Notes - Session 2 - Databases

DB Isolation Levels:

SQL Scripts:

```
Unset
USE algoprep;

SELECT @@transaction_ISOLATION;

SET autocommit=0;

// Repeatable read

START TRANSACTION;

SELECT * FROM persons where id=1;

UPDATE persons SET fname = "abhi_t1" WHERE id = 1;

SELECT * FROM persons where id=1;


// moving to read committed

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT @@transaction_ISOLATION;

START TRANSACTION;

SELECT * FROM persons where id=1;

UPDATE persons SET fname = "abhi_read_committed" WHERE id = 1;

SELECT * FROM persons where id=1;


// moving to read uncommitted

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT @@transaction_ISOLATION;
```

```
START TRANSACTION;

SELECT * FROM persons where id=1;

UPDATE persons SET fname = "abhi_uncommitted_t1" WHERE id = 1;

SELECT * FROM persons where id=1;


// moving to serializable

SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT @@transaction_ISOLATION;

START TRANSACTION;

SELECT * FROM persons where id=1;

UPDATE persons SET fname = "abhi_serializable_t1" WHERE id = 1;

SELECT * FROM persons where id=1;
```

Real Life Examples:

1. Read Uncommitted:
    - Use Case:

      Rarely used in practice due to its minimal isolation guarantees.

    - Example:

      In a high-throughput logging system where immediate visibility of log entries is essential, you might use this isolation level. Even if a transaction hasn't been committed, other transactions can read the uncommitted data.

2. Read Committed:
    - Use Case:

      Used in most transactional systems where some level of isolation is needed.

    - Example

      In an e-commerce platform, when updating product stock levels, you want to prevent users from seeing negative stock values or purchasing items that are

out of stock. Read Committed ensures that only committed changes are visible to other transactions.

3. Repeatable Read:
    - Use Case:

      Useful when you need to maintain a consistent snapshot of data during a transaction.

    - Example:

      In a reservation system for a hotel, you don't want a customer to book a room that has become unavailable since they first viewed the available rooms. Repeatable Read ensures that the set of rooms a customer sees during their booking process remains constant.

4. Serializable:
    - Use Case:

      When strict isolation is required, often used in financial systems.

    - Example:

      In a banking application, you want to ensure that two transactions, such as transferring money between accounts, don't interfere with each other. Serializable isolation ensures that transactions occur as if they were executed one after the other, preventing conflicts.

5. Snapshot Isolation:
    - Use Case:

      Useful when you need to provide consistent views of data without locking. MySQL does not support Snapshot Isolation.

    - Example:

      In a content management system, multiple authors may be editing articles simultaneously. Snapshot Isolation allows each author to work on their copy of an article without locking it, and changes are merged when saving without blocking others.

6. No Isolation:
    - Use Case:

      Rarely used in real-world scenarios, as it provides no isolation between transactions.

- Example:

  In a single-user application where there's no concurrent access or where data consistency isn't critical, you might choose not to enforce any isolation level. However, this is uncommon in multi-user database systems.

Scaling Databases

Vertical Scaling

Pros

- Add more CPU or RAM to the DB
- Faster way to handle more load to handle unexpected scale

Cons

- Requires downtime during reboot
- Has physical hardware limitations

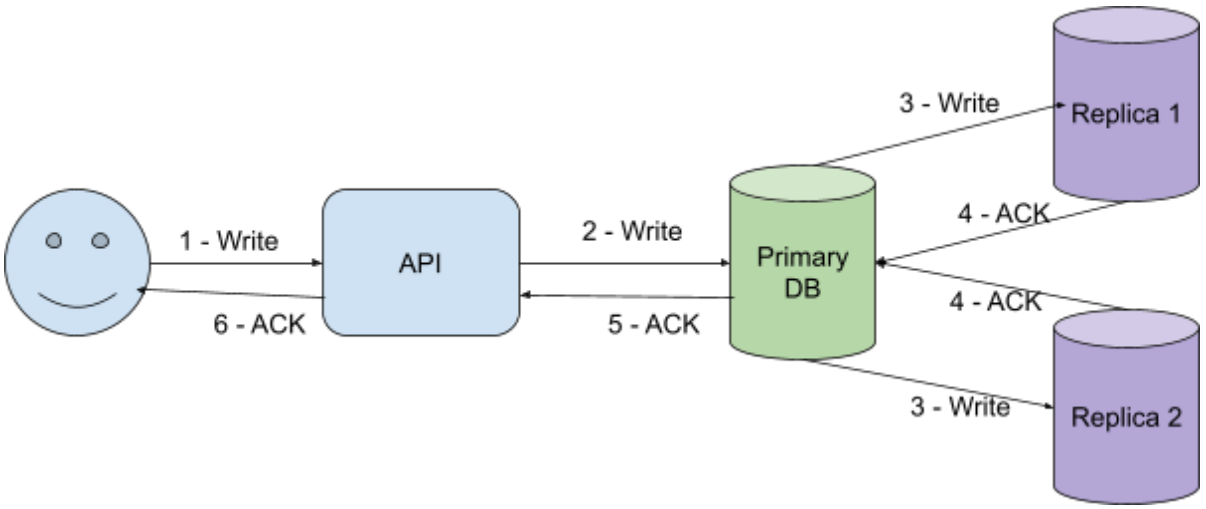
Horizontal Scaling

- Better suited for read heavy applications

DB Replication

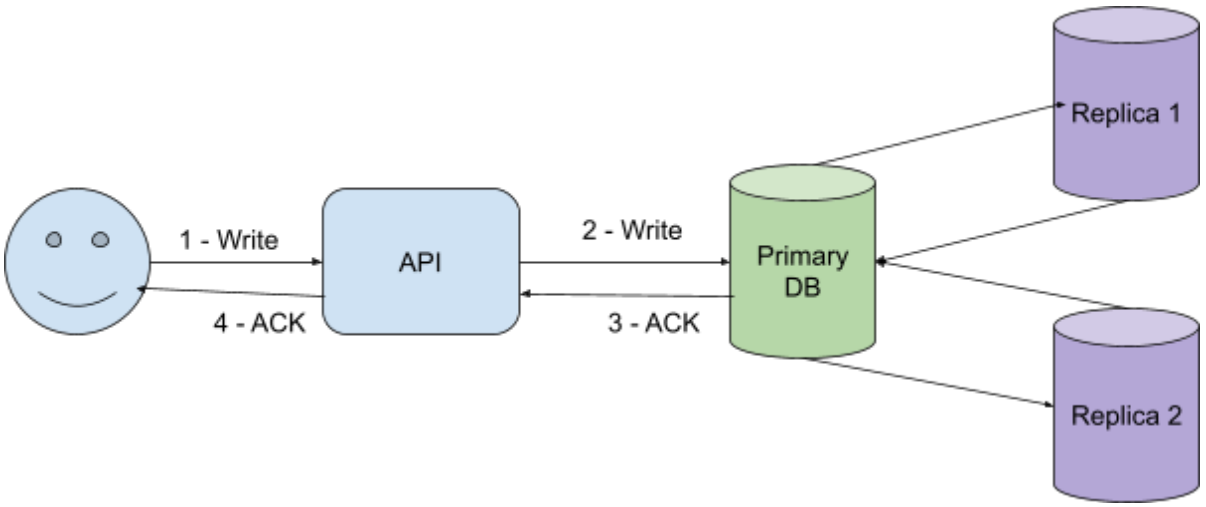
Synchronous Replication

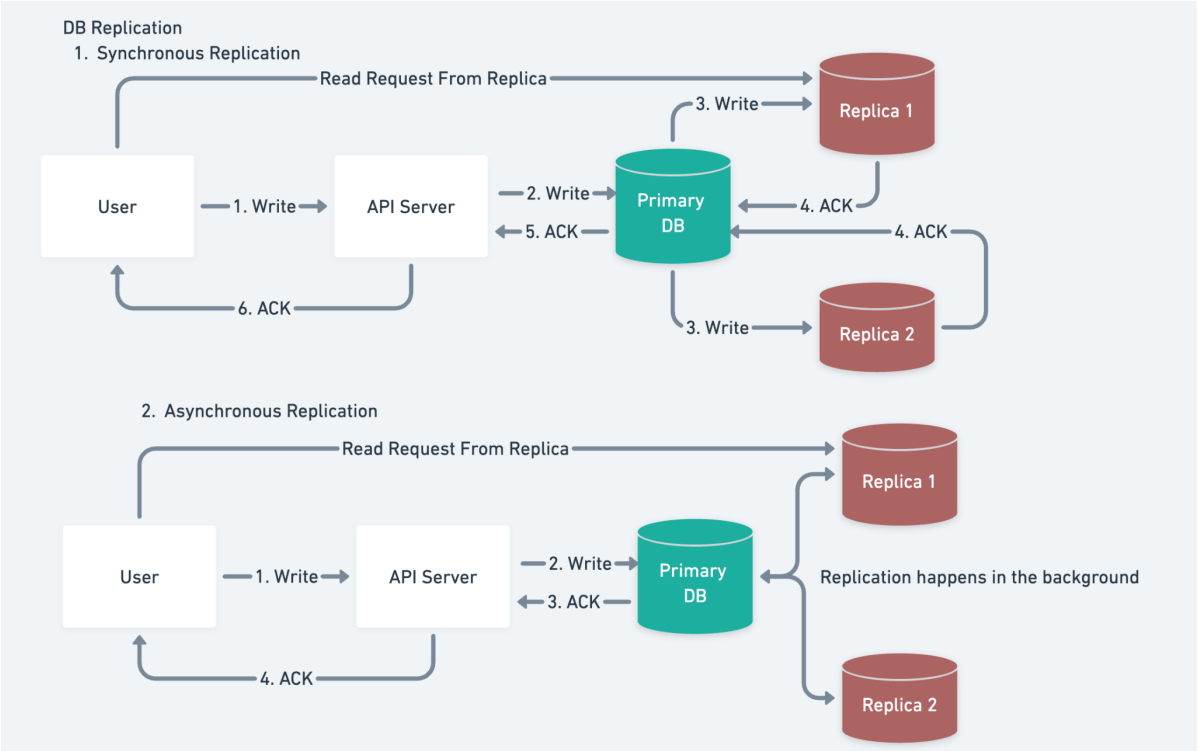
- Strong consistency
- Zero Replication Lag
- Slower Writes



Asynchronous Replication

- Eventual Consistency
- Replication Lag
- Faster Writes




DB Replication
1. Synchronous Replication
Read Request From Replica
3. Write
Replica 1
User
1. Write
API Server
2. Write
Primary DB
4. ACK
5. ACK
4. ACK
6. ACK
3. Write
Replica 2
2. Asynchronous Replication
Read Request From Replica
Replica 1
User
1. Write
API Server
2. Write
Primary DB
3. ACK
Replication happens in the background
4. ACK
Replica 2