¿Qué son las funciones de ventana?

Las funciones de ventana (Window Functions) permiten efectuar cálculos o operaciones sobre un conjunto de filas (una "ventana") sin colapsar esas filas en una sola. Es decir, a diferencia de las funciones de agregación tradicionales con GROUP BY (que devuelven una fila por grupo), las funciones de ventana:

1. Mantienen cada fila del resultado.

2. Agregan o adjuntan valores calculados para cada fila, basados en otras filas relacionadas (definidas por la "ventana").

De esta forma, puedes:

- Calcular totales o promedios acumulados (por ejemplo, SUM(col) OVER (...)).

- Asignar números de fila o rankings (ROW_NUMBER(), RANK(), etc.).

- Consultar valores de filas anteriores o posteriores (LAG(), LEAD()).

Todo ello sin perder el detalle de cada fila individual.

¿Por qué son útiles?

1. Flexibilidad: Con un GROUP BY tradicional, si calculas SUM o COUNT, desaparecen las filas individuales y solo obtienes una fila por grupo. Con las funciones de ventana, sigues viendo todas las filas, pero además tienes un valor agregado.

2. Análisis avanzado: Permiten crear reportes que muestren, por ejemplo, la venta acumulada día a día, o el promedio de ventas de un mes junto al detalle de cada día, etc.

3. Ranking y comparaciones: Cosas como "el top 5 de ventas por región" o "qué porcentaje representa cada fila respecto al total" se hacen mucho más simples

con funciones de ventana que con subconsultas complicadas.

Estructura o sintaxis básica

Una función de ventana generalmente se define así:

```
<funcion_de_ventana>() OVER (
   [PARTITION BY <columna(s)>]
   [ORDER BY <columna(s)>]
   [frame_clause_opcional]
)
```

Donde:

1. <funcion_de_ventana>() puede ser:
    - Funciones de ranking:
      ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE(n).
    - Funciones de agregación en modo ventana:
      SUM(col), AVG(col), MIN(col), MAX(col), COUNT(col), etc.
    - Funciones de desplazamiento (offset):
      LAG(col, offset), LEAD(col, offset), FIRST_VALUE(col), LAST_VALUE(col).
    - Funciones de distribución:
      PERCENT_RANK(), CUME_DIST(), etc.
2. OVER(...) indica que la función se va a comportar como función de ventana, y no como función de agregación normal.
3. PARTITION BY <columna(s)> define cómo se divide (o particiona) el conjunto de resultados en subgrupos. La función se aplicará dentro de cada partición.
    - Si no lo usas, la partición será todo el conjunto de filas.

4. ORDER BY <columna(s)> define el orden de las filas dentro de cada partición.
    - Es fundamental en funciones de ranking y desplazamiento.
    - Para funciones de agregación en ventana (como SUM), al incluir ORDER BY, a menudo se activa la lógica de acumulados (running total).
5. Frame Clause (por ejemplo, ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW):
    - Permite definir exactamente cuáles filas se incluyen en el cálculo para la fila actual.
    - Por defecto, muchas bases de datos asumen RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW para ciertas funciones (como SUM OVER(ORDER BY ...)), lo que se traduce en un acumulado desde la primera fila de la partición hasta la fila actual.

Ejemplos básicos

Para ilustrar, supongamos una tabla Empleados con datos como:

EmpleadoID	Nombre	DepartamentoID	Salario
1	Ana	10	2000.00
2	Carlos	10	2500.00
3	Fernanda	20	1800.00
4	Pedro	20	3000.00
5	Lucía	10	2200.00

Funciones de agregación en ventana: SUM, AVG, etc.

Objetivo: mostrar cada empleado con su salario y la suma total de salarios en su departamento, sin colapsar las filas.

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

Cómo funciona:

- PARTITION BY DepartamentoID: cada departamento es una partición.
- SUM(Salario) se calcula dentro de cada departamento.
- El resultado se repite en cada fila del mismo departamento, pero no desaparecen las filas individuales.

Resultado (ejemplo supuesto):

EmpleadoID	Nombre	DepartamentoID	Salario	SumaSalariosDepto
1	Ana	10	2000.00	6700.00
2	Carlos	10	2500.00	6700.00
5	Lucía	10	2200.00	6700.00
3	Fernanda	20	1800.00	4800.00
4	Pedro	20	3000.00	4800.00

Funciones de ranking: ROW_NUMBER, RANK, DENSE_RANK

Objetivo: asignar un número de fila a cada empleado, ordenado por salario de mayor a menor.

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

- ORDER BY Salario DESC: primero el salario más alto.
- ROW_NUMBER() asigna números consecutivos 1,2,3..., sin importar empates.

Resultado (ejemplo):

EmpleadoID	Nombre	Salario	RowNumDescSal
4	Pedro	3000.00	1
2	Carlos	2500.00	2
5	Lucía	2200.00	3
1	Ana	2000.00	4
3	Fernanda	1800.00	5

RANK y DENSE_RANK

- RANK(): si hay empates, dos filas pueden tener "rango 1", y la siguiente fila salta al rango 3.

- DENSE_RANK(): en caso de empates, también hay varias filas con rango 1, pero la siguiente pasa a rango 2 (no salta).

Ejemplo, si dos empleados tienen salario 2500, con RANK(), la siguiente fila podría ser rango 3; con DENSE_RANK(), la siguiente fila sería rango 2.

Funciones de desplazamiento (LAG, LEAD)

Permiten acceder a valores de filas anteriores o posteriores. Ejemplo:

```sql
SELECT
    EmpleadoID,
    Salario,
    LAG(Salario, 1, 0) OVER (ORDER BY EmpleadoID) AS SalarioAnterior
FROM Empleados
ORDER BY EmpleadoID;
```

- LAG(Salario, 1, 0) dice: "Dame el Salario de la fila anterior" (offset 1).

- Si es la primera fila y no hay fila anterior, devuelve 0 (el valor por defecto).

- Así, en la misma fila ves el salario actual y el anterior.

Ventanas con "frame clause" y acumulados

Si quieres un acumulado (running total) por orden de EmpleadoID o Fecha, podrías hacer:

```sql
SELECT
    EmpleadoID,
    Salario,
    SUM(Salario) OVER (
        ORDER BY EmpleadoID
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS SalarioAcumulado
FROM Empleados
ORDER BY EmpleadoID;
```

- ORDER BY EmpleadoID dentro de OVER define el orden.

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW indica que en la suma se incluyan todas las filas desde el principio de la partición hasta la fila actual.

- Cada fila se mostrará con el total acumulado de salarios hasta esa fila.

¿Dónde aparece esta cláusula?

La sintaxis general de una función de ventana con frame clause puede verse así:

<función_de_ventana>(columna)

OVER (

  [PARTITION BY columnaParticion]

  ORDER BY columnaOrden

  ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

- PARTITION BY (opcional): divide el conjunto de filas en “ventanas” más pequeñas (p.ej. por departamento).

- ORDER BY: define el orden dentro de cada partición.

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW: define el rango de filas que serán incluidas en el cálculo para la fila actual.

Desglose de la cláusula

- ROWS: indica que la base para contar es por fila, no por valor de columna (que se usaría con RANGE).
- BETWEEN UNBOUNDED PRECEDING: significa que, para la fila actual, la ventana inicia en la primera fila de la partición (no hay límite superior de filas anteriores).
- AND CURRENT ROW: la ventana cierra en la fila actual, incluida.

Por tanto, en cada fila, la función de ventana utilizará todas las filas hasta esa fila, en orden, para hacer el cálculo.

Ejemplo práctico: suma acumulada

Imagina una tabla simplificada VentasDiarias:

Fecha	Ventas
2023-01-01	100
2023-01-02	200
2023-01-03	150
2023-01-04	300

Si quieres un acumulado diario de las ventas, podrías usar:

```sql
SELECT
    Fecha,
    Ventas,
    SUM(Ventas) OVER (
        ORDER BY Fecha
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS VentasAcumuladas
FROM VentasDiarias
ORDER BY Fecha;
```

- Explicación fila a fila:

    1. Fila 1 (Fecha = 2023-01-01): La ventana (unbounded preceding → current row) incluye la fila 1. Suma = 100.

    2. Fila 2 (Fecha = 2023-01-02): Incluye filas 1 y 2, la suma = 100 + 200 = 300.

3. Fila 3 (Fecha = 2023-01-03): Incluye filas 1, 2 y 3; la suma = 100 + 200 + 150 = 450.

4. Fila 4 (Fecha = 2023-01-04): Incluye filas 1, 2, 3 y 4; la suma = 100 + 200 + 150 + 300 = 750.

Resultado:

Fecha	Ventas	VentasAcumuladas
2023-01-01	100	100
2023-01-02	200	300
2023-01-03	150	450
2023-01-04	300	750

Diferencia entre ROWS y RANGE

- ROWS BETWEEN ...: se basa estrictamente en la posición física de las filas (row-by-row).

- RANGE BETWEEN ...: se basa en valores de las columnas del ORDER BY. Por ejemplo, si hay filas con el mismo valor en la columna usada para ordenar, RANGE las trata como un “grupo” y puede producir diferencias en cómo se incluyen esas filas.

En muchos casos, ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW y RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW pueden dar resultados equivalentes, pero no siempre (especialmente si hay empates en el ORDER BY).

Usos típicos

1. Cálculo de acumulados: suma progresiva, conteo progresivo, máximo progresivo, etc.

2. Análisis de series temporales: comparar la cifra actual de ventas con la suma o el promedio desde el inicio del periodo.

3. Tablas con secuencia numérica: si tienes un ID incremental, puedes hacer acumulados basados en ese ID.

Comparación con GROUP BY

GROUP BY crea una fila por grupo y es útil si solo quieres resultados agregados.

Ejemplo:

```sql
SELECT DepartamentoID, COUNT(*) AS NumEmpleados
FROM Empleados
GROUP BY DepartamentoID;
```

- Esto produce una fila por cada DepartamentoID.

Funciones de ventana mantienen cada fila. Por ejemplo:

```sql
SELECT
    EmpleadoID,
    DepartamentoID,
    COUNT(*) OVER (PARTITION BY DepartamentoID) AS NumEmpleadosEnDepto
FROM Empleados;
```

Tendrás todas las filas (empleados), y en cada una se muestra cuántos empleados hay en su departamento, sin perder el detalle de cada empleado.

Orden lógico y performance

1. Las funciones de ventana se calculan después de FROM/WHERE/GROUP BY/HAVING pero antes de ORDER BY final de la consulta.

2. Performance:

    - A veces, puede ser más pesado si ORDER BY en la ventana no tiene índices.

    - Se recomienda verificar planes de ejecución y, si es posible, incluir índices en columnas muy usadas en PARTITION BY o ORDER BY.