Lesson 10 -> Data Replication

1️⃣ What Is Data Replication?

Data replication is the process of **copying the same data to multiple nodes**.

So instead of:

```
Data → 1 database
```

You have:

```
Data → DB1 → DB2 → DB3
```

Each replica stores the **same logical data**.

2️⃣ Why Do We Need Data Replication?

1. High Availability

If one node fails, others can serve traffic.

2. Read Scalability

Reads can be distributed across replicas.

3. Low Latency

Serve users from the nearest replica (geo-replication).

4. Fault Tolerance

Protection against hardware failures.

5. Disaster Recovery

Backups + replicas = safety net.

3️⃣ Types of Data Replication

3.1 Synchronous Replication

How it works

- Write is successful **only after all replicas confirm**

```
Client → Leader → Replica1 ✔ → Replica2 ✔ → ACK
```

Pros

✅ Strong consistency
✅ No stale reads

Cons

❌ High latency
❌ Poor availability

Used in:

- Banking
- Financial systems

3.2 Asynchronous Replication

How it works

- Leader commits first
- Replicas catch up later

```
Client → Leader ✔ → Replica (later)
```

Pros

✅ Low latency
✅ High availability

Cons

❌ Replication lag
❌ Potential data loss

Used in:

- MySQL
- MongoDB
- Most web apps

3.3 Semi-Synchronous Replication

How it works

- Leader waits for **at least one replica**

Pros

✅ Balanced consistency
✅ Lower data loss risk

Cons

❌ Slower than async
❌ Complex

Used in:

- MySQL semi-sync mode

4️⃣ Replication Models (Very Important)

4.1 Single-Leader (Primary–Replica)

Model

- One leader handles writes
- Replicas handle reads

```
Writes → Leader Reads → Replicas
```

Pros

✅ Simple
✅ Strong write consistency

Cons

❌ Leader bottleneck
❌ Failover complexity

Real DBs

- MySQL

- PostgreSQL
- MongoDB (primary-secondary)

4.2 Multi-Leader Replication

Model

- Multiple leaders accept writes
- Leaders replicate to each other

```
Write → Leader A Write → Leader B
```

Pros

✅ High write availability
✅ Geo-distributed writes

Cons

❌ Conflict resolution
❌ Complex logic

Real DBs

- Cassandra (conceptually)
- MySQL multi-master
- CouchDB

4.3 Leaderless Replication

Model

- No single leader
- Clients write to multiple nodes

```
Client → Node1 Client → Node2 Client → Node3
```

Pros

✅ No single point of failure
✅ High availability

Cons

❌ Eventual consistency
❌ Read repair complexity

Real DBs

- Cassandra
- DynamoDB
- Riak

5️⃣ Read–Write Strategy

Defines:

- **Where writes go**
- **Where reads come from**

Common Strategies

Write

- Usually to leader (or quorum)

Read

- From leader → strong consistency
- From replicas → eventual consistency

Read Preferences

- Read-after-write consistency
- Read-your-writes
- Monotonic reads

Very important for:

- User-facing ML features
- Recommendations
- Personalization

6️⃣ What Is Replication Lag?

Replication lag = delay between leader write and replica update.

Example:

```
Write at T0 → Leader Replica updates at T0 + 2s
```

Causes

- Network latency
- High write throughput
- Disk IO
- Replica overload

Why Replication Lag is Dangerous

- Users see stale data
- ML models get outdated features
- Counters become inconsistent

Mitigation

- Read from leader
- Sticky sessions
- Version checks
- Async tolerance

7️⃣ Database Comparison (Read–Write & Replication Models)

DB	Replication Model	Write Strategy	Read Strategy	Consistency
MySQL	Single leader	Leader only	Leader / Replica	Strong → Eventual
MongoDB	Primary–Secondary	Primary	Secondary / Primary	Tunable
Cassandra	Leaderless	Quorum	Quorum	Eventual

DB	Replication Model	Write Strategy	Read Strategy	Consistency
DynamoDB	Leaderless	Quorum	Quorum	Eventual / Strong
Redis	Primary–Replica	Primary	Replica	Eventual
Redis Cluster	Partitioned leader	Partition leader	Replica	Eventual

---

Quick DB Insights

MySQL

- Simple
- Strong consistency
- Scaling writes is hard

MongoDB

- Flexible read preferences
- Good balance
- Still leader-based

Cassandra

- Designed for scale
- Tunable consistency
- Complex operationally

DynamoDB

- Fully managed
- Global tables
- Expensive at scale

Redis

- Extremely fast
- Often used as cache
- Data durability trade-offs

ML & Retail Context (Very Important)

Feature Stores

- Async replication
- Read-your-writes required

Online Inference

- Read from nearest replica
- Accept eventual consistency

Training Pipelines

- Replica lag acceptable
- Throughput more important

Architect-Level Summary

- Replication improves availability and read scalability
- Consistency vs availability is always a trade-off
- ML systems must tolerate stale reads
- Leaderless systems scale better but are harder to reason about