

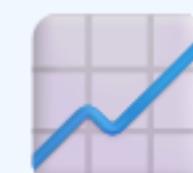
# ⚡ SQL Window Functions

## Complete Masterclass Guide



### ROW\_NUMBER()

Assigns unique sequential integers to rows within a partition



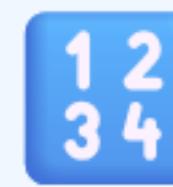
### RANK()

Ranks rows with gaps for equal values



### DENSE\_RANK()

Ranks rows without gaps for equal values



### NTILE(n)

Divides rows into n roughly equal groups



### LAG()

Accesses previous row's value



### LEAD()

Accesses next row's value



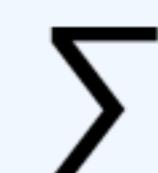
### FIRST\_VALUE()

Gets first value in window frame



### LAST\_VALUE()

Gets last value in window frame



### SUM() OVER()

Running/cumulative totals

# ROW\_NUMBER()

Assign unique sequential numbers

## Example 1: Rank employees by salary

```
SELECT
    employee_name,
    department,
    salary,
    ROW_NUMBER() OVER (
        ORDER BY salary DESC
    ) AS rank
FROM employees;
```

## Output

employee_name	department	salary	rank
John	IT	90000	1
Sarah	Sales	85000	2
Mike	IT	80000	3
Lisa	HR	75000	4
Tom	Sales	75000	5

## Example 2: Rank by department

```
SELECT
    employee_name,
    department,
    salary,
    ROW_NUMBER() OVER (
        PARTITION BY department
        ORDER BY salary DESC
    ) AS dept_rank
FROM employees;
```

## Output

employee_name	department	salary	dept_rank
John	IT	90000	1
Mike	IT	80000	2
Sarah	Sales	85000	1
Tom	Sales	75000	2

# RANK()

Rank with gaps for equal values

## Example 1: Rank students by score

```
SELECT
    student_name,
    score,
    RANK() OVER (
        ORDER BY score DESC
    ) AS rank
FROM students;
```

## Output

student_name	score	rank
Alice	95	1
Bob	95	1
Charlie	92	3
Diana	90	4
Eve	90	4
Frank	85	6

## Example 2: Rank products by sales

```
SELECT
    product_name,
    category,
    units_sold,
    RANK() OVER (
        PARTITION BY category
        ORDER BY units_sold DESC
    ) AS sales_rank
FROM products;
```

## Output

product_name	category	units_sold	sales_rank
Laptop Pro	Electronics	1500	1
Phone X	Electronics	1500	1
Tablet Air	Electronics	1200	3
Shirt	Clothing	800	1
Jeans	Clothing	600	2

# DENSE\_RANK()

Rank without gaps for equal values

## Example 1: Rank students with DENSE\_RANK

```
SELECT
    student_name,
    score,
    DENSE_RANK() OVER (
        ORDER BY score DESC
    ) AS dense_rank
FROM students;
```

## Output

student_name	score	dense_rank
Alice	95	1
Bob	95	1
Charlie	92	2
Diana	90	3
Eve	90	3
Frank	85	4

## Example 2: Compare RANK vs DENSE\_RANK

```
SELECT
    product_name,
    price,
    RANK() OVER (
        ORDER BY price DESC
    ) AS rank,
    DENSE_RANK() OVER (
        ORDER BY price DESC
    ) AS dense_rank
FROM products;
```

## Output

product_name	price	rank	dense_rank
Laptop	1000	1	1
Phone	1000	1	1
Tablet	800	3	2
Watch	600	4	3
Headphones	600	4	3

# NTILE()

Divide rows into groups

## Example 1: Divide into 4 quartiles

```
SELECT  
    customer_name,  
    total_spent,  
    NTILE(4) OVER (  
        ORDER BY total_spent DESC  
    ) AS quartile  
FROM customers;
```

## Output

customer_name	total_spent	quartile
John	5000	1
Sarah	4500	1
Mike	4000	2
Lisa	3500	2
Tom	3000	3
Emma	2500	3
Alex	2000	4

## Example 2: Divide students into 3 performance groups

```
SELECT  
    student_name,  
    score,  
    NTILE(3) OVER (  
        ORDER BY score DESC  
    ) AS performance_group  
FROM students;
```

## Output

student_name	score	performance_group
Alice	95	1
Bob	92	1
Charlie	90	2
Diana	88	2
Eve	85	3
Frank	82	3

# LAG()

Get value from previous row

## Example 1: Compare monthly sales

```
SELECT
    month,
    sales,
    LAG(sales) OVER (
        ORDER BY month
    ) AS prev_sales
FROM monthly_sales;
```

## Output

month	sales	prev_sales
Jan	10000	NULL
Feb	12000	10000
Mar	15000	12000
Apr	14000	15000

## Example 2: Price changes

```
SELECT
    date,
    price,
    LAG(price) OVER (
        ORDER BY date
    ) AS prev_price
FROM stock_prices;
```

## Output

date	price	prev_price
2024-01-01	100	NULL
2024-01-02	102	100
2024-01-03	98	102
2024-01-04	105	98

# LEAD()

Get value from next row

## Example 1: Next exam score

```
SELECT
    student,
    score,
    LEAD(score) OVER (
        ORDER BY score DESC
    ) AS next_score
FROM exam_results;
```

## Output

student	score	next_score
Alice	95	92
Bob	92	88
Charlie	88	85
Diana	85	NULL

## Example 2: Next day's temperature

```
SELECT
    date,
    temperature,
    LEAD(temperature) OVER (
        ORDER BY date
    ) AS next_day_temp
FROM weather;
```

## Output

date	temperature	next_day_temp
2024-01-01	72	75
2024-01-02	75	70
2024-01-03	70	68
2024-01-04	68	NULL

# FIRST\_VALUE()

Get first value in window

## Example 1: First salary in department

```
SELECT
    employee,
    department,
    salary,
    FIRST_VALUE(salary) OVER (
        PARTITION BY department
        ORDER BY hire_date
    ) AS first_salary
FROM employees;
```

## Output

employee	department	salary	first_salary
John	Sales	50000	50000
Mike	Sales	55000	50000
Sarah	IT	70000	70000
Tom	IT	75000	70000

## Example 2: First product price

```
SELECT
    product,
    date,
    price,
    FIRST_VALUE(price) OVER (
        PARTITION BY product
        ORDER BY date
    ) AS first_price
FROM prices;
```

## Output

product	date	price	first_price
Laptop	2024-01-01	1000	1000
Laptop	2024-02-01	950	1000
Phone	2024-01-01	800	800
Phone	2024-02-01	750	800

# LAST\_VALUE()

Get last value in window

## Example 1: Latest price for each product

```
SELECT
    product,
    date,
    price,
    LAST_VALUE(price) OVER (
        PARTITION BY product
        ORDER BY date
    ) AS latest_price
FROM product_prices;
```

## Output

product	date	price	latest_price
Laptop	2024-01-01	1000	900
Laptop	2024-02-01	950	900
Laptop	2024-03-01	900	900
Phone	2024-01-01	800	700

## Example 2: Final exam score

```
SELECT
    student,
    exam,
    score,
    LAST_VALUE(score) OVER (
        PARTITION BY student
        ORDER BY exam_date
    ) AS final_score
FROM student_scores;
```

## Output

student	exam	score	final_score
Alice	Midterm	85	90
Alice	Final	90	90
Bob	Midterm	78	82
Bob	Final	82	82

# SUM() OVER()

Calculate running totals

## Example 1: Daily running total

```
SELECT  
    date,  
    sales,  
    SUM(sales) OVER (  
        ORDER BY date  
    ) AS running_total  
FROM daily_sales;
```

## Output

date	sales	running_total
Jan 1	1000	1000
Jan 2	1500	2500
Jan 3	1200	3700
Jan 4	1800	5500

## Example 2: Category running total

```
SELECT  
    category,  
    month,  
    revenue,  
    SUM(revenue) OVER (  
        PARTITION BY category  
        ORDER BY month  
    ) AS cat_running_total  
FROM category_revenue;
```

## Output

category	month	revenue	cat_running_total
Electronics	Jan	15000	15000
Electronics	Feb	18000	33000
Clothing	Jan	8000	8000
Clothing	Feb	9500	17500