Video-16

Topics to cover:

- Window Functions in SQL

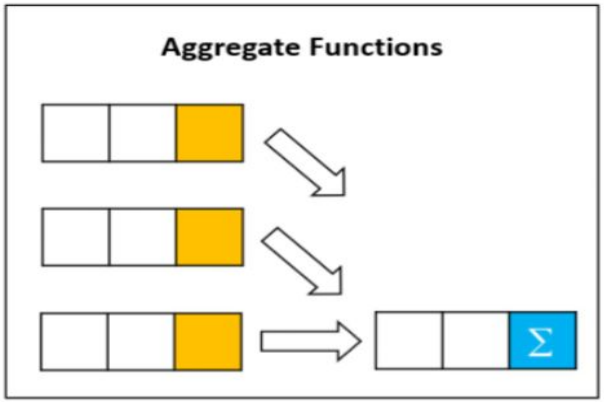
Window Functions

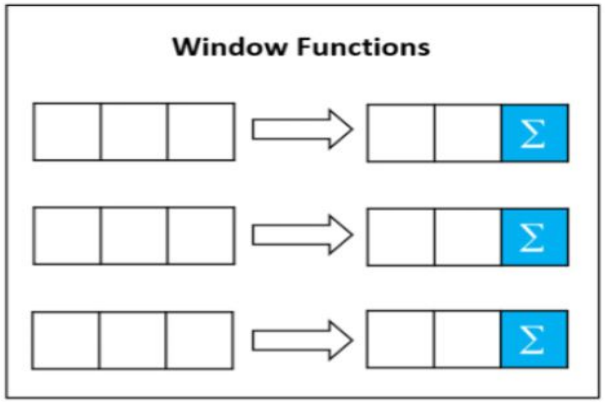
Why we need Window Functions?

When we use group by instead of window function we are getting aggregated data, here actually we are losing some of our attribute or data. That was the biggest disadvantage.

In order to give more flexibility plus keeping the originality of the data we are using the window function.

- **Definition**: Window functions compute values over a "window" (a set of rows) defined by the OVER() clause.





Syntax:

```
SELECT column_name(s),
function_name (expression) OVER (  [PARTITION BY column]
                                   [ORDER BY column]
                                   [ROWS or RANGE frame_spec] )
FROM table_name;
```

• **function_name** : Select a function : Aggregation Functions, Ranking Functions, Value Functions.

• **Expression** : is the name of the column that we want the window function operated on. This may not be necessary depending on what window function is used.

• **OVER** : Define a Window.

PARTITION BY : Divides rows into groups (like GROUP BY but for window functions).

ORDER BY : Defines the order of rows within each partition.This is optional.

Frame specification (ROWS / RANGE): Defines the subset of rows used for calculation (e.g., preceding, following). This is optional and usually not used.

Different Types of Window Functions

Aggregation Functions:

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

Ranking Functions:

- ROW_NUMBER(): Assigns a unique row number to each row, ranking start from 1 and keep increasing till the end of last row
- RANK(): Assigns a rank to each row. Rows with equal values receive the same rank, with the next row receiving a rank which skips the duplicate rankings.
- DENSE_RANK(): Similar to RANK(), but does not skip rankings if there are duplicates.

Value Functions:

- LEAD(): Returns the value of the next row.
- LAG(): Returns the value of the previous row
- FIRST_VALUE(): Returns the first value in the window
- LAST_VALUE(): Returns the last value in the window.

Create sample table to understand Aggregate Window Functions

```
CREATE TABLE sales (
    id INT,
    employee VARCHAR(50),
    department VARCHAR(50),
    sales_amount INT
);

INSERT INTO sales VALUES
(1, 'Alice', 'Electronics', 500),
(2, 'Bob', 'Electronics', 800),
(3, 'Charlie', 'Electronics', 800),
(4, 'David', 'Grocery', 400),
(5, 'Eva', 'Grocery', 300),
(6, 'Frank', 'Clothing', 300),
(7, 'Grace', 'Clothing', 700);

SELECT * FROM sales;
```

 # Aggregate Functions

```
SELECT employee, department, sales_amount,
    SUM(sales_amount)   OVER (PARTITION BY department ORDER BY department) AS dept_sum,
    AVG(sales_amount)   OVER (PARTITION BY department ORDER BY department) AS dept_avg,
    COUNT(*)            OVER (PARTITION BY department ORDER BY department) AS dept_count,
    MIN(sales_amount)   OVER (PARTITION BY department ORDER BY department) AS dept_min,
    MAX(sales_amount)   OVER (PARTITION BY department ORDER BY department) AS dept_max
FROM sales;
```

/* Note: You can use a single window function or combine multiple window functions in your query, depending on your specific requirement. */

-- single Aggregate Functions

```
SELECT employee, department, sales_amount,
    SUM(sales_amount)   OVER (PARTITION BY department ORDER BY department) AS dept_sum
FROM sales;
```

Create sample table to understand Ranking Window Functions

```
CREATE TABLE shop_sales (
    sales_date DATE,
    shop_id VARCHAR(5),
    sales_amount DECIMAL(10,2)
);

INSERT INTO shop_sales VALUES
('2024-01-01', 'S001', 400.00),
('2024-01-02', 'S001', 400.00),
('2024-01-03', 'S001', 300.00),
('2024-01-04', 'S001', 200.00),
('2024-01-05', 'S002', 600.00),
('2024-01-06', 'S002', 600.00),
('2024-01-07', 'S003', 500.00),
('2024-01-07', 'S003', 500.00),
('2024-01-07', 'S003', 300.00);

SELECT * FROM shop_sales;
```

 # Ranking Functions:

```
-- Without partition by
SELECT *,
    ROW_NUMBER() OVER(ORDER BY sales_amount) AS row_num,
    RANK() OVER(ORDER BY sales_amount) AS rank_val,
    DENSE_RANK() OVER(ORDER BY sales_amount) AS dense_rank_val
FROM shop_sales;

-- With partition by
SELECT *,
    ROW_NUMBER() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS row_num,
    RANK() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS rank_val,
    DENSE_RANK() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS dense_rank_val
FROM shop_sales;
```

/* Note: You can use a single window function or combine multiple window functions in your query, depending on your specific requirement. */

```
-- Row_Number()
SELECT *,
    ROW_NUMBER() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS row_num
FROM shop_sales;

-- Rank()
SELECT *,
    RANK() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS rank_val
FROM shop_sales;

-- Dense_Rank()
SELECT *,
    DENSE_RANK() OVER(PARTITION BY shop_id ORDER BY sales_amount DESC) AS dense_rank_val
FROM shop_sales;
```

 # Value Functions:

-- Without partition by

```
SELECT employee, department, sales_amount,
    LAG(sales_amount) OVER (ORDER BY sales_amount) AS prev_sales,
          LEAD(sales_amount) OVER (ORDER BY sales_amount) AS next_sales,
          FIRST_VALUE(sales_amount) OVER (ORDER BY sales_amount) AS first_sales,
          LAST_VALUE(sales_amount) OVER (ORDER BY sales_amount
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_sales
FROM sales;
```

-- With partition by

```
SELECT employee, department, sales_amount,
    LAG(sales_amount, 1) OVER (PARTITION BY department ORDER BY sales_amount) AS prev_sales,
          LEAD(sales_amount, 1) OVER (PARTITION BY department ORDER BY sales_amount) AS next_sales,
          FIRST_VALUE(sales_amount) OVER (PARTITION BY department ORDER BY sales_amount) AS first_sales,
          LAST_VALUE(sales_amount) OVER (PARTITION BY department ORDER BY sales_amount
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_sales
FROM sales;
```

 # Frame Clause

/* - The frame start can be:
- UNBOUNDED PRECEDING (starts at the first row of the partition)
- N PRECEDING (starts N rows before the current row)
- CURRENT ROW (starts at the current row)

- The frame end can be:
- UNBOUNDED FOLLOWING (ends at the last row of the partition)
- N FOLLOWING (ends N rows after the current row)
- CURRENT ROW (ends at the current row) */