Breaking down WINDOW FUNCTIONS in SQL

Dawn Choo

Window functions in SQL are powerful but complex.

So let’s break it down..

What do window functions do?

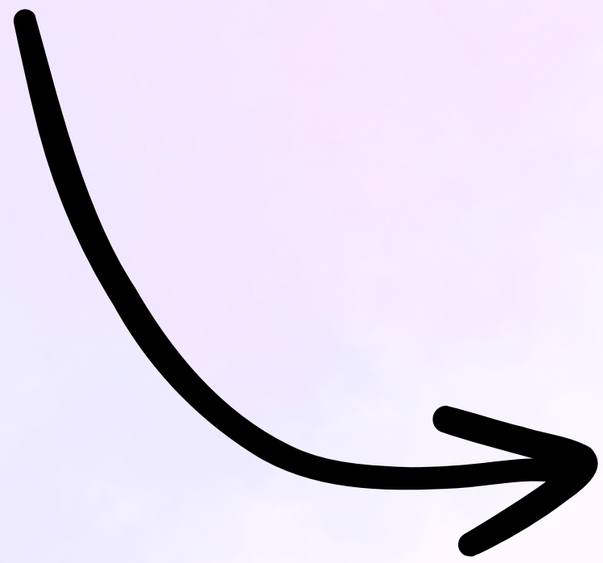
They allow access to data in other rows.

Sales Month	Sales
January	$1000
February	$2000
March	$1000
April	$3000

Unlike aggregation functions, they return **one result per row**.

Sales Month	Sales	Cumulative Sales
January	$1000	$1000
February	$2000	$3000
March	$1000	$4000
April	$3000	$7000

Let’s look at how a window function query is constructed.



We are going to calculate the cumulative total sales by products using data in this table.

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

First, use an **aggregate function** or **window function** to define how to aggregate over the window.

In this case, we are using SUM( ).

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
        )
```

Next, all window functions are defined by the **OVER( )** keyword

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
        )
```

PARTITION BY ( ) defines how to group the data.

In this case, group by the column Product.

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

ORDER BY ( ) defines how to sort the data within the groups.

In this case, sort by the column Sales Month.

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

ROWS BETWEEN defines the relative window to aggregate on

In this case, from the start of the window to the current row.

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

UNBOUNDED PRECEDING
means the start of the window is
at the beginning of the group

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

CURRENT ROW means the end of the window is the current row

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales
January	Watches	$1000
February	Watches	$2000
March	Watches	$1000
January	Computers	$3000
February	Computers	$3000
March	Computers	$5000

Here is the final result:
cumulative sum of sales by product

Cumulative Sales by Product

```
SELECT
  SUM(sales) OVER (
    PARTITION BY product
    ORDER BY sales_month
    ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW
          )
```

Sales Month	Product	Sales	Cumulative Sales
January	Watches	$1000	$1000
February	Watches	$2000	$3000
March	Watches	$1000	$4000
January	Computers	$3000	$3000
February	Computers	$3000	$6000
March	Computers	$5000	$11000

Found this useful?

Save it

Follow me

Repost it

Dawn Choo