Dateneo

Guía definitiva

Funciones Ventana en SQL

Dateneo

Funciones ventana

📌 ¿Qué son?

- Funciones que permiten realizar cálculos sobre un conjunto de filas sin agrupar los datos.
- Después de aplicarlas, obtenemos **la misma cantidad de filas** que en la entrada.

🤔 Para qué usarlas

✅ Ranking y ordenamiento
✅ Acumulados y promedios móviles
✅ Comparación entre filas

Sintaxis general

```
SELECT <columna_1>, <columna_2>,
  <función_ventana> OVER (
    PARTITION BY <...>
    ORDER BY <...>
    <ventana_deslizante>) <alias_columna_ventana>
FROM <nombre_tabla>;
```

Dateneo

📌 Elementos principales

✅ **OVER**() → Define el contexto de la función ventana.

✅ **PARTITION BY** → Agrupa las filas antes de aplicar la función (opcional).

✅ **ORDER BY** → Define el orden dentro de cada grupo.

✅ **Ventana deslizante** → Controla cuántas filas se consideran en la operación.

💡 Apliquemos todo con ejemplos

Tabla base

Comenzamos con una tabla de transacciones como la siguiente, donde registramos fecha, usuario, categoría y monto de venta.

123 id	A-Z usuario	fecha	123 ventas	A-Z categoria
1	A	2024-01-01 00:00:00.000	100	Electrónica
2	A	2024-01-02 00:00:00.000	200	Electrónica
3	A	2024-01-03 00:00:00.000	150	Ropa
4	A	2024-01-05 00:00:00.000	300	Hogar
5	A	2024-01-06 00:00:00.000	250	Electrónica
6	B	2024-01-01 00:00:00.000	80	Hogar

Dateneo

Caso más simple

Sin uso de PARTITION ni ORDER

➡ Incluir una columna que muestre **total de ventas** en toda la tabla.

```
SELECT * ,
  SUM(ventas) OVER () AS total_ventas
FROM transacciones
```

123 id	A-Z usuario	fecha	123 ventas	A-Z categoria	123 total_ventas
1	A	2024-01-01 00:00:00.000	100	Electrónica	7.170
2	A	2024-01-02 00:00:00.000	200	Electrónica	7.170
3	A	2024-01-03 00:00:00.000	150	Ropa	7.170
4	A	2024-01-05 00:00:00.000	300	Hogar	7.170
5	A	2024-01-06 00:00:00.000	250	Electrónica	7.170
6	B	2024-01-01 00:00:00.000	80	Hogar	7.170

☝ Utiliza la función **SUM** pero no consolida los datos, no confundir con SUM + GROUP BY.

Dateneo

Casos más interesantes 🤩

Uso de PARTITION BY

➡ Total de **ventas por usuario**.

```
SELECT usuario, fecha, ventas,
    SUM(ventas) OVER (PARTITION BY usuario) AS
total_ventas_usuario
FROM transacciones
```

usuario	fecha	ventas	total_ventas_usuario
A	2024-01-01 00:00:00.000	100	1.000
A	2024-01-02 00:00:00.000	200	1.000
A	2024-01-03 00:00:00.000	150	1.000
A	2024-01-05 00:00:00.000	300	1.000
A	2024-01-06 00:00:00.000	250	1.000
B	2024-01-01 00:00:00.000	80	1.050

Total usuario A

☝ El **PARTITION** “resetea” el cálculo, genera grupos por cada valor que exista en la(s) columna(s) de partición.

Dateneo

Casos más interesantes 🤩

Uso de PARTITION BY con ORDER BY

➡ Total **acumulado** de **ventas por usuario por fecha**.

```
SELECT usuario, fecha, ventas,
  SUM(ventas)
  OVER (PARTITION BY usuario ORDER BY fecha) AS
acumulado_ventas
FROM transacciones
```

A-Z usuario	fecha	123 ventas	123 acumulado_ventas
A	2024-01-01 00:00:00.000	100	100
A	2024-01-02 00:00:00.000	200	300
A	2024-01-03 00:00:00.000	150	450
A	2024-01-05 00:00:00.000	300	750
A	2024-01-06 00:00:00.000	250	1.000
B	2024-01-01 00:00:00.000	80	80

+ 100 =

+ 100 + 200 =

Total usuario A

Comienza de nuevo para el usuario B

☝🏻 La inclusión del **ORDER BY** hace que la suma (o la función que usemos) se calcule fila por fila según el criterio de ordenamiento.

🔢 En este caso al total de ventas del día (la fila donde estamos parados) se le suman las ventas de las fechas anteriores (esto por el criterio del ORDER BY).

Dateneo

Casos más interesantes 🤩

Usemos otra función, LAG

➡ Para cada venta, ¿cuál fue el monto de la venta anterior del mismo usuario y cuántos días transcurrieron entre cada transacción?

```
SELECT usuario, fecha, ventas,
LAG(ventas) OVER (PARTITION BY usuario ORDER BY fecha)
AS monto_anterior,
fecha - LAG(fecha) OVER (PARTITION BY usuario ORDER BY
fecha) AS dias_desde_ultima_compra
FROM transacciones
```

usuario	fecha	ventas	monto_anterior	dias_desde_ultima_compra
A	2024-01-01 00:00:00.000	100	[NULL]	[NULL]
A	2024-01-02 00:00:00.000	200	100	1
A	2024-01-03 00:00:00.000	150	200	1
A	2024-01-05 00:00:00.000	300	150	2
A	2024-01-06 00:00:00.000	250	300	1
B	2024-01-01 00:00:00.000	80	[NULL]	[NULL]

05/01 - 03/01 (fecha de la venta próxima anterior)

☝ LAG nos posiciona N filas (por defecto es 1) hacia atrás, según el criterio de ordenamiento.

✅ Podemos utilizar cualquier columna, en este caso fecha y ventas.

💡 El resultado de una función ventana podemos utilizarlo en otra operación, en este caso para calcular dias_desde_ultima_compra.

Dateneo

Casos más avanzados 🧐

Calculo de cuantiles con NTILE

➡ Queremos segmentar los montos de venta en 3 niveles (bajo, medio y alto) y luego saber para cada venta en qué segmento cae.

```
SELECT usuario, fecha, ventas,
NTILE(3) OVER (ORDER BY ventas) AS nivel_gasto
FROM transacciones
```

usuario	fecha	ventas	nivel_gasto
A	2024-01-01 00:00:00.000	100	1
A	2024-01-02 00:00:00.000	200	2
A	2024-01-03 00:00:00.000	150	1
A	2024-01-05 00:00:00.000	300	2
A	2024-01-06 00:00:00.000	250	2
B	2024-01-01 00:00:00.000	80	1
B	2024-01-03 00:00:00.000	220	2
B	2024-01-04 00:00:00.000	300	2
B	2024-01-05 00:00:00.000	50	1
B	2024-01-06 00:00:00.000	400	3

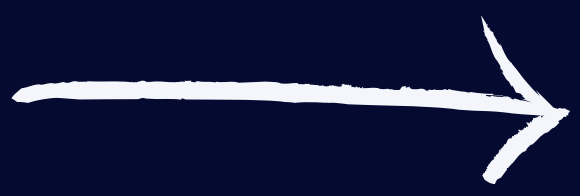
Gasto **bajo**

Gasto **medio**

Gasto **alto**

☝ NTILE divide un conjunto de filas en n grupos de tamaño l**o más uniforme posible**. Cada fila recibe un número de 1 a n según el orden definido en ORDER BY.

❗ Es una aproximación al concepto de **cuantiles** en estadística pero no garantiza que cada grupo tenga exactamente la misma cantidad de elementos.



Dateneo

Casos más avanzados 🧐

Juguemos con la ventana deslizante (time frame)

💡 Es posible definir el rango de filas dentro de una partición sobre el que se realiza el cálculo en funciones ventana.

📌 Podemos definir tres cosas:

1️⃣ El criterio de la ventana:
- 🔹 ROWS → Usa una cantidad fija de filas.
- 🔹RANGE → Usa un rango basado en valores, como fechas.

2️⃣ El límite inferior (inicio del rango):
- ⏪ UNBOUNDED PRECEDING → Incluye todas las filas previas.
- 🔹 CURRENT ROW → Comienza desde la fila actual.
- 🔢 n PRECEDING → Considera las últimas n filas anteriores.

3️⃣ El límite superior (fin del rango):
- ⏩ UNBOUNDED FOLLOWING → Incluye todas las filas siguientes.
- 🔹 CURRENT ROW → Termina en la fila actual.
- 🔢 n FOLLOWING → Considera las próximas n filas.

📌 Si no especificamos nada, se usa la ventana por defecto:
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
✔️ Esto significa que la función operará sobre todas las filas anteriores y la fila actual dentro de la partición.

Dateneo

Casos más avanzados 🧐

Es complejo, veamos un ejemplo

➡ ¿Cuál es el total de ventas en los **últimos 7 días** por usuario?

```
SELECT usuario, fecha, ventas,
SUM(ventas) OVER (PARTITION BY usuario ORDER BY fecha
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
) AS acumulado_7_dias
FROM transacciones
```

usuario	fecha	ventas	acumulado_7_dias
E	2024-01-01 00:00:00.000	90	90
E	2024-01-04 00:00:00.000	180	270
E	2024-01-06 00:00:00.000	250	520
E	2024-01-08 00:00:00.000	400	920
E	2024-01-10 00:00:00.000	800	1.720

☝ En este caso utilizamos ROWS con 6 como límite inferior: queremos que sume comenzando desde 6 filas hacia atrás hasta la actual inclusive.

📅 Como el criterio de ordenación es fecha, esto no necesariamente significa 7 días exactos, sino las últimas 7 filas registradas para cada usuario.

❌ Si bien puede ser válido en algún contexto, como en este caso hay saltos de fechas, ROWS no garantiza que estemos sumando exactamente los últimos 7 días.

Dateneo

Casos más avanzados 🧐

Es complejo, veamos un ejemplo

➡ ¿Cuál es el total de ventas en los **últimos 7 días** por usuario?

```
SELECT usuario, fecha, ventas,
SUM(ventas) OVER (PARTITION BY usuario ORDER BY fecha
RANGE BETWEEN INTERVAL '6' DAY PRECEDING AND
CURRENT ROW
) AS acumulado_7_dias
FROM transacciones
```

Este monto ahora no se computa porque la fecha (01/01) no entra dentro del rango de 6 días anteriores al 10/01

usuario	fecha	ventas	acumulado_7_dias
E	2024-01-01 00:00:00.000	90	90
E	2024-01-04 00:00:00.000	180	270
E	2024-01-06 00:00:00.000	250	520
E	2024-01-08 00:00:00.000	400	830
E	2024-01-10 00:00:00.000	800	1.630

☝ En este caso utilizamos RANGE con INTERVAL '6' DAY como límite inferior: queremos que sume comenzando desde 6 **días** hacia atrás hasta la actual inclusive.

✅Esto asegurará que solo se sumen las ventas dentro de los últimos 7 días efectivos, sin importar si hay días sin transacciones.

Descubre más recursos en

www.dateneo.com