Frame Clause:

→ Frame clause which rows are used for calculation for the current row inside a window.

> If the (window) tells, where you are allowed to look, the (frame) tells; how much you actually look

Window:

> Defined by (partitionBy) + (orderBy)
> ~~Defines~~ Decides the set of rows available

Frame:

> Defined by rowsBetween / rangeBetween
> Decides the subset of rows used for calculation

Think of reading a book:

> Partition: which chapter
> orderBy: page order
> frame: how many pages you read around the current page

*Frame clause controls the range of rows used for aggregation, not grouping or ordering.

Default Frame:

If you do not explicitly specify a frame clause, spark automatically applies one.

Rows Between unBounded Preceeding and Current row

Note:

{Default frame is only applies in orderBy() present.}

①
```
sum('salary').over(
        Window.partitionBy('dept').orderBy('salary')
)
```

orderBy is present
so; default frame is applied

②
```
sum('salary').over(
        Window.partitionBy('dept')
)
```

No orderBy()
so; no default frame

Default frame is active only when orderBy() is used

---

rowsBetween():

defines the frame using the number of rows relative to current row.

unBounded Preceding → From the first row of the partition
current row → only the current row
unBounded following → Till the last row of the partition
negative number → previous rows
positive number → next rows

1) rowsBetween(-1, 0)
   - -1 → previous row
   - 0 → current row

2) rowsBetween(0,0) # only current row

3) rowsBetween(Window.unboundedPreceding, Window.currentRow)
   # all previous rows + current row

4) rowsBetween(0, Window.unboundedFollowing)
   # current row + all next rows

5) rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing)
   # use all rows in the partition

---

rangeBetween():

defines the frame based on the value range of the `orderBy` column, not row positions.

Example:

Assume the current row has value X in the `orderBy` column.

rangeBetween(-100, 0)

↓

means:

Include all rows whose `orderBy` value is Between (X-100 and X)

Note: rangeBetween() only works when:
- we have exactly one `orderBy` Column
- That Column is numeric (or) date/timestamp

If orderBy = salary

1000
1200
1500
1800

⇒ rangeBetween(-200, 0)

⇒ Include rows whose salary is within [current_salary - 200 to current_salary]

Best Ex:

current salary	Rows used
1000	(1000)
1200	[1000, 1200]
1500	[1500]
1800	[1800]

```
rangeBetween(
    Window.unBoundedPreceding,
    Window.unboundedfollowing
)
```

All rows in the partition.

---

rangeBetween() (vs) rowsBetween()

rowsBetween() → Counts Rows

rangeBetween() → Counts values

Data

No	Salary
1	1000
2	1200
3	1500
4	1800

① rowsBetween(-1, 0):

Salary	rows used
1000	[1000]
1200	[1000, 1200]
1500	[1200, 1500]
1800	[1500, 1800]

② rangeBetween(-200, 0)

salary	rows used
1000	[1000]
1200	[1000, 1200]
1500	[1500]
1800	[1800]

Aspect	rowsBetween	rangeBetween
Based on	row position	Column values
uses row numbers	Yes	NO
Depends on gaps	NO	Yes
Supports strings	Yes	NO
multiple orderBy	Yes	NO
Predictability	High	Medium

When to use each:

> use `rowsBetween()` when:

- You want previous/next rows
- You want predictable sliding window
- You're new to frame clauses

> use `rangeBetween()` when:

- Time-based windows (last 7 days)
- value-based windows (within 1000₹)
- Business logic depends on value range

1. Default frame:

```
sum("salary").over(
        Window.partitionBy('dept')
              .orderBy('salary')
    )
```

o/p

salary	running_sum
1000	1000
1200	2200
1500	3700
1800	5500

{Rows-Between unBounded-preceding & current row}

2. Current row only:

```
sum('salary').over(
        Window.partitionBy('dept')
              .orderBy('salary')
              .rowsBetween(0,0)
    )
```

o/p

Salary	running_sum
1000	1000
1200	1200
1500	1500
1800	1800

3 Previous + current row:

```
sum('salary').over(
        Window.partitionBy('dept')
              .orderBy('salary')
              .rowsBetween(-1,0)
    )
```

o/p

Salary	running_sum
1000	1000
1200	2200
1500	2700
1800	3300

4. Current + next row:

```
sum('salary').over(
        Window.partitionBy('dept')
              .orderBy('salary')
              .rowsBetween(0,1)
    )
```

o/p

salary	running_sum
1000	2200
1200	2700
1500	3300
1800	1800

5. Entire partition:

```
sum('salary').over(
    Window.partitionBy('dept')
          .orderBy('salary')
          .rowsBetween(
              Window.unboundedPreceding,
              Window.unboundedFollowing
          )
)
```

salary	running-salary
1000	5500
1200	5500
1500	5500
1800	5500

6. rangeBetween(-200,0):

```
sum('salary').over(
    Window.partitionBy('dept')
          .orderBy('salary')
          .rangeBetween(-200,0)
)
```

salary	running-salary
1000	1000
1200	2200
1500	1500
1800	1800

Interview remainders:

- `orderBy()` not only just sorts data, it activates the default frame, changing aggregation behaviour
- `Frame clause` is optional but when "orderBy()" is used it automatically applies the default frame
- `rangeBetween()` has strict rules & unpredictable results if gaps exists.