**🔄 What is Denormalization?**

It’s the **intentional duplication of data** to improve **read performance**.

🧠 Think of it as the **opposite of normalization**. Instead of splitting data into multiple tables (to avoid redundancy), we merge/repeat data to make fetching faster.

**🧠 Why Denormalize?**

| **Reason** | **Explanation** |
| --- | --- |
| ⚡ Speed | JOINs are expensive; flattening avoids them |
| 📈 Read-heavy apps | Analytics/reporting systems need fast SELECTs |
| 🔄 Simplifies queries | No complex subqueries/joins |
| 🧩 Caching | Easy to cache denormalized data |
| 💰 Cost | Reduce DB compute if JOINs are slowing down app |

**🚫 Why *not* always use it?**

| **Problem** | **Impact** |
| --- | --- |
| 🚨 Redundancy | Duplicated data = storage wastage |
| 🤕 Update Anomalies | You change in one place but forget others |
| ❌ Loss of Integrity | More chances of inconsistent data |
| 🔧 Harder to maintain | Complex sync logic for updates/deletes |

**🔧 Ride-Sharing Example (Before vs After)**

**✅ Normalized**

**Tables:**

* Drivers(driver\_id, name)
* Rides(ride\_id, driver\_id, fare)

Query:

SELECT d.name, r.fare

FROM Rides r

JOIN Drivers d ON r.driver\_id = d.driver\_id;

**❌ Denormalized**

Single table:

Rides(ride\_id, driver\_id, driver\_name, fare)

Insert:

INSERT INTO Rides VALUES (1, 'D01', 'Ahmed', 300);

No JOIN needed now:

SELECT driver\_name, fare FROM Rides;

Faster read, but if Ahmed changes his name? Gotta update every row where he appears.

**💼 When to Denormalize? (Interview Tip)**

Use it when:

* **Read >> Write**
* Performance is **critical**
* You have **aggregated or historical** data
* You're building **data warehouses**, not transactional systems