



# AGGREGATE FUNCTIONS

Baraa Khatib Salkini  
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

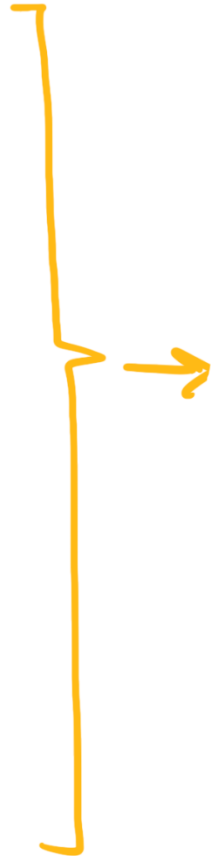
Sales

35

15

20

10



COUNT(\*) 4

SUM 80

AVG 20

MAX 35

MIN 10



# WINDOW FUNCTIONS

## BASICS

Baraa Khatib Salkini  
SQL Course | Window Functions



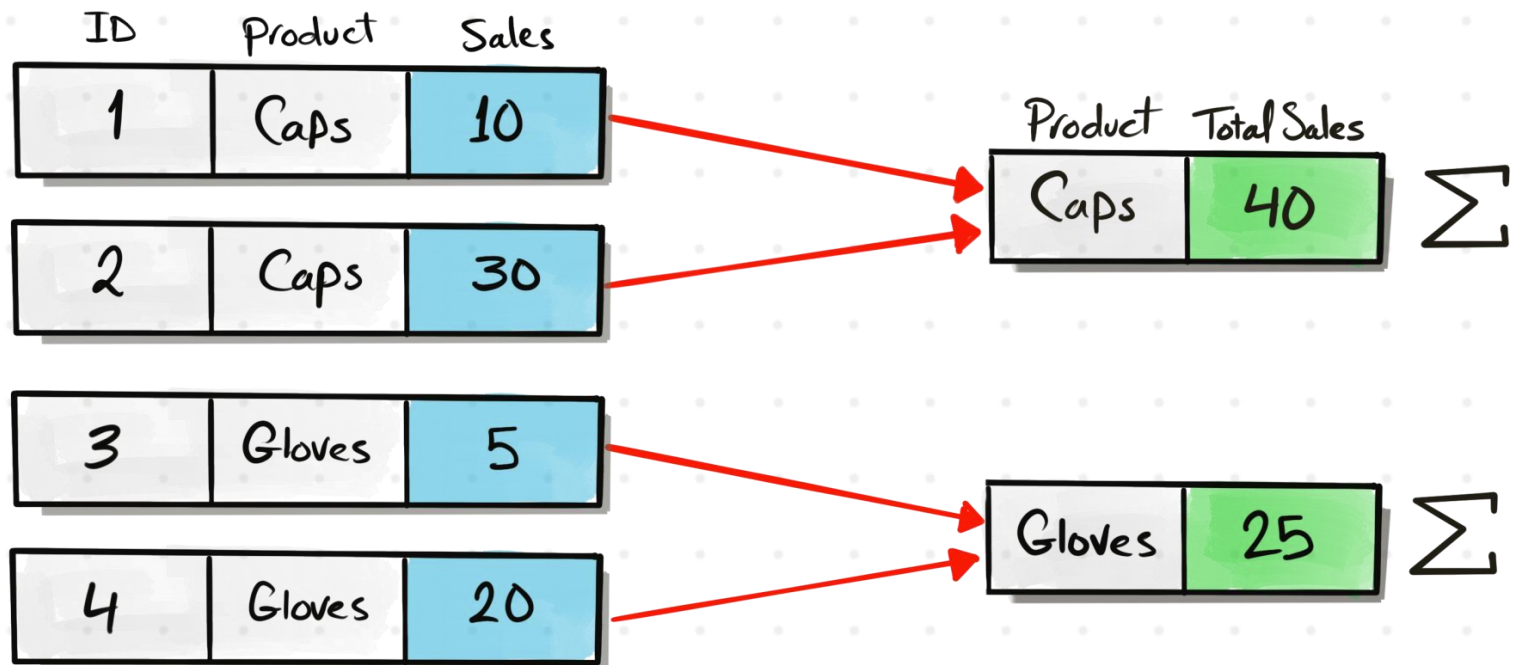


# **WINDOW FUNCTIONS**

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

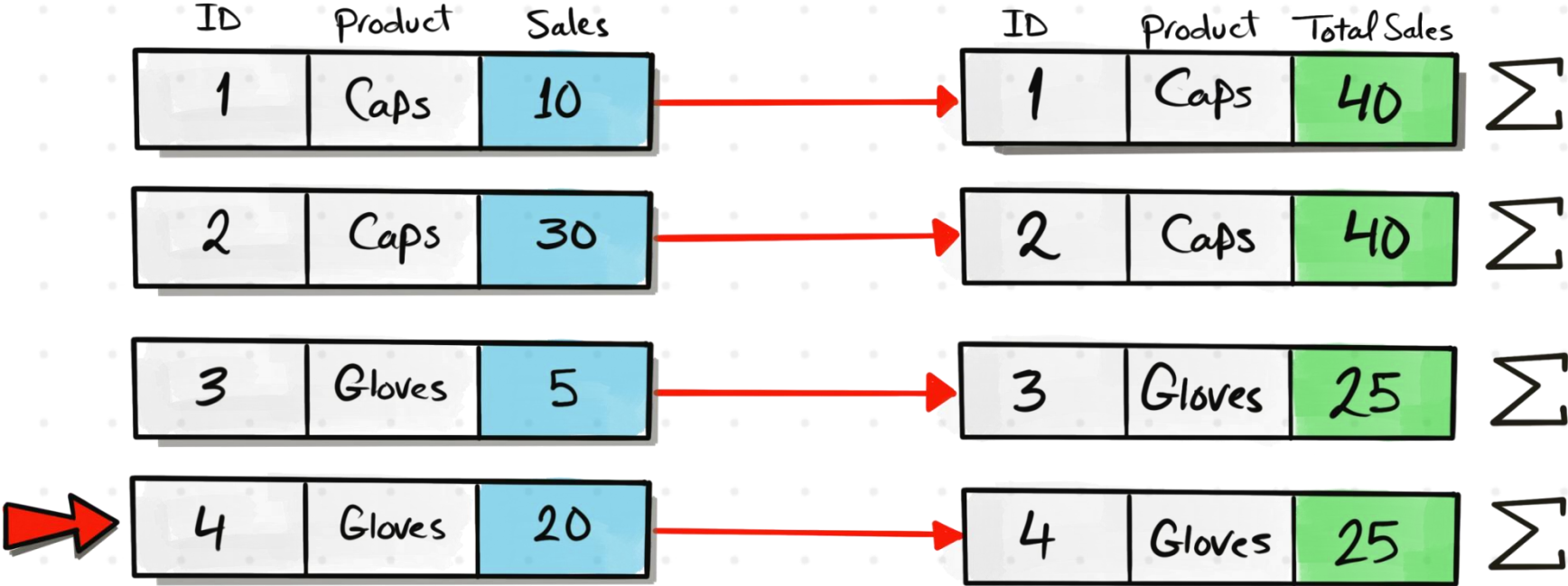
# GROUP BY

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



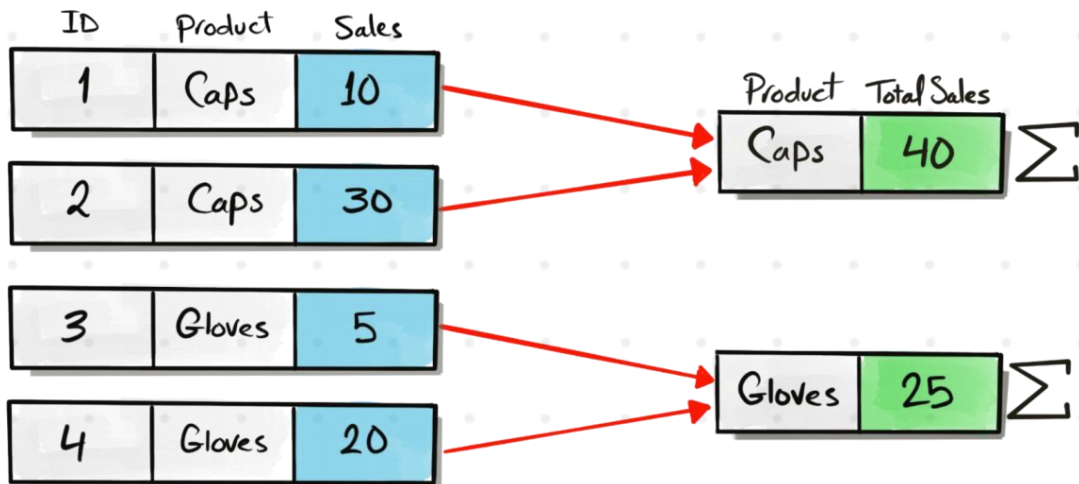
# WINDOW Functions

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



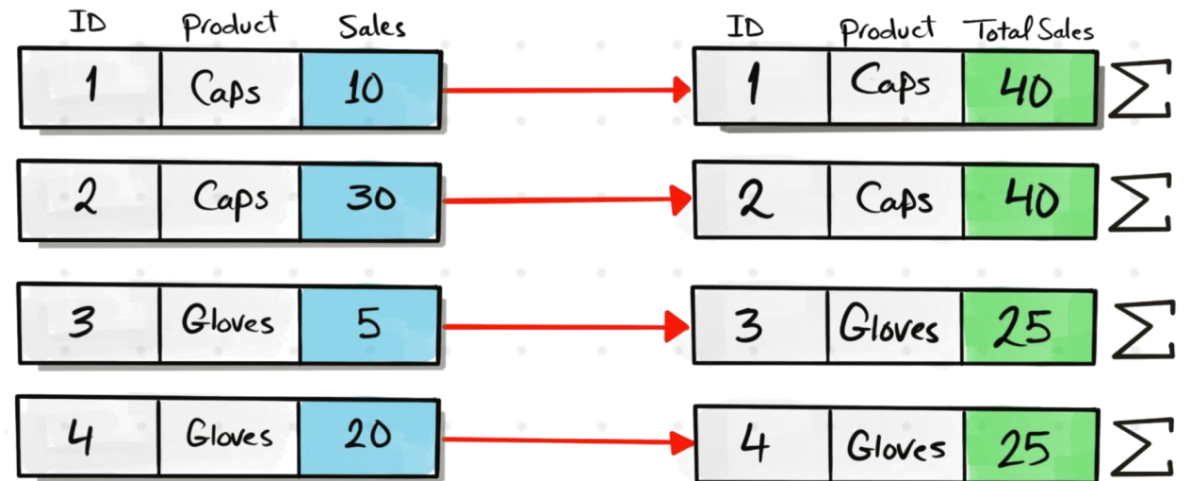
Row-Level Calculation

## GROUP BY



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

## Window Functions



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*)

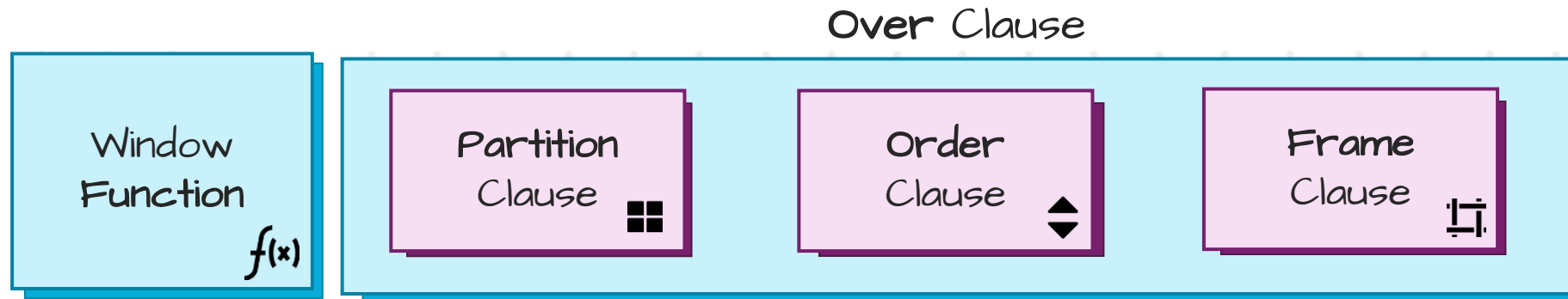
## **GROUP BY**

**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

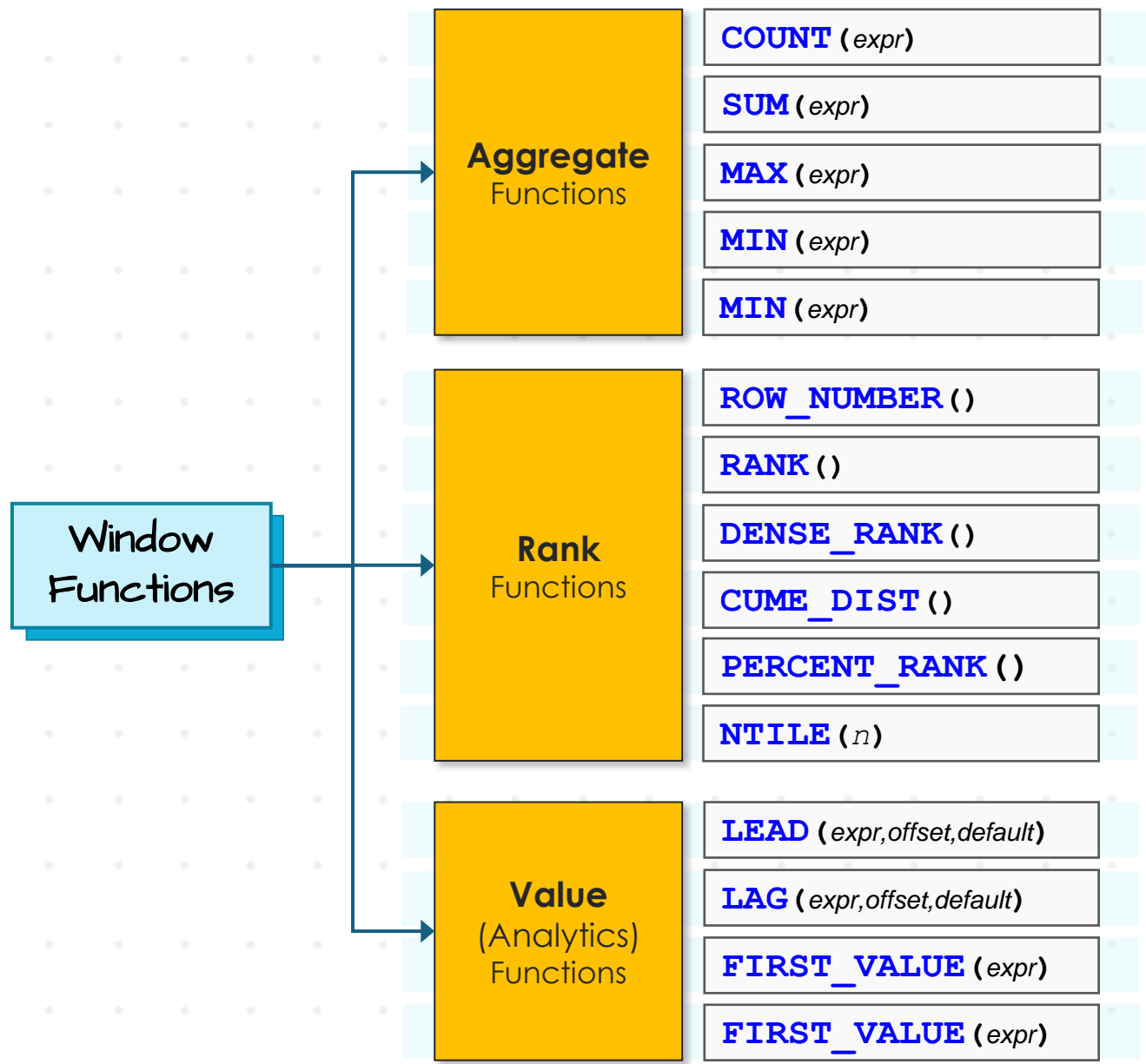
f(\*) Window  
Function

**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

$f(x)$  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

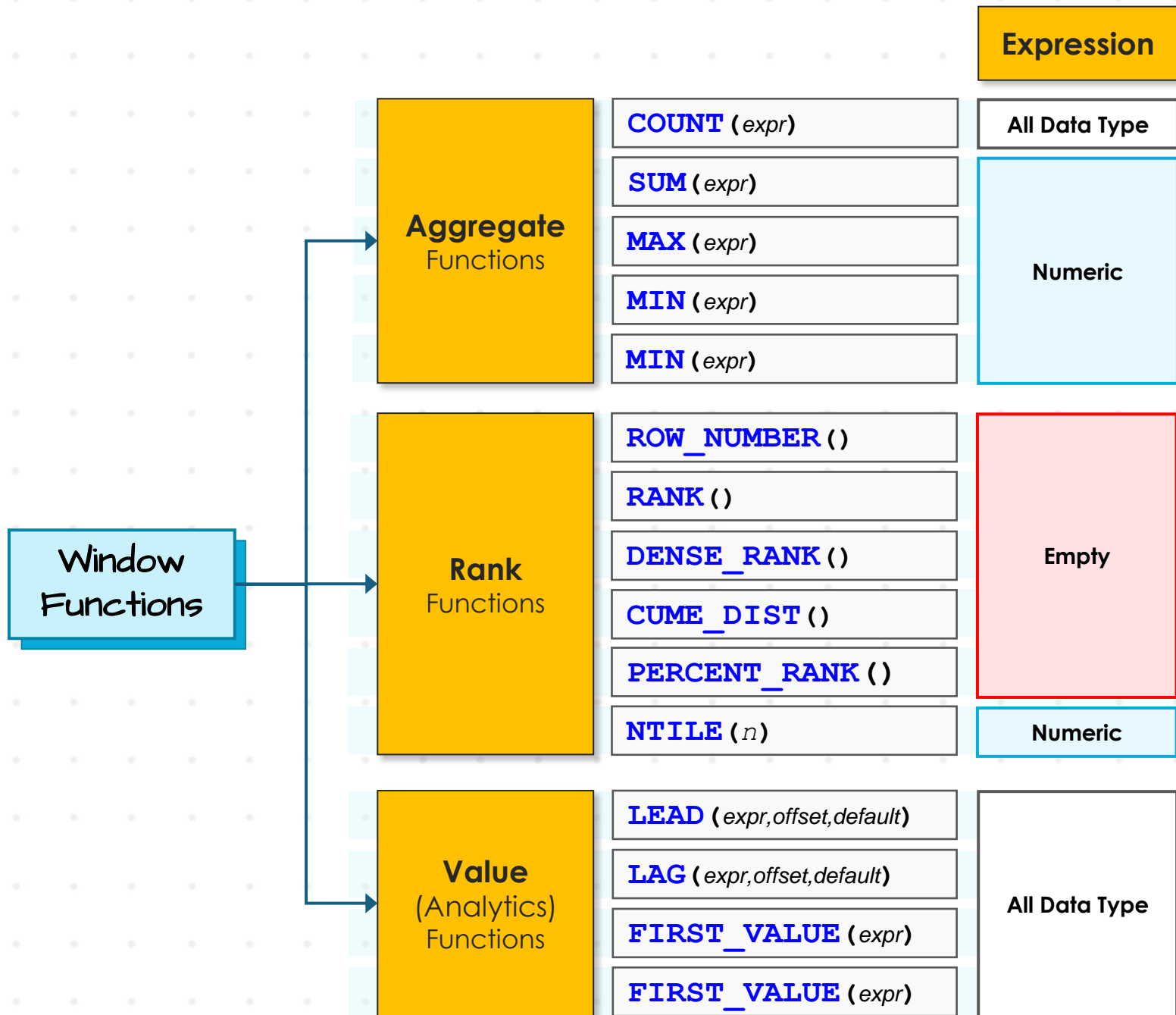
`NTEIL(2) OVER (ORDER BY OrderDate)`

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

$f(x)$  Window Function

Define the Window

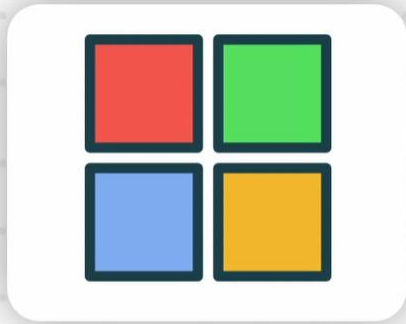
Over Clause

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

Function Expression

Partition Clause 

Divides the dataset into windows (Partitions)



## **PARTITION BY**

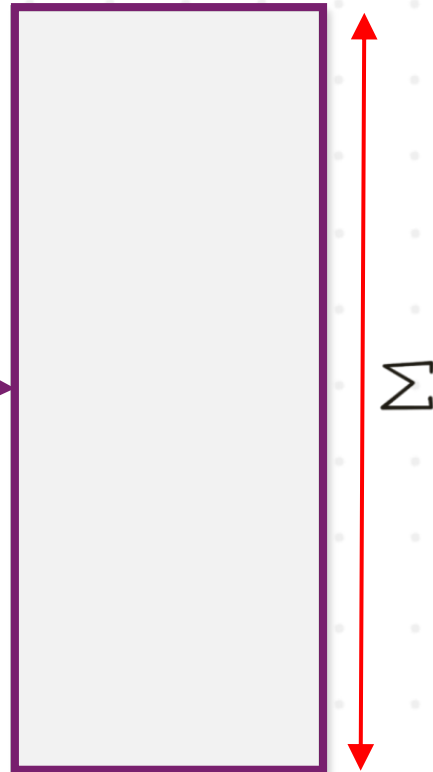
**Divides the result set into partitions (Windows)**

# Partition By

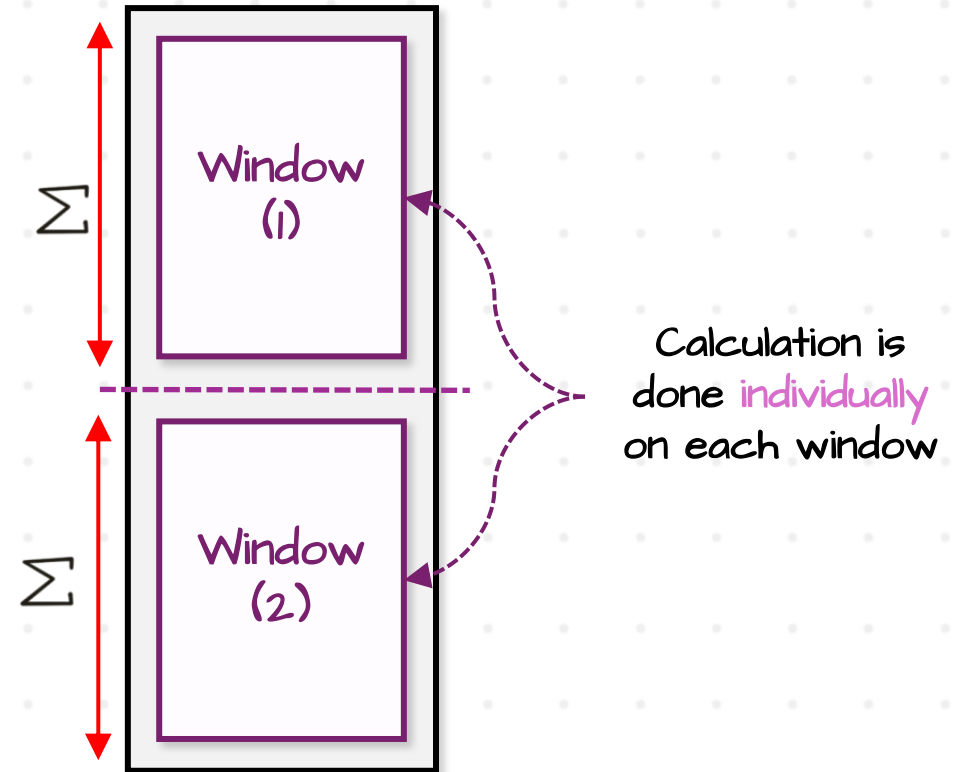
**PARTITION BY** divides the rows into groups, based on the column/s

`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

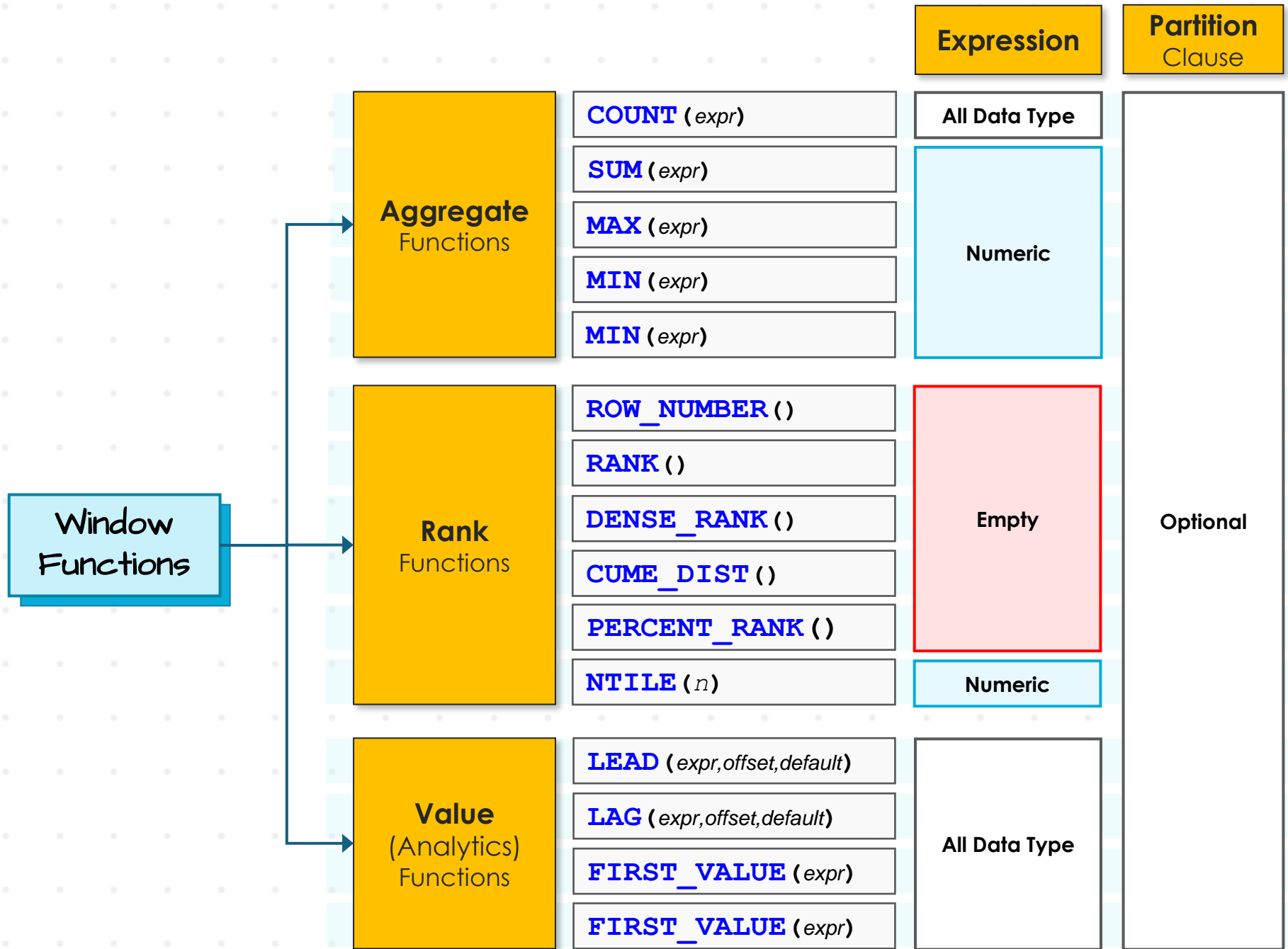
Total sales for each Product

```
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)
```



# Window Syntax

Calculation used on the Window

$f(x)$  Window Function

Define the Window

Over Clause

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

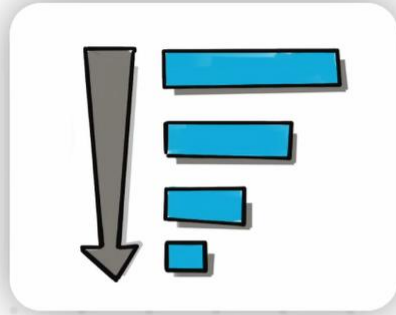
Function Expression

Partition Clause 

Order Clause 

Divides the dataset into windows (Partitions)

Sort the data in a window



# **ORDER BY**

**Sort the data within a window**

( Ascending | Descending )

**Window Functions**

**Aggregate Functions**

- COUNT (expr)
- SUM (expr)
- MAX (expr)
- MIN (expr)
- MIN (expr)

**Expression**

All Data Type  
 Numeric

**Partition Clause**

Optional

**Order Clause**

Optional

**Rank Functions**

- ROW\_NUMBER ()
- RANK ()
- DENSE\_RANK ()
- CUME\_DIST ()
- PERCENT\_RANK ()
- NTILE (n)

Empty  
 Numeric

Optional

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 

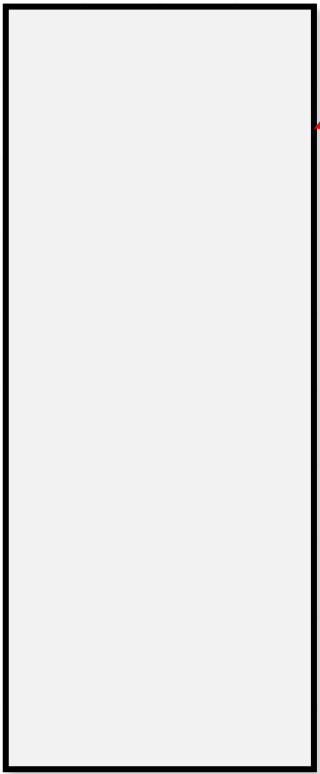
Sort the data in a window

Frame Clause 

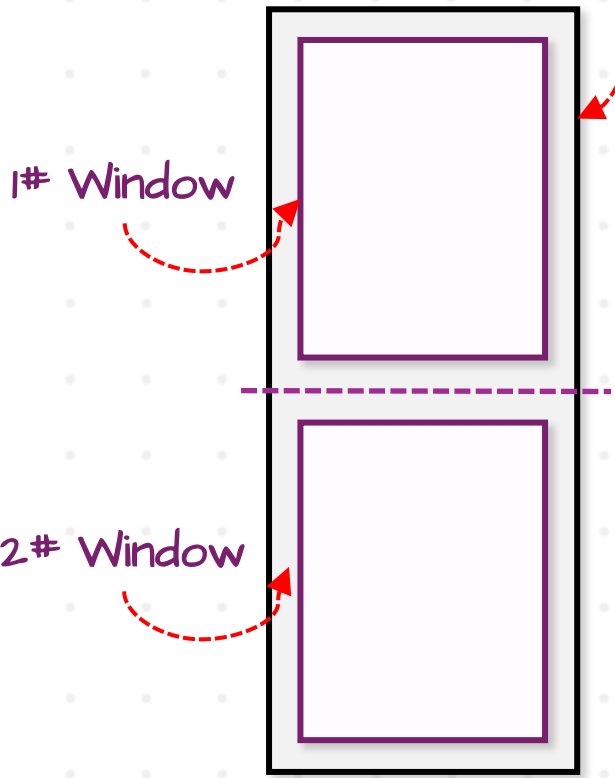
Define a subset of rows in a window

# Frame Clause

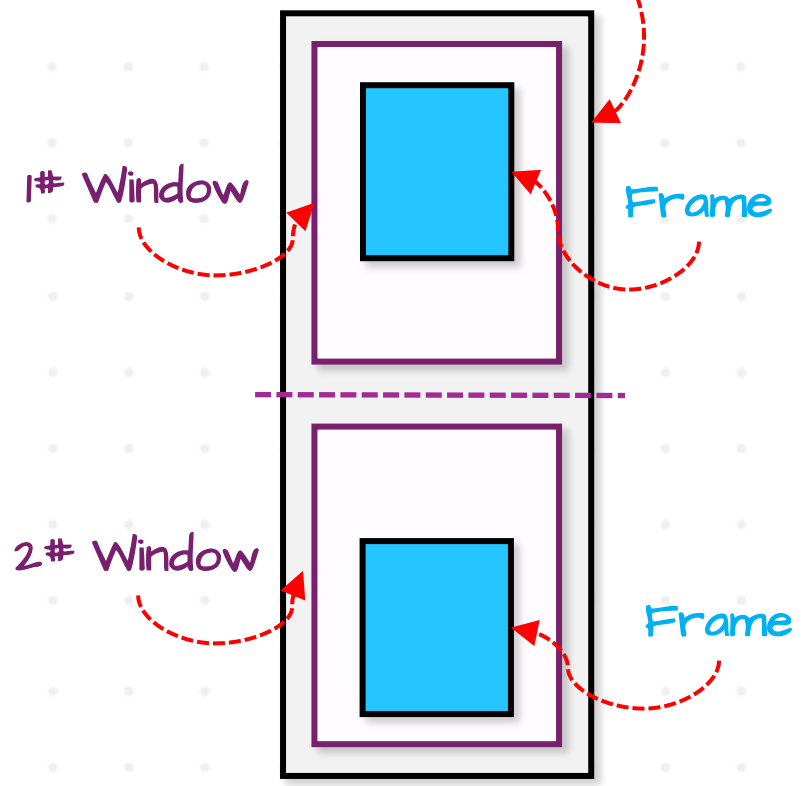
Entire Data



Entire Data



Entire Data



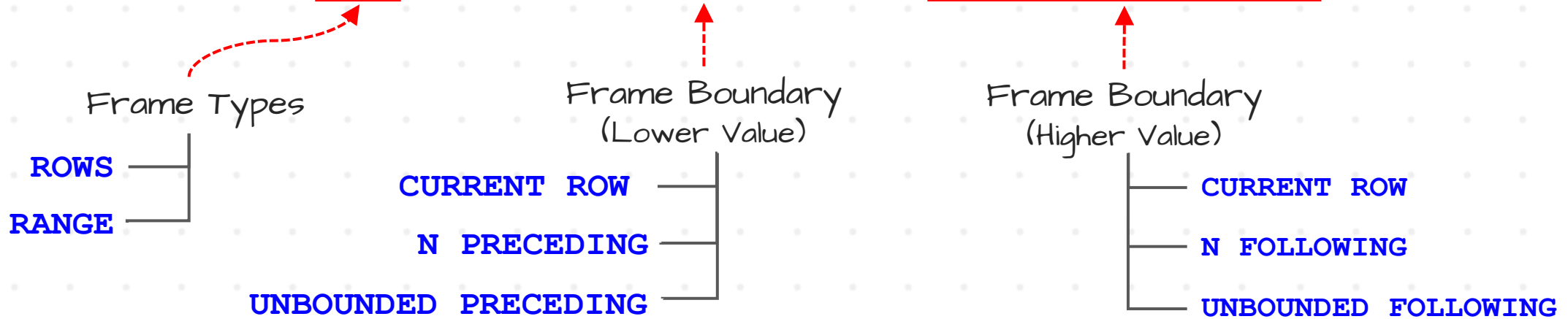
# Window Functions

|                             |                                     | Expression    | Partition Clause | Order Clause | Frame Clause   |
|-----------------------------|-------------------------------------|---------------|------------------|--------------|----------------|
| Aggregate Functions         | COUNT ( <i>expr</i> )               | All Data Type |                  | Optional     | Optional       |
|                             | SUM ( <i>expr</i> )                 | Numeric       |                  |              |                |
|                             | AVG ( <i>expr</i> )                 |               |                  |              |                |
|                             | MAX ( <i>expr</i> )                 |               |                  |              |                |
|                             | MIN ( <i>expr</i> )                 |               |                  |              |                |
| Rank Functions              | ROW_NUMBER ()                       | Empty         | Optional         | Required     | Not allowed    |
|                             | RANK ()                             |               |                  |              |                |
|                             | DENSE_RANK ()                       |               |                  |              |                |
|                             | CUME_DIST ()                        |               |                  |              |                |
|                             | PERCENT_RANK ()                     |               |                  |              |                |
|                             | NTILE ( <i>n</i> )                  | Numeric       |                  |              |                |
| Value (Analytics) Functions | LEAD ( <i>expr,offset,default</i> ) | All Data Type | Required         | Not allowed  | Optional       |
|                             | LAG ( <i>expr,offset,default</i> )  |               |                  |              |                |
|                             | FIRST_VALUE ( <i>expr</i> )         |               |                  |              | Should be used |
|                             | FIRST_VALUE ( <i>expr</i> )         |               |                  |              |                |

# Frame

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

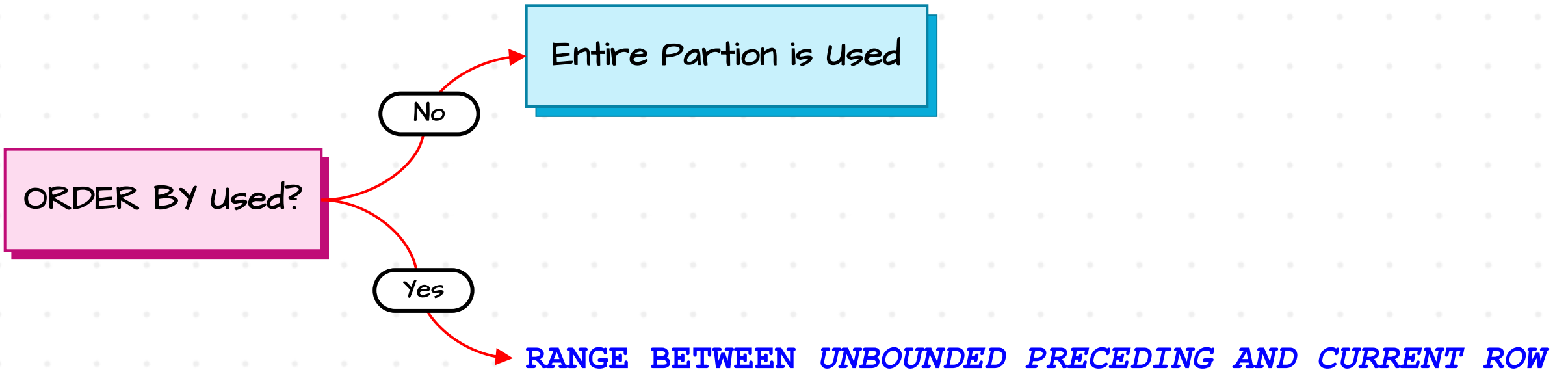
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)



## Rules

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

# Frame



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

# Window Rules

## #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**



# WINDOW AGGREGATE FUNCTIONS

Baraa Khatib Salkini  
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    |



$\Sigma$

One Aggregated Value

175



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 | <b>COUNT</b> ( <i>expr</i> ) | All Data Type    | Optional     | Optional     |
|                     | <b>SUM</b> ( <i>expr</i> )   | Numeric Values   |              |              |
|                     | <b>AVG</b> ( <i>expr</i> )   | Numeric Values   |              |              |
|                     | <b>MIN</b> ( <i>expr</i> )   | Numeric Values   |              |              |
|                     | <b>MAX</b> ( <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)`

`COUNT (Sales) 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     |

| 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)`

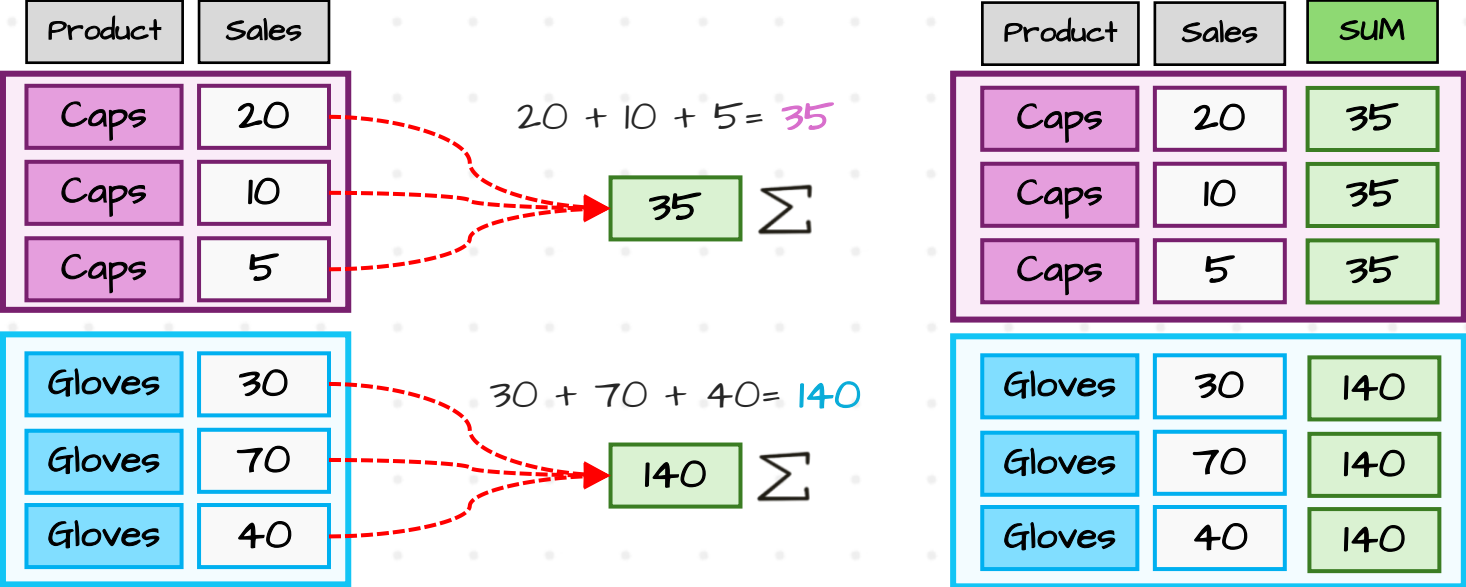
| Product | Sales | SUM |
|---------|-------|-----|
| Caps    | 20    | 35  |
| Caps    | 10    | 35  |
| Caps    | 5     | 35  |
| Gloves  | 30    | 100 |
| Gloves  | 70    | 100 |
| Gloves  | NULL  | 100 |

$$20 + 10 + 5 = 35$$

$$30 + 70 = 100$$

# 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   |
| Caps    | 10    | 5   |
| Caps    | 5     | 5   |
| Gloves  | 30    | 30  |
| Gloves  | 70    | 30  |
| Gloves  | 40    | 30  |

5 is the lowest sales for Caps

30 is the lowest sales for Gloves

# MAX

Returns the **maximum** value in a window

```
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  | 40    | 70  |

20 is the highest sales for Caps

70 is the highest sales for Gloves

# 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 |

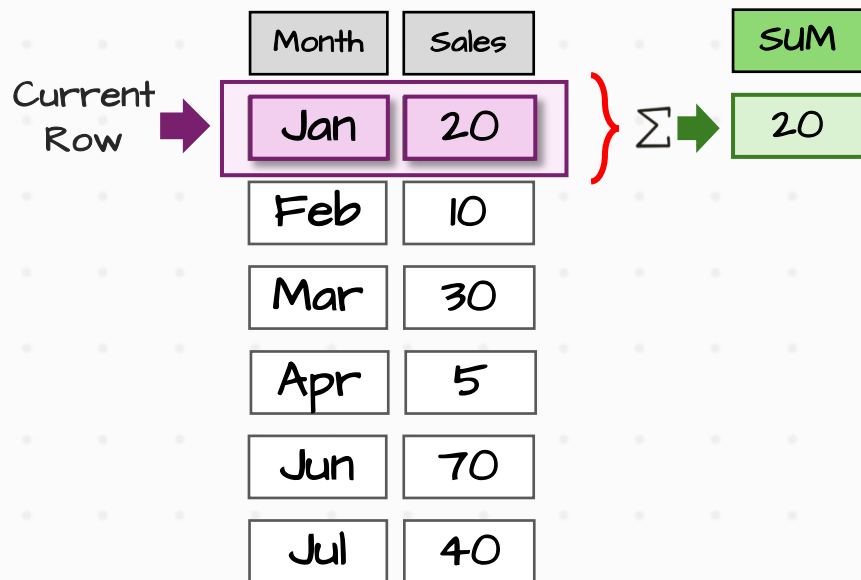
## Running Total

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

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

Default Frame

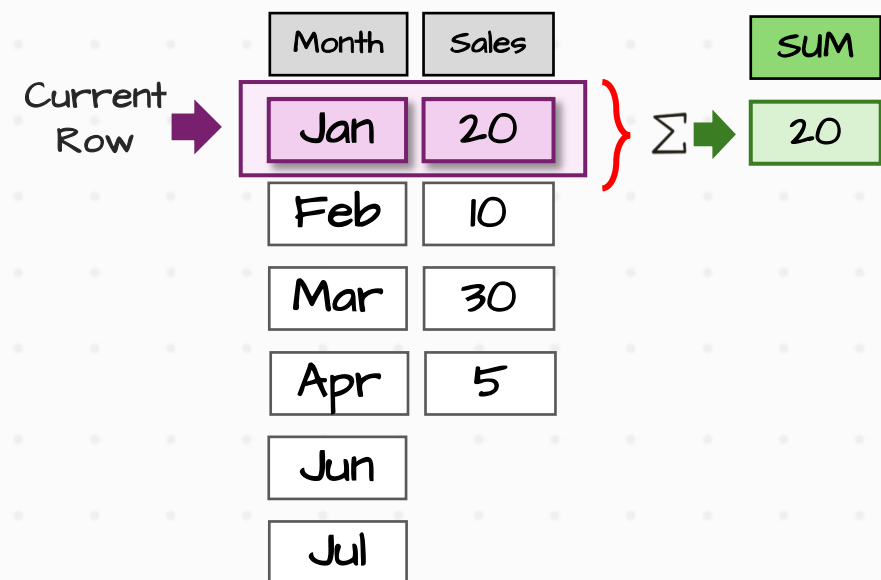
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



## Rolling Total

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

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



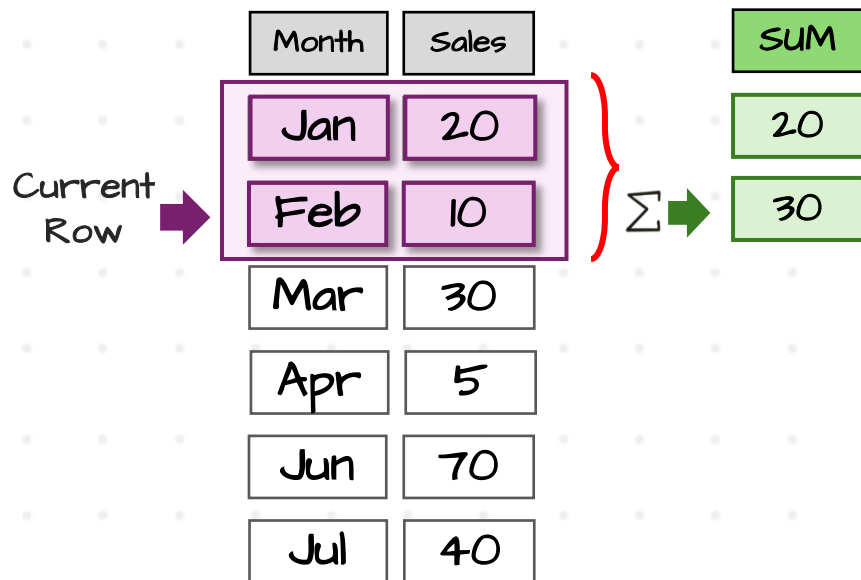
## Running Total

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

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

Default Frame

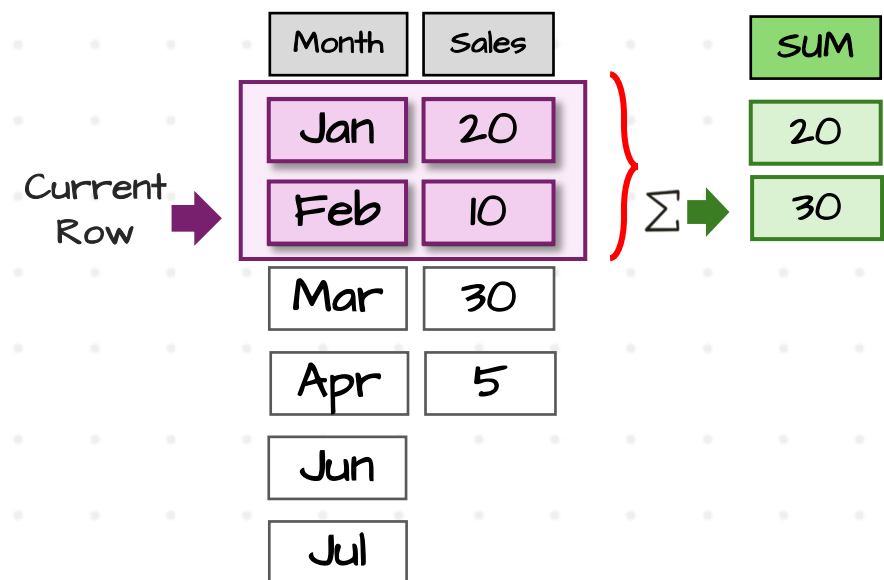
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



## Rolling Total

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

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



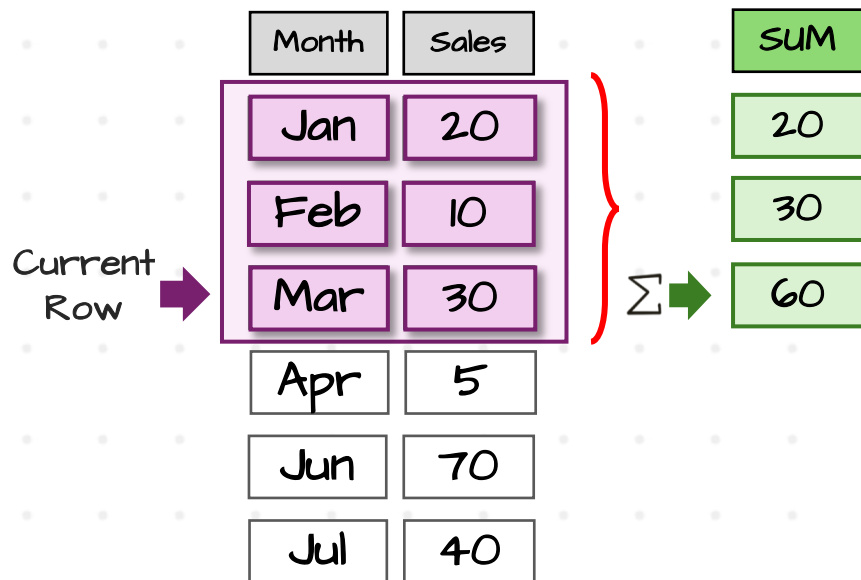
## Running Total

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

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

Default Frame

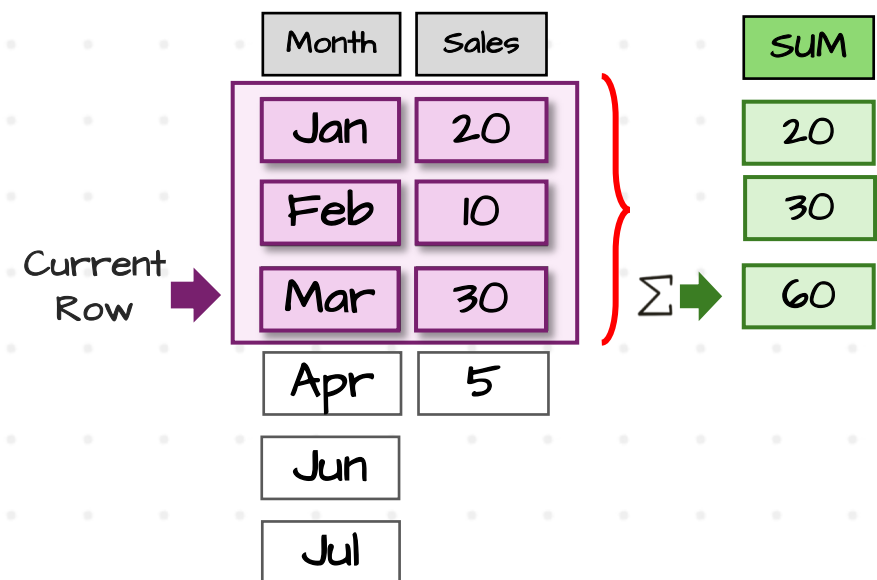
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



## 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**

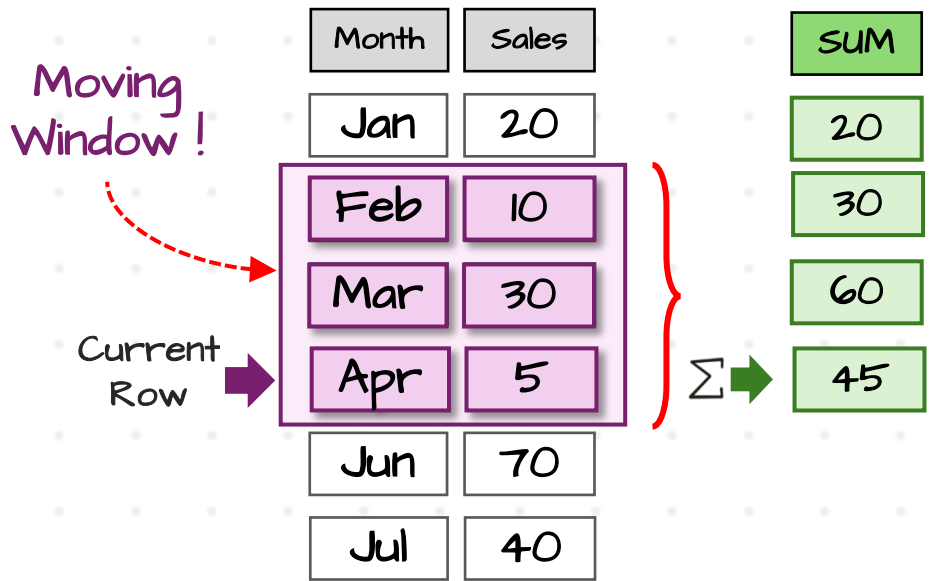
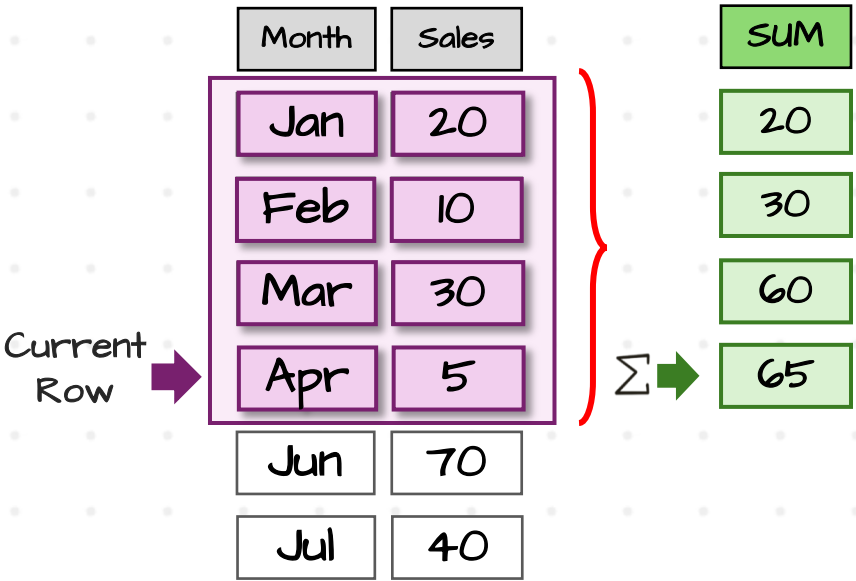
```
SUM(Sales) OVER(  
ORDER BY Month)
```

# Rolling Total

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

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

Default Frame  
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



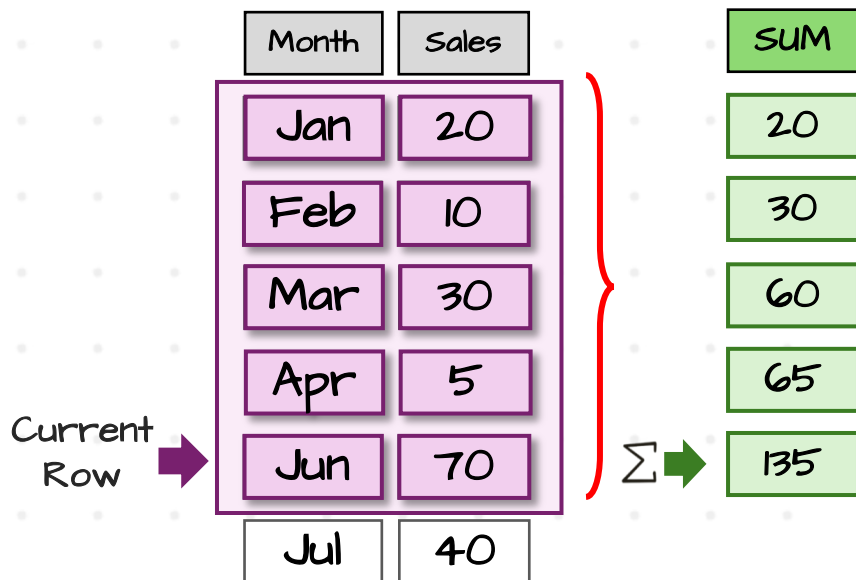
## Running Total

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

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

Default Frame

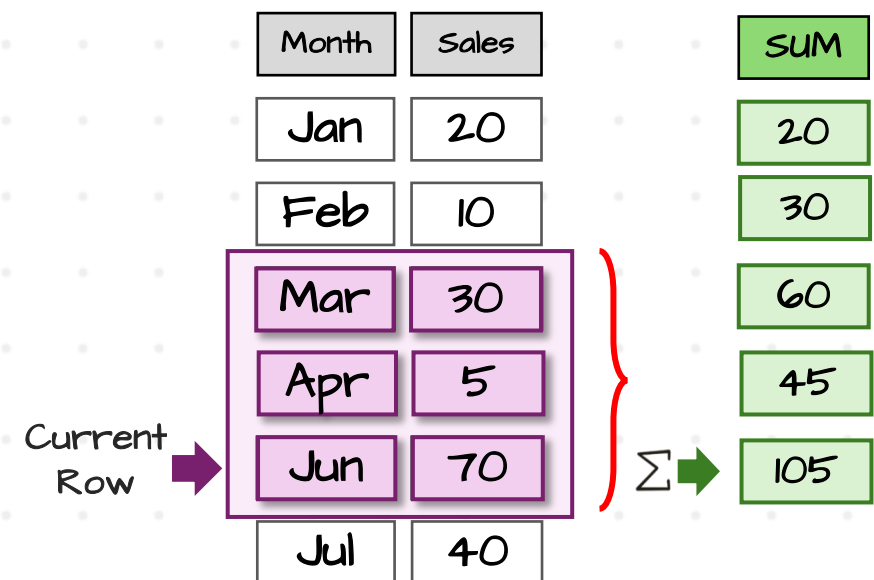
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW



## 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

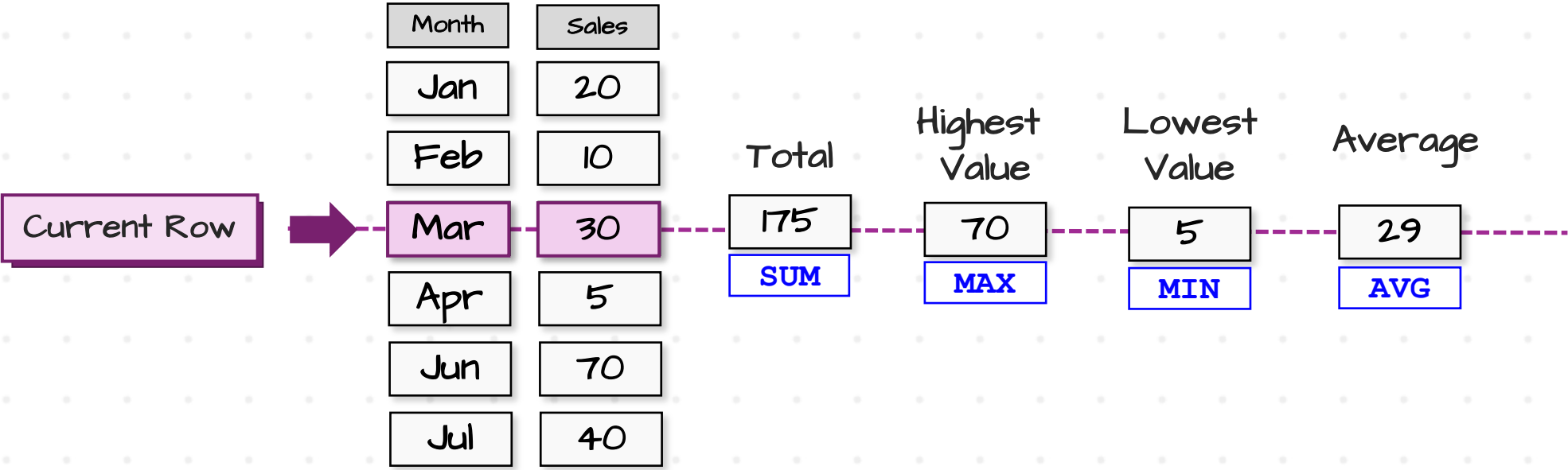
```
SUM(Sales) OVER(  
ORDER BY MONTH  
ROWS 2 PRECEDING)
```

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  
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 ()

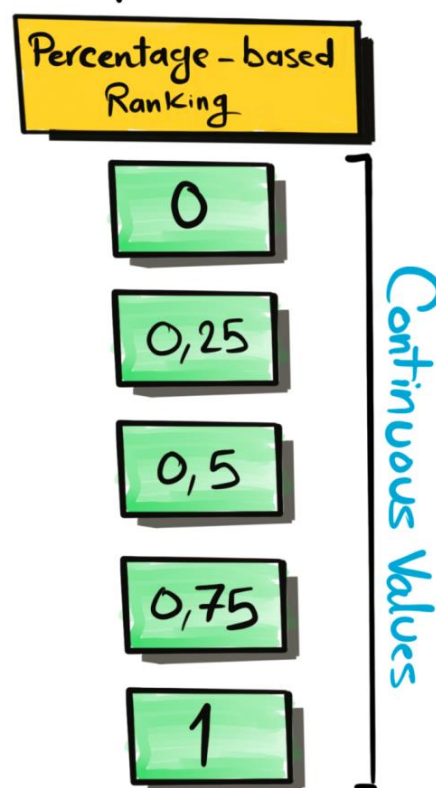
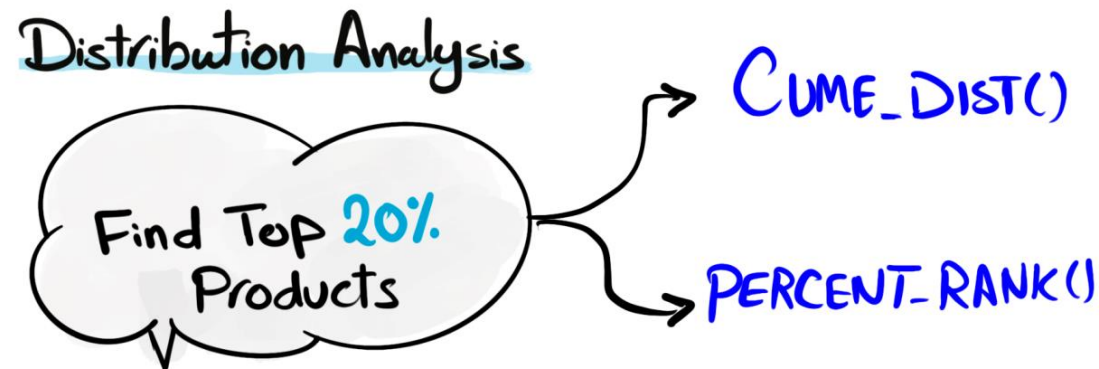
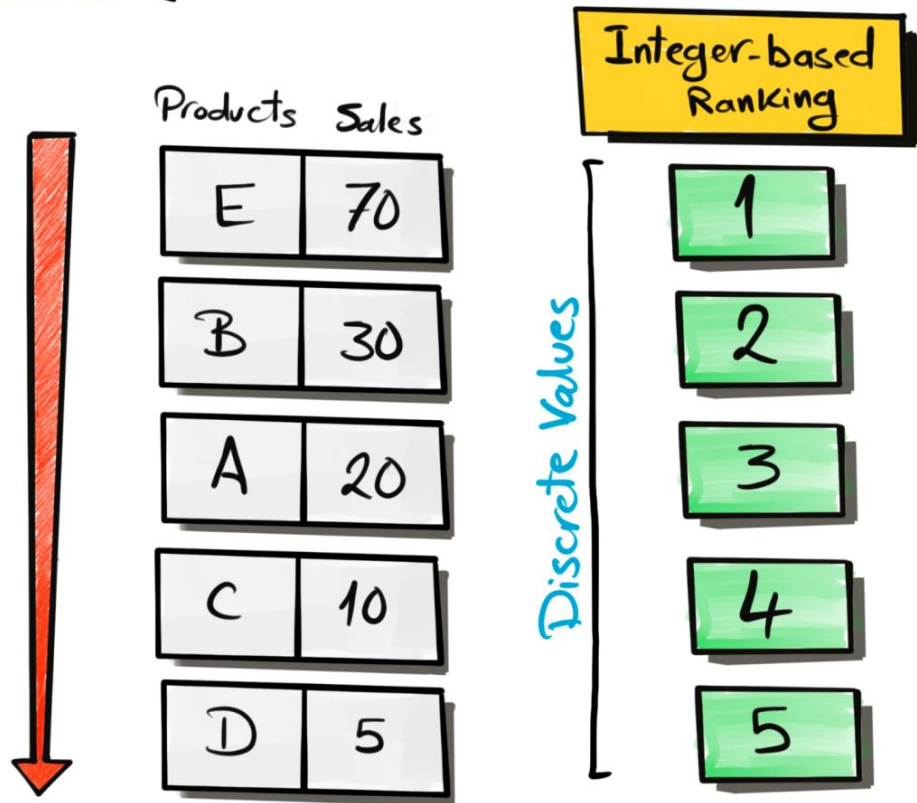
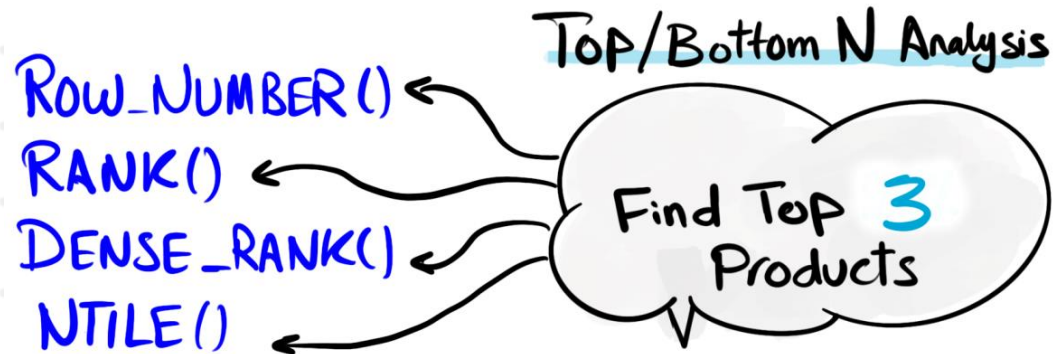
Return a specific value in a window to be compared with the value of current row



| 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)
```

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

We have a **tie** here!

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)

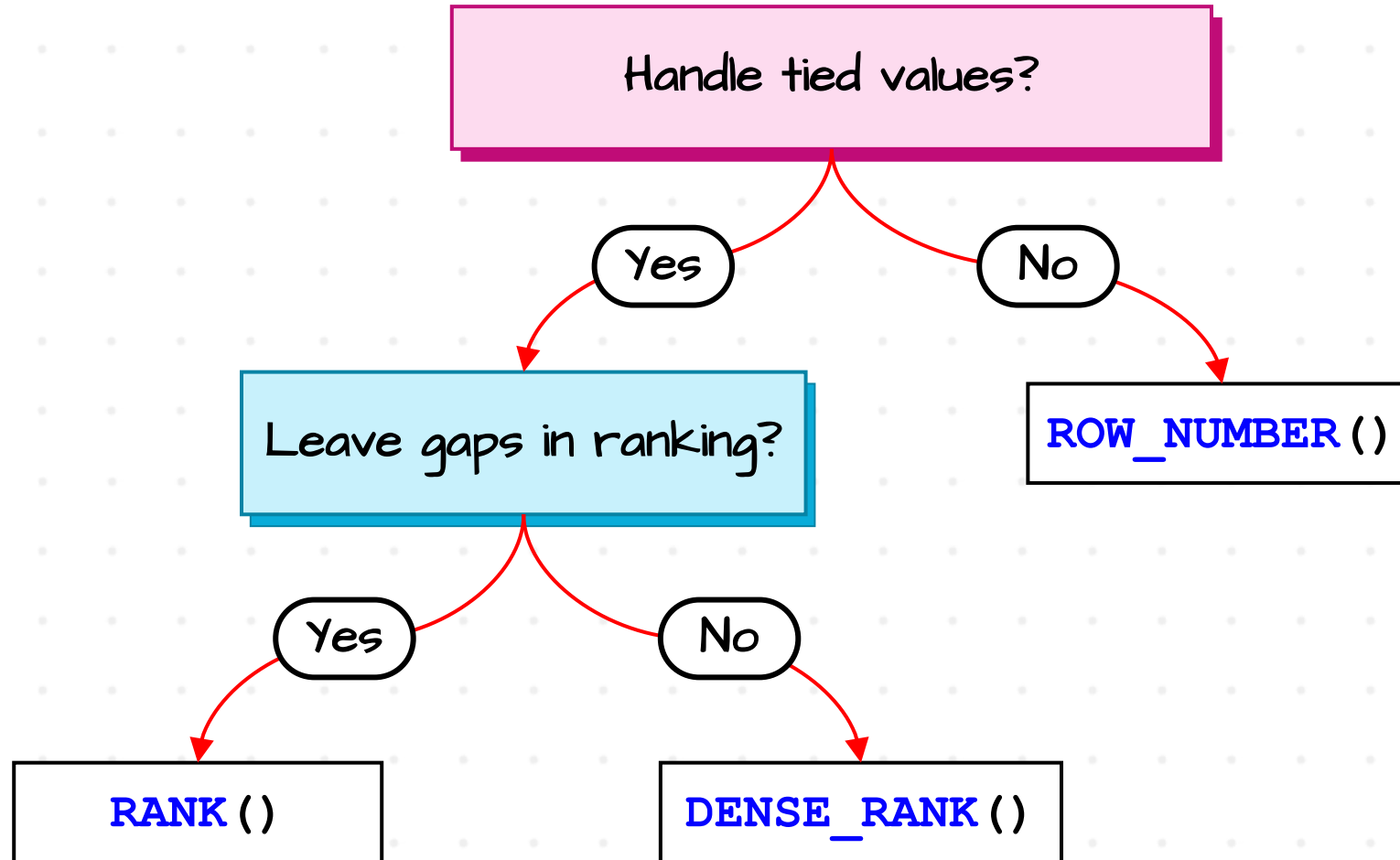
| Sales | Rank |
|-------|------|
| 100   | 1    |
| 80    | 2    |
| 80    | 2    |
| 50    | 3    |
| 20    | 4    |

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)**

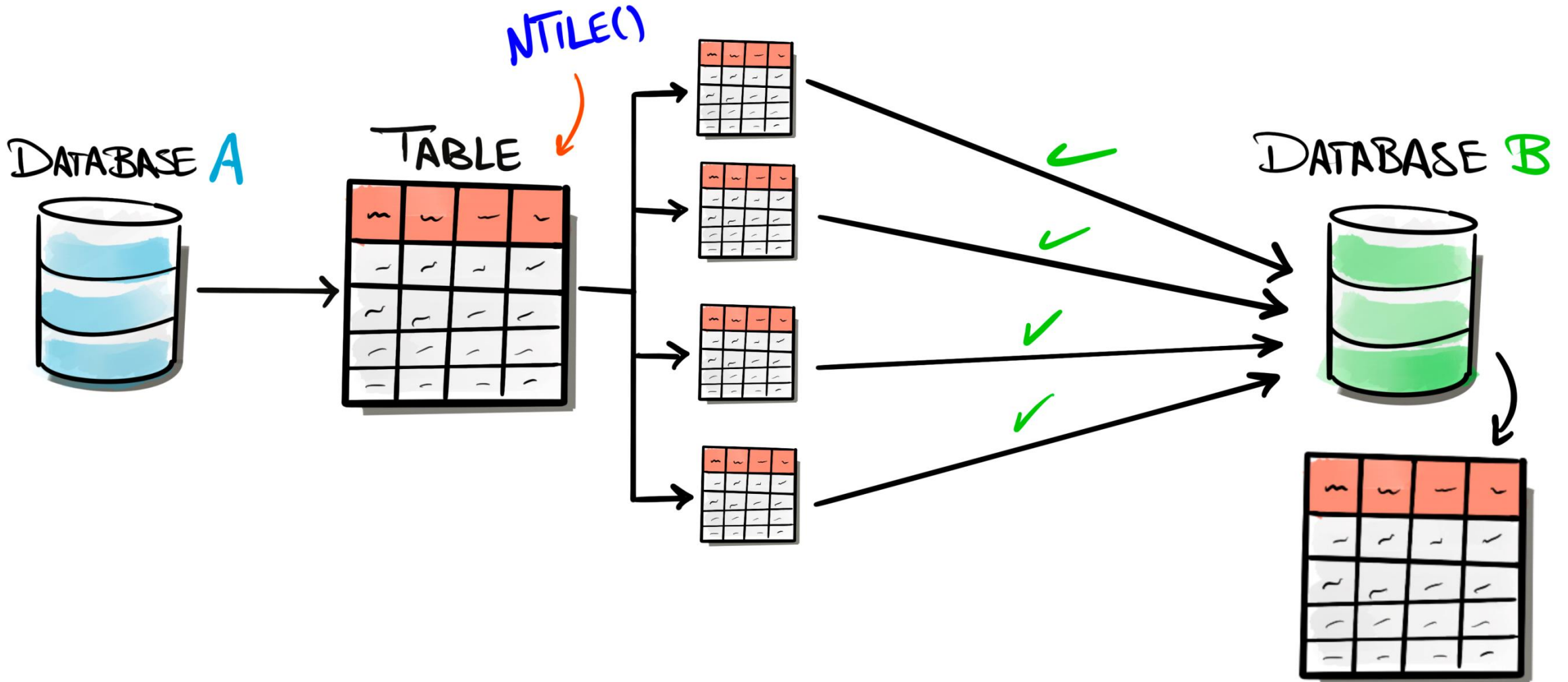
**DATA ANALYST**

**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) |
|-------|-----------|
| 100   | 1         |
| 80    | 1         |
| 80    | 1         |
| 50    | 2         |
| 20    | 2         |

Bucket (1) →

Bucket (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)

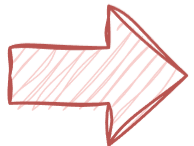
$$\text{Bucket Size} = \frac{5}{3} = 1.7$$

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)
```

Current Row



| Sales | Dist |
|-------|------|
| 20    | 0,2  |
| 50    |      |
| 60    |      |
| 80    |      |
| 100   |      |

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

$$\text{CUME\_DIST}(20) = \frac{1}{5} = 0,2$$

# 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



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

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

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)
```

|               | Sales | Rank | Dist |
|---------------|-------|------|------|
| Current Row → | 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



# WINDOW VALUE FUNCTIONS

Baraa Khatib Salkini  
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)

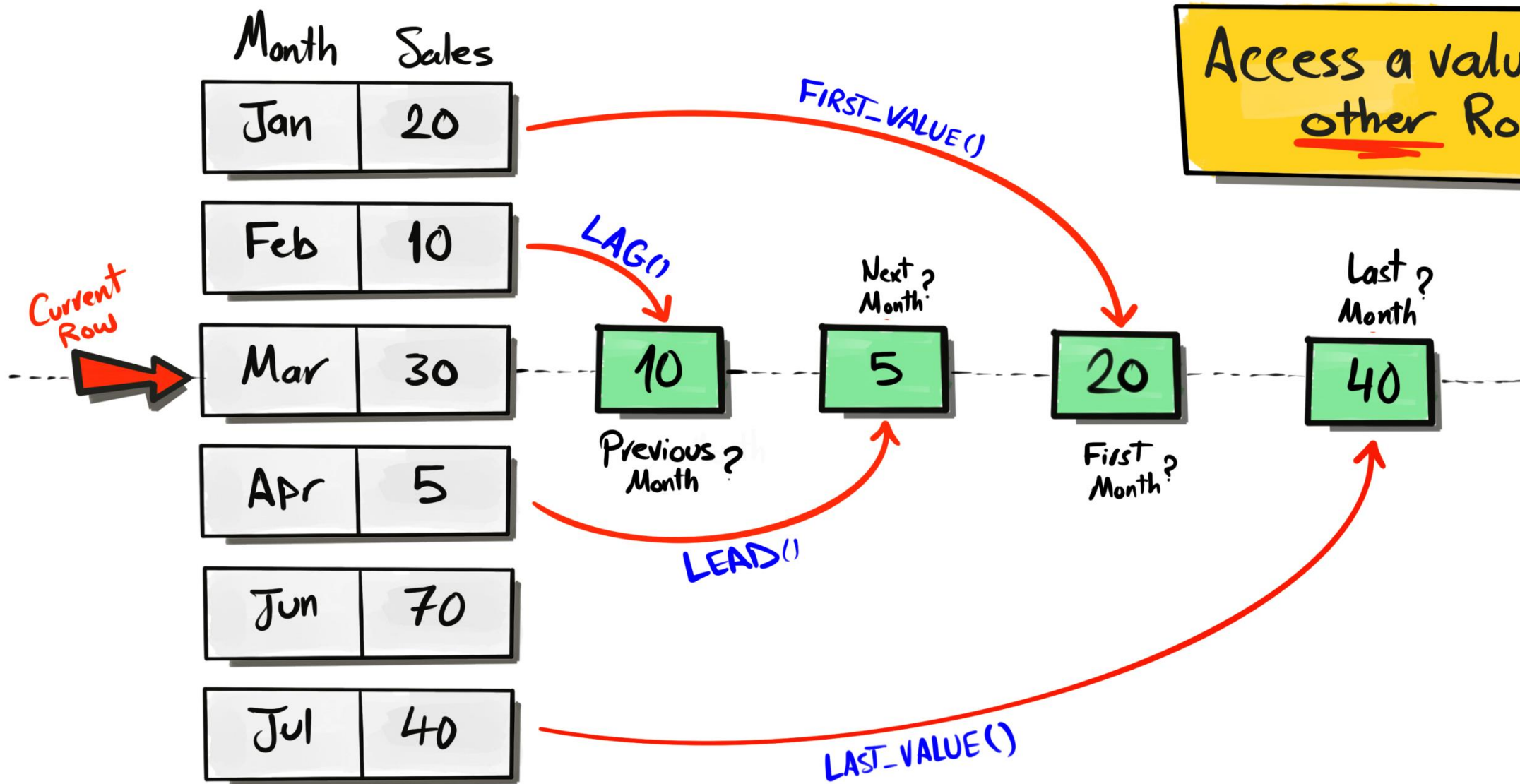
**LAST\_VALUE** (*expr*)

Returns the last value in a window

**LAST\_VALUE**(Sales) **OVER** (**ORDER BY** OrderDate)

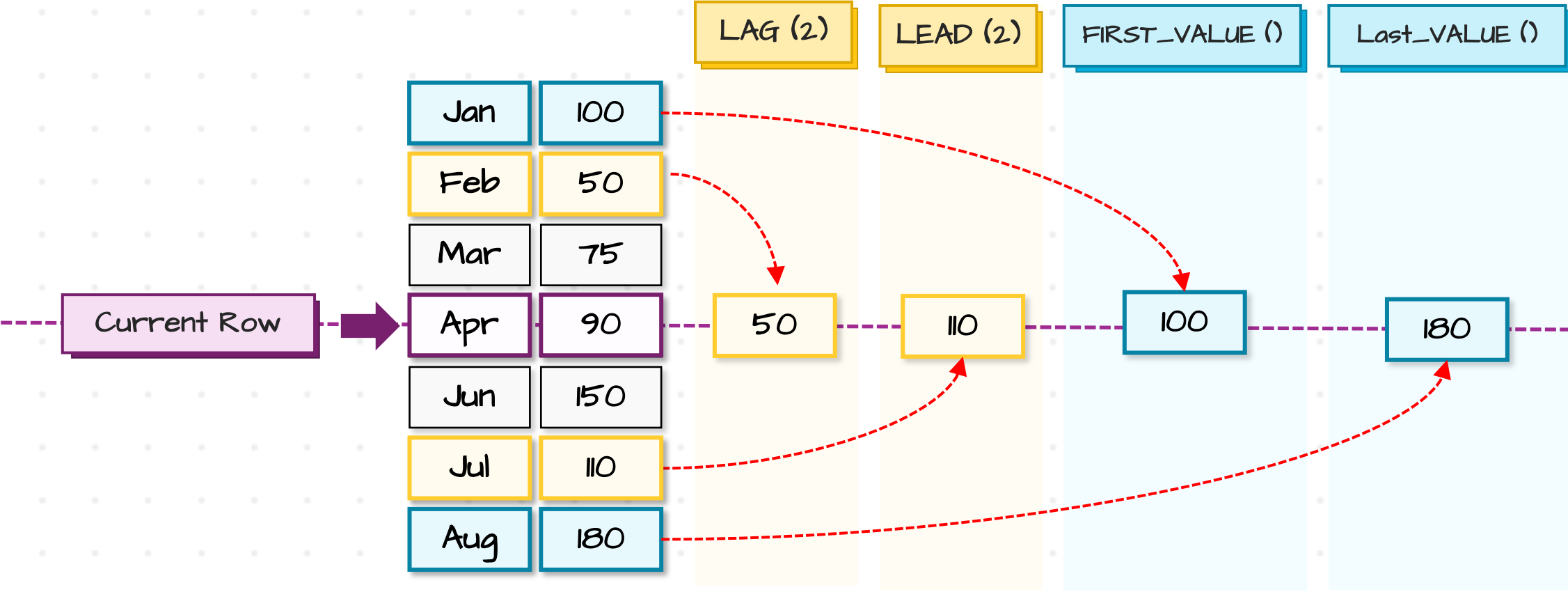
# Value Functions

|  | Expression    | Partition Clause | Order Clause | Frame Clause   |
|--|---------------|------------------|--------------|----------------|
| <b>Value</b><br>(Analytics)<br>Functions                     | All Data Type | Optional         | Required     | Not allowed    |
| <b>LEAD</b> ( <i>expr</i> , <i>offset</i> , <i>default</i> ) |               |                  |              | Optional       |
| <b>LAG</b> ( <i>expr</i> , <i>offset</i> , <i>default</i> )  |               |                  |              | Should be used |
| <b>FIRST_VALUE</b> ( <i>expr</i> )                           |               |                  |              |                |
| <b>LAST_VALUE</b> ( <i>expr</i> )                            |               |                  |              |                |



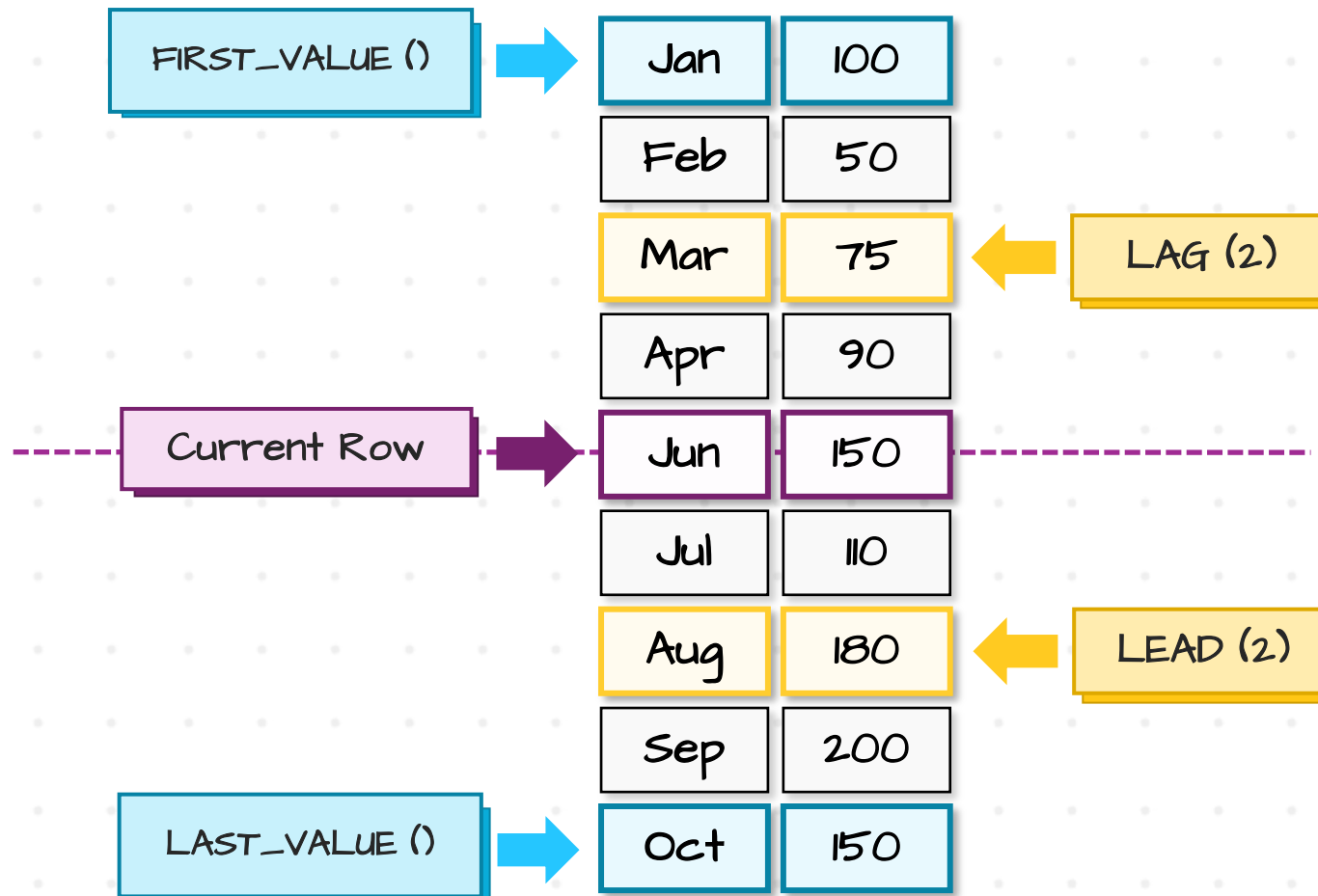
Access a value from other Row

# Value Functions

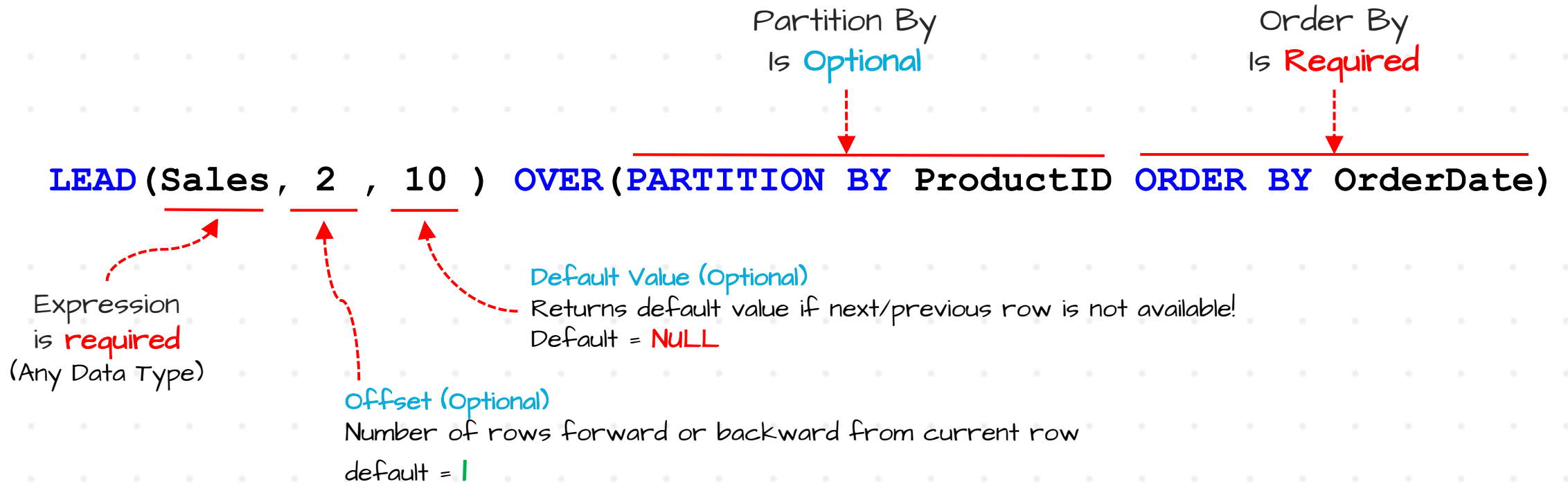


# Value Functions

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



# LEAD & LAG



# 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)
```

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

# LAG

Access **Previous** Row

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

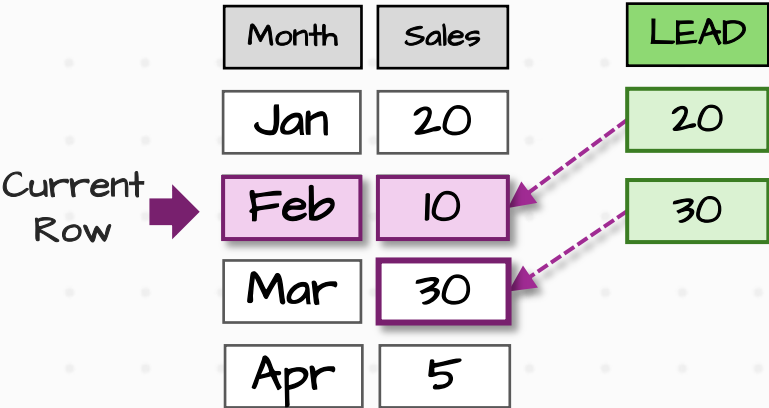
First Row has **No** Previous Row !

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

# LEAD

Access **Next** Row

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

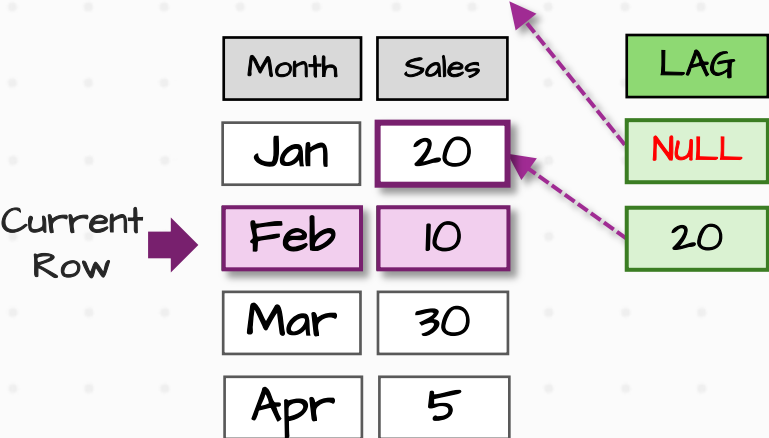


# LAG

Access **Previous** Row

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

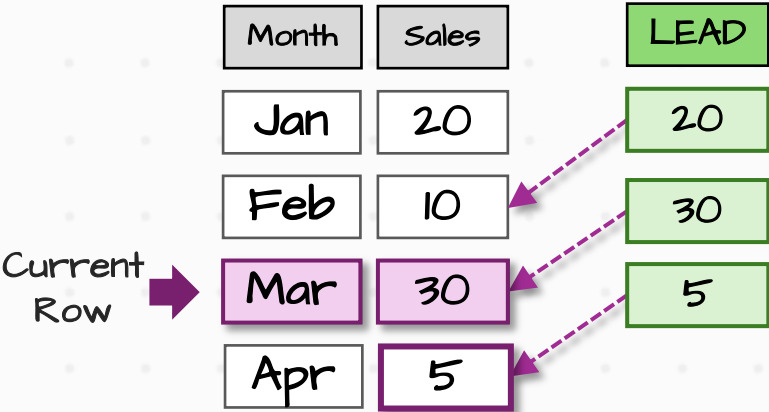
First Row has **No** Previous Row !



# LEAD

Access **Next** Row

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

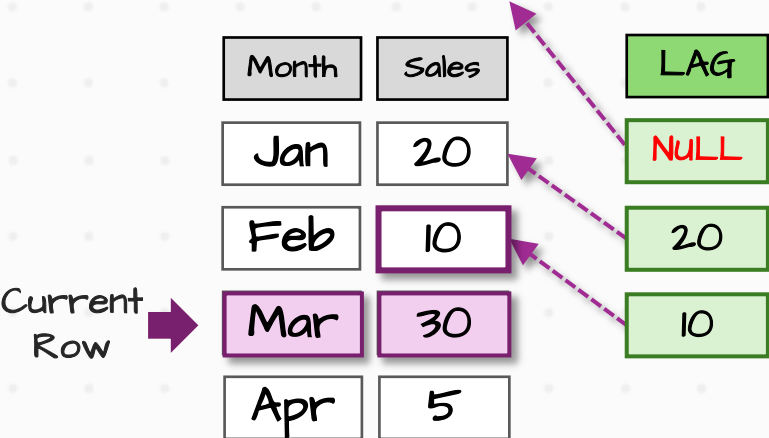


# LAG

Access **Previous** Row

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

First Row has **No** Previous 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 →

## LEAD & LAG

# 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

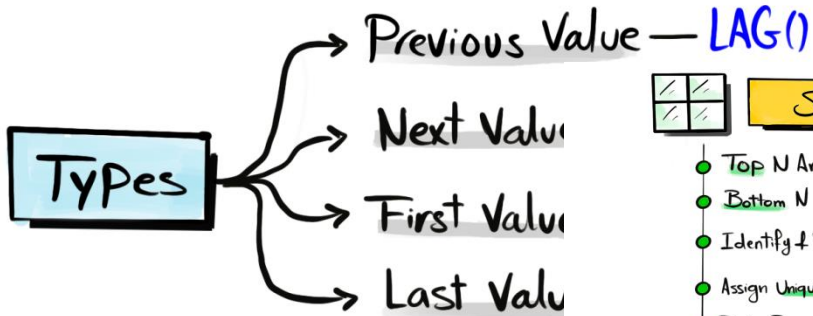
`LAST_VALUE(Sales) OVER (ORDER BY Month  
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)`

| 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 result for each month. The 'Current Row' is highlighted for April, and the 'UNBOUNDED FOLLOWING' range is indicated by a blue arrow pointing to the right, showing that the window includes all rows from the current row to the end of the partition.

# WINDOW VALUE (ANALYTICAL) FUNCTIONS

Allow Access specific Value from another Row



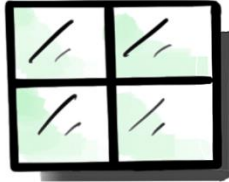
## SQL WINDOW USE CASES

- 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  $\begin{matrix} \nearrow \text{Highest} \\ \searrow \text{Lowest} \end{matrix}$
- Outlier Detection
- Running Total
- Rolling Total
- Moving Average

## Rules

- Expression — Any Data Type
- ORDER BY — Required
- FRAME — Optional

- Time Series Analysis: MoM & YoY
- Time Gaps Analysis: Customer Retention
- Comparison Analysis: Extreme  $\begin{matrix} \nearrow \text{Highest} \\ \searrow \text{Lowest} \end{matrix}$



# 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