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>).

Consulta

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

2. Explicación
    - OVER (ORDER BY Salario DESC) indica que, para asignar números de fila, se ordenan primero los empleados por Salario (de mayor a menor).
    - ROW_NUMBER() asigna 1 al primero, 2 al segundo, etc.
    - No hay PARTITION BY, así que se considera todo el conjunto de filas una sola “ventana”.

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.

Consulta

```
SELECT
  E.EmpleadoID,  E.Nombre,  E.DepartamentoID, E.Salario,
  SUM(E.Salario) OVER (
    PARTITION BY E.DepartamentoID
  ) AS SumaSalariosDepto
FROM Empleados E;
```

2. Explicación
    - PARTITION BY E.DepartamentoID significa que se agrupa (particiona) por departamento.
    - SUM(E.Salario) se calcula dentro de cada departamento, pero sin perder el detalle de cada fila.
    - Cada fila muestra el salario individual y la suma total de su 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).

Consulta con subconsulta derivada

```
WITH VentasPorEmpleado AS (
    SELECT
        V.EmpleadoID,
        SUM(V.Cantidad) AS TotalUnidades
    FROM Ventas V
    GROUP BY V.EmpleadoID
)
SELECT
    E.EmpleadoID,
    E.Nombre,
    VPE.TotalUnidades,
    ROW_NUMBER() OVER (
        ORDER BY VPE.TotalUnidades DESC
```

```
) AS RankingUnidades

FROM Empleados E

JOIN VentasPorEmpleado VPE ON E.EmpleadoID = VPE.EmpleadoID;
```

2. Explicación

    - El CTE VentasPorEmpleado agrupa por empleado y obtiene SUM(Cantidad).
    - En la consulta principal, hacemos un JOIN con Empleados para poder mostrar el Nombre junto al TotalUnidades.
    - ROW_NUMBER() OVER (ORDER BY VPE.TotalUnidades DESC) asigna 1 al empleado con más unidades vendidas, 2 al siguiente, etc.

(Si quisieras hacerlo directamente en una sola consulta, podrías usar SUM(V.Cantidad) OVER (...), pero es más claro el ejemplo con un CTE para ver el paso a paso.)

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.

Consulta

```
WITH MontosPorEmpleado AS (
    SELECT
        E.EmpleadoID,
        E.Nombre,
        SUM(V.Cantidad * V.PrecioUnitario) AS MontoTotal
    FROM Empleados E
    JOIN Ventas V ON E.EmpleadoID = V.EmpleadoID
    GROUP BY E.EmpleadoID, E.Nombre
)
SELECT
    EmpleadoID,
    Nombre,
    MontoTotal,
    RANK() OVER (ORDER BY MontoTotal DESC) AS RankVentas,
    DENSE_RANK() OVER (ORDER BY MontoTotal DESC) AS DenseRankVentas
FROM MontosPorEmpleado;
```

2. Explicación

    - Paso 1: Con el CTE MontosPorEmpleado, sumamos (Cantidad * PrecioUnitario) para cada empleado.

    - Paso 2: RANK() OVER (ORDER BY MontoTotal DESC) asigna rangos. Si dos empleados empatan en 1er lugar, ambos tienen rango 1 y el siguiente tiene rango 3.

- Paso 3: DENSE_RANK() OVER (ORDER BY MontoTotal DESC) no salta el rango tras un empate (si dos empatan en 1, el siguiente es 2, no 3).

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.

Consulta

```
SELECT
  VentaID,  EmpleadoID,  ProductoID,
  (Cantidad * PrecioUnitario) AS MontoActual, LAG((Cantidad * PrecioUnitario), 1, 0)
    OVER (ORDER BY FechaVenta) AS MontoAnterior
FROM Ventas
ORDER BY FechaVenta;
```

2. Explicación
    - ORDER BY FechaVenta dentro del OVER indica el orden cronológico.
    - Para cada fila, LAG(..., 1, 0) toma el Monto de la fila anterior (1 fila atrás).

- Si es la primera venta y no hay “anterior”, devuelve 0.

(Si quisieras comparar por empleado, usarías PARTITION BY EmpleadoID ORDER BY FechaVenta. Cada empleado tendría su secuencia independiente.)

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).

Consulta

```
SELECT
  FechaVenta,
  SUM(Cantidad * PrecioUnitario) AS MontoDelDia,
  SUM(SUM(Cantidad * PrecioUnitario)) OVER (
    ORDER BY FechaVenta
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS MontoAcumulado
FROM Ventas
GROUP BY FechaVenta
ORDER BY FechaVenta;
```

2. Explicación
    - Primero, agrupamos por FechaVenta para obtener MontoDelDia.
    - En la misma fila, aplicamos SUM(...) OVER(...) con un frame:
        - ORDER BY FechaVenta define el orden cronológico de días.
        - ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW indica que se acumulan todas las filas desde la primera fecha hasta la fecha actual.
    - Así obtenemos un acumulado progresivo día a día.

*(Algunos SGBD permiten hacer esto directamente sin agrupar primero, usando SUM(Cantidad * PrecioUnitario) OVER (PARTITION BY … ORDER BY …), pero la agrupación previa hace más explícito el "monto por día".)*

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).

2. Pasos

    - Necesitamos la tabla unida: Ventas + Productos (para saber la categoría) y agrupar o sumar.

    - Definir MONTH(FechaVenta) como “MesVenta”.

Consulta (posible con un CTE)

```
WITH VentasMesCat AS (
   SELECT
      MONTH(V.FechaVenta) AS MesVenta,
      P.CategoriaID,
      V.ProductoID,
      SUM(V.Cantidad * V.PrecioUnitario) AS TotalCatMes
   FROM Ventas V
   JOIN Productos P ON V.ProductoID = P.ProductoID
   GROUP BY MONTH(V.FechaVenta), P.CategoriaID, V.ProductoID
)
SELECT
   MesVenta,
   CategoriaID,
   ProductoID,
   TotalCatMes,
   NTILE(4) OVER (
      PARTITION BY MesVenta, CategoriaID
      ORDER BY TotalCatMes DESC
```

) AS Cuartil

FROM VentasMesCat

ORDER BY MesVenta, CategoriaID, TotalCatMes DESC;

3. Explicación

    - El CTE VentasMesCat consolida la venta total (TotalCatMes) por MesVenta y CategoriaID.

    - En la consulta principal, NTILE(4) crea 4 grupos dentro de cada partición (MesVenta, CategoriaID), ordenados por TotalCatMes DESC.

    - Cuartil 1 tendrá los productos más vendidos en ese mes/categoría, hasta Cuartil 4 con los menos vendidos.