Window functions are one of the most **powerful parts of SQL** — and they often get asked in **data engineering & analyst interviews**. Let’s build this from **basic → advanced** step by step.

**Step 1: Basics of Window Functions**

**Objective:** Understand what a window function is, its syntax, and how it differs from regular aggregate functions.

**Key Concepts:**

1. **Window vs Aggregate Functions**:
   * Aggregate functions like SUM() collapse rows.
   * Window functions calculate over a “window” but **keep all rows**.

1️ - SUM() as a regular aggregate function

When you use SUM() **without a window function**, it **collapses rows**.

**Example:** **SELECT branch\_id, SUM(balance) AS total\_balance**

**FROM accounts**

**GROUP BY branch\_id;**

**What happens here:**

* SQL **groups all rows by branch\_id**.
* Then it sums the balances **per branch**.
* The result **has one row per branch**, not one row per account.

So yes, it **does not take data 1 by 1**, it **aggregates all rows in the group together**.

**2️- SUM() as a Window Function**

When you use SUM() **with OVER()**, it **does NOT collapse rows**.

**Example:** **SELECT**

**account\_id,**

**branch\_id,**

**balance,**

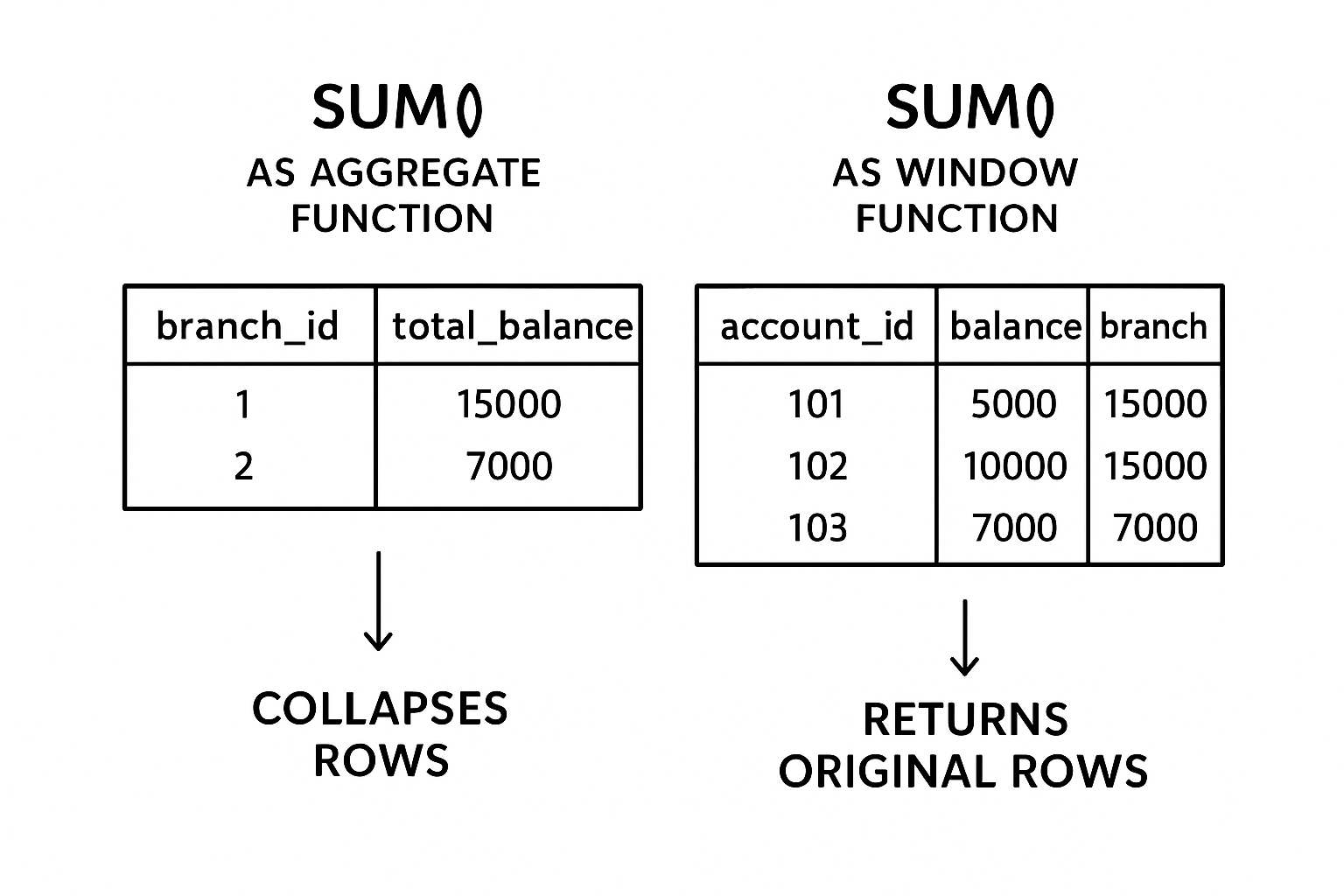
**SUM(balance) OVER (PARTITION BY branch\_id) AS branch\_total**

**FROM accounts;**

**What happens here:**

* SQL calculates the **sum of balances per branch**.
* **BUT each account row still appears**.
* Every account now also shows the **total balance of its branch**.

**Output:**



Take **branch 1**:

| **account\_id** | **branch\_id** | **balance** |
| --- | --- | --- |
| 101 | 1 | 5000 |
| 102 | 1 | 10000 |

* Each account still **keeps its own balance**.
  + Account 101 → balance 5000
  + Account 102 → balance 10000
* The **window SUM** calculates **total for the branch** = 5000 + 10000 = 15000.

So when we do:

SELECT

account\_id,

branch\_id,

balance,

SUM(balance) OVER (PARTITION BY branch\_id) AS branch\_total

FROM accounts;

We get:

| **account\_id** | **branch\_id** | **balance** | **branch\_total** |
| --- | --- | --- | --- |
| 101 | 1 | 5000 | 15000 |
| 102 | 1 | 10000 | 15000 |

✅ Important:

* **branch\_total is not the user’s balance**.
* It is **the sum of all balances in the branch**.
* Each user sees their own balance **plus** the total of their branch.

Think of it like this: **“Your balance is 5000, but your branch as a whole has 15000.**

✅ **So your practice order is:**  
OVER() → PARTITION BY → ORDER BY → ROW\_NUMBER/RANK → LEAD/LAG → ROWS BETWEEN

**Step 1: OVER() (base keyword)**

* Syntax: function() OVER ()
* Without PARTITION BY or ORDER BY, the function works on the entire table.

SELECT

account\_id,

balance,

SUM(balance) OVER() AS total\_balance

FROM accounts;

* **OVER is the heart of all window functions**.
* A screenshot of a computer

  AI-generated content may be incorrect.
* **How SQL thinks:**
* SUM(price) OVER() → means “calculate the sum of **all prices** in the table.”
* Since there is **no PARTITION BY** or **ORDER BY**, it looks at the entire dataset.
* SQL does not collapse the rows — it keeps every row from the orders table.
* Then it adds a new column (total) with the **grand total (5755)** next to each row.

**Why does total repeat in every row?**

Because:

* SUM(price) OVER() = one single value (grand total of all prices).
* SQL doesn’t collapse rows → so it **copies that value into every row**.

**🔑 What happens here?**

* SUM(price) → add up all prices.
* OVER(PARTITION BY Category) → do the sum **separately for each Category**.
* SQL **does not collapse rows** — every row still appears, but with the category total shown.

SELECT

orderid,

customerid,

CustomerName,

ProductName,

Price,

Category,

SUM(price) OVER(PARTITION BY Category) AS total

FROM orders;

**📊 Example with your dataset (let’s assume Categories)**

| **orderid** | **customerid** | **CustomerName** | **ProductName** | **Price** | **Category** |
| --- | --- | --- | --- | --- | --- |
| 1001 | 1 | Rajesh Kumar | iPhone 14 Pro | 1200.00 | Electronics |
| 1002 | 2 | Priya Sharma | Nike Shoes | 120.00 | Fashion |
| 1003 | 1 | Rajesh Kumar | Samsung TV | 800.00 | Electronics |
| 1004 | 3 | John Smith | MacBook Air | 1500.00 | Electronics |
| 1005 | 2 | Priya Sharma | Saree | 60.00 | Fashion |
| 1006 | 4 | Anita Gupta | iPad | 700.00 | Electronics |
| 1007 | 1 | Rajesh Kumar | AirPods | 150.00 | Electronics |
| 1008 | 3 | John Smith | T-Shirt | 25.00 | Fashion |
| 1009 | 2 | Priya Sharma | Dell Laptop | 1000.00 | Electronics |
| 1010 | 4 | Anita Gupta | Smart Watch | 200.00 | Electronics |

**✅ Output**

| **orderid** | **CustomerName** | **ProductName** | **Price** | **Category** | **total** |
| --- | --- | --- | --- | --- | --- |
| 1001 | Rajesh Kumar | iPhone 14 Pro | 1200 | Electronics | 5350.00 |
| 1003 | Rajesh Kumar | Samsung TV | 800 | Electronics | 5350.00 |
| 1004 | John Smith | MacBook Air | 1500 | Electronics | 5350.00 |
| 1006 | Anita Gupta | iPad | 700 | Electronics | 5350.00 |
| 1007 | Rajesh Kumar | AirPods | 150 | Electronics | 5350.00 |
| 1009 | Priya Sharma | Dell Laptop | 1000 | Electronics | 5350.00 |
| 1010 | Anita Gupta | Smart Watch | 200 | Electronics | 5350.00 |
| 1002 | Priya Sharma | Nike Shoes | 120 | Fashion | 205.00 |
| 1005 | Priya Sharma | Saree | 60 | Fashion | 205.00 |
| 1008 | John Smith | T-Shirt | 25 | Fashion | 205.00 |

**💡 Meaning of total here:**

* For **Electronics**, total = 1200 + 800 + 1500 + 700 + 150 + 1000 + 200 = **5350**.
* For **Fashion**, total = 120 + 60 + 25 = **205**.
* Each row shows its own price **plus the total for that category**.

👉 So yes, your query correctly gives **category-wise total product price** without losing the row details.

**📝 Practice Questions**

**Q1.** Show each order with the grand total price of all orders.

**Q2.** Show each order with the total spent by each customer.

**Q3.** Show each order with the total product price in each category.

**Q4.** Show each order with a row number for each customer’s orders, ordered by price (highest first).

**Q5.** Show each order with the previous order’s price, based on orderid.

**Finally completed this:**

**Basic Questions (1–10)**

1. Show each order with a grand total of all orders using OVER().
2. Show each order with total spent by each customer.
3. Show each order with total product sales per category.
4. Show each order with row number per customer ordered by OrderDate ASC.
5. Show each order with row number per category ordered by Price DESC.
6. Show each order with the cumulative sum of TotalAmount per customer ordered by OrderDate.
7. Show each order with the cumulative sum of TotalAmount per category ordered by Price.
8. Show each order with ROW\_NUMBER() for all orders sorted by OrderID ASC.
9. Show each order with ROW\_NUMBER() partitioned by CustomerID and ordered by Quantity DESC.
10. Show each order with cumulative total for all orders using OVER() without PARTITION BY.

**Intermediate Questions (11–20)**

1. Assign a unique rank per customer’s order based on TotalAmount using ROW\_NUMBER().
2. Show top 2 most expensive orders per customer using ROW\_NUMBER().
3. Show first purchased product per customer using ROW\_NUMBER().
4. Show last purchased product per customer using ROW\_NUMBER() + ORDER BY OrderDate DESC.
5. Show row number per category with tie-breaking by Price DESC, then OrderDate ASC.
6. Show the rank of each order per customer based on Price using ROW\_NUMBER().
7. Show the row number per product (if the same product appears multiple times) ordered by OrderDate.
8. Show cumulative number of products bought per customer using ROW\_NUMBER().
9. Show each order with total quantity purchased per customer using SUM(TotalAmount) OVER(PARTITION BY CustomerID).
10. Show row number per category for top 3 orders by TotalAmount.

**Advanced Questions (21–30)**

1. Show first and second most expensive order per customer using ROW\_NUMBER() and filtering.
2. Show orders where ROW\_NUMBER() = 1 per customer (top purchase).
3. Show orders where ROW\_NUMBER() = 2 per customer (second top purchase).
4. Show cumulative total of TotalAmount per category and per customer (nested partition).
5. Show each customer’s top N purchases in each category using ROW\_NUMBER().
6. Show row number per customer ordered by TotalAmount DESC, then OrderDate ASC (tie-breaker).
7. Show row number of each order per month (use MONTH(OrderDate) as partition).
8. Show each order with the rank per customer ignoring ties using ROW\_NUMBER().
9. Show top selling product per category using ROW\_NUMBER() partitioned by Category and ordered by TotalAmount DESC.
10. Combine ROW\_NUMBER() with OVER(PARTITION BY CustomerID ORDER BY TotalAmount DESC) to show **top order per customer** and cumulative sum per customer in the same query.