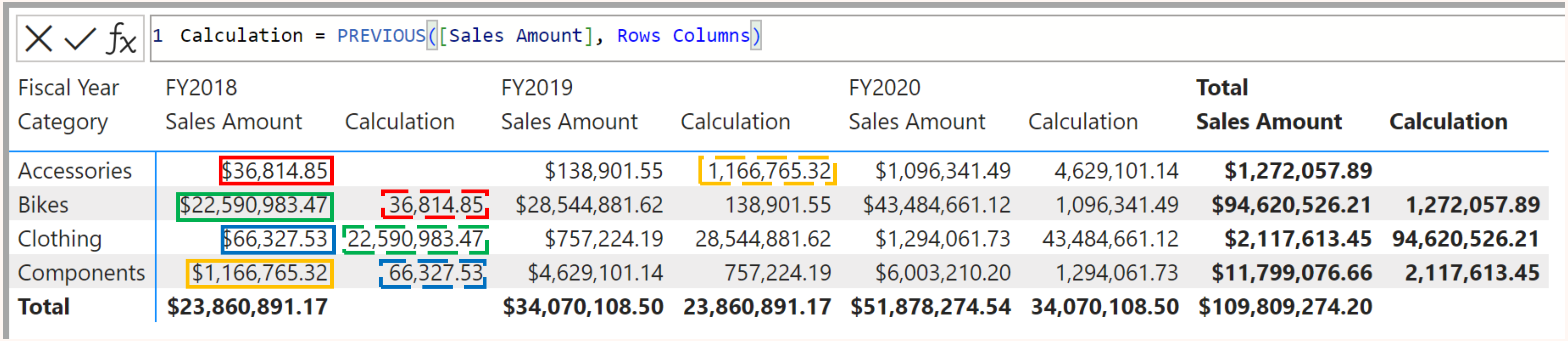


```
Calculation = FIRST([Sales Amount], Columns)
```

Fiscal Year	FY2018		FY2019		FY2020		**Total**	
Category	Sales Amount	Calculation	Sales Amount	Calculation	Sales Amount	Calculation	**Sales Amount**	**Calculation**
Accessories	$36,814.85	36,814.85	$138,901.55	36,814.85	$1,096,341.49	36,814.85	**$1,272,057.89**	**1,272,057.89**
Bikes	$22,590,983.47	22,590,983.47	$28,544,881.62	22,590,983.47	$43,484,661.12	22,590,983.47	**$94,620,526.21**	**94,620,526.21**
Clothing	$66,327.53	66,327.53	$757,224.19	66,327.53	$1,294,061.73	66,327.53	**$2,117,613.45**	**2,117,613.45**
Components	$1,166,765.32	1,166,765.32	$4,629,101.14	1,166,765.32	$6,003,210.20	1,166,765.32	**$11,799,076.66**	**11,799,076.66**
Total	**$23,860,891.17**	**23,860,891.17**	**$34,070,108.50**	**23,860,891.17**	**$51,878,274.54**	**23,860,891.17**	**$109,809,274.20**	**109,809,274.20**

Example: Axis

`PREVIOUS([Sales Amount], Rows Columns)` →
for each cell, retrieve the previous Sales Amount from the *Row* above it or from the last cell in the previous *Column*

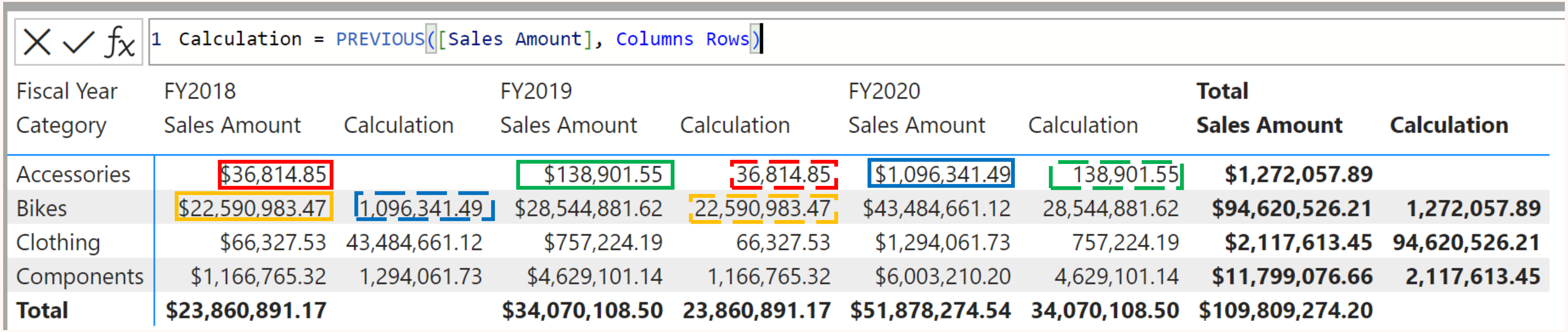


```
Calculation = PREVIOUS([Sales Amount], Rows Columns)
```

Fiscal Year	FY2018		FY2019		FY2020		**Total**	
Category	Sales Amount	Calculation	Sales Amount	Calculation	Sales Amount	Calculation	**Sales Amount**	**Calculation**
Accessories	$36,814.85		$138,901.55	1,166,765.32	$1,096,341.49	4,629,101.14	**$1,272,057.89**	
Bikes	$22,590,983.47	36,814.85	$28,544,881.62	138,901.55	$43,484,661.12	1,096,341.49	**$94,620,526.21**	**1,272,057.89**
Clothing	$66,327.53	22,590,983.47	$757,224.19	28,544,881.62	$1,294,061.73	43,484,661.12	**$2,117,613.45**	**94,620,526.21**
Components	$1,166,765.32	66,327.53	$4,629,101.14	757,224.19	$6,003,210.20	1,294,061.73	**$11,799,076.66**	**2,117,613.45**
Total	**$23,860,891.17**		**$34,070,108.50**	**23,860,891.17**	**$51,878,274.54**	**34,070,108.50**	**$109,809,274.20**	

Example: Axis

`PREVIOUS([Sales Amount], Columns Rows)` →
for each cell, retrieve the previous Sales Amount from the *Column* to the left of it or from the last cell of the previous *Row*



```
Calculation = PREVIOUS([Sales Amount], Columns Rows)
```

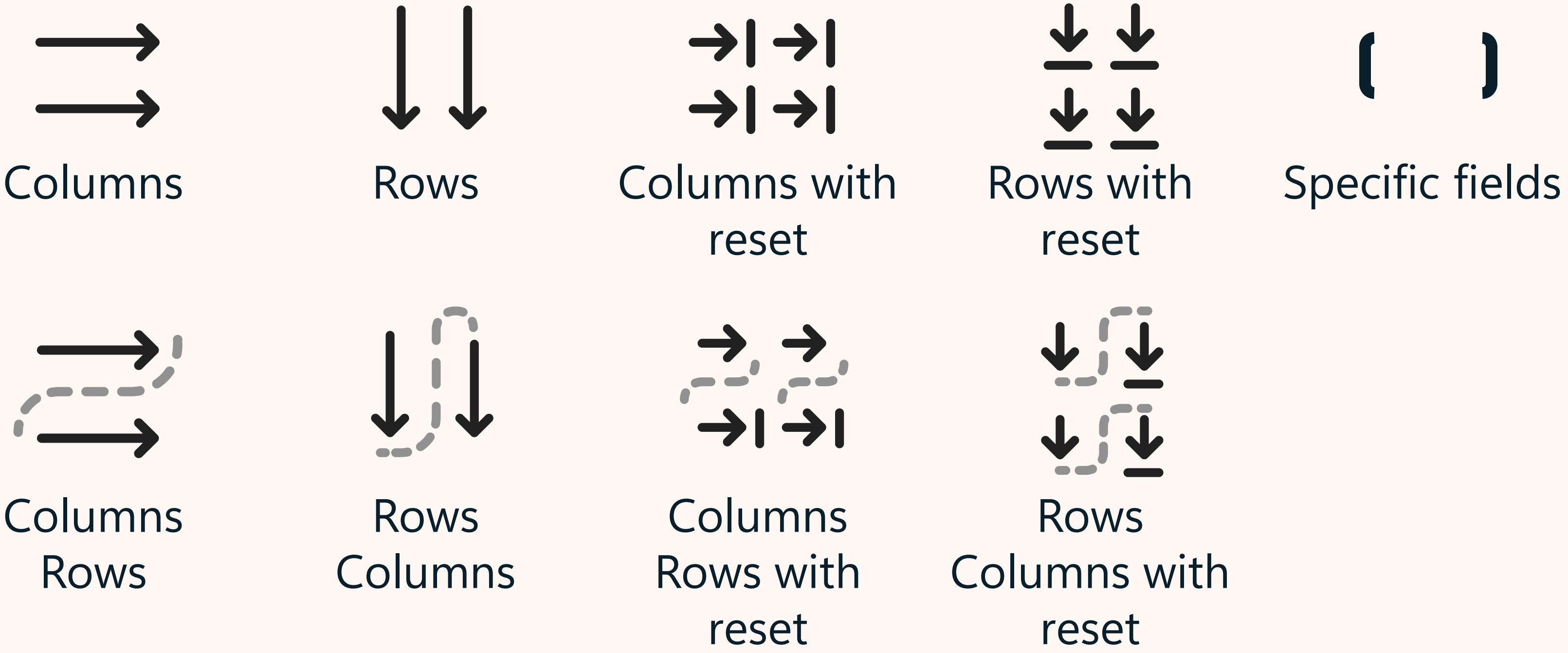
Fiscal Year	FY2018		FY2019		FY2020		**Total**	
Category	Sales Amount	Calculation	Sales Amount	Calculation	Sales Amount	Calculation	**Sales Amount**	**Calculation**
Accessories	$36,814.85		$138,901.55	36,814.85	$1,096,341.49	138,901.55	**$1,272,057.89**	
Bikes	$22,590,983.47	1,096,341.49	$28,544,881.62	22,590,983.47	$43,484,661.12	28,544,881.62	**$94,620,526.21**	**1,272,057.89**
Clothing	$66,327.53	43,484,661.12	$757,224.19	66,327.53	$1,294,061.73	757,224.19	**$2,117,613.45**	**94,620,526.21**
Components	$1,166,765.32	1,294,061.73	$4,629,101.14	1,166,765.32	$6,003,210.20	4,629,101.14	**$11,799,076.66**	**2,117,613.45**
Total	**$23,860,891.17**		**$34,070,108.50**	**23,860,891.17**	**$51,878,274.54**	**34,070,108.50**	**$109,809,274.20**	

Concept: Reset

- None: continue counting, never restart
- `LowestParent`: start over for each parent of the lowest level on the specified axis
- `HighestParent`: start over on the highest level on the specified axis
- [n]: start over on the nth level on the specified axis
- [Field reference] (example: `reset([Year]`)

Concept: Direction

- Direction is the combination of *Axis* and *Reset*
- The ‘specific fields’ “direction” does not rely on *Axis* but instead references fields



Columns

Rows

Columns with reset

Rows with reset

Specific fields

Columns Rows

Rows Columns

Columns Rows with reset

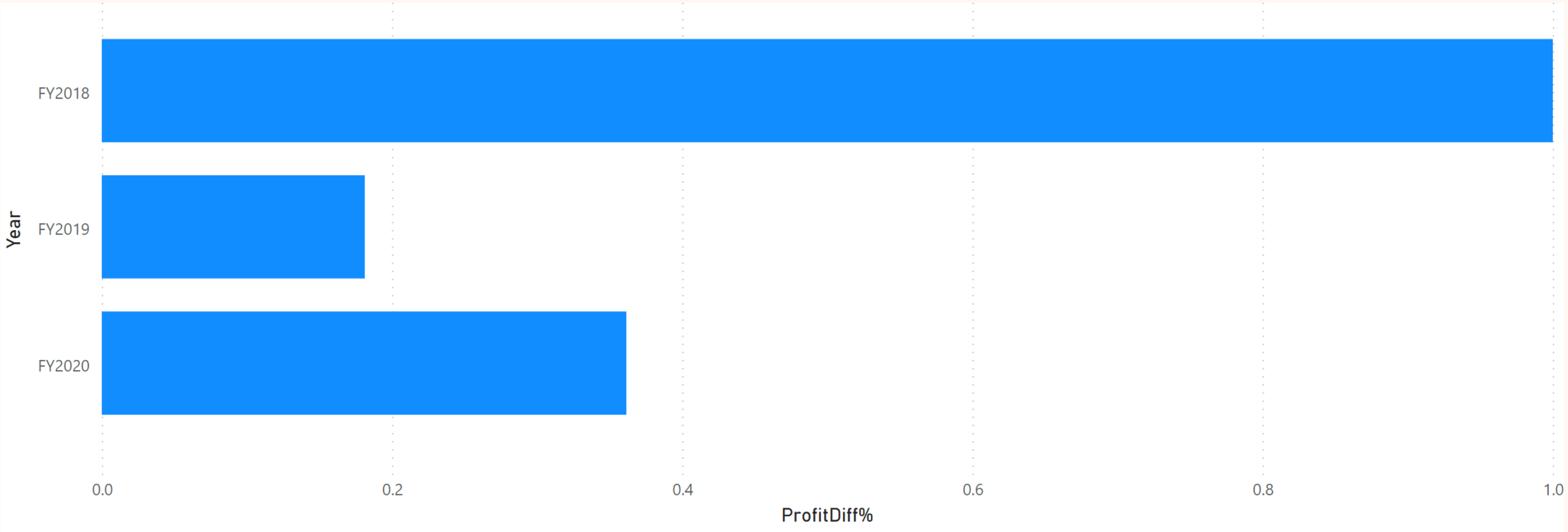
Rows Columns with reset

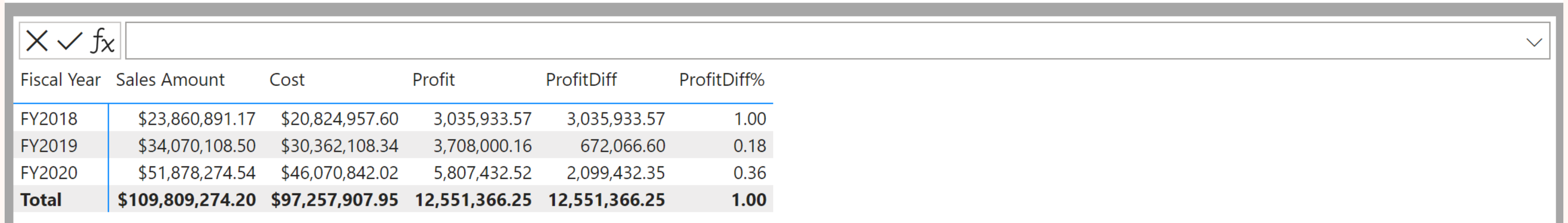
Concept: Hidden

- All fields on the visual are on the visual matrix, but not all fields on the visual matrix are visible on the visual.
- Remember: visual calcs can refer to what’s on the visual matrix, not to anything else.
- Allows for “partial results”

Example: Hidden

- `Profit = [Sales Amount] – [Cost] (hidden)`
- `ProfitDiff = [Profit] – Previous([Profit]) (hidden)`
- `ProfitDiff% = DIVIDE([ProfitDiff], [Profit])`





Fiscal Year	Sales Amount	Cost	Profit	ProfitDiff	ProfitDiff%
FY2018	$23,860,891.17	$20,824,957.60	3,035,933.57	3,035,933.57	1.00
FY2019	$34,070,108.50	$30,362,108.34	3,708,000.16	672,066.60	0.18
FY2020	$51,878,274.54	$46,070,842.02	5,807,432.52	2,099,432.35	0.36
Total	**$109,809,274.20**	**$97,257,907.95**	**12,551,366.25**	**12,551,366.25**	**1.00**

Demo

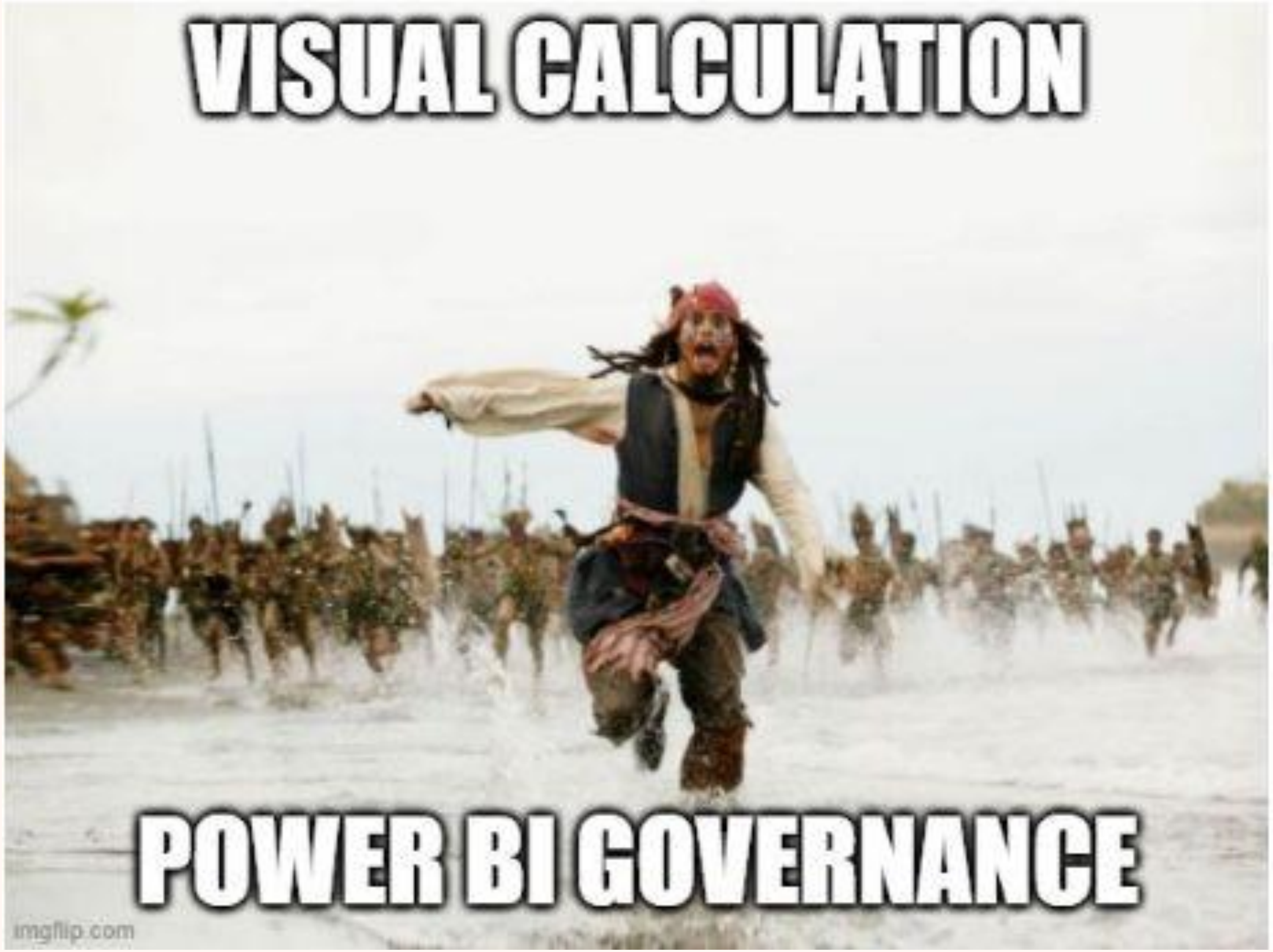
But where is `calculate`?

Only in the **center** the user is exposed to `calculate` and worry about complexities such as context transition

Higher-level (medium and high) functions have signatures that expand into calculate statements.

Example:

```
first([Sales], [Year]) →
calculate([Sales], first[Year]) →
calculate([Sales], index(1, [Year])
```


VISUAL CALCULATION
POWER BI GOVERNANCE
imgflip.com

Resources

- Documentation of released functions: INDEX, OFFSET, WINDOW, RANK, ROWNUMBER, ORDERBY, PARTITIONBY, MATCHBY
- Understanding ORDERBY, PARTITIONBY and MATCHBY
- Using visual calculations
  https://aka.ms/visual-calculations-docs
- Calculations options
  https://aka.ms/powerbi-calculation-options
- Provide feedback on visual calculations
  https://aka.ms/visual-calculations-feedback

Questions?

Rate this session

Marc Lelijveld

Technical Evangelist & Architect
At Macaw

in linkedin.com/in/MarcLelijveld

Data-Marc.com