Window Functions

Window functions let you calculate things like totals, ranks, or comparisons for each row without grouping rows together. Instead, they look at a "window" of rows while keeping each row in the result.

Categories of Window functions

- Aggregate function
- Ranking functions
- Analytical functions

How Window Functions Work

Window functions allow you to perform calculations across a set of rows that are related to the current row. They are called "window functions" because they operate within a window of rows, defined by the OVER() clause. Unlike aggregate functions that group and collapse rows, window functions retain all rows while adding calculated values.

The term "window" refers to the subset of rows that a function can "see" when performing its calculation. You can control the size and boundaries of this window using the PARTITION BY and ORDER BY clauses within the OVER() statement.

Window Function Syntax

<function>() OVER ([PARTITION BY column] **[ORDER BY column]** [ROWS | RANGE clause])

Key Components:

- **Function:** The actual function to be applied, such as SUM(), AVG(), ROW_NUMBER(), etc.
- **OVER():** Defines the window within which the function operates.
- **PARTITION BY (optional):** Divides the data into groups (or "partitions"). The window function operates separately within each partition.If omitted, the window includes all rows.

- **ORDER BY :** Specifies the order of rows within the window.Many window functions (like ranking) is mandatory to use ORDER BY.
- **ROWS or RANGE (optional):** Specifies the boundaries of the window relative to the current row.

  Examples:
  - ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING: Includes one row before and one row after the current row.
  - ROWS UNBOUNDED PRECEDING: Includes all rows from the start of the partition to the current row.

Understanding How a Window Is Created

Let’s imagine we’re in a classroom with students, their marks, and their sections (e.g., Section A and Section B). A window is like a group of rows related to the current row. Each window is defined by specific rules using the OVER() clause.

Problem:

We want to calculate the following for each student:

1. Total marks of all students in their section.
2. Rank of the student within their section.
3. Class average marks for comparison.

Student Data (Imagine this as a table):

Student Name	Section	Marks
Alice	A	85
Bob	A	90
Charlie	A	80
David	B	70
Eve	B	95
Frank	B	85

Without any partitioning or ordering, the window includes all rows in the table.

```
SELECT StudentName, Marks, SUM(Marks) OVER () AS TotalMarks
FROM Students;
```

Student Name	Marks	TotalMarks
Alice	85	505
Bob	90	505
Charlie	80	505
David	70	505
Eve	95	505
Frank	85	505

When we add PARTITION BY Section, the window is split into smaller windows for each section.

SELECT StudentName, Section, Marks, SUM(Marks) OVER (PARTITION BY Section) AS SectionTotal FROM Students;

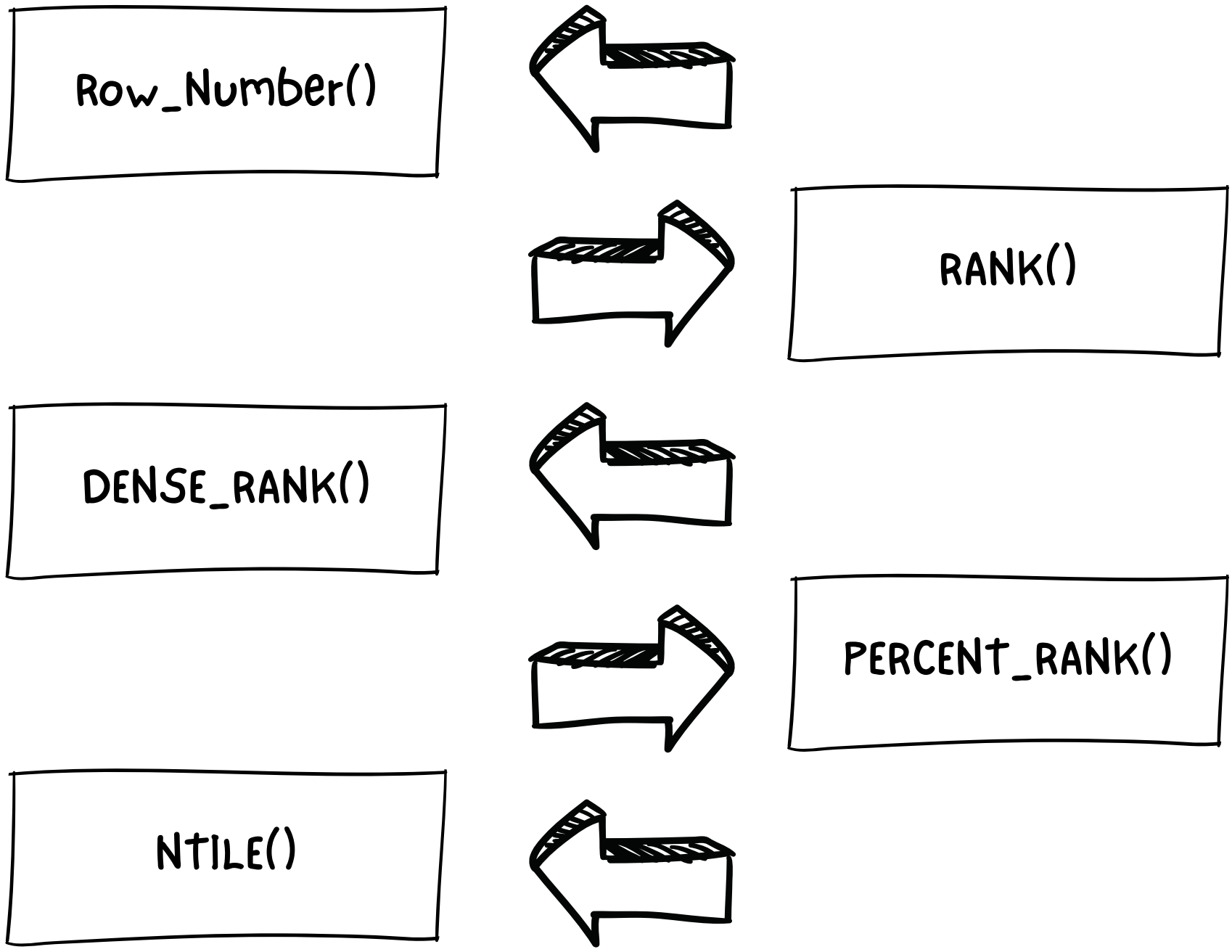
Student Name	Section	Marks	SectionTotal
Alice	A	85	255
Bob	A	90	255
Charlie	A	80	255
David	B	70	250
Eve	B	95	250
Frank	B	85	250

When we add ORDER BY Marks, the rows within each partition (window) are sorted by marks.

```
SELECT StudentName, Section, Marks,
    RANK() OVER (PARTITION BY Section ORDER BY Marks DESC) AS Rank
FROM Students;
```

Student Name	Section	Marks	Rank
Bob	A	90	1
Alice	A	85	2
Charlie	A	80	3
Eve	B	95	1
Frank	B	85	2
David	B	70	3

Ranking functions



Ranking Functions

Student Name	Section	Marks
Alice	A	85
Bob	A	90
Charlie	A	85
David	B	70
Eve	B	95
Frank	B	85

ROW_NUMBER()

Assigns a unique number to each row within a partition (ordered by marks):

```
SELECT StudentName, Section, Marks,
    ROW_NUMBER() OVER (PARTITION BY Section ORDER BY Marks DESC) AS
RowNum FROM Students;
```

Student Name	Section	Marks	RowNum
Bob	A	90	1
Alice	A	85	2
Charlie	A	85	3
Eve	B	95	1
Frank	B	85	2
David	B	70	3

RANK()

Assigns the same rank to identical marks but leaves gaps in ranking

```
SELECT StudentName, Section, Marks,
    RANK() OVER (PARTITION BY Section ORDER BY Marks DESC) AS Rank
FROM Students;
```

Student Name	Section	Marks	Rank
Bob	A	90	1
Alice	A	85	2
Charlie	A	85	2
Eve	B	95	1
Frank	B	85	2
David	B	70	3

DENSE_RANK() Works like RANK() but avoids gaps

```
SELECT StudentName, Section, Marks,
    DENSE_RANK() OVER (PARTITION BY Section ORDER BY Marks DESC) AS
DenseRank FROM Students;
```

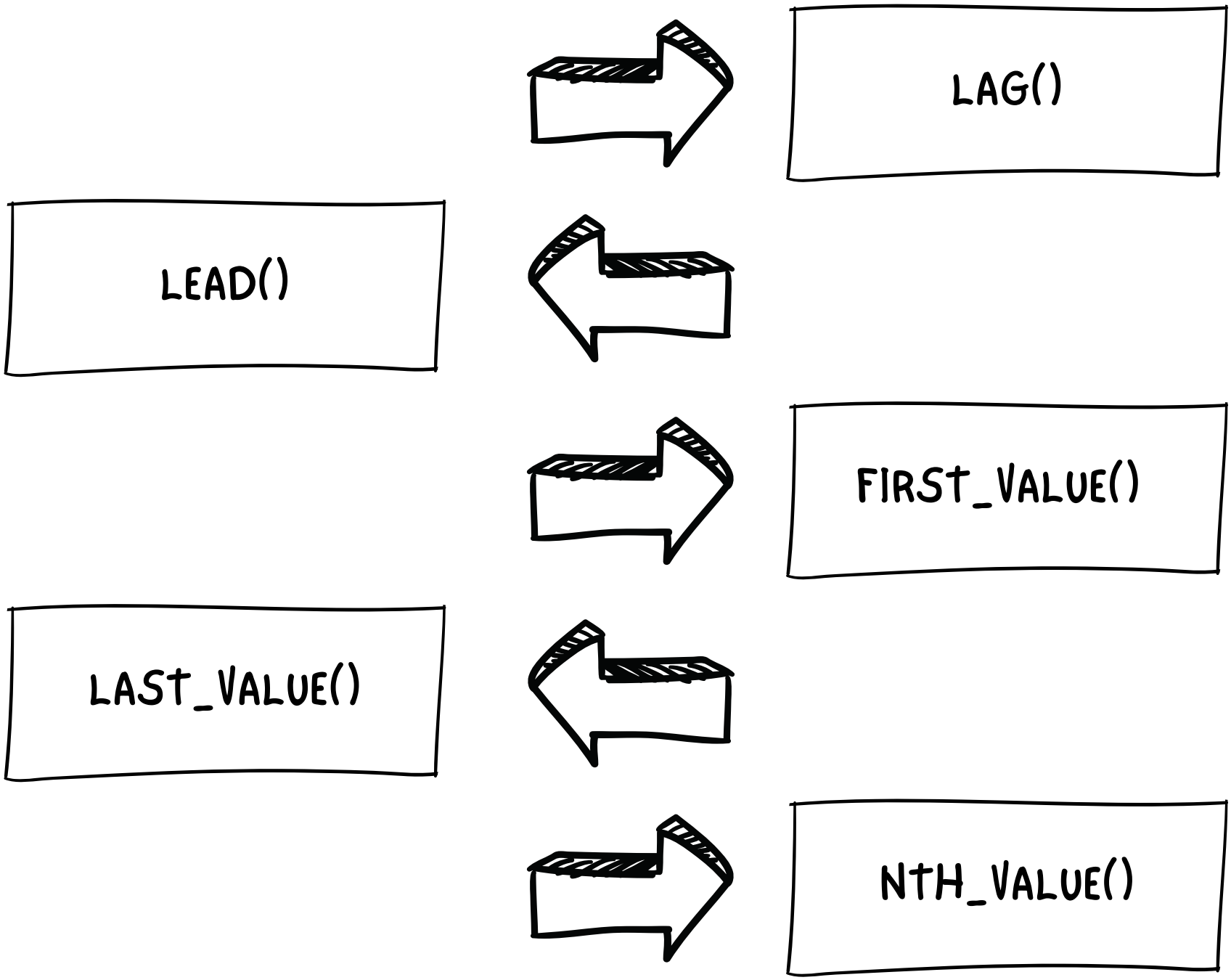
Student Name	Section	Marks	DenseRank
Bob	A	90	1
Alice	A	85	2
Charlie	A	85	2
Eve	B	95	1
Frank	B	85	2
David	B	70	3

NTILE(n) Divides rows into n equal-sized buckets

```
SELECT StudentName, Section, Marks,
    NTILE(2) OVER (PARTITION BY Section ORDER BY Marks DESC)
AS NTile FROM Students;
```

Student Name	Section	Marks	NTile
Bob	A	90	1
Alice	A	85	1
Charlie	A	85	2
Eve	B	95	1
Frank	B	85	1
David	B	70	2

Analytical functions



Analytical Functions

Analytical functions compute values based on a group of rows without collapsing them into a single row.

LAG() and LEAD()

```
SELECT StudentName, Section, Marks,
    LAG(Marks) OVER (PARTITION BY Section ORDER BY Marks DESC) AS PrevMarks,
    LEAD(Marks) OVER (PARTITION BY Section ORDER BY Marks DESC) AS NextMarks FROM Students;
```

Student Name	Section	Marks	PrevMarks	NextMarks
Bob	A	90	NULL	85
Alice	A	85	90	85
Charlie	A	85	85	NULL
Eve	B	95	NULL	85
Frank	B	85	95	70
David	B	70	85	NULL

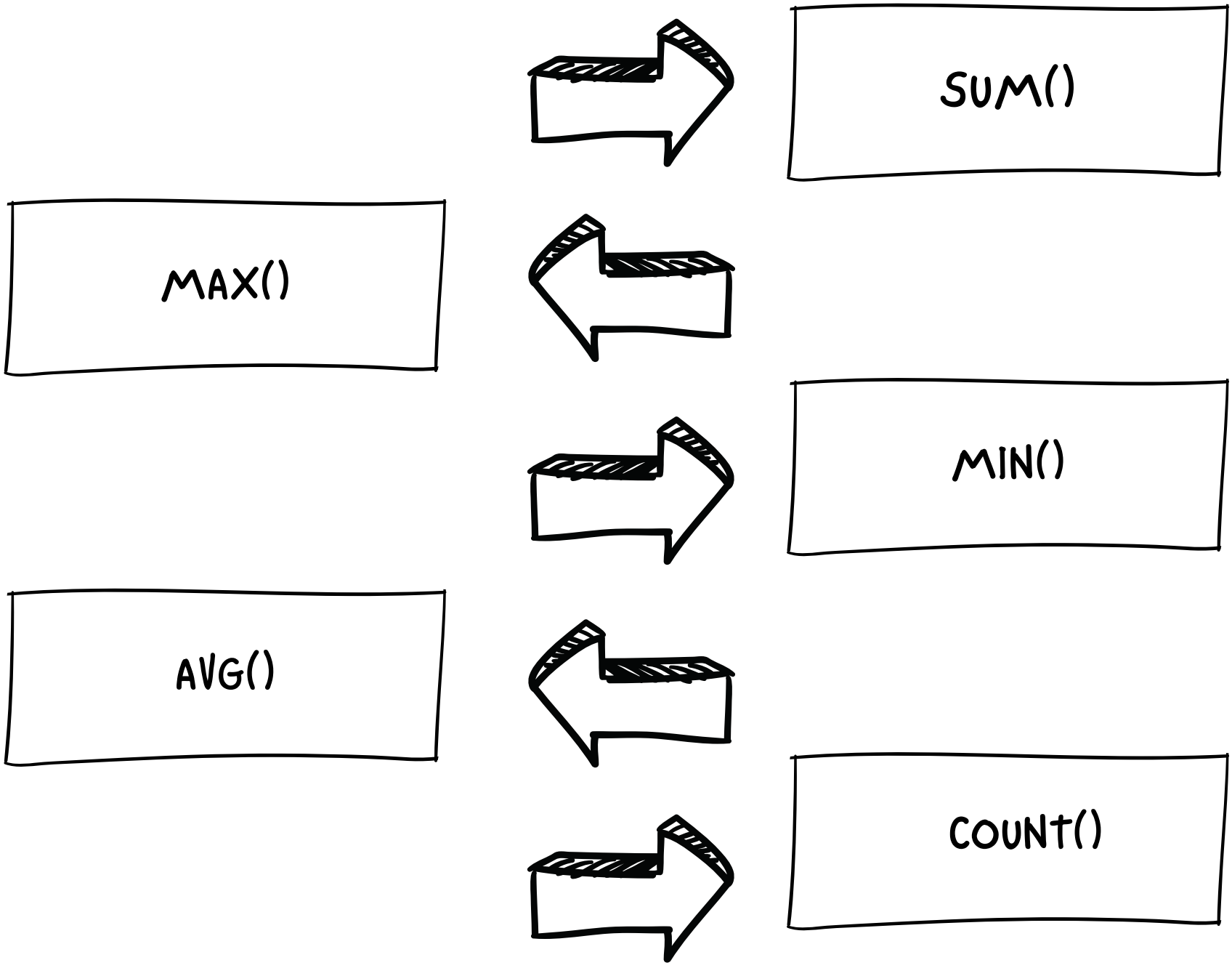
FIRST_VALUE() and LAST_VALUE()

Find the top and bottom scorers in each section

```
SELECT StudentName, Section, Marks,
    FIRST_VALUE(Marks) OVER (PARTITION BY Section ORDER BY Marks
DESC) AS TopScore,
    LAST_VALUE(Marks) OVER (PARTITION BY Section ORDER BY Marks ASC
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
AS BottomScore FROM Students;
```

Student Name	Section	Marks	TopScore	BottomScore
Bob	A	90	90	80
Alice	A	85	90	80
Charlie	A	85	90	80
Eve	B	95	95	70
Frank	B	85	95	70
David	B	70	95	70

Aggregate function



Aggregate Functions

SUM(): Adds all values.

AVG(): Calculates the average.

COUNT(): Counts rows.

MIN(): Finds the smallest value.

MAX(): Finds the largest value.

```
SELECT StudentName, Section, Marks,
    SUM(Marks) OVER (PARTITION BY Section) AS SectionTotal,
    AVG(Marks) OVER (PARTITION BY Section) AS SectionAvg,
    MAX(Marks) OVER (PARTITION BY Section) AS SectionMax,
    MIN(Marks) OVER (PARTITION BY Section) AS SectionMin FROM
Students;
```

Student Name	Section	Marks	SectionTotal	SectionAvg	SectionMax	SectionMin
Alice	A	85	255	85	90	80
Bob	A	90	255	85	90	80
Charlie	A	80	255	85	90	80
David	B	70	250	83.33	95	70
Eve	B	95	250	83.33	95	70
Frank	B	85	250	83.33	95	70