Data Analytics

–

window functions

gamma
tech school

1. ¿Qué es una función ventana?

- Permite particionar la información de una tabla.
- Se basará en la fila actual para aplicar la función requerida en base a la partición indicada.
- Facilitan consultas que requieren de subconsultas complejas.
- Nos ahorran el uso de tablas temporales.
- **ALERTA**: a pesar de parecerse, no confundir con la cláusula GROUP BY.

```
SELECT

    …,

    funcion OVER()

FROM tabla;
```

2. Estructura

- Para definir una función ventana se necesita la cláusula **OVER()** a continuación de la función que se quiere aplicar a esa ventana.
- Dentro de los paréntesis de la cláusula **OVER()** se puede indicar cómo particionar los datos para crear cada ventana y estipular su orden:
  - PARTITION BY: según los datos de qué columna se va a hacer cada partición.
  - ORDER BY: el orden en el que van a organizarse los datos para aplicar la función.

```
SELECT

    …,

    funcion OVER (

        PARTITION BY columnaX

        ORDER BY columnaY)

FROM …;
```

* La función ventana se escribe en una misma línea de código, a no ser que por su extensión necesite dividirse en varias líneas.

3. Sintaxis de la función ventana

Funciones de agregación

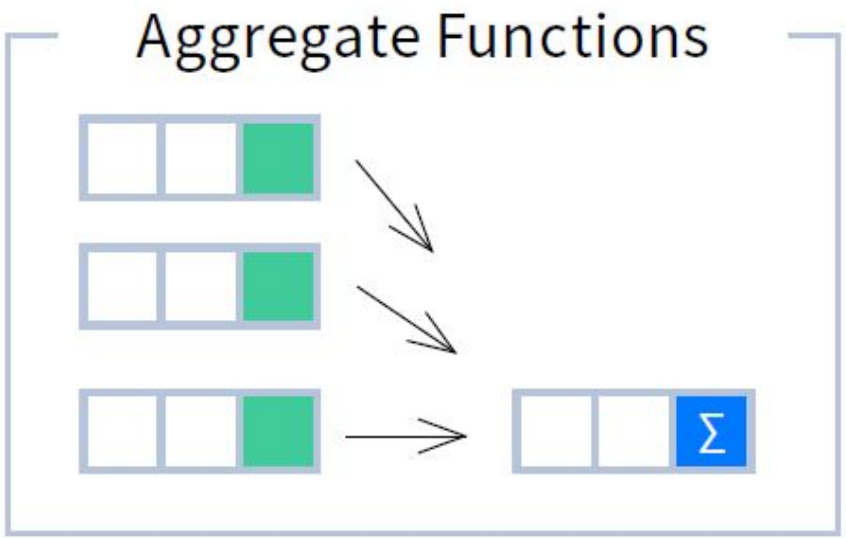
- SUM()
- MIN()
- MAX()
- COUNT()
- AVG()

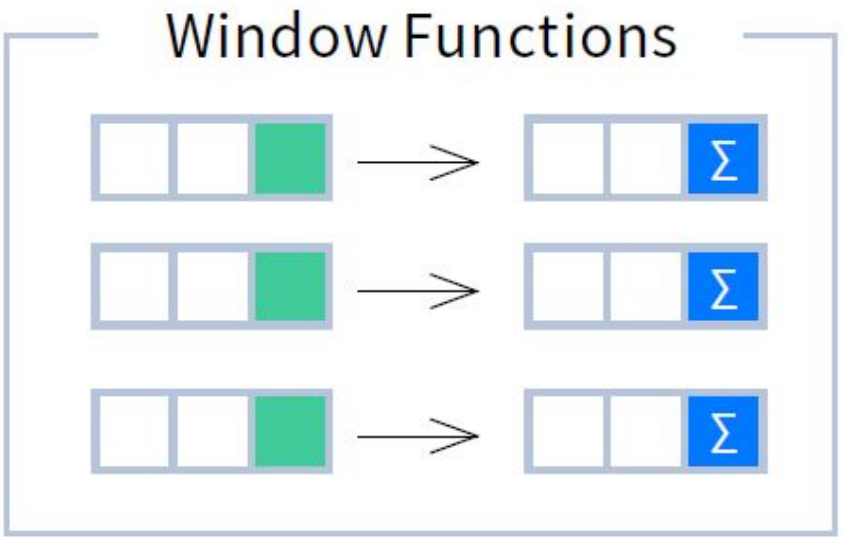
Además de cualquier tipo de operación necesaria.

Principales Funciones ventana

- ROW_NUMBER(): añade una columna numerando cada fila.
- RANK(): genera un ranking, teniendo en cuenta resultados con empate, pero manteniendo la cronología de la numeración.
- DENSE_RANK(): genera un ranking, teniendo en cuenta resultados con empate sin mantener la cronología de la numeración.
- LAG(): genera una columna con el dato de la fila anterior.
- LEAD(): genera una columna con el dato de la fila siguiente.
- ...

4. Función ventana VS group by





5. PARTITION BY

month	city	sold
1	Rome	200
2	Paris	500
1	London	100
1	Paris	300
2	Rome	300
2	London	400
3	Rome	400

PARTITION BY city

month	city	sold	sum
1	Paris	300	800
2	Paris	500	800
1	Rome	200	900
2	Rome	300	900
3	Rome	400	900
1	London	100	500
2	London	400	500

6. PARTITION BY <...> ORDER BY <...>

sold	city	month
200	Rome	1
500	Paris	2
100	London	1
300	Paris	1
300	Rome	2
400	London	2
400	Rome	3

```
PARTITION BY city ORDER BY month
```

sold	city	month
300	Paris	1
500	Paris	2
200	Rome	1
300	Rome	2
400	Rome	3
100	London	1
400	London	2

7. Rankings

city	price	row_number	rank	dense_rank
		over(order by price)		
Paris	7	1	1	1
Rome	7	2	1	1
London	8.5	3	3	2
Berlin	8.5	4	3	2
Moscow	9	5	5	3
Madrid	10	6	6	4
Oslo	10	7	6	4

7. Ranking

- Facturación total de cada cliente y el total para calcular el porcentaje sobre la facturación total.

```
SELECT
      cus.customer_id
    , cus.first_name
    , cus.last_name
    , SUM(p.amount) customer_amount
    , SUM(SUM(p.amount)) OVER() total_amount
    , SUM(p.amount)/SUM(SUM(p.amount)) OVER()*100
percentatge_of_total
FROM customer cus
JOIN payment p
      ON cus.customer_id = p.customer_id
GROUP BY 1, 2, 3
ORDER BY 6 DESC;
```

8. LAG() / LEAD()

`lag(sold) OVER(ORDER BY month)`

order by month ↓

month	sold	
1	500	NULL
2	300	500
3	400	300
4	100	400
5	500	100

`lead(sold) OVER(ORDER BY month)`

order by month ↓

month	sold	
1	500	300
2	300	400
3	400	100
4	100	500
5	500	NULL

`lag(sold, 2, 0) OVER(ORDER BY month)`

order by month ↓

month	sold	
1	500	0
2	300	0
3	400	500
4	100	300
5	500	400

offset=2 ↓

`lead(sold, 2, 0) OVER(ORDER BY month)`

order by month ↓

month	sold	
1	500	400
2	300	100
3	400	500
4	100	0
5	500	0

offset=2 ↑