Lecture 18: SQL Aggregation

COUNT, SUM, AVG, MAX, MIN

Dr. Innocent Nyalala



IIT Madras Zanzibar

13 April 2026

Outline

Running Example

`Orders(OrderID, StudentID, ItemName, Qty, Price, OrderDate, Status)`

`Students(StudentID, Name, Year, Hostel)`

University Food Court

The Big 5 Aggregate Functions

Function	What it computes	Food Court Example	NULLs?
CO	Counts rows or va	How many orders today?	COUNT(*) includes
SU	Adds numeric val	Total revenue this week?	Ignores NULLs
AV	Mean of valu	Avg order price per student?	Ignores NULLs
MA	Largest value	Most expensive item?	Ignores NULLs
MI	Smallest value	Cheapest item?	Ignores NULLs

Aggregate Function

Takes a **set of values**, returns **one value**.
Always used in `SELECT` or `HAVING`.

```
SELECT  COUNT(*), SUM(Price),
        AVG(Price), MAX(Price),
        MIN(Price)
FROM    Orders;
```

COUNT Variants — What Gets Counted?

OID	SID	Item	Price
1	S01	Biryani	250
2	S02	Juice	80
3	S01	*NULL*	*NULL*
4	S03	Pizza	300
5	S02	Coffee	60

Query	Result	Why
`COUNT(*)`	**5**	All rows
`COUNT(Price)`	**4**	Skips NULL
`COUNT(Item)`	**4**	Skips NULL
`COUNT(DISTINCT Item)`	**4**	Unique non-NULL
`COUNT(DISTINCT SID)`	**3**	S01, S02, S03

NULL rows

Row 3 has `NULL` for Item and Price.
Watch what each COUNT does!

```sql
SELECT  COUNT(*),
        COUNT(Price),
        COUNT(DISTINCT StudentID)
FROM    Orders;
```

SUM and AVG — Revenue Analytics

StudentID	Item	Price
S01	Biryani	250
S02	Juice	80
S01	Pizza	300
S03	Coffee	60
S02	Biryani	250
	Total	**940**
	Average	**188**

```sql
-- Total food court revenue
SELECT SUM(Price) AS TotalRevenue
FROM   Orders;
-- Result: 940

-- Average order value
SELECT AVG(Price) AS AvgOrder
FROM   Orders;
-- Result: 188.00

-- Revenue for Biryani only
SELECT SUM(Price) AS BiryaniRevenue
FROM   Orders
WHERE  ItemName = 'Biryani';
-- Result: 500
```

Key insight

MAX and MIN — Extremes in the Data

Item	Price	SID	OrderDate
Biryani	**300**	S01	2026-04-10
Juice	80	S02	2026-04-11
Pizza	250	S01	**2026-04-13**
Coffee	**60**	S03	2026-04-09
Samosa	40	S04	2026-04-08

MAX=300
Biryani

MIN=40
Samosa

```sql
-- Priciest item?
SELECT MAX(Price) AS MaxPrice
FROM   Orders;
-- Result: 300

-- Cheapest item?
SELECT MIN(Price) AS MinPrice
FROM   Orders;
-- Result: 40

-- Most recent order?
SELECT MAX(OrderDate) AS LastOrder
FROM   Orders;
-- Result: 2026-04-13

-- Alphabetically first student
   name?
```

NULL Handling in Aggregates

Before: Column with NULLs

OrderID	Price
1	250
2	*NULL*
3	80
4	*NULL*
5	300

⚠ NULL ≠ Zero
A missing price is NOT zero revenue!

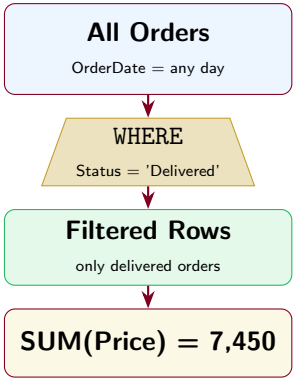
After: What each aggregate returns

Aggregate	Result	NULLs?
`COUNT(*)`	5	Included
`COUNT(Price)`	3	Skipped
`SUM(Price)`	630	Skipped
`AVG(Price)`	210	Skipped (avg of 3)
`MAX(Price)`	300	Skipped

Danger Zone

`AVG(Price) = 630/3 = 210`, NOT 630/5=126.
NULLs silently shrink the denominator!

Combining Aggregates with WHERE




```sql
-- Revenue: delivered orders only
SELECT SUM(Price) AS Revenue
FROM   Orders
WHERE  Status = 'Delivered';

-- Orders this week
SELECT COUNT(*) AS WeekOrders,
       AVG(Price) AS AvgPrice
FROM   Orders
WHERE  OrderDate >= '2026-04-07'
  AND  OrderDate <= '2026-04-13';

-- How many students bought Biryani
   ?
SELECT COUNT(DISTINCT StudentID)
FROM   Orders
WHERE  ItemName = 'Biryani';
```

Think–Pair–Share

Schema Reminder

```
Orders(OrderID, StudentID, ItemName,
Qty, Price, OrderDate, Status)
Students(StudentID, Name, Year,
Hostel)
```

1. How many **distinct items** were ever ordered?
2. What is the **total quantity** of all orders?
3. Find the **average price** of items ordered by Year-2 students.
4. Which **order date** had the most recent transaction?

2 minutes: discuss with your neighbour

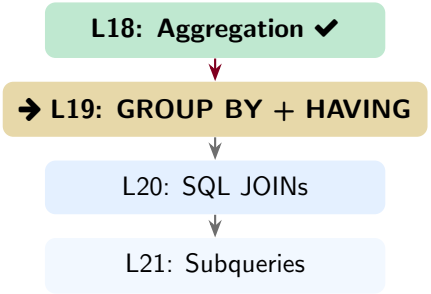
Hint

Q3 requires joining `Orders` and `Students` first — we cover JOINs in L20!

Summary and What's Next

Key Takeaways

- `COUNT(*)` counts all rows incl. NULLs
- `SUM/AVG/MAX/MIN` ignore NULLs
- `COUNT(DISTINCT col)` counts unique values
- `WHERE` filters **before** aggregation
- Aggregates collapse many rows → one result



Common Mistake

Using `WHERE Price > AVG(Price)` — **illegal!**
Must use `HAVING` for aggregate filters.

Next Lecture

GROUP BY: aggregate **per group**.
HAVING: filter the **groups** themselves.