Recomendaciones finales

1. Experimenta variando ORDER BY y PARTITION BY para ver cómo cambian los resultados.

2. Prueba RANGE en lugar de ROWS en tu frame clause, sobre todo si hay empates en la columna de orden.

3. Combina las funciones de ventana con WHERE, JOIN, CTE, etc., para producir reportes más complejos.

Ejercicio 1: Enumerar empleados según su salario (ROW_NUMBER)

Objetivo: Asignar un número de fila a cada empleado, ordenado por Salario de forma descendente (el que más gana será el ROW_NUMBER 1).

1. Análisis

    - Queremos ver cada empleado con su salario y un número secuencial.

    - ROW_NUMBER() requiere usar OVER(ORDER BY <columna>).

Ejercicio 2: Sumar salarios dentro de cada departamento (función de agregación con OVER)

Objetivo: Mostrar el salario de cada empleado y la suma total de salarios de su departamento. Sin usar GROUP BY (para no colapsar filas).

1. Análisis

    - Con un GROUP BY DepartamentoID, obtendríamos una sola fila por departamento.

    - Con funciones de ventana, podemos seguir viendo cada empleado y además mostrar la suma agregada del departamento.

Ejercicio 3: Calcular total de ventas por empleado y enumerar filas (ROW_NUMBER con PARTITION BY)

Objetivo: Para cada empleado, calcular su total de unidades vendidas (sumando Cantidad), y asignar un número de fila para saber qué empleado está en primer lugar, segundo, etc., ordenado de mayor a menor cantidad.
(Esta vez haremos una subconsulta o CTE para obtener la suma, y luego aplicaremos la función de ventana)

1. Análisis
    - Primero, debemos saber cuántas unidades vendió cada empleado (SUM(Cantidad) en la tabla Ventas).
    - Luego, asignar un ROW_NUMBER() ordenado por ese total (descendente).

Ejercicio 4: Ranking de empleados por monto de ventas (RANK vs. DENSE_RANK)

Objetivo: Ordenar empleados según el monto total vendido (SUM(Cantidad * PrecioUnitario)), pero usando RANK() y DENSE_RANK() para ver cómo se tratan los empates.

1. Análisis
    - Queremos ver si existen empates en el monto total.
    - RANK() saltará números cuando haya empates, DENSE_RANK() no.

Ejercicio 5: Comparar la venta de cada registro con la venta anterior (LAG)

Objetivo: Para cada venta (cada fila en Ventas), mostrar el monto (Cantidad * PrecioUnitario) y el monto de la venta anterior (ordenado por FechaVenta).

1. Análisis

- LAG(expresion, [offset], [default]) devuelve el valor de una fila anterior dentro de la partición/orden definido.

- Si no hay fila anterior (p.ej. la primera venta), se puede especificar un valor por defecto.

Ejercicio 6: Total acumulado de ventas por fecha (running total con window frame)

Objetivo: Mostrar, para cada fecha, la suma acumulada de ventas en todo el periodo. Se asume la tabla Ventas registra cada transacción con su FechaVenta, Cantidad, y PrecioUnitario.

1. Análisis

    - Un “running total” (total acumulado) requiere definir un OVER(ORDER BY ...) con un frame que vaya desde el principio (UNBOUNDED PRECEDING) hasta la fila actual (CURRENT ROW).

Ejercicio 7 (avanzado): Ranking por mes y categoría de producto (NTILE)

Objetivo: Para cada mes y categoría de producto, asignar a cada producto un “cuartil” (usando NTILE(4)) según el monto total vendido en ese mes-categoría.

1. Análisis

    - Queremos particionar los datos por mes y categoría (tomados de Productos.CategoriaID).

    - Dentro de cada partición (mes + categoría), ordenamos por el total de ventas (monto).

    - NTILE(4) divide esas filas en 4 grupos aproximadamente iguales (cuartiles).
