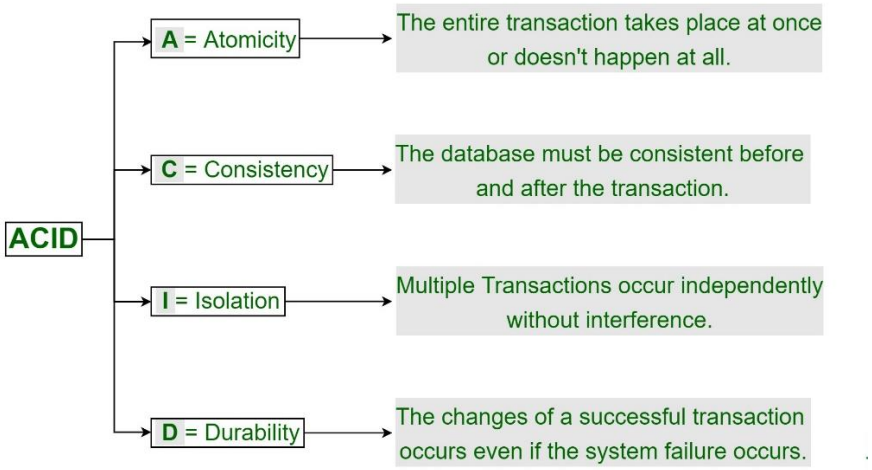
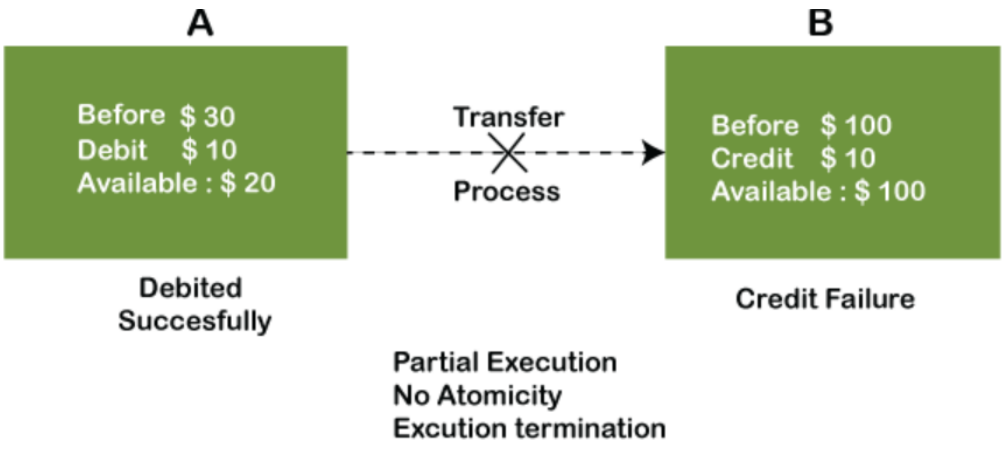
Experiment:8

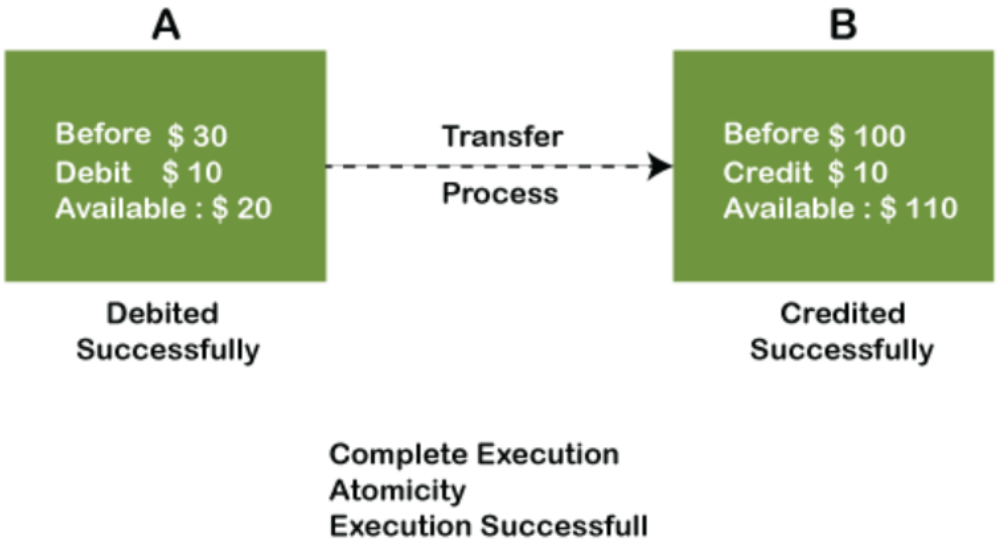
ACID & Indexes

MongoDB offers ACID transactions with some key differences compared to traditional relational databases. ACID stands for Atomicity, Consistency, Isolation, and Durability, which are essential properties for ensuring data integrity during database operations.



Atomicity: Ensures that all operations within a transaction are executed as a single unit. If any part fails, the entire transaction is rolled back, leaving the data unchanged. MongoDB guarantees atomicity for single-document updates. For multi-document transactions, MongoDB offers functionalities to achieve atomicity.

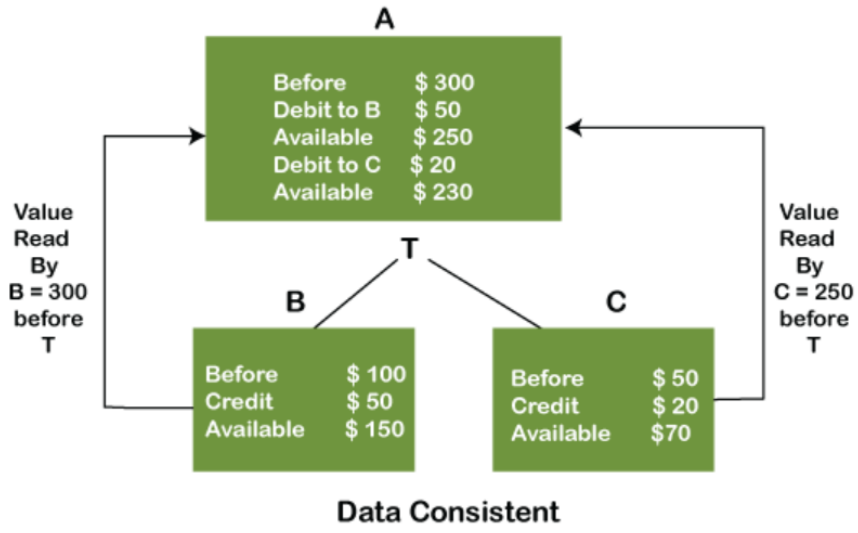




Consistency: Maintains data integrity by enforcing pre-defined rules before and after a transaction. MongoDB relies on schema validation and user-defined constraints to maintain consistency.

The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

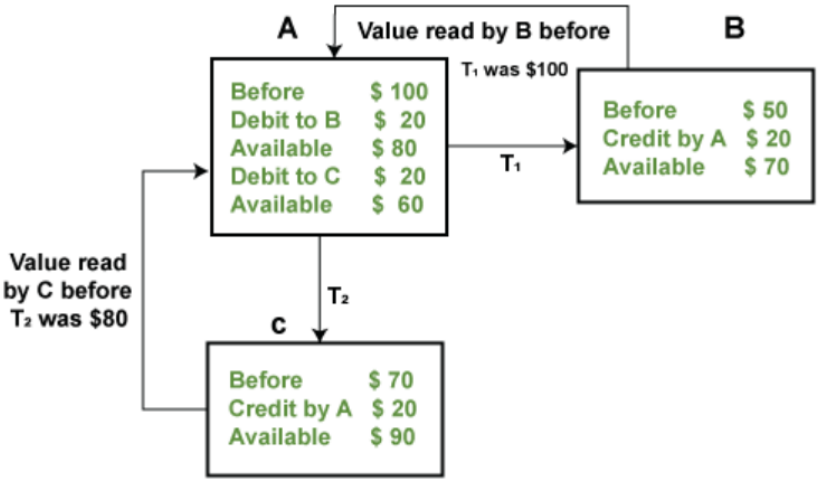
Example:



Isolation: Guarantees that concurrent transactions do not interfere with each other, ensuring predictable outcomes. MongoDB uses read locks and write locks to achieve isolation.

-

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



Isolation - Independent execution of T_1 & T_2 by A

Durability: Ensures that successful transactions persist to disk, even in case of system failures. MongoDB uses journaling to ensure data durability.

While MongoDB offers ACID transactions, it's important to understand that it prioritizes flexibility and scalability compared to strict ACID guarantees found in relational databases. This trade-off makes MongoDB a good choice for many applications but may not be suitable for scenarios requiring the strongest data consistency.

Replication (Master - Slave)

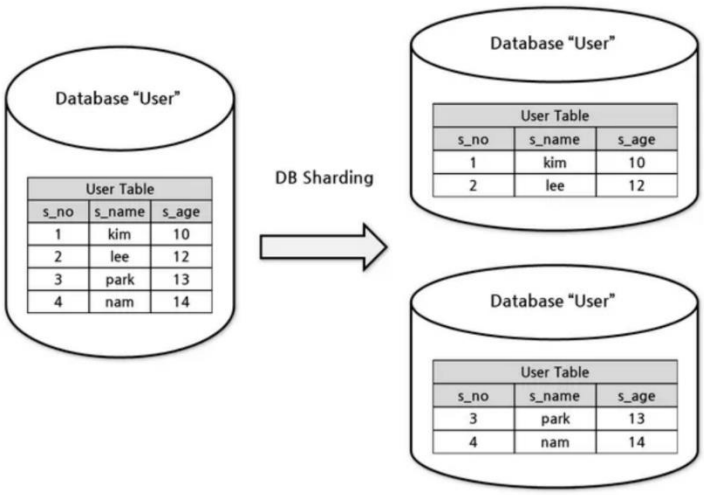
Replica Set: The current approach to replication in MongoDB. It consists of multiple mongod instances forming a set, with one designated as the **primary** and the others as **secondaries**. The primary receives all writes and replicates them to secondaries. This offers high availability with automatic failover to a secondary if the primary fails.

Master/Slave (deprecated): This terminology refers to an older approach similar to replica sets, but with some key differences. There was a single **master** that received all writes, and one or

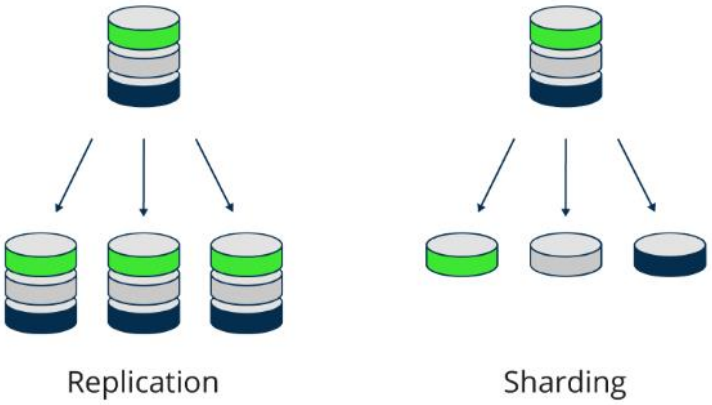
more **slaves** that replicated data from the master. However, slaves didn't participate in elections for becoming the new master upon failure.

Sharding

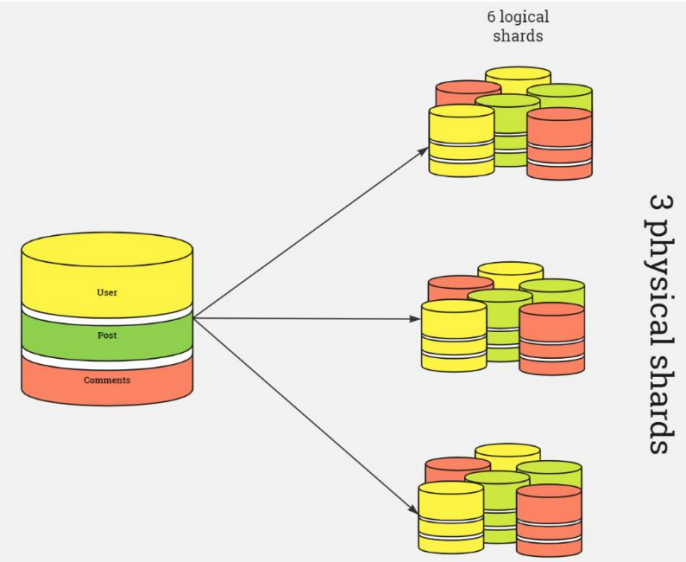
Sharding in MongoDB is a method for distributing data across multiple servers. This horizontal scaling approach allows you to handle massive datasets and high-volume operations that a single server can't efficiently manage.



Replication VS Sharding



Replication + Sharding

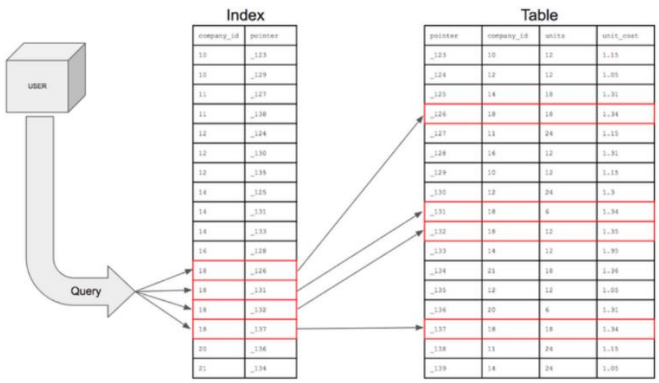


Indexes

In MongoDB, indexing is a fundamental concept for optimizing the performance of queries that retrieve data from collections (similar to tables in relational databases). Indexes are special data structures that act like reference manuals for your collection's documents. They essentially store a subset of your data in an organized way to enable faster retrieval based on specific fields.

- **Without Indexing:** When a query searches for documents based on a particular field, MongoDB needs to scan every single document in the collection. This can be very slow, especially for large collections.

- **With Indexing:** When you create an index on a field, MongoDB builds a separate structure that stores the values of that field along with references (pointers) to the corresponding documents. This allows MongoDB to quickly locate documents matching the query criteria by navigating the index structure instead of scanning every document.



Types of Indexes

MongoDB offers various indexing options to optimize queries based on your data and access patterns. Here's a breakdown of the common types of indexes in MongoDB:

1. Single-Field Index:

- The most basic and widely used type.
- Created on a single field within a document.
- When a query filters or sorts based on that specific field, the index significantly improves performance by enabling faster retrieval.

2. Compound Index:

- Involves multiple fields in a specified order.
- Useful for queries that involve filtering or sorting on combinations of those fields.
- MongoDB will traverse the index based on the order you define the fields.
- Example: If you frequently query for documents based on both "name" and "age", a compound index on (name, age) would be beneficial.

3. Unique Index:

- Enforces data integrity by ensuring each indexed field value is unique across the collection.
- Useful for scenarios where duplicate values for a specific field are not allowed, such as unique IDs or usernames.
- Attempts to insert documents with duplicate values on the unique index field will fail.

4. Multikey Index:

- Primarily used for indexing arrays within documents.
- Creates separate entries in the index for each element within the array field.
- Useful for queries that filter or sort based on values within an array.

5. Text Index:

- Enables full-text search capabilities on specific text fields.
- Allows you to search for documents containing keywords or phrases within the indexed text field.
- Ideal for applications where data search is a core functionality.

6. Geospatial Indexes:

- Designed for efficient querying of geospatial data like coordinates.
- MongoDB offers two options:
    - **2dsphere Index:** Uses spherical geometry for Earth-like distances.
    - **2d Index:** Uses planar geometry for flat surface representations.

- Beneficial for location-based applications or queries involving geospatial coordinates.

7. Hashed Index:

- Less common and typically used internally by MongoDB for specific system fields like the _id field.
- Stores a hash of the indexed field value instead of the actual value.
- Useful for equality comparisons on the hashed field.