Notes - Session 3 - Databases

Replication Strategies

There are 3 types of replication strategies

1. Leader - Follower Replication
2. Multi-Leader Replication
3. Leaderless Replication

Leader - Follower Replication?

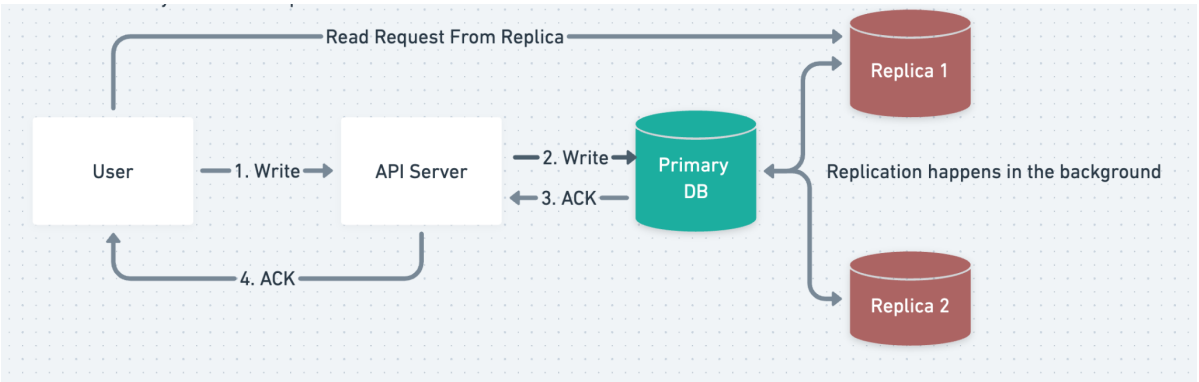
Primary node is responsible for handling all write requests
Follower nodes are used for handling read requests

Appropriate when our workload is read-heavy

For write-heavy applications, leader-follower replication is not a good idea ever

We can scale for readability, by adding more read replicas

Follower nodes can handle read requests even when the leader is down



What happens when the primary node fails?

In case of failure of the primary node, a secondary node can be appointed as a primary node, which speeds up the process of recovering the initial primary node. There are two approaches to select the new primary node: manual and automatic.

In a manual approach, an operator decides which node should be the primary node and notifies all secondary nodes.

In an automatic approach, when secondary nodes find out that the primary node has failed, they appoint the new primary node by conducting an election known as a leader election.

Replication Methods:

1. Statement-based replication

In the statement-based replication approach, the primary node saves all statements that it executes, like insert, delete, update, and so on, and sends them to the secondary nodes to perform.

Used in MySQL till version 5.1

Challenges with Statement-based replication:

1. Any non-deterministic function (such as NOW()) might result in distinct writes on the follower and leader.
2. If a write statement is dependent on a prior write, and both of them reach the follower in the wrong order, the outcome on the follower node will be uncertain.

2. Write-ahead log (WAL) shipping

The primary node saves the query before executing it in a log file known as a write-ahead log file. It then uses these logs to copy the data onto the secondary nodes.

Used in PostgreSQL and Oracle

Challenges with WAL:

1. WAL defines data at a very low level. It's tightly coupled with the inner structure of the database engine, which makes upgrading software on the leader and followers complicated.

3. Logical (row-based) log replication

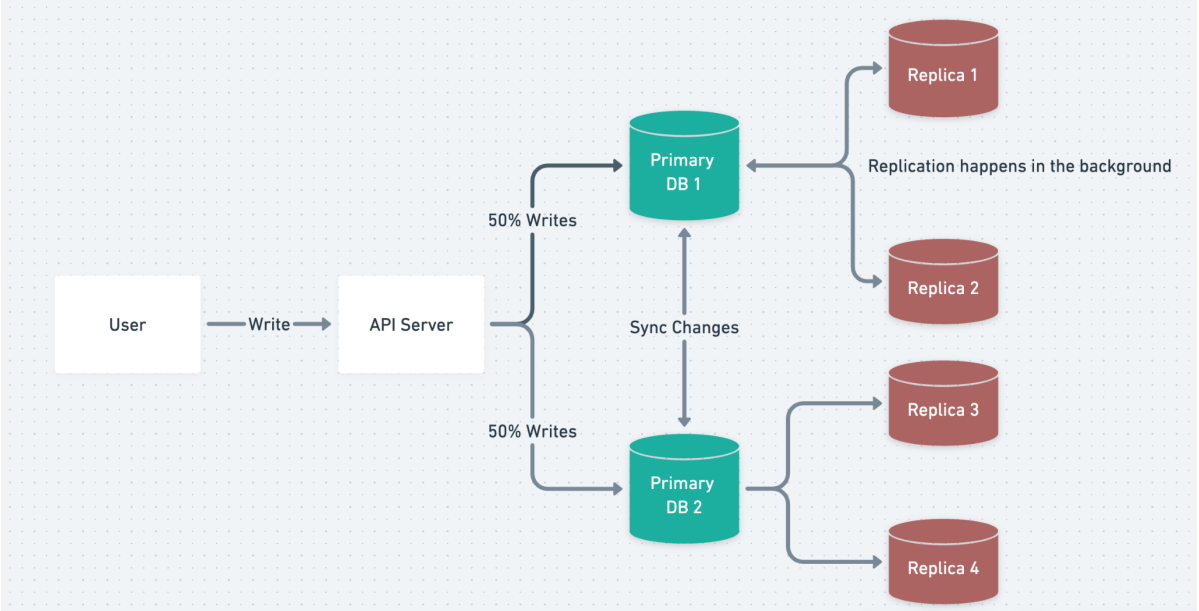
All secondary nodes replicate the actual data changes. For example, if a row is inserted or deleted in a table, the secondary nodes will replicate that change in that specific table

The binary log records change to database tables on the primary node at the record level. To create a replica of the primary node, the secondary node reads this data and changes its records accordingly.

Row-based replication doesn't have the same difficulties as WAL because it doesn't require information about data layout inside the database engine.

Multi-Leader Replication

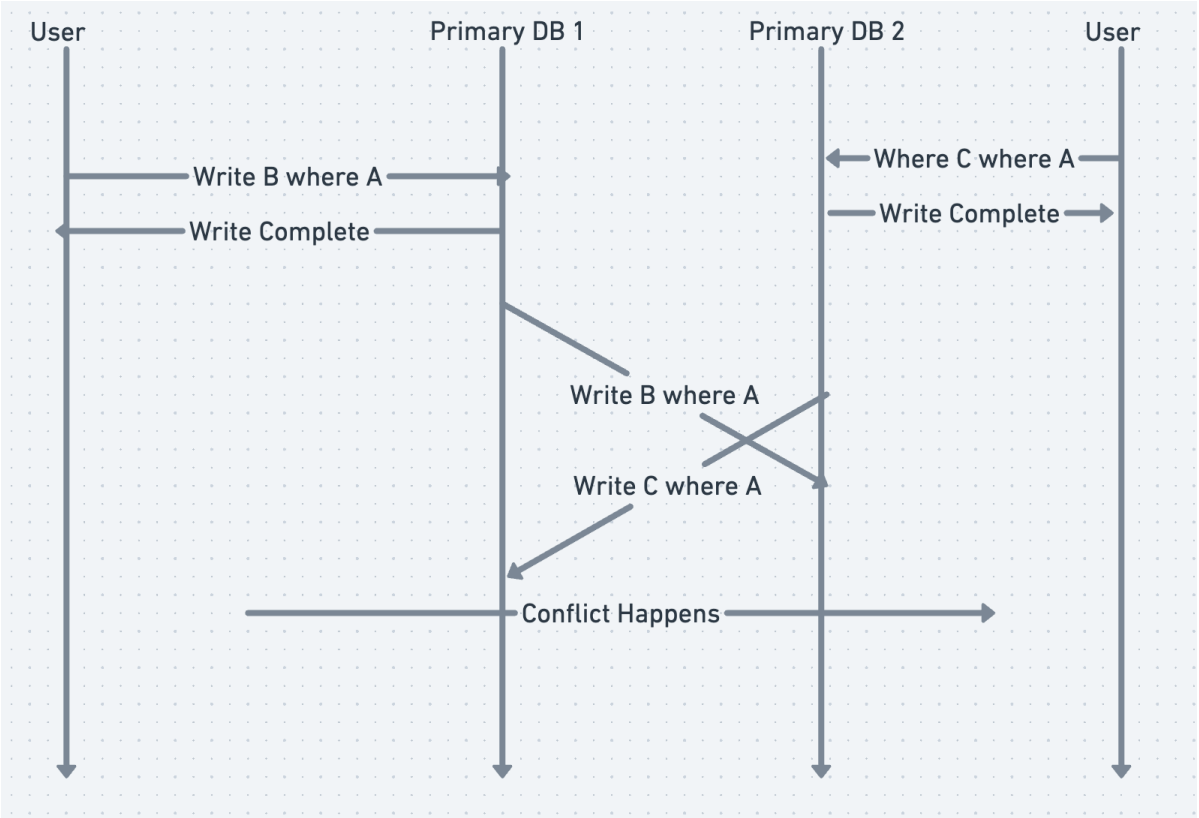
There are multiple primary nodes that process the writes and send them to all other primary and secondary nodes to replicate



Useful in applications in which we can continue work even if we're offline.

Example: Calendar apps
We can set our meetings even if we don't have access to the internet. Once we're online, it replicates its changes from our local database (our mobile phone or laptop acts as a primary node) to other nodes.

Conflicts can happen in Multi-Leader Replication:
Since all the primary nodes concurrently deal with the write requests, they may modify the same data, which can create a conflict between them.



Handling Conflicts

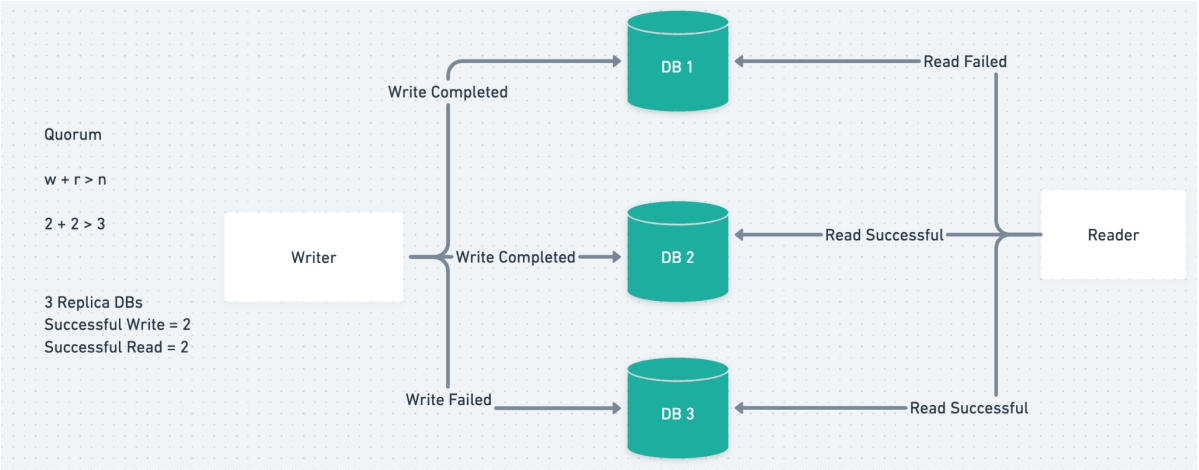
1. Last Write Wins
2. Custom Logic for applications (example Github Merge)

Leaderless Replication

Leaderless replication helps solve the following challenges faced in leader-follower replication:

- Primary node is a bottleneck for write
- Single point of failure
- Difficult to implement write scalability

In leaderless replication, all the nodes have equal weightage and can accept reads and writes requests.



Data inconsistency can exist in leaderless replication too. This is because when several nodes accept write requests, it may lead to concurrent writes.

Handling Conflicts

We use Quorums (explained in the recordings) to handle conflicts in leaderless replication