📌 Window Functions en SQL

Las **Window Functions** en SQL permiten realizar cálculos sobre un conjunto de filas **sin agrupar los datos**. A diferencia de GROUP BY, estas funciones **no colapsan filas**, sino que generan valores adicionales para cada fila dentro de una ventana definida.

Se usan para **ranking, acumulados, promedios móviles y cálculos sobre subconjuntos de datos**.

🔹 Sintaxis general

```
FUNCIÓN_WINDOW() OVER (
    PARTITION BY columna  -- (Opcional) Agrupa en subconjuntos
    ORDER BY columna  -- (Opcional) Define el orden de la ventana
)
```

📌 Claves importantes:

- **PARTITION BY**: Divide los datos en grupos (opcional).
- **ORDER BY**: Define el orden dentro de la ventana.

🔹 Tipos de Window Functions en SQL

1. Ranking Functions (Orden y posiciones)

🔹 RANK() - Asigna un ranking con saltos

```
SELECT Nombre, Salario,
       RANK() OVER (ORDER BY Salario DESC) AS Ranking
FROM Empleados;
```

📌 Si hay empates en el salario, deja huecos en los rankings.

🔹 DENSE_RANK() - Ranking sin huecos

```
SELECT Nombre, Salario,
       DENSE_RANK() OVER (ORDER BY Salario DESC) AS Ranking
FROM Empleados;
```

📌 Igual que RANK(), pero sin saltos en la numeración.

🔹 **ROW_NUMBER() - Numera filas de forma única**

```sql
SELECT Nombre, Salario,
       ROW_NUMBER() OVER (ORDER BY Salario DESC) AS Fila
FROM Empleados;
```

📌 **Cada fila tiene un número único, sin importar empates.**

---

2. Aggregate Functions (Suma, promedio, acumulados)

🔹 **SUM() - Suma acumulada**

```sql
SELECT Nombre, Departamento, Salario,
       SUM(Salario) OVER (PARTITION BY Departamento ORDER BY Salario) AS Salario_Acumulado
FROM Empleados;
```

📌 **Calcula una suma acumulativa por departamento.**

🔹 **AVG() - Promedio dentro de una ventana**

```sql
SELECT Nombre, Departamento, Salario,
       AVG(Salario) OVER (PARTITION BY Departamento) AS Promedio_Departamento
FROM Empleados;
```

📌 **Calcula el salario promedio por departamento sin agrupar.**

🔹 **LAG() - Valor anterior**

```sql
SELECT Nombre, Salario,
       LAG(Salario) OVER (ORDER BY Salario DESC) AS Salario_Anterior
FROM Empleados;
```

📌 **Muestra el salario de la fila anterior.**

🔹 **LEAD() - Valor siguiente**

```sql
SELECT Nombre, Salario,
       LEAD(Salario) OVER (ORDER BY Salario DESC) AS Salario_Siguiente
FROM Empleados;
```

📌 **Muestra el salario de la fila siguiente.**

💡 **Proyecto práctico: Ranking de ventas por vendedor**

📌 **Escenario**

Una empresa quiere analizar las **ventas de sus vendedores** y determinar:

1. **El ranking de cada vendedor según su monto total de ventas.**
2. **El total acumulado de ventas por vendedor.**
3. **Comparar la venta actual con la anterior (diferencia de ventas).**

📌 **Tablas**

Ventas

VentaID	Vendedor	Monto	Fecha
1	Ana	500	2025-01-01
2	Luis	800	2025-01-02
3	Marta	1000	2025-01-03
4	Ana	700	2025-01-04
5	Luis	1200	2025-01-05

📌 **Objetivo**

1. **Obtener un ranking de ventas por vendedor**
2. **Calcular el total acumulado de ventas por vendedor**
3. **Comparar cada venta con la venta anterior**

📌 **Pista:** Usa RANK(), SUM() y LAG().

🚀 **Intenta escribir la consulta que genere el siguiente resultado:**

Vendedor	Monto	Ranking	Total_Acumulado	Diferencia_Anterior
Ana	500	2	500	NULL
Ana	700	1	1200	200
Luis	800	2	800	NULL
Luis	1200	1	2000	400
Marta	1000	1	1000	NULL

📌 **Reto**

✍️ **Escribe tu consulta SQL para resolverlo**.
Cuando la tengas, dime si necesitas ayuda o más desafíos 🚀