# **InnoDB Clustering**

* InnoDB Cluster is a high-availability solution built into MySQL.
* It allows you to create a self-healing, fault-tolerant cluster of MySQL servers.
* It uses:
  + Group Replication → to sync data automatically between servers.
  + MySQL Router → to route app traffic to the correct server.

✅ So if one database server fails, the cluster keeps working automatically!

**Components**

|  |  |
| --- | --- |
| Component | Description |
| MySQL Server | Normal MySQL server running Group Replication. |
| Group Replication Plugin | Handles real-time syncing between servers. |
| MySQL Shell | Command line tool to configure/manage the cluster. |
| MySQL Router | Smart proxy that connects Java apps to the correct DB server automatically. |

**Architecture**

[Java App] → [MySQL Router] → [MySQL InnoDB Cluster]

↳ [Primary MySQL Node]

↳ [Secondary MySQL Node]

↳ [Secondary MySQL Node]

**Concepts and Terms**

|  |  |
| --- | --- |
| Concept | Explanation |
| Quorum | Majority agreement is needed (e.g., 2 out of 3 nodes active). |
| Single-Primary Mode | Only 1 node accepts writes (default and safest). |
| Multi-Primary Mode | All nodes can write (advanced, riskier if not handled carefully). |
| Failover | Automatic switch to secondary node if primary fails. |
| Consistency Level | How strictly replication is handled (eventual, majority, etc.). |

**Connection Pooling (HikariCP)**

Without pooling:

* Java app → New DB connection → Query → Close connection → Repeat for every request.

With pooling:

* Java app → Gets a ready-made connection from pool → Query → Return it → Very Fast!
* **HikariCP** is **the fastest and most popular** connection pool library in Java.
* Used by **Spring Boot** as **default** since Spring Boot 2.x.
* It's extremely **lightweight**, **high performance**, and **low latency**

## **Steps Behind Server Voting:**

### **1. Heartbeat messages**

* All nodes (servers) **constantly** send **heartbeat signals** to each other (tiny network pings).
* Every server is **aware** of the other servers' health.

Example heartbeat every **5 seconds** (by default).

### **2. Detecting Failure**

* If a server **stops responding** (no heartbeat), other servers **detect** that.
* There's a **timeout period** (for example 10 seconds) to decide that the server is "dead".

### **3. Start Voting (Majority Consensus)**

* The remaining **alive servers** start a **consensus protocol** (similar to Raft/Paxos algorithms).
* They decide:
  + **Who should become Primary** (if Primary was lost).
  + Whether the cluster can continue safely (do we have Quorum?).

### **4. Election Rules**

* **Who becomes Primary** is decided based on:
  + **Member UUID** (server unique IDs).
  + **Server Priority** (if configured).
  + **Lowest Server UUID** (default if no priority is given).