



DATA WITH BARAA

AGGREGATE FUNCTIONS

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Aggregate Functions



Aggregate Functions

Data Types		
Any Types	COUNT	Counts the number of rows
Only Numbers	SUM	Add up all values in a column
	AVG	Find the average of values
Any Types	MAX	Gets the highest value
	MIN	Gets the lowest value

Aggregate Functions

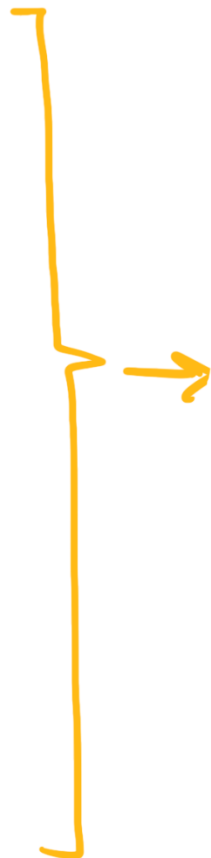
Sales

35

15

20

10



COUNT(*)

4

SUM

80

AVG

20

MAX

35

MIN

10



DATA WITH BARAA

WINDOW FUNCTIONS

BASICS

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Window Functions



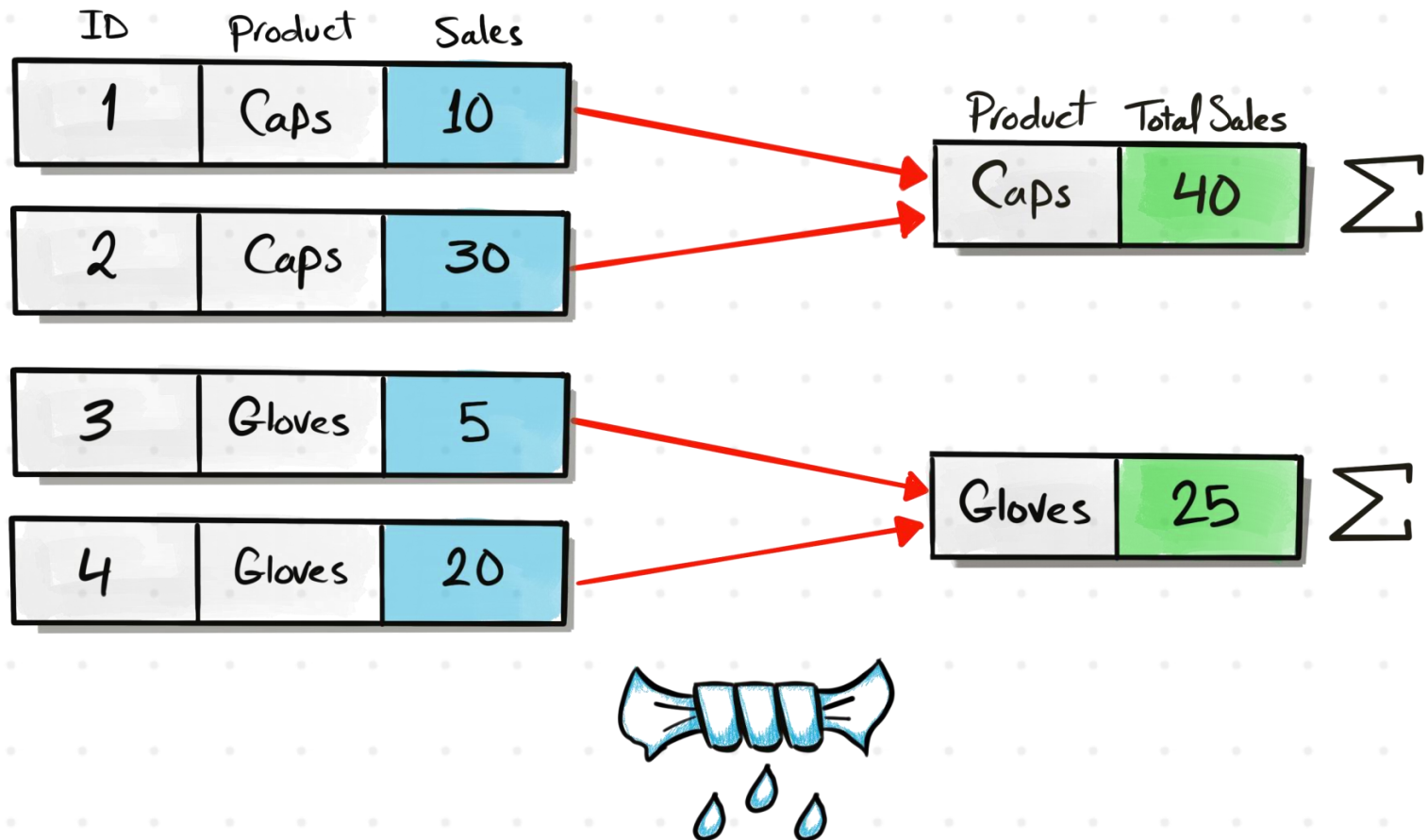


WINDOW FUNCTIONS

**Perform calculations (e.g. aggregation)
on a specific subset of data,
without losing the level of details of rows.**

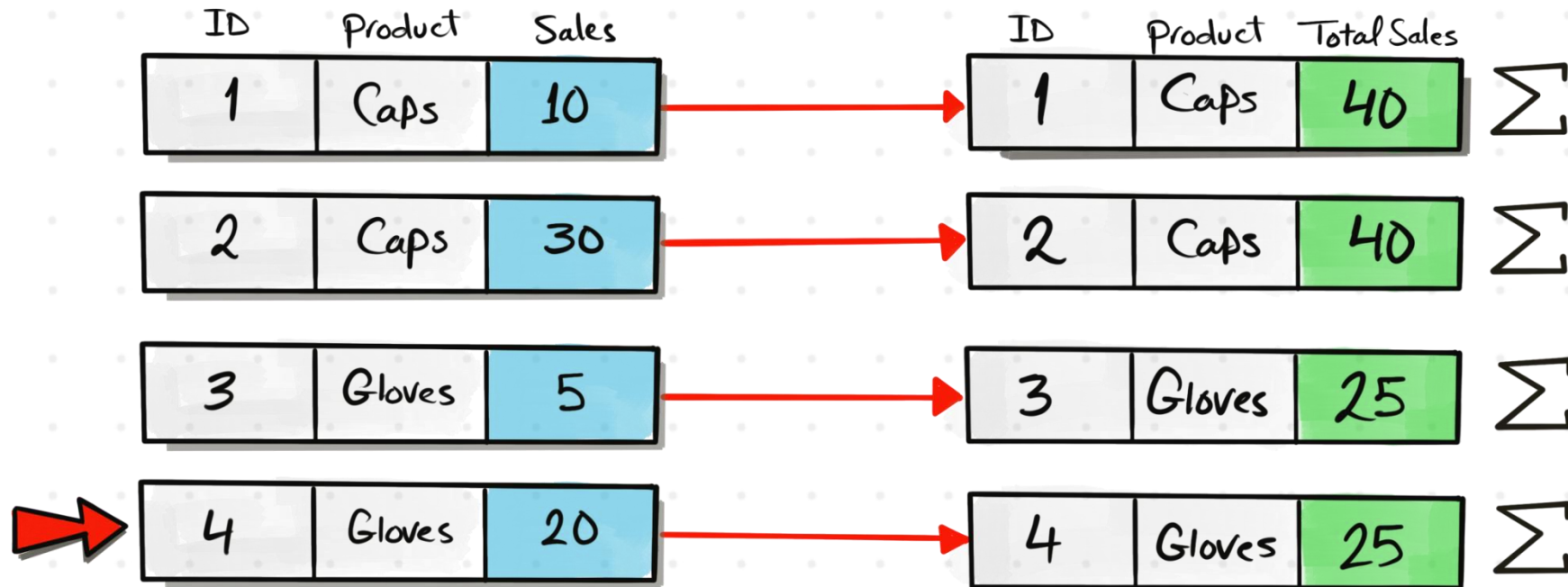
GROUP BY

Aggregates and groups rows **based** on column/s into **summary rows**



WINDOW Functions

Compute **aggregates** but **keep details** of individual rows at the same time



Row-Level Calculation

GROUP BY

ID	Product	Sales	
1	Caps	10	
2	Caps	30	
3	Gloves	5	
4	Gloves	20	

Product	Total Sales	
Caps	40	Σ
Gloves	25	Σ

Diagram illustrating GROUP BY: Rows are grouped by Product (Caps and Gloves). The Sales values are summed for each group to calculate Total Sales (40 for Caps, 25 for Gloves).

It **collapse** groups of rows into a single row
(Group-Level-Calculations)

Window Functions

ID	Product	Sales	
1	Caps	10	
2	Caps	30	
3	Gloves	5	
4	Gloves	20	

ID	Product	Total Sales	
1	Caps	40	Σ
2	Caps	40	Σ
3	Gloves	25	Σ
4	Gloves	25	Σ

Diagram illustrating Window Functions: Rows are not collapsed. Each row is calculated individually, showing the total sales for its product group (40 for Caps, 25 for Gloves) next to its original sales value.

It **doesn't collapse** rows into a single row
(Row-Level-Calculations)

GROUP BY

Functions

Aggregate Functions

COUNT (*expr*)

SUM (*expr*)

MAX (*expr*)

MIN (*expr*)

MIN (*expr*)

WINDOW

Functions

Aggregate Functions

COUNT (*expr*)

SUM (*expr*)

MAX (*expr*)

MIN (*expr*)

MIN (*expr*)

Rank Functions

ROW_NUMBER ()

RANK ()

DENSE_RANK ()

CUME_DIST ()

PERCENT_RANK ()

NTILE (*n*)

Value (Analytics) Functions

LEAD (*expr, offset, default*)

LAG (*expr, offset, default*)

FIRST_VALUE (*expr*)

FIRST_VALUE (*expr*)

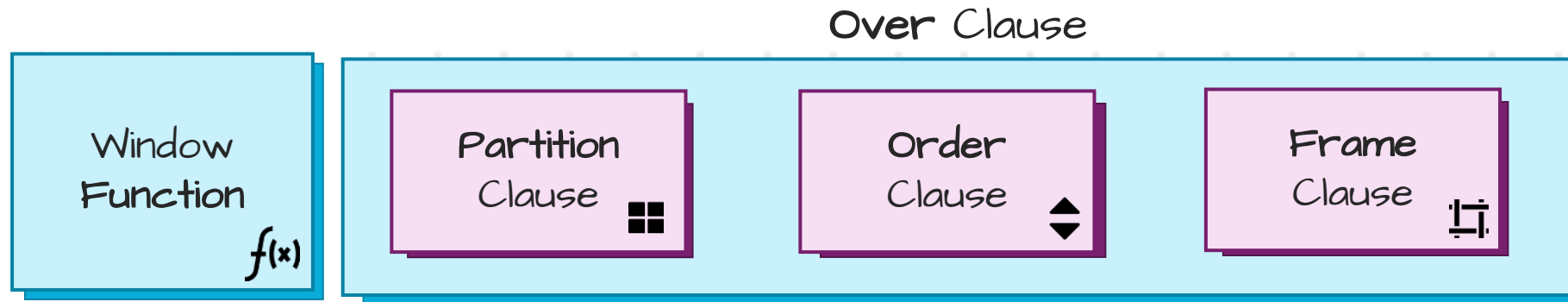
GROUP BY

**Simple Data Analysis
(Aggregations)**

WINDOW

**Advanced Data Analysis
(Aggregations + Details)**

Window Syntax



Window Syntax

```
AVG(Sales) OVER ( PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING )
```


Window Syntax

Calculation used
on the Window

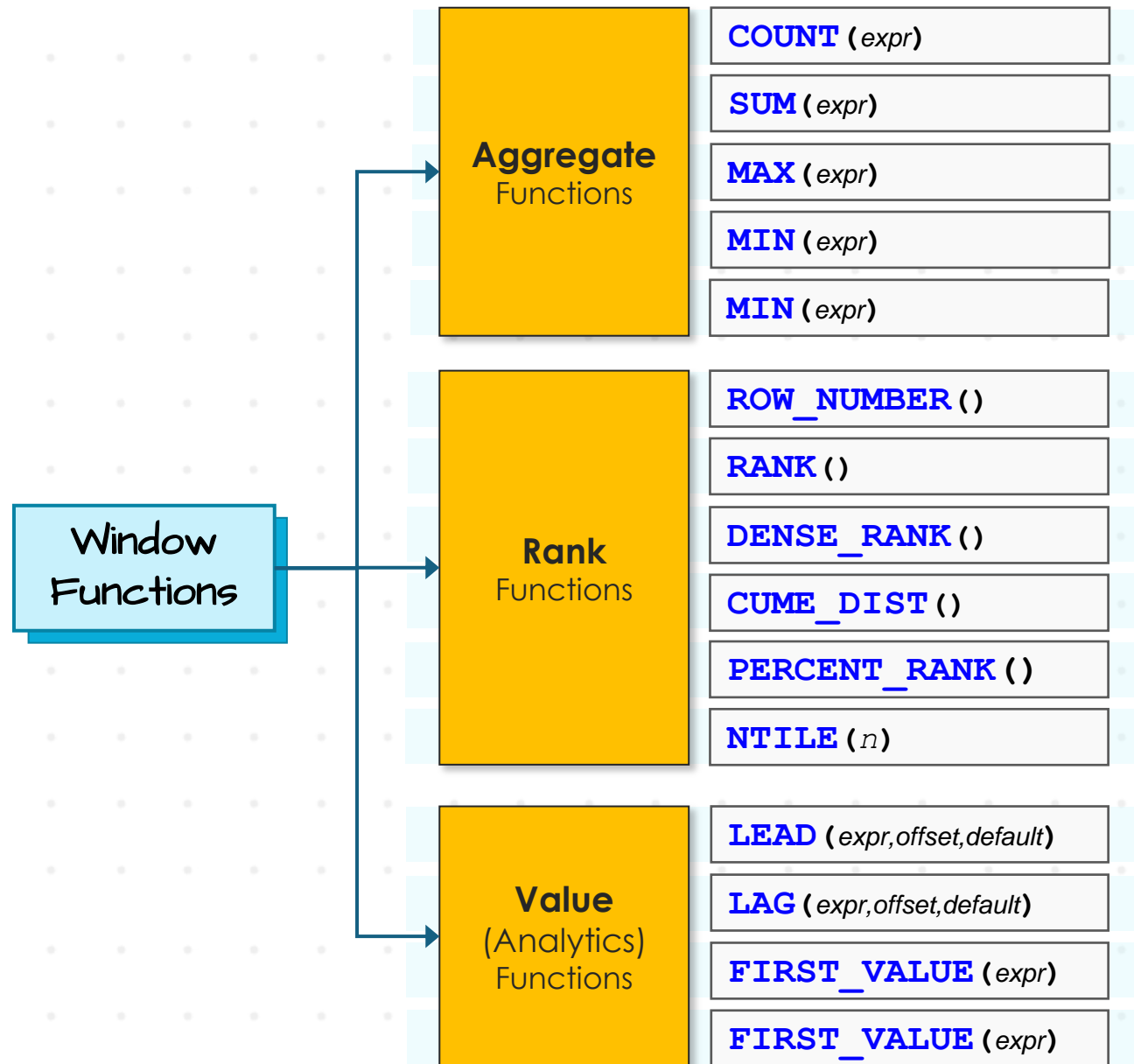
f(*) Window
Function

AVG (Sales) OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)



WINDOW FUNCTIONS

Perform calculations within a window



Window Syntax

Calculation used
on the Window

f(*) Window
Function

AVG (Sales) OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function
Expression

FUNCTION EXPRESSION

Arguments you pass to a function

Window Expressions

Empty

`RANK() OVER (ORDER BY OrderDate)`

Column

`AVG(Sales) OVER (ORDER BY OrderDate)`

Number

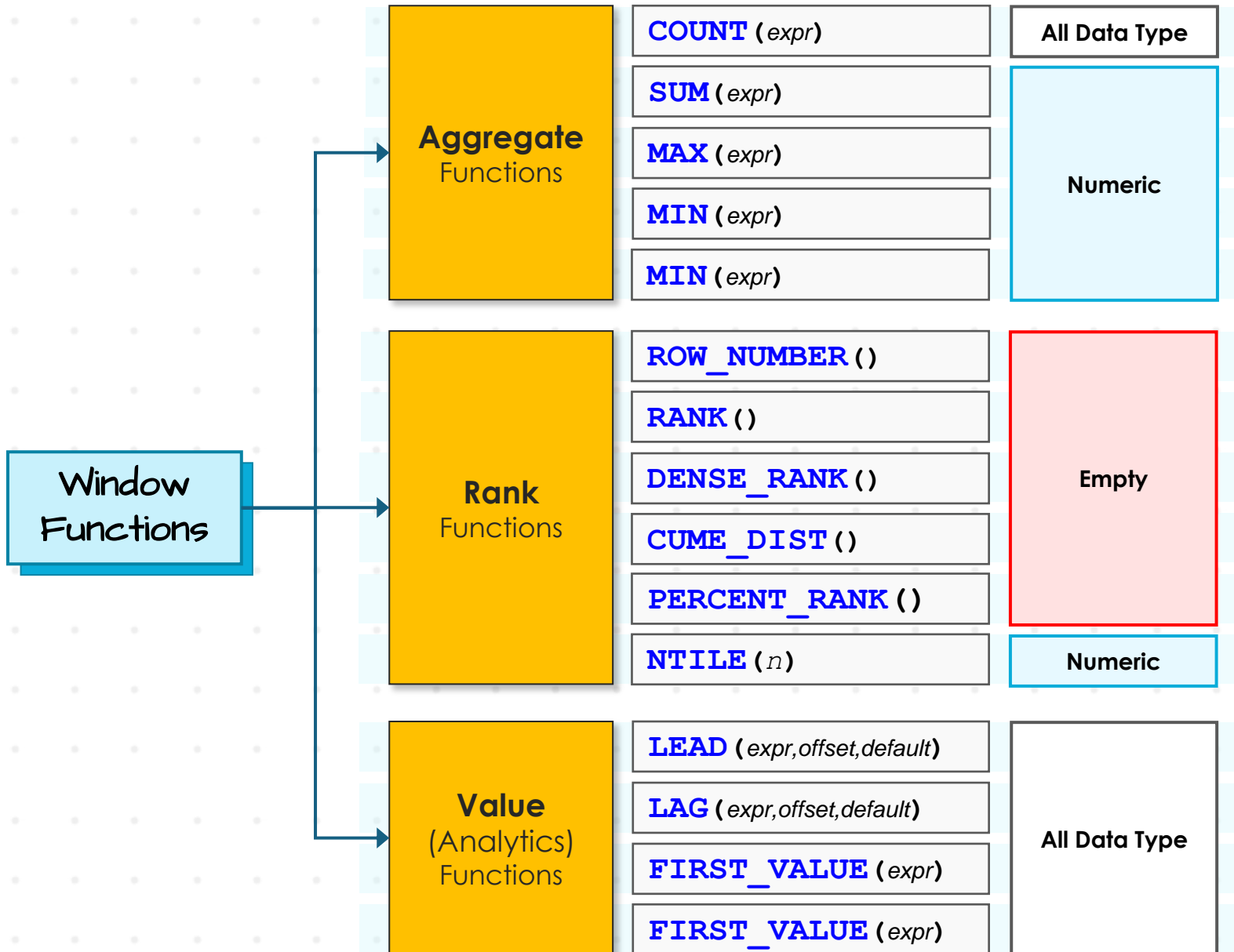
`NTEIL(2) OVER (ORDER BY OrderDate)`

Multiple Arguments

`LEAD(Sales,2,10) OVER (ORDER BY OrderDate)`

Conditional Logic

`SUM(CASE WHEN Sales > 100 THEN 1 ELSE 0 END) OVER (ORDER BY OrderDate)`



Window Syntax

Calculation used
on the Window

f(*) Window
Function

Define the
Window

Over
Clause

AVG (Sales)

OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function
Expression

OVER CLAUSE

Tells SQL that the function used is a **window function**

Window Syntax

Calculation used
on the Window

$f(*)$ Window
Function

Define the
Window

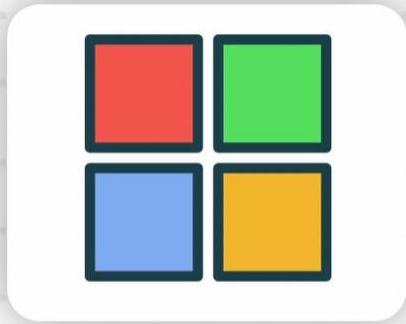
Over
Clause

AVG (Sales) **OVER** (**PARTITION BY** Category **ORDER BY** OrderDate **ROWS UNBOUNDED PRECEDING**)

Function
Expression

Partition
Clause 

Divides the dataset into
windows (Partitions)



PARTITION BY

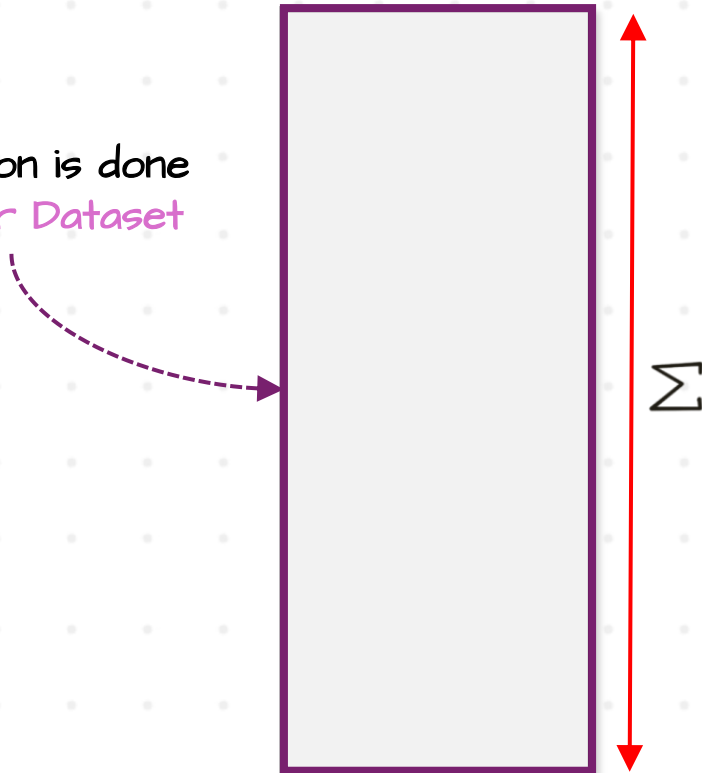
Divides the result set into partitions (Windows)

Partition By

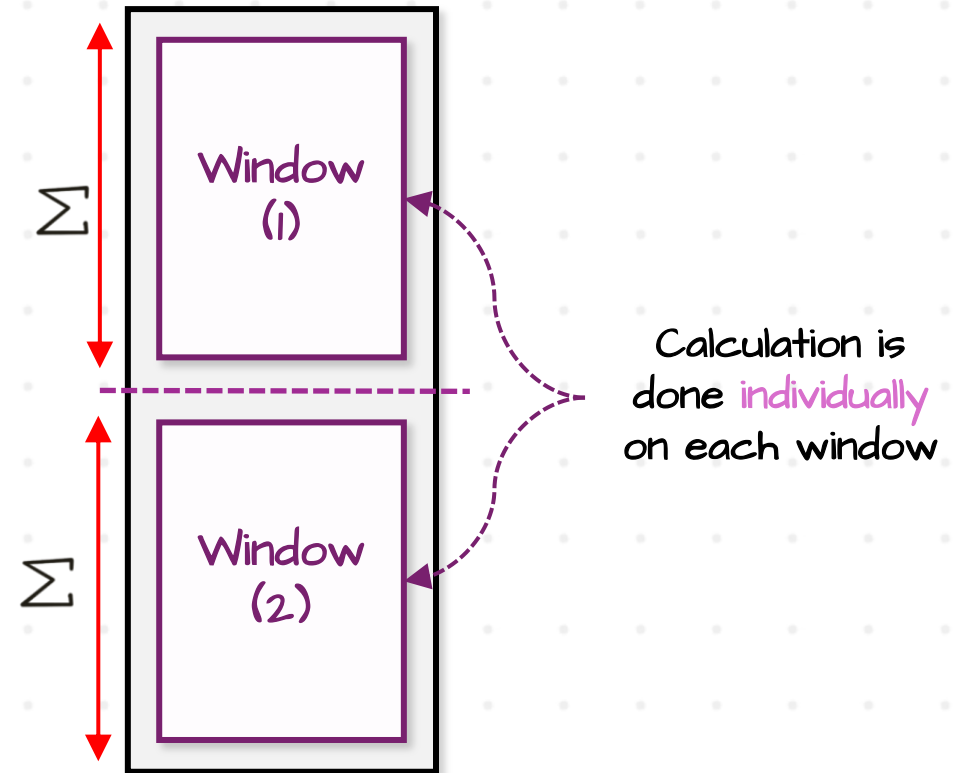
PARTITION BY divides the rows into groups, based on the column/s

`SUM(Sales) OVER ()`

Calculation is done
on *entier Dataset*



`SUM(Sales) OVER (PARTITION BY Product)`



Partition By

PARTITION BY divides the rows into groups, based on the column/s

Without
Partition By

Total sales across all rows (Entire Result Set)

```
SUM(Sales) OVER ()
```

Partition By
Single Column

Total sales for each Product

```
SUM(Sales) OVER (PARTITION BY Product)
```

Partition By
Combined-Columns

Total sales for each combination of Product and Order Status

```
SUM(Sales) OVER (PARTITION BY Product, OrderStatus)
```

		Expression	Partition Clause
Window Functions	Aggregate Functions	<div>COUNT (expr)</div> <div>SUM (expr)</div> <div>MAX (expr)</div> <div>MIN (expr)</div> <div>MIN (expr)</div>	<div>All Data Type</div> <div>Numeric</div>
	Rank Functions	<div>ROW_NUMBER ()</div> <div>RANK ()</div> <div>DENSE_RANK ()</div> <div>CUME_DIST ()</div> <div>PERCENT_RANK ()</div> <div>NTILE (n)</div>	<div>Empty</div> <div>Numeric</div>
	Value (Analytics) Functions	<div>LEAD (expr,offset,default)</div> <div>LAG (expr,offset,default)</div> <div>FIRST_VALUE (expr)</div> <div>FIRST_VALUE (expr)</div>	<div>All Data Type</div>

Window Syntax

Calculation used on the Window

$f(*)$ Window Function

Define the Window

Over Clause

AVG (Sales) **OVER** (**PARTITION BY** Category **ORDER BY** OrderDate **ROWS UNBOUNDED PRECEDING**)

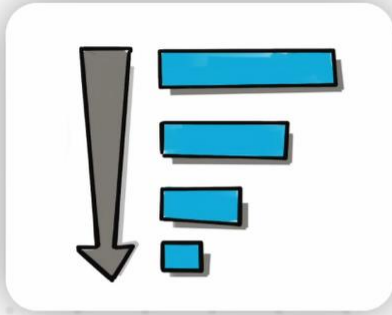
Function Expression

Partition Clause 

Divides the dataset into windows (Partitions)

Order Clause 

Sort the data in a window



ORDER BY

Sort the data within a window

(Ascending | Descending)

		Expression	Partition Clause	Order Clause
Window Functions	Aggregate Functions	COUNT (expr) SUM (expr) MAX (expr) MIN (expr) MIN (expr)	All Data Type Numeric	Optional
	Rank Functions	ROW_NUMBER () RANK () DENSE_RANK () CUME_DIST () PERCENT_RANK () NTILE (n)	Empty Numeric	Required
	Value (Analytics) Functions	LEAD (expr,offset,default) LAG (expr,offset,default) FIRST_VALUE (expr) FIRST_VALUE (expr)	All Data Type	Required

Window Syntax

Calculation used on the Window

$f(*)$ Window Function

Define the Window

Over Clause

AVG (Sales) OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function Expression

Partition Clause 

Divides the dataset into windows (Partitions)

Order Clause 

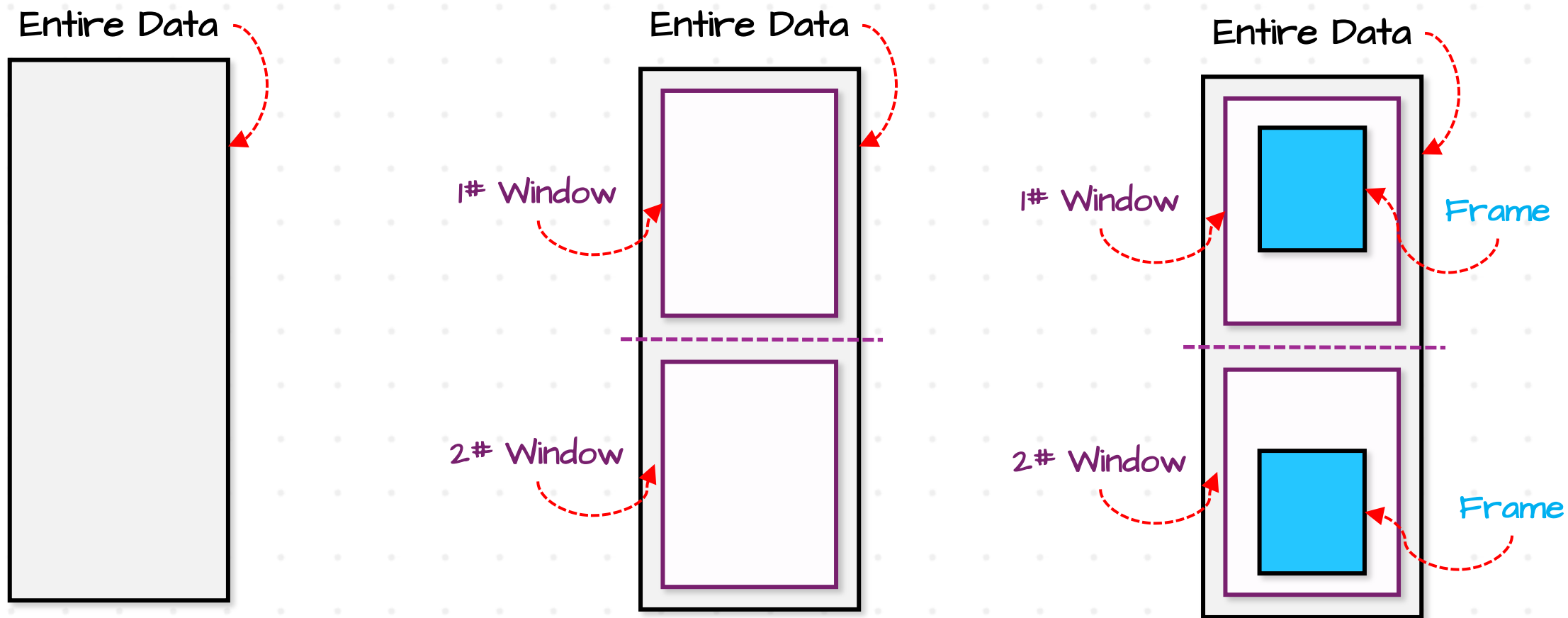
Sort the data in a window

Frame Clause 

Define a subset of rows in a window



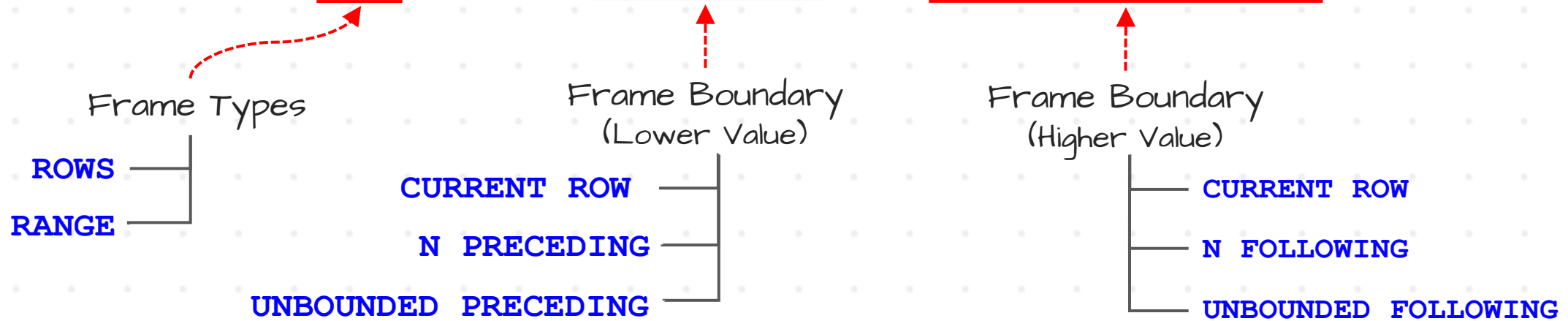
Frame Clause



		Expression	Partition Clause	Order Clause	Frame Clause
Window Functions	Aggregate Functions	COUNT (expr)	Optional	Optional	Optional
		SUM (expr)			
		AVG (expr)			
		MAX (expr)			
		MIN (expr)			
	Rank Functions	ROW_NUMBER ()		Required	Not allowed
		RANK ()			
		DENSE_RANK ()			
		CUME_DIST ()			
		PERCENT_RANK ()			
		NTILE (n)			
	Value (Analytics) Functions	LEAD (expr,offset,default)		Required	Not allowed
		LAG (expr,offset,default)			
		FIRST_VALUE (expr)			Optional
		FIRST_VALUE (expr)			Should be used

AVG(Sales) OVER (PARTITION BY Category ORDER BY OrderDate

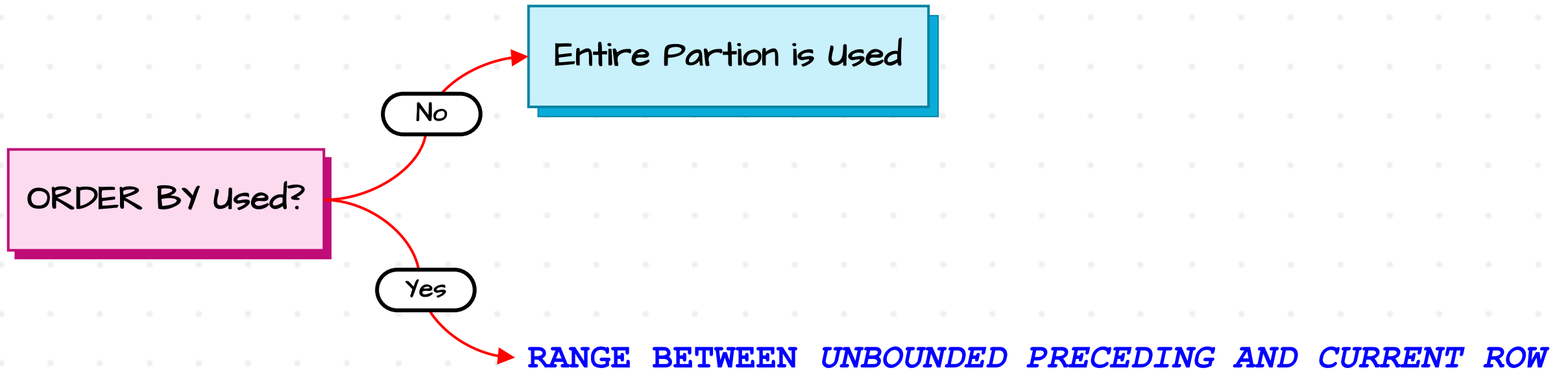
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)



Rules

- Frame Clause can only be used **together** with order by clause.
- Lower Value must be **BEFORE** the higher Value.

Frame



COMPACT FRAME

For only PRECEDING, the CURRENT ROW can be skipped

NORMAL FORM

ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING

SHORT FORM

ROWS 2 FOLLOWING

#1 RULE

Window functions can be used ONLY
in **SELECT** and **ORDER BY** Clauses

#2 RULE

Nesting Window Functions is not allowed !

#3 RULE

SQL **execute** WINDOW Functions **after** WHERE Clause

#4 RULE

Window Function can be used together with GROUP BY
in the same query, ONLY if the same **columns** are used

Window Rules

Inner Window Function

AVG (SUM(Sales) OVER ()) OVER (ORDER BY OrderDate)

Outer Window Function



Not allowed to nest
window functions !

Windowed functions cannot be used in the context of another windowed function or aggregate.

SQL WINDOW FUNCTIONS

Performs calculations on subset of data without losing details

Window v/s Group By

- Window is more powerfull & Dynamic than Group By.
- Data Analysis
 - Advanced → Window
 - Simple → Group By
- Use Group By + window in same Query, only if same Column used.

Components

Window Functions + Window Definition **OVER**



Rules

- Nesting is not allowed!
- Window Can be used only in **SELECT** and **ORDER BY**
- SQL executes window **AFTER** filtering data using **WHERE**

$f(x)$ Window Functions

Aggregate

- SUM ()
- AVG ()
- COUNT ()
- MAX ()
- MIN ()

Perform calculations on a set of rows and return a **single aggregated** value for each row

Rank

- ROW_NUMBER ()
- RANK ()
- DENSE_RANK ()
- NTILE ()
- CUME_DIST ()
- PERCENT_RANK ()

Assign a **rank** to each row in a window

Value

- LAG ()
- LEAD ()
- FIRST_VALUE ()
- LAST_VALUE ()

Return a **specific value** in a window to be compared with the value of **current row**



DATA WITH BARAA

WINDOW AGGREGATE FUNCTIONS

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Window Aggregate Functions



$f(x)$ Window Functions

Aggregate

SUM ()

AVG ()

COUNT ()

MAX ()

MIN ()

Perform calculations on a set of rows and return a **single aggregated** value for each row

Rank

ROW_NUMBER ()

RANK ()

DENSE_RANK ()

NTILE ()

CUME_DIST ()

PERCENT_RANK ()

Assign a rank to each row in a window

Value

LAG ()

LEAD ()

FIRST_VALUE ()

LAST_VALUE ()

Return a specific value in a window to be compared with the value of current row

Month	Sales
Jan	20
Feb	10
Mar	30
Apr	5
Jun	70
Jul	40

 Σ

175

One Aggregated Value



Aggregation is combining multiple values into a single summary

Aggregate Functions

AVG(Sales) OVER (PARTITION BY ProductID ORDER BY Sales)

↑
Expression
is **required**
(Only **Numeric** Values)

↑
Partition By
Is **Optional**

↑
Order By
Is **Optional**

Aggregate Functions

		Expression	Partition Clause	Order Clause	Frame Clause
Aggregate Functions	COUNT (<i>expr</i>)	All Data Type	Optional	Optional	Optional
	SUM (<i>expr</i>)	Numeric Values			
	AVG (<i>expr</i>)	Numeric Values			
	MIN (<i>expr</i>)	Numeric Values			
	MAX (<i>expr</i>)	Numeric Values			

Aggregate Functions

Aggregate Functions

COUNT (*expr*)

Returns the number of Rows in a window

COUNT (*) **OVER** (**PARTITION BY** Product)

SUM (*expr*)

Returns the sum of values in a window

SUM(Sales) **OVER** (**PARTITION BY** Product)

AVG (*expr*)

Returns the average of values in a window

SUM(Sales) **OVER** (**PARTITION BY** Product)

MIN (*expr*)

Returns the minimum value in a window

SUM(Sales) **OVER** (**PARTITION BY** Product)

MAX (*expr*)

Returns the maximum value in a window

SUM(Sales) **OVER** (**PARTITION BY** Product)

COUNT Function

Returns the **number of Rows** in a window

```
COUNT (*) OVER (PARTITION BY Product)
```

Product	Sales	Count
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	3
Gloves	70	3
Gloves	40	3

3 Orders for **Caps**

3 Orders for **Gloves**

COUNT Function

Count the number of Rows
including NULLs

```
COUNT (*) OVER (PARTITION BY Product)
```

```
COUNT (1) OVER (PARTITION BY Product)
```

Product	Sales	Count
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	3
Gloves	70	3
Gloves	NULL	3

This Row *is* counted

Count the number of Rows
excluding NULLs

```
COUNT(Sales) OVER (PARTITION BY Product)
```

Column

Product	Sales	Count
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	2
Gloves	70	2
Gloves	NULL	2

This Row *won't be* counted

COUNT Function

COUNT (1) OVER (PARTITION BY Product)

COUNT (*) OVER (PARTITION BY Product)

Product	Sales	COUNT
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	3
Gloves	70	3
Gloves	NULL	3

COUNT (Sales) OVER (PARTITION BY Product)

Product	Sales	COUNT
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	2
Gloves	70	2
Gloves	NULL	2

SUM Function

Returns the **sum** of values in a window

```
SUM(Sales) OVER(PARTITION BY Product)
```

* Is **not** allowed!

Product	Sales	SUM
Caps	20	35
Caps	10	35
Caps	5	35
Gloves	30	140
Gloves	70	140
Gloves	40	140

$20 + 10 + 5 = 35$

$30 + 70 + 40 = 140$

SUM Function

`SUM(Sales) OVER(PARTITION BY Product)`

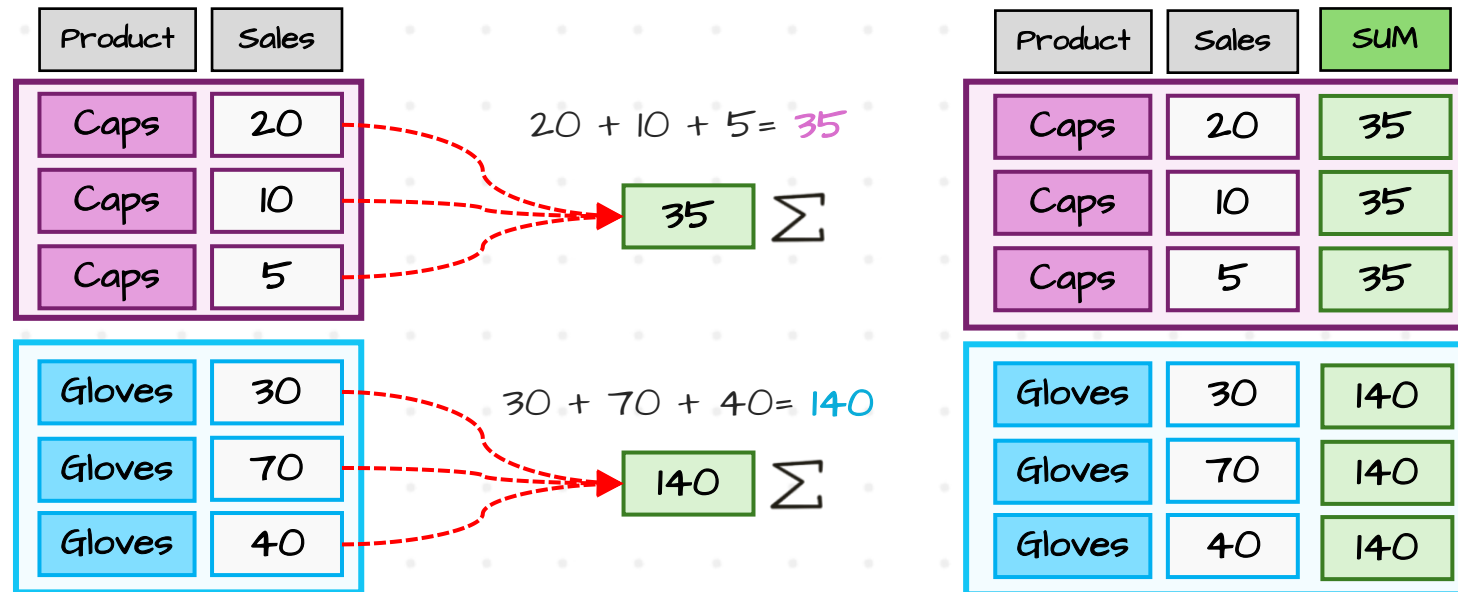
Product	Sales	SUM
Caps	20	35
Caps	10	35
Caps	5	35
Gloves	30	100
Gloves	70	100
Gloves	NULL	100

$$20 + 10 + 5 = 35$$

$$30 + 70 = 100$$

COUNT Function

Perform calculations on a set of rows and return a **single aggregated value** for each row



AVG Function

Returns the **average** of values in a window

AVG (**Sales**) **OVER** (**PARTITION BY** Product)

* Is **not** allowed!

Product	Sales	Avg
Caps	20	
Caps	10	
Caps	5	
Gloves	30	46
Gloves	70	46
Gloves	40	46

$$\frac{20 + 10 + 5}{3} = ||$$

$$\frac{30 + 70 + 40}{3} = 46$$

AVG Function

Default Average Function
exclude NULLs



`AVG(Sales) OVER(PARTITION BY Product)`

Product	Sales	AVG
Caps	20	
Caps	10	
Caps	5	
Gloves	30	50
Gloves	70	50
Gloves	NULL	50

$$\frac{20 + 10 + 5}{3} = ||$$

$$\frac{30 + 70}{2} = 50$$

Deal with Nulls using **COALESCE** to
include NULLs



`AVG(COALESCE(Sales, 0)) OVER(PARTITION BY Product)`

Replace NULL with 0

Product	Sales	AVG
Caps	20	
Caps	10	
Caps	5	
Gloves	30	50
Gloves	70	50
Gloves	0	50

$$\frac{20 + 10 + 5}{3} = ||$$

$$\frac{30 + 70 + 0}{3} = 33$$

AVG Function

`AVG (COALESCE (Sales,0)) OVER (PARTITION BY Product)`

Product	Sales	AVG
Caps	20	
Caps	10	
Caps	5	
$(20 + 10 + 5) / 3 = $		
Gloves	30	33
Gloves	70	33
Gloves	0	33
$(30 + 70 + 0) / 3 = 33$		

`AVG (Sales) OVER (PARTITION BY Product)`

MIN

Returns the **minimum** value in a window

```
MIN(Sales) OVER(PARTITION BY Product)
```

Product	Sales	MIN	
Caps	20	5	5 is the lowest sales for Caps
Caps	10	5	
Caps	5	5	
Gloves	30	30	30 is the lowest sales for Gloves
Gloves	70	30	
Gloves	40	30	

MAX

Returns the **maximum** value in a window



```
MAX(Sales) OVER(PARTITION BY Product)
```

Product	Sales	MAX	
Caps	20	20	20 is the highest sales for Caps
Caps	10	20	
Caps	5	20	
Gloves	30	70	70 is the highest sales for Gloves
Gloves	70	70	
Gloves	40	70	

MAX & MIN Function

Find the highest sales for each product



MIN(Sales) **OVER**(PARTITION BY Product)



Product	Sales	MIN
Caps	20	5
Caps	10	5
Caps	5	5
Gloves	30	0
Gloves	70	0
Gloves	0	0

Find the lowest sales for each product

MAX(Sales) **OVER**(PARTITION BY Product)



Product	Sales	MAX
Caps	20	20
Caps	10	20
Caps	5	20
Gloves	30	70
Gloves	70	70
Gloves	0	70

RUNNING TOTAL

Aggregate all values from the beginning up to the current point without dropping off older data.

ROLLING TOTAL

Aggregate all values within a fixed time window (e.g. 30 days).
As new data is added, the oldest data point will be dropped.



Rolling/Shifting Window

Running Total

`SUM(Sales) OVER (ORDER BY Month)`

Default

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

	Month	Sales	SUM
UNBOUNDED PRECEDING	Jan	20	20
	Feb	10	30
	Mar	30	60
	Apr	5	65
	Jun	70	135
Current Row	Jul	40	175

Rolling Total

`SUM(Sales) OVER (ORDER BY MONTH
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)`

	Month	Sales	SUM
	Jan	20	20
	Feb	10	30
	Mar	30	60
2 PRECEDING	Apr	5	45
	Jun	70	105
Current Row	Jul	40	105

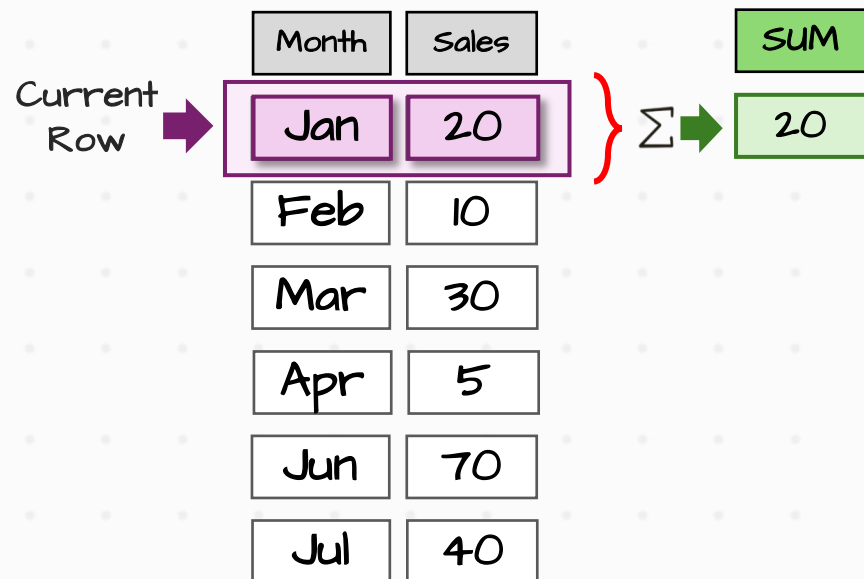
Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(  
ORDER BY Month)
```

Default Frame

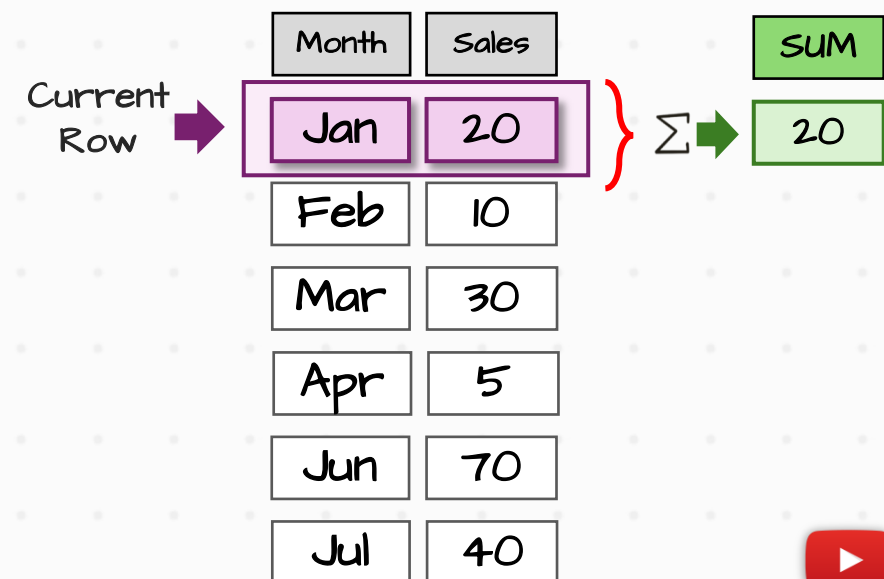
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(  
ORDER BY Month ROWS 2 PRECEDING)
```



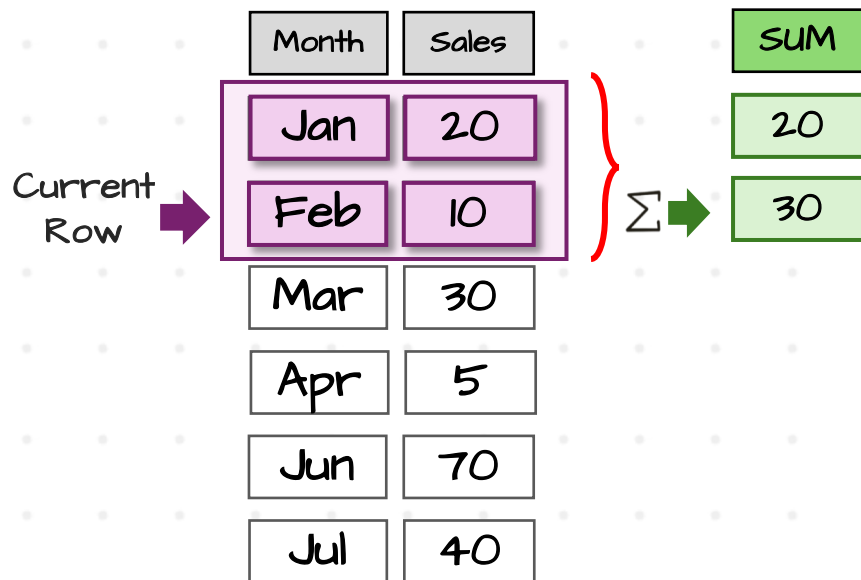
Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(  
ORDER BY Month)
```

Default Frame

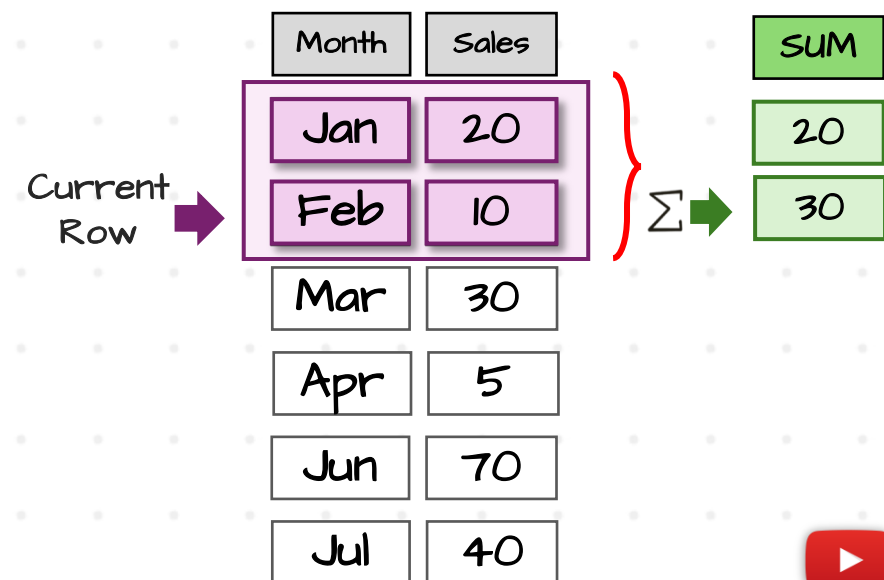
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(  
ORDER BY Month ROWS 2 PRECEDING)
```



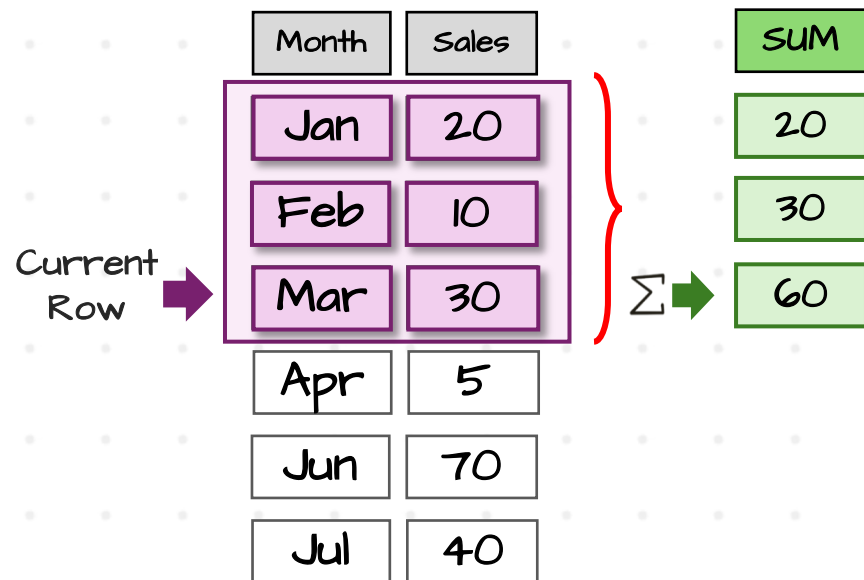
Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(  
ORDER BY Month)
```

Default Frame

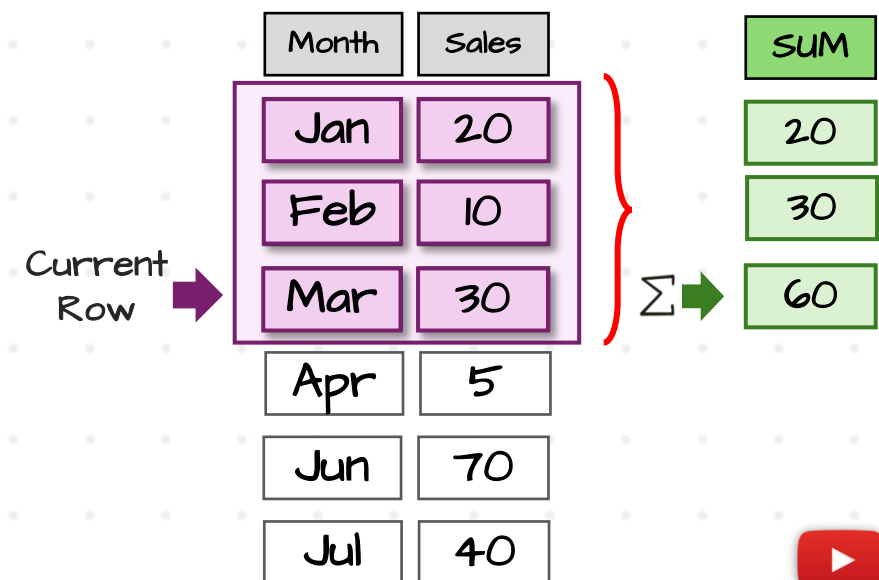
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(  
ORDER BY Month ROWS 2 PRECEDING)
```



Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(  
ORDER BY Month)
```

Default Frame

RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	65
Jun	70	
Jul	40	

Current Row →

Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(  
ORDER BY Month ROWS 2 PRECEDING)
```

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	45
Jun	70	
Jul	40	

Current Row →

Moving Window !

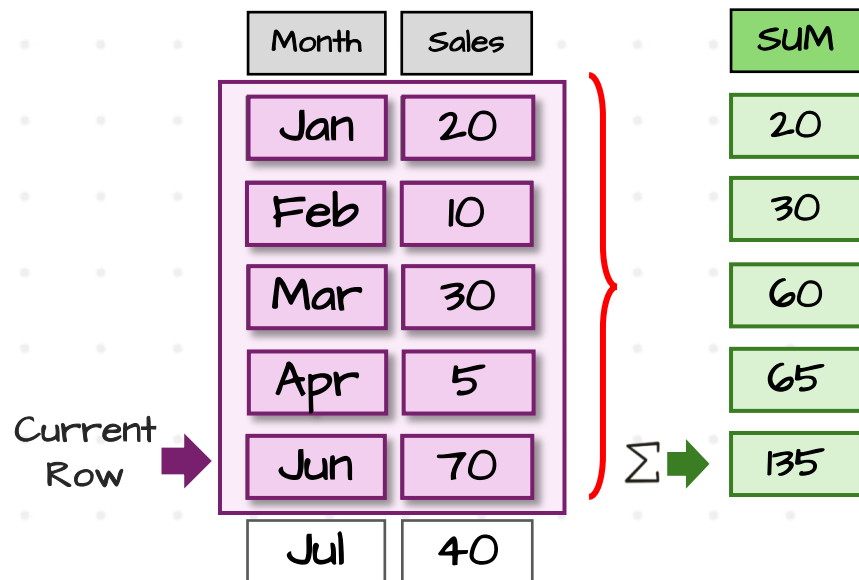
Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(
ORDER BY Month)
```

Default Frame

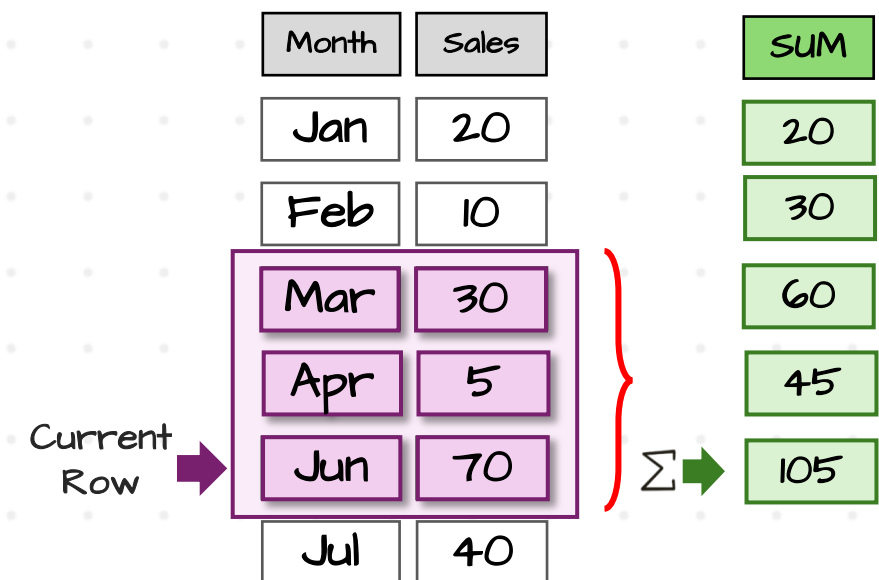
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(
ORDER BY Month ROWS 2 PRECEDING)
```



Running Total

Summarize all values from the **first row** up to the **current row**

```
SUM(Sales) OVER(
ORDER BY Month)
```

Default Frame

RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	65
Jun	70	135
Jul	40	175

Current Row →

Rolling Total

Summarize a **fixed** number of consecutive rows calculated within a moving window

```
SUM(Sales) OVER(
ORDER BY Month ROWS 2 PRECEDING)
```

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	45
Jun	70	105
Jul	40	115

Current Row →

Overall Total

```
SUM(Sales) OVER()
```

Overview of entire data

Month	Sales	SUM
Caps	20	175
Caps	10	175
Caps	30	175
Gloves	5	175
Gloves	70	175
Gloves	40	175

Total Per Groups

```
SUM(Sales) OVER(  
PARTITION BY Product)
```

Compare Categories

Month	Sales	SUM
Caps	20	60
Caps	10	60
Caps	30	60
Gloves	5	105
Gloves	70	105
Gloves	40	105

Running Total

```
SUM(Sales) OVER(  
ORDER BY MONTH)
```

progress over time

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	65
Jun	70	135
Jul	40	175

Rolling Total

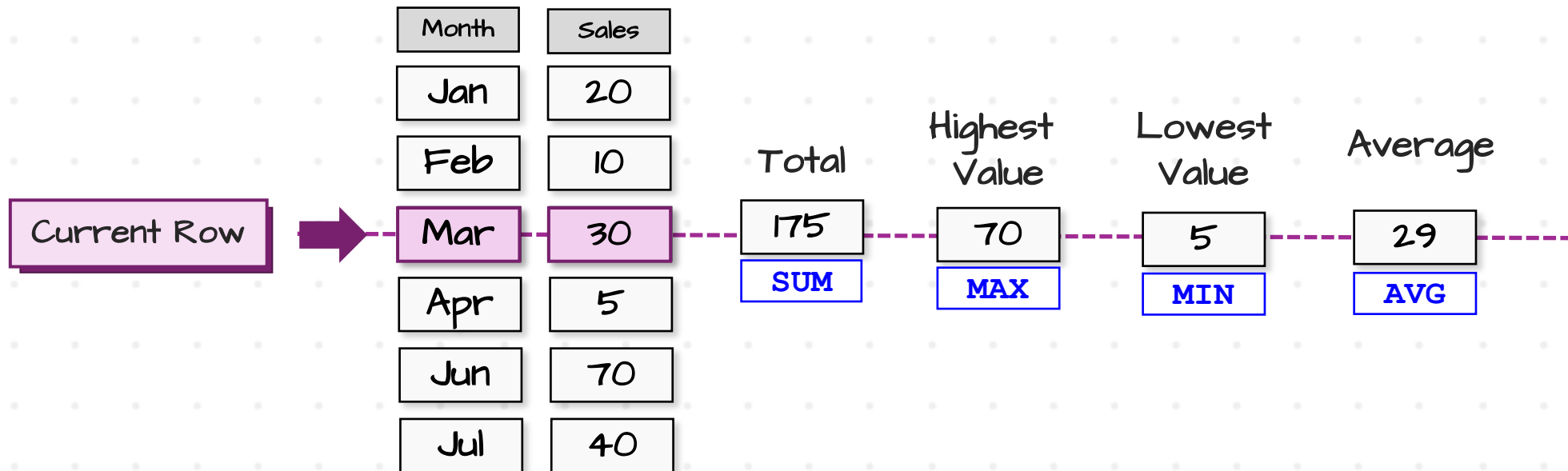
```
SUM(Sales) OVER(  
ORDER BY MONTH  
ROWS 2 PRECEDING)
```

progress over time in
specific fixed window

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	45
Jun	70	105
Jul	40	105

Comparison Use Cases

Compare the **current value** and aggregated value of **window functions**



WINDOW AGGREGATE FUNCTIONS

Aggregate set of values and return a single aggregated value

Rules

- Expressions
 - Numbers (All Functions)
 - Any Data Type - `COUNT()`
- All Clauses are Optional

Use Cases

- Overall Analysis
- Total Per Groups Analysis
- Part-to-Whole Analysis
- Comparison Analysis
 - Average
 - Extreme: Highest/Lowest
- Identify Duplicates
- Outlier Detection
- Running Total
- Rolling Total
- Moving Average



WINDOW RANKING FUNCTIONS

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Window Rank Functions



$f(x)$ Window Functions

Aggregate

- SUM ()
- AVG ()
- COUNT ()
- MAX ()
- MIN ()

Perform calculations on a set of rows and return a single aggregated value for each row

Rank


- ROW_NUMBER ()
- RANK ()
- DENSE_RANK ()
- NTILE ()
- CUME_DIST ()
- PERCENT_RANK ()

Assign a **rank** to each row in a window

Value

- LAG ()
- LEAD ()
- FIRST_VALUE ()
- LAST_VALUE ()

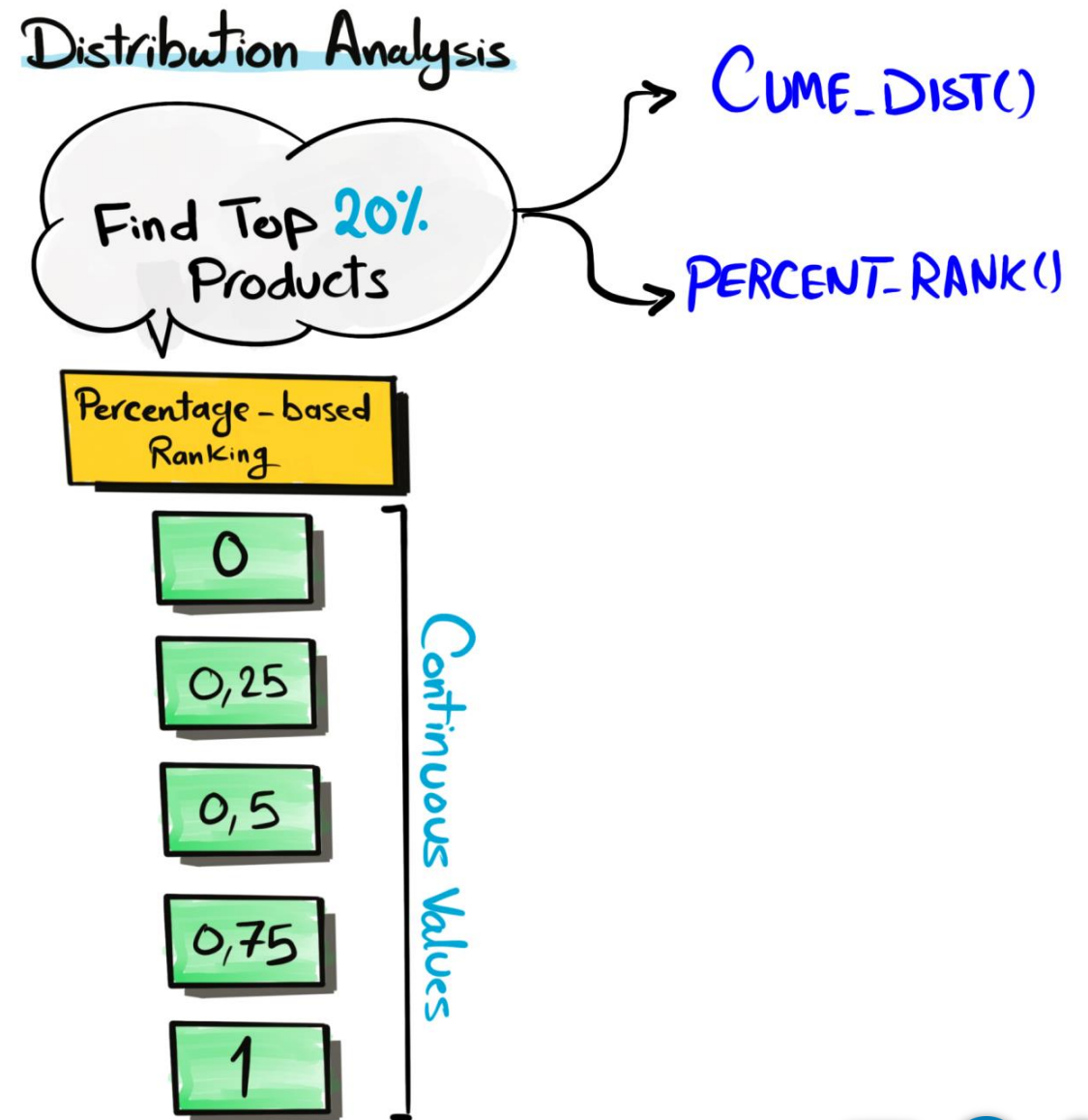
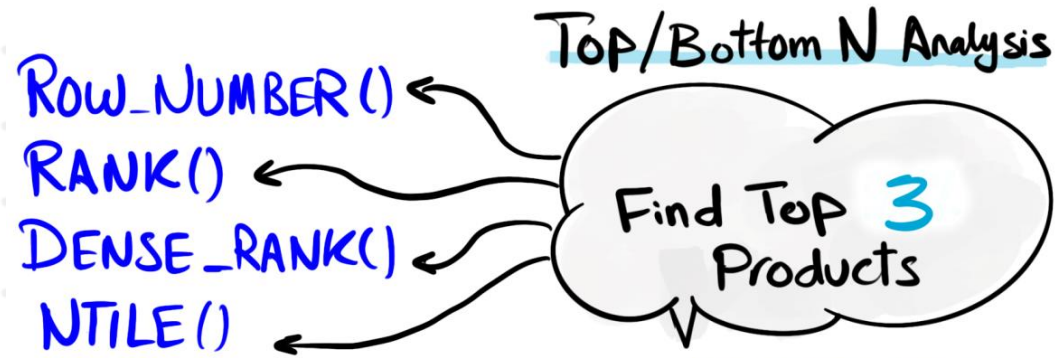
Return a specific value in a window to be compared with the value of current row



Month	Sales	Rank
Jun	70	1
Jul	40	2
Mar	30	3
Jan	20	4
Feb	10	5
Apr	5	6

Assign a Rank (Number)
for each Row





Ranking Function

RANK() **OVER** (**PARTITION BY** ProductID **ORDER BY** Sales)

Expression
must be **empty**

Partition By
Is **Optional**

Order By
Is **required**

Ranking Function

		Expression	Partition Clause	Order Clause	Frame Clause
Rank Functions	ROW_NUMBER ()	Empty	Optional	Required	Not allowed
	RANK ()				
	DENSE_RANK ()				
	CUME_DIST ()				
	PERCENT_RANK ()				
	NTILE (n)	Number			

Ranking Function

Rank Functions

ROW_NUMBER()

Assign a unique number to each in a window

ROW_NUMBER() OVER (ORDER BY Sales)

RANK()

Assign a rank to each row in a window, with gaps

RANK() OVER (ORDER BY Sales)

DENSE_RANK()

Assign a rank to each row in a window, without gaps

DENSE_RANK() OVER (ORDER BY Sales)

CUME_DIST()

calculates the cumulative distribution of a value within a set of values

CUME_DIST() OVER (ORDER BY Sales)

PERCENT_RANK()

Returns the percentile ranking number of a row.

PERCENT_RANK() OVER (ORDER BY Sales)

NTILE(n)

Divides the rows into a specified number of approximately equal groups

NTILE(2) OVER (ORDER BY Sales)

ROW_NUMBER

Assign a **unique sequential integer** to each row with in a window

```
ROW_NUMBER() OVER (ORDER BY Sales DESC)
```

Sales	Rank
100	1
80	2
80	3
50	4
20	5

We have a **tie** here!

Row_Number() assigns a **unique rank** to each of row

RANK

Assign a **rank** to each row with in a window

```
RANK() OVER(ORDER BY Sales DESC)
```

We have a **tie** here!

Sales	Rank
100	1
80	2
80	2
50	4
20	5

RANK () assigns a **same rank** for both of them

RANK () leaves a **GAP** in Ranking after a tie

DENSE_RANK

Assign a **rank** to each row with in a window, but **does not leave gaps** in the ranking

```
DENSE_RANK() OVER (ORDER BY Sales DESC)
```

Sales	Rank
100	1
80	2
80	2
50	3
20	5

We have a **tie** here!

DENSE_RANK () assigns a **same rank** for both of them

DENSE_RANK () doesn't leaves a **GAP in Ranking** after a tie

ROW_NUMBER()

`ROW_NUMBER() OVER(ORDER BY Sales DESC)`

Sales	Rank
100	1
80	2
80	3
50	4
20	5

Unique Rank

Does NOT handle Ties

No Gaps in Ranks

RANK()

`RANK() OVER(ORDER BY Sales DESC)`

Sales	Rank
100	1
80	2
80	2
50	4
20	5

Shared Rank

Handles Ties

Gaps in Ranks

DENSE_RANK()

`DENSE_RANK() OVER(ORDER BY Sales DESC)`

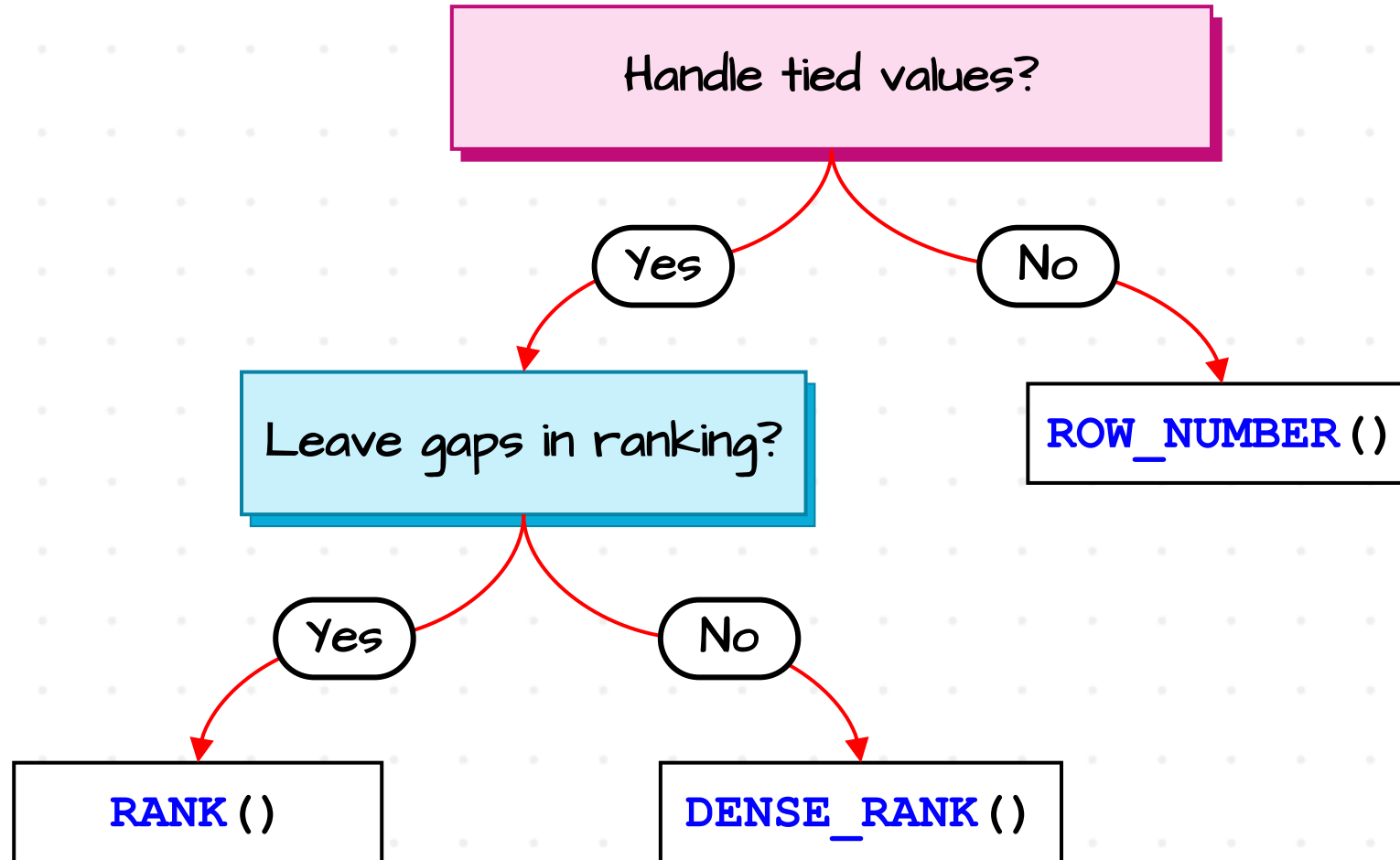
Sales	Rank
100	1
80	2
80	2
50	3
20	4

Shared Rank

Handles Ties

No Gaps in Ranks

Which One To Use?



NTILE ()

Divides the rows into a specified number of
approximately equal groups (Buckets)

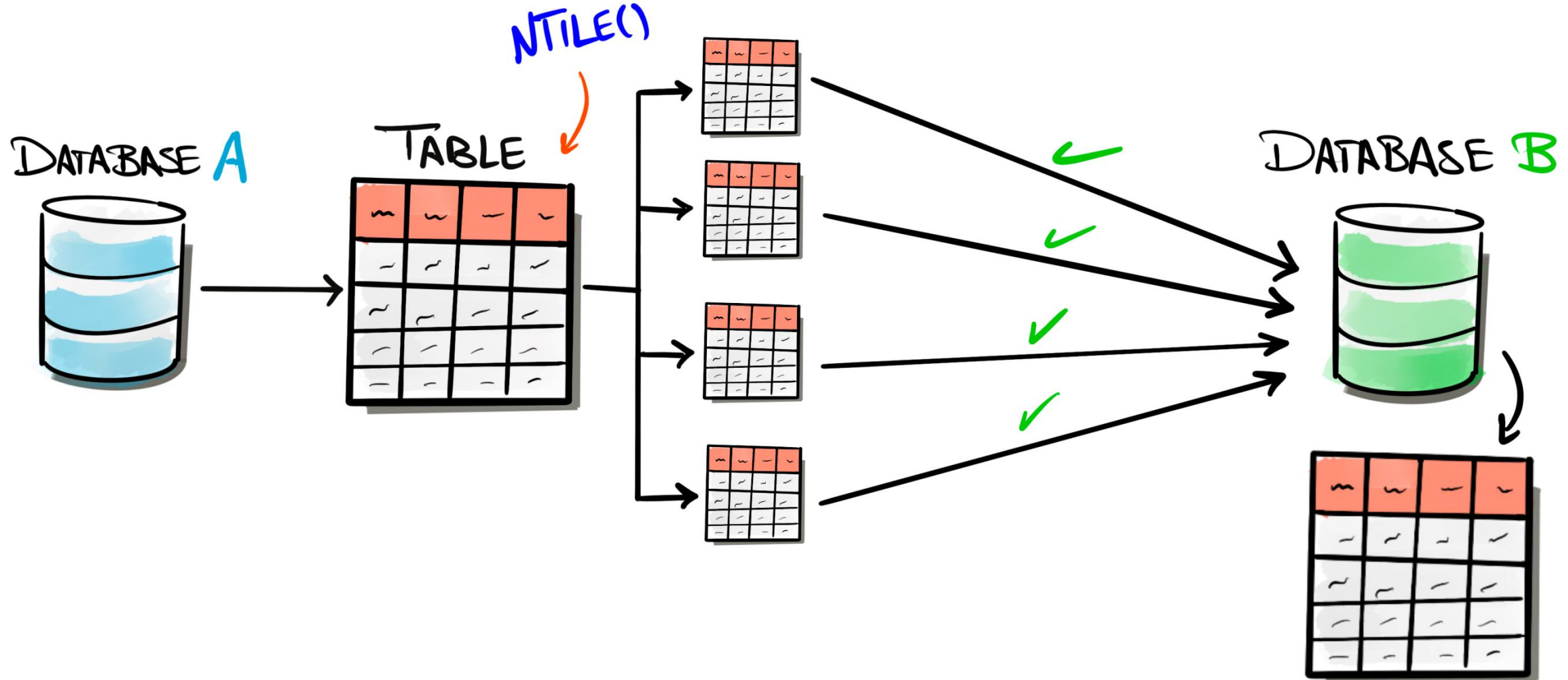
DATA ANALYST

Data Segmentation

DATA ENGINEER

**Equalizing
load processing**

NTILE use Case



NTILE Use Case

`NTILE (2) OVER (ORDER BY Sales DESC)`

Sales	NTILE
100	1
80	1
80	1
50	2
30	2

Bucket Size = $\frac{\text{Number of Rows}}{\text{Number of Buckets}}$

$$2 = \frac{5}{2}$$

NTILE

Divides the rows into a specified number of approximately **equal groups** (buckets)

```
NTILE (2) OVER (ORDER BY Sales DESC)
```

Number of Buckets

	Sales	NTILE (2)
Bucket (1)	100	1
	80	1
Bucket (2)	80	1
	50	2

$$\text{Bucket Size} = \frac{\text{Number of Rows}}{\text{Number of Buckets}}$$

(Nr of Rows in each Bucket)



$$\text{Bucket Size} = \frac{4}{2} = 2$$

NTILE

Divides the rows into a specified number of approximately **equal groups** (buckets)

```
NTILE (2) OVER (ORDER BY Sales DESC)
```

Number of Buckets 

	Sales	NTILE (2)
Bucket (1) 	100	1
	80	1
	80	1
Bucket (2) 	50	2
	20	2

$$\text{Bucket Size} = \frac{\text{Number of Rows}}{\text{Number of Buckets}}$$

(Nr of Rows in each Bucket)

$$\text{Bucket Size} = \frac{5}{2} = 2.5$$




Larger groups come **first** then smaller groups

NTILE

Divides the rows into a specified number of approximately **equal groups** (buckets)

```
NTILE (3) OVER (ORDER BY Sales DESC)
```

Number of Buckets 

	Sales	NTILE (3)
Bucket (1) 	100	1
	80	1
Bucket (2) 	80	1
	50	2
Bucket (3) 	20	2

$$\text{Bucket Size} = \frac{\text{Number of Rows}}{\text{Number of Buckets}}$$

(Nr of Rows in each Bucket)

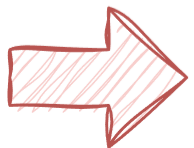
$$\text{Bucket Size} = \frac{5}{3} = 1.7$$

Larger groups come **first** then smaller groups

CUME_DIST

`CUME_DIST() OVER (ORDER BY Sales DESC)`

Sales	DIST
100	0,2
80	0,6
80	0,6
50	0,8
30	1



$$\text{CUME_DIST} = \frac{\text{Position Nr}}{\text{Number of Rows}}$$

$$\text{CUME_DIST} = \frac{5}{5}$$

CUME_DIST ()

Cumulative Distribution calculates the **distribution of data** points within a window

$$\frac{\text{Position Nr}}{\text{Number of Rows}}$$

PERCENT_RANK ()

Calculates the **relative position** of each row

$$\frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

`CUME_DIST() OVER (ORDER BY Sales DESC)`

`PERCENT_RANK() OVER (ORDER BY Sales DESC)`

Sales	DIST	Per
100	0,2	0
80	0,6	0,25
80	0,6	0,25
50	0,8	0,75
30	1	1

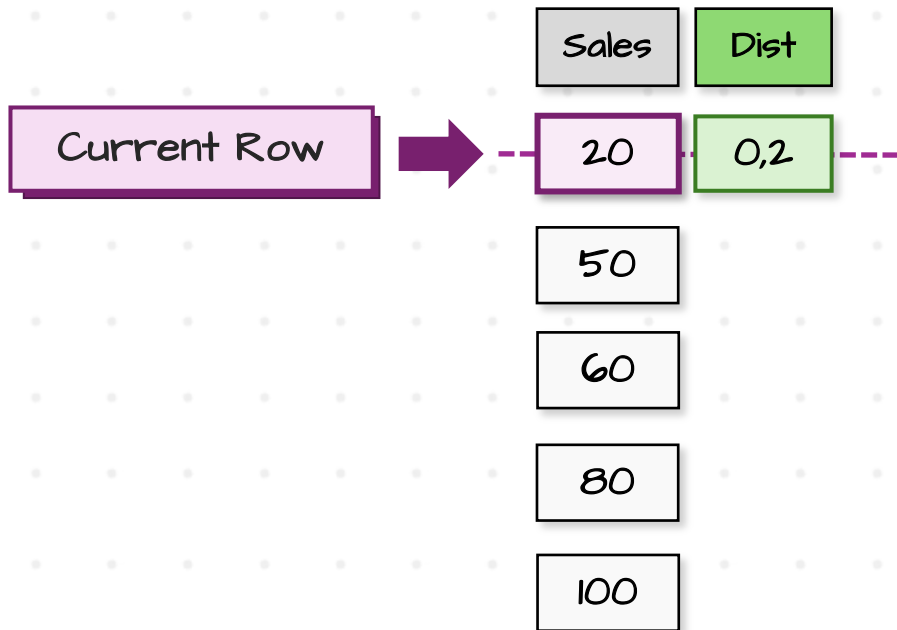
$$\text{CUME_DIST} = \frac{\text{Position Nr}}{\text{Number of Rows}}$$

$$\text{Percent_Rank} = \frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

CUME_DIST

Calculates the **relative position** of a specified value in a group of values.

`CUME_DIST() OVER (ORDER BY Sales)`



$$CUME_DIST(x) = \frac{\text{Number of Rows less than or equal to } x}{\text{Total Number of Rows}}$$

$$CUME_DIST(20) = \frac{1}{5} = 0,2$$

CUME_DIST

Calculates the **relative position** of a specified value in a group of values.

`CUME_DIST() OVER (ORDER BY Sales)`

Sales	Dist
20	0,2
50	0,4
60	0,6
80	
100	

Current Row



$$CUME_DIST(x) = \frac{\text{Number of Rows less than or equal to } x}{\text{Total Number of Rows}}$$

$$CUME_DIST(60) = \frac{3}{5} = 0,6$$

CUME_DIST

Calculates the **relative position** of a specified value in a group of values.

`CUME_DIST() OVER (ORDER BY Sales)`

Sales	Dist
20	0,2
50	0,4
60	0,6
80	0,8
100	1

Current Row



100

1

$$\text{CUME_DIST}(x) = \frac{\text{Number of Rows less than or equal to } x}{\text{Total Number of Rows}}$$

$$\text{CUME_DIST}(100) = \frac{5}{5} = 1$$

It returns values **greater than 0** and **less and equal to 1**

PERCENT_RANK

Returns the **percentile ranking** number of a row

```
PERCENT_RANK() OVER (ORDER BY Sales)
```

Current Row	Sales	Rank	Dist
	20	1	0
	50	2	
	60	3	
	80	4	
	100	5	

$$\text{PERCENT_RANK}(x) = \frac{\text{Rank of } x-1}{\text{Total Number of Rows}-1}$$

$$\text{PERCENT_RANK}(20) = \frac{0}{4} = 0$$

PERCENT_RANK

Returns the **percentile ranking** number of a row

```
PERCENT_RANK() OVER (ORDER BY Sales)
```

Sales	Rank	Dist
20	1	0
50	2	0,25
60	3	0,5
80	4	
100	5	

Current Row



$$\text{PERCENT_RANK}(x) = \frac{\text{Rank of } x-1}{\text{Total Number of Rows} - 1}$$

$$\text{PERCENT_RANK}(60) = \frac{2}{4} = 0.5$$

PERCENT_RANK

Returns the **percentile ranking** number of a row

```
PERCENT_RANK() OVER (ORDER BY Sales)
```

Sales	Rank	Dist
20	1	0
50	2	0,25
60	3	0,5
80	4	0,75
100	5	1

Current Row



$$\text{PERCENT_RANK}(x) = \frac{\text{Rank of } x-1}{\text{Total Number of Rows}-1}$$

$$\text{PERCENT_RANK}(100) = \frac{4}{4} = 1$$

It returns values between 0 and 1

PERCENT_RANK

Returns the **percentile ranking** number of a row

```
PERCENT_RANK() OVER (ORDER BY Sales)
```

Sales	Rank	Dist
20	1	0
50	2	0,25
60	3	0,5
80	4	0,75
100	5	1

Current Row →

Lowest Position

Highest Position

WINDOW RANK FUNCTIONS

Assign a RANK for each row within a window



Rules

- Expression → Empty
- ORDER BY → Required
- FRAME → Not Allowed

Use Cases

- Top N Analysis
- Bottom N Analysis
- Identify & Remove Duplicates
- Assign Unique IDs & Pagination
- Data Segmentation
- Data Distribution Analysis
- Equalizing Load Processing



DATA WITH BARAA

WINDOW VALUE FUNCTIONS

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Window Value Functions



$f(x)$ Window Functions

Aggregate

- SUM ()
- AVG ()
- COUNT ()
- MAX ()
- MIN ()

Perform calculations on a set of rows and return a single aggregated value for each row

Rank

- ROW_NUMBER ()
- RANK ()
- DENSE_RANK ()
- NTILE ()
- CUME_DIST ()
- PERCENT_RANK ()

Assign a rank to each row in a window

Value

- LAG ()
- LEAD ()
- FIRST_VALUE ()
- LAST_VALUE ()

Return a **specific value** in a window to be compared with the value of **current row**

Value Functions

Value (Analytics) Functions

LEAD (*expr*,*offset*,*default*)

Returns the value from a previous row

LEAD(Sales,2,0) **OVER** (**ORDER BY** OrderDate)

LAG (*expr*,*offset*,*default*)

Returns the value from a subsequent row

LAG(Sales,2,0) **OVER** (**ORDER BY** OrderDate)

FIRST_VALUE (*expr*)

Returns the first value in a window

FIRST_VALUE(Sales) **OVER** (**ORDER BY** OrderDate)

LAST_VALUE (*expr*)

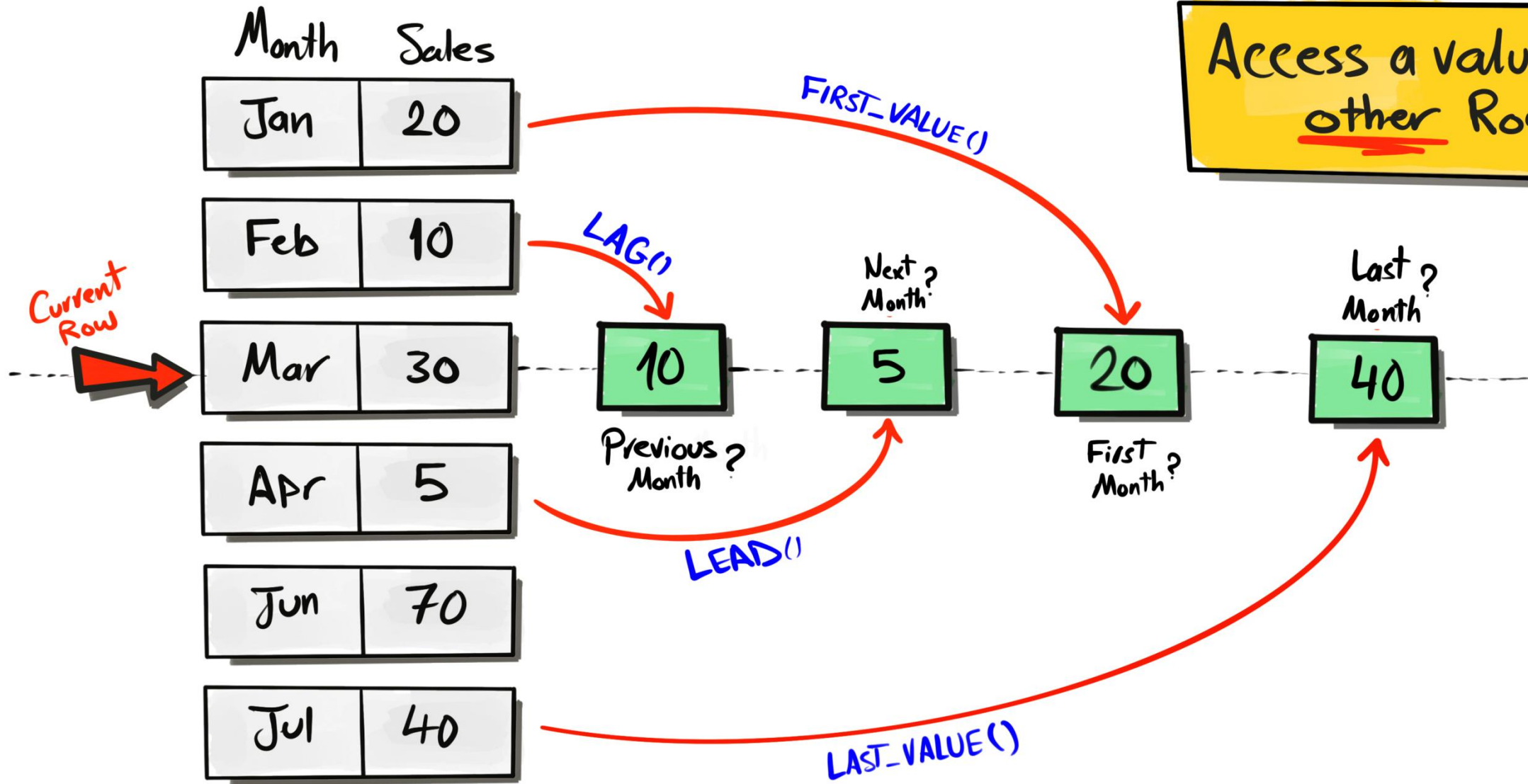
Returns the last value in a window

LAST_VALUE(Sales) **OVER** (**ORDER BY** OrderDate)

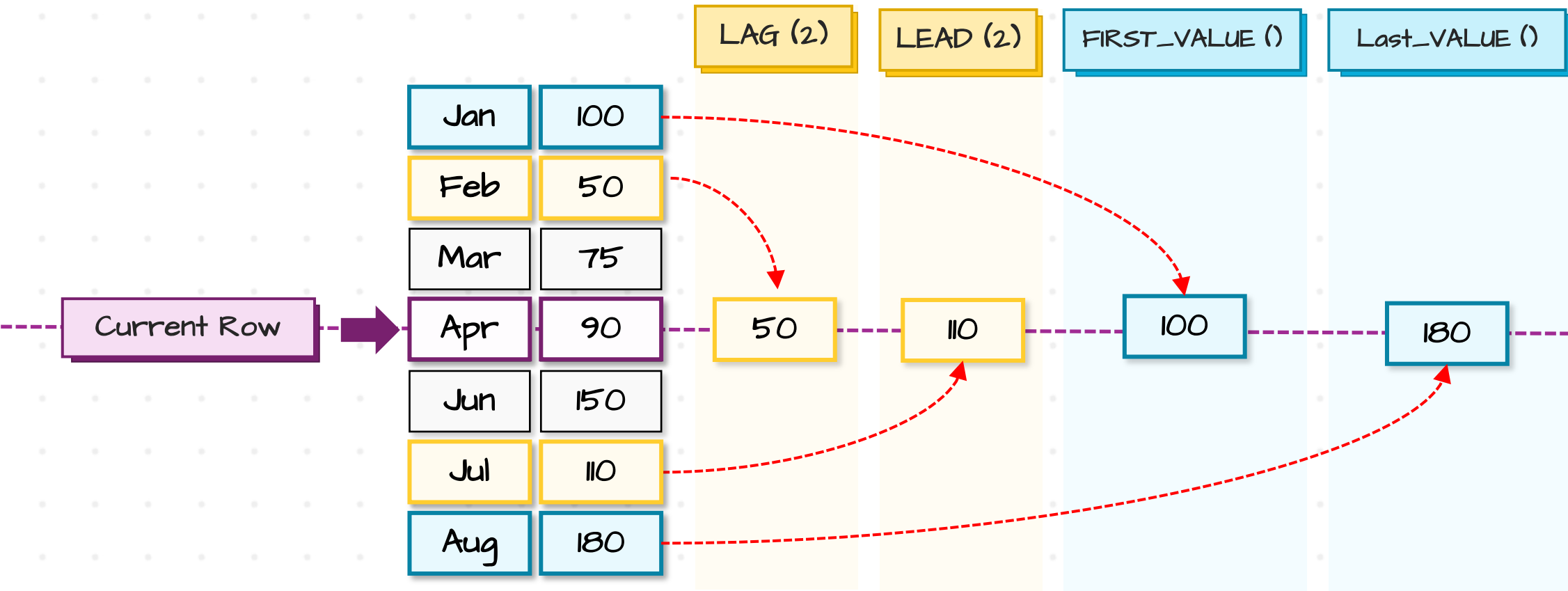
Value Functions

		Expression	Partition Clause	Order Clause	Frame Clause
Value (Analytics) Functions	LEAD (expr,offset,default)	All Data Type	Optional	Required	Not allowed
	LAG (expr,offset,default)				
	FIRST_VALUE (expr)				Optional
	LAST_VALUE (expr)				Should be used

Access a value from
other Row

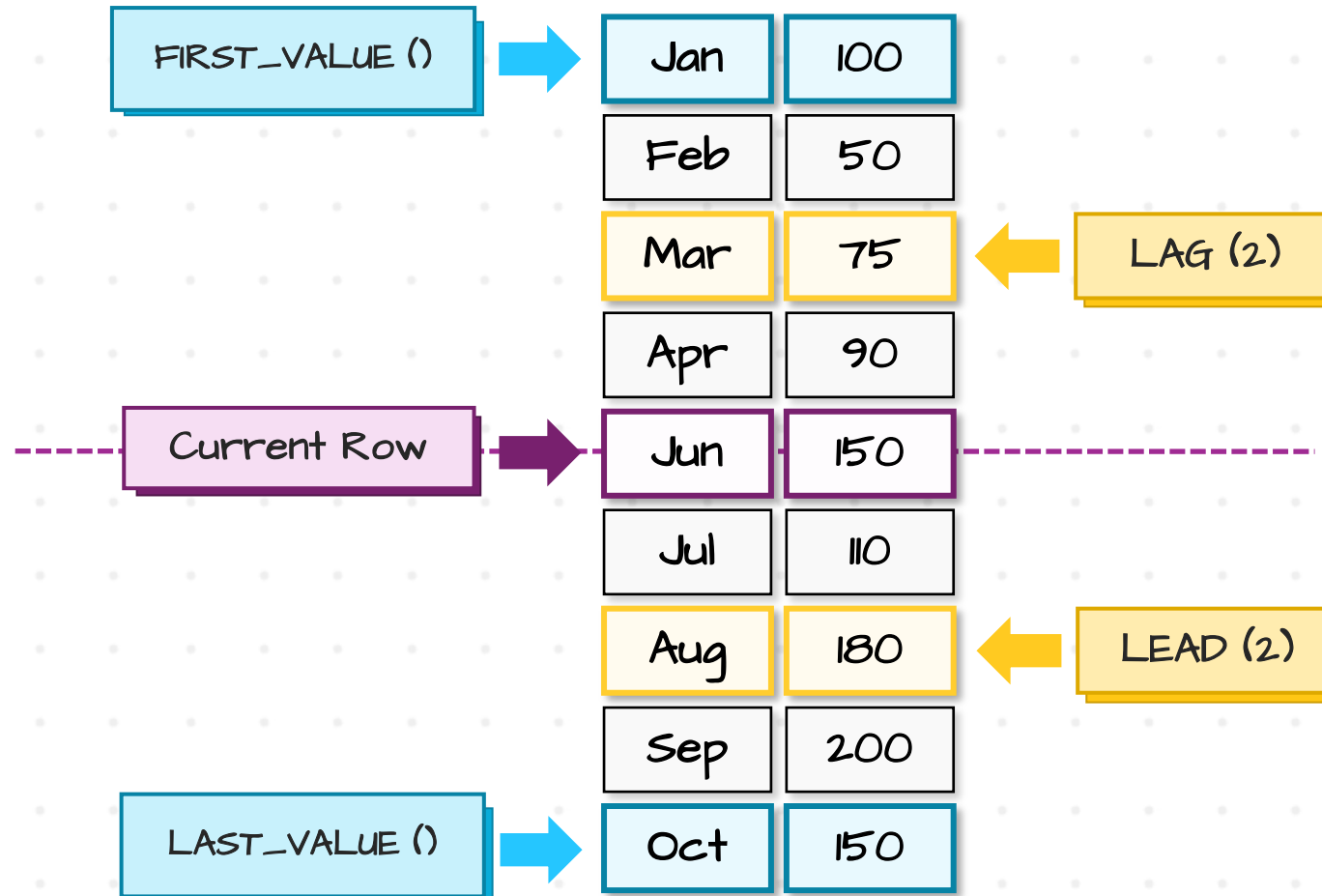


Value Functions



Value Functions

Return a **specific value** in a window to be compared with the value of current row



LEAD & LAG

Partition By
Is **Optional**

Order By
Is **Required**

LEAD (Sales , 2 , 10) **OVER** (PARTITION BY ProductID ORDER BY OrderDate)

Expression
is **required**
(Any Data Type)

Default Value (Optional)

Returns default value if next/previous row is not available!
Default = **NULL**

offset (Optional)

Number of rows forward or backward from current row
default = **1**

LEAD & LAG

`LEAD(Sales) OVER (ORDER BY Month)`

Month	Sales	LEAD
Jan	20	10
Feb	10	30
Mar	30	5
Apr	5	NULL



Find Sales of
the next month

`LAG(Sales) OVER (ORDER BY Month)`

Month	Sales	LAG
Jan	20	NULL
Feb	10	20
Mar	30	10
Apr	5	30



Find Sales of
the previous month

LEAD

Access **Next** Row

```
LEAD (Sales) OVER ( ORDER BY Month)
```

Current Row →	Month	Sales	LEAD
	Jan	20	20
	Feb	10	
	Mar	30	
	Apr	5	

LAG

Access **Previous** Row

```
LAG (Sales) OVER ( ORDER BY Month)
```

First Row has **No** Previous Row !

Current Row →	Month	Sales	LAG
	Jan	20	NULL
	Feb	10	
	Mar	30	
	Apr	5	

LEAD

Access **Next** Row

```
LEAD (Sales) OVER ( ORDER BY Month)
```

Month	Sales	LEAD
Jan	20	20
Feb	10	30
Mar	30	
Apr	5	

Current Row →

LAG

Access **Previous** Row

```
LAG (Sales) OVER ( ORDER BY Month)
```

First Row has **No** Previous Row !

Month	Sales	LAG
Jan	20	NULL
Feb	10	20
Mar	30	
Apr	5	

Current Row →

LEAD

Access **Next** Row

```
LEAD (Sales) OVER ( ORDER BY Month)
```

Month	Sales	LEAD
Jan	20	20
Feb	10	30
Mar	30	5
Apr	5	

Current Row →

LAG

Access **Previous** Row

```
LAG (Sales) OVER ( ORDER BY Month)
```

First Row has **No** Previous Row !

Month	Sales	LAG
Jan	20	NULL
Feb	10	20
Mar	30	10
Apr	5	

Current Row →

LEAD

Access **Next** Row

```
LEAD (Sales) OVER ( ORDER BY Month)
```

Month	Sales	LEAD
Jan	20	20
Feb	10	30
Mar	30	5
Apr	5	NULL

Current Row →

Last Row has **No** Next Row !

LAG

Access **Previous** Row

```
LAG (Sales) OVER ( ORDER BY Month)
```

First Row has **No** Previous Row !

Month	Sales	LAG
Jan	20	NULL
Feb	10	20
Mar	30	10
Apr	5	30

Current Row →

TIME SERIES ANALYSIS

Year-over-Year (YoY)

Analyze the overall growth or decline of the business's performance over time

Month-over-Month (MoM)

Analyze short-term trends and discover patterns in seasonality

FIRST & LAST

`FIRST_VALUE(Sales) OVER (ORDER BY Month)`

	Month	Sales	First
UNBOUNDED PRECEDING	Jan	20	20
	Feb	10	20
	Mar	30	20
Current Row	Apr	5	20

`LAST_VALUE(Sales) OVER (ORDER BY Month)`

	Month	Sales	Last
UNBOUNDED PRECEDING	Jan	20	20
	Feb	10	10
	Mar	30	30
Current Row	Apr	5	5

Default

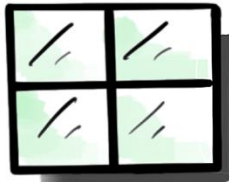
`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`

FIRST & LAST

`LAST_VALUE(Sales) OVER (ORDER BY Month
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)`

Month	Sales	Last
Jan	20	5
Feb	10	5
Mar	30	5
Apr	5	5

Diagram illustrating the window function `LAST_VALUE(Sales) OVER (ORDER BY Month ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)`. The table shows the current row (Apr) and the unbounded following rows (Mar, Feb, Jan) used for the calculation. The 'Last' column shows the result of the function, which is 5 for all rows.



SQL WINDOW USE CASES

- Top N Analysis
- Bottom N Analysis
- Identify & Remove Duplicates
- Assign Unique IDs & Pagination
- Data Segmentation
- Data Distribution Analysis
- Equalizing Load Processing
- Overall Analysis
- Total Per Groups Analysis

- Part-to-Whole Analysis
- Time Series Analysis: MoM & YoY
- Time Gaps Analysis: Customer Retention
- Comparison Analysis: Extreme ↗ Highest
↘ Lowest
- Outlier Detection
- Running Total
- Rolling Total
- Moving Average