The Master-Slave Database Concept

1.What problem does it solve?

- When a website or app grows, **one database server is not enough**.
- Thousands or millions of users start sending **read (view)** and **write (insert/update)** requests.
- A single database doing **both reads and writes** becomes **slow and overloaded**.

Master–Slave architecture is used to handle this load properly.

Basic idea (in simple words)

- **One database = Master**
- **Many databases = Slaves**

Each has a **fixed role**, just like an old, well-organized system where everyone knows their duty.

2.Roles of Master and Slave

Master Database

- Stores the **true and latest data**
- Handles **WRITE operations only**
  - Insert
  - Update
  - Delete

Only **one authority** writes data, so mistakes and conflicts are avoided.

Slave Databases

- Get **copies of data** from the Master
- Handle **READ operations only**
  - Fetch data
  - Show data to users

Many slaves can serve many users at the same time.

3.Why this architecture is useful?

- **High traffic handling**
- **Faster response time (low latency)**
- **Better availability**

- **System does not slow down**
- **Master is protected from overload**

Without this, one DB would do:

- Reads
- Writes
  → Result: **slow website for everyone**

4.Relation with CQRS (Command Query Responsibility Segregation)

- **Commands (Write operations)** → Master
- **Queries (Read operations)** → Slaves

Reads and writes are **separated**, which makes the system **clean and scalable**.

5.How data stays same everywhere? (Replication)

- **Database Replication** copies data from **Master → Slaves**

Types of Replication

1. Synchronous Replication

- Master waits until Slaves confirm data is copied
- **Strong consistency**
- Slightly slower

2. Asynchronous Replication

- Master does not wait
- Slaves update after some time
- **Faster but small delay possible**

Real-World Example (Very Clear)

Online Shopping Website (like Amazon)

- **Placing an order** → Master DB
- **Updating product stock** → Master DB
- **Browsing products** → Slave DB
- **Reading reviews** → Slave DB
- **Viewing order history** → Slave DB

Imagine:

- **1 Master** = Head Office (final decision maker)

- **Many Slaves** = Branch offices (only show information)

If everyone calls the **Head Office** for simple questions → chaos
Instead:

- Branch offices handle questions
- Head Office only handles **important updates**

6.What happens without Master–Slave?

- One DB does everything
- Too many read + write requests
- **High load → slow queries → poor user experience**