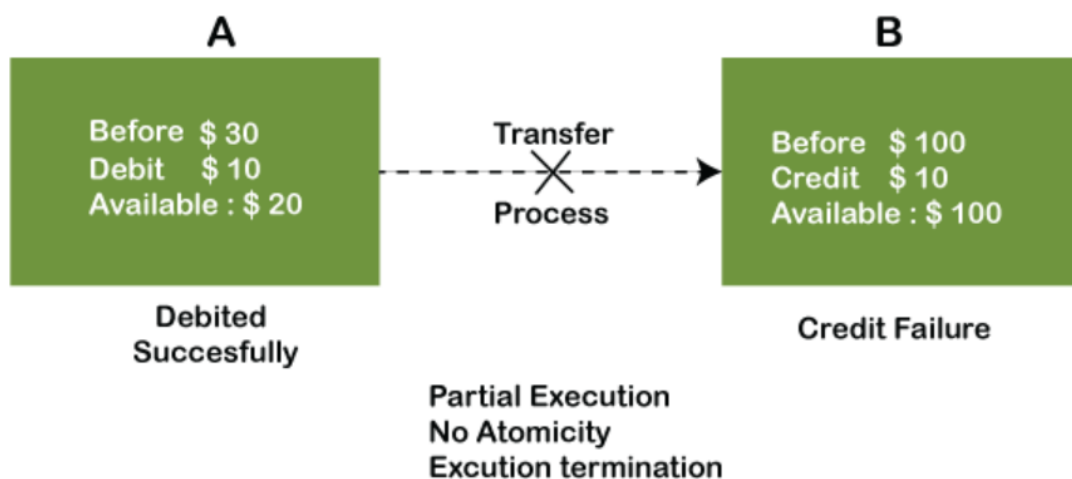


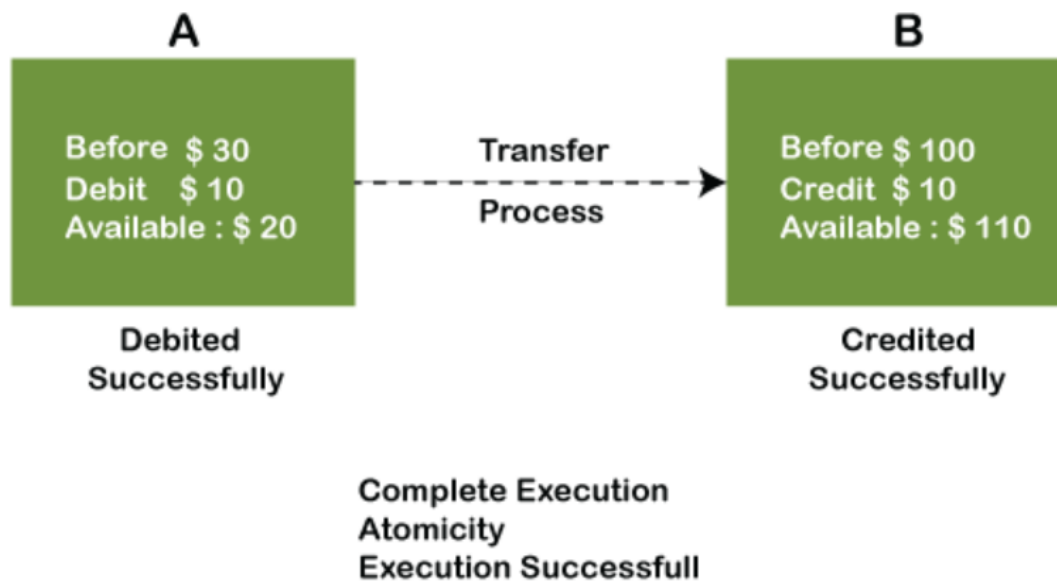
CLASS 8: ACIDS AND INDEXES

ATOMICITY:

In MongoDB, atomicity refers to the property of operations being either fully completed or not at all within a single transaction or document update.

MongoDB uses the concept of atomic operations at the document level. This means that for operations like updates or inserts on a single document, MongoDB ensures that these operations are atomic. If multiple operations are part of a transaction, MongoDB provides support for multi-document transactions that ensure all operations within the transaction are either committed or rolled back as a whole. This ensures data integrity and consistency, preventing partial updates that could lead to inconsistent states in the database. MongoDB's support for atomic operations and transactions helps maintain the reliability and accuracy of data across complex operations and distributed systems.

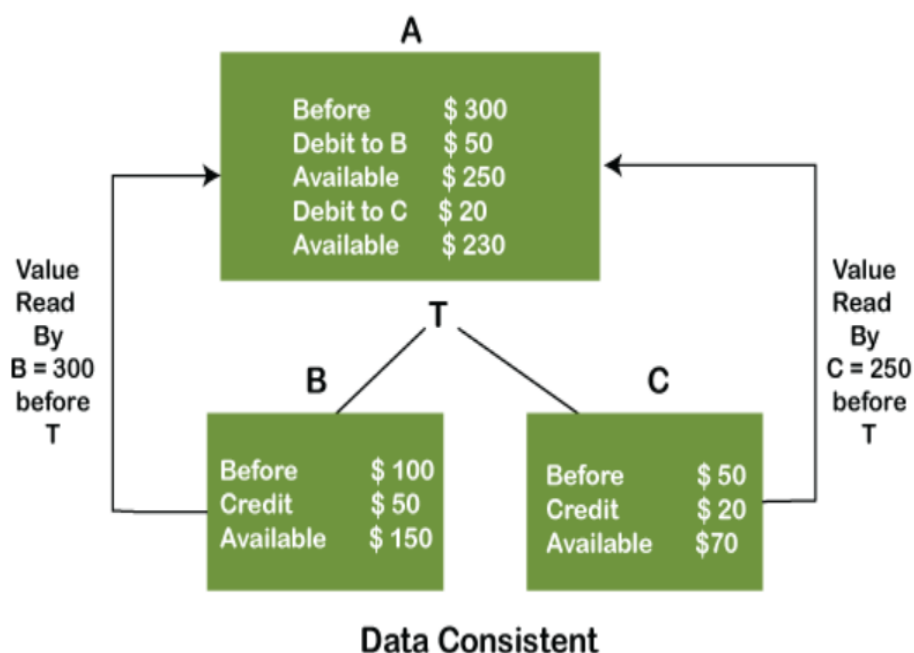




CONSISTENCY:

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:



EVENTUAL CONSISTENCY:

- Not consistent at that moment but gradually

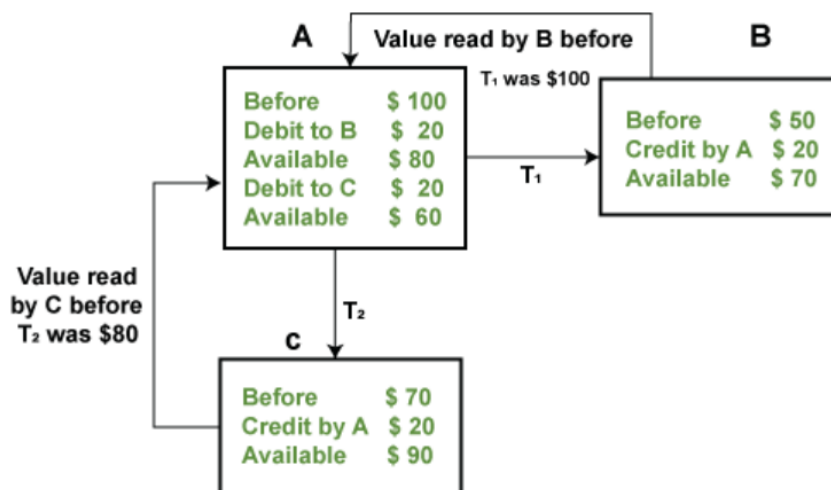
EXAMPLE:

If Virat Kohli posts in instagram, It wont be available at that moment to all.

ISOLATION:

In MongoDB, isolation refers to the property that ensures transactions or operations are executed independently of each other, without interference or dependency. MongoDB provides isolation through its transaction model, which allows multiple operations within a transaction to be executed in a way that each transaction is isolated from others until it is committed. This means changes made within one transaction are not visible to other transactions until the transaction commits, ensuring that transactions are executed as if they were the only operations running on the database. MongoDB's support for multi-document transactions further enhances isolation by allowing developers to perform complex operations across multiple documents or collections while ensuring each transaction's changes are isolated until they are finalized. This isolation mechanism helps maintain data integrity and consistency, ensuring reliable and predictable behavior across concurrent transactions in MongoDB databases.

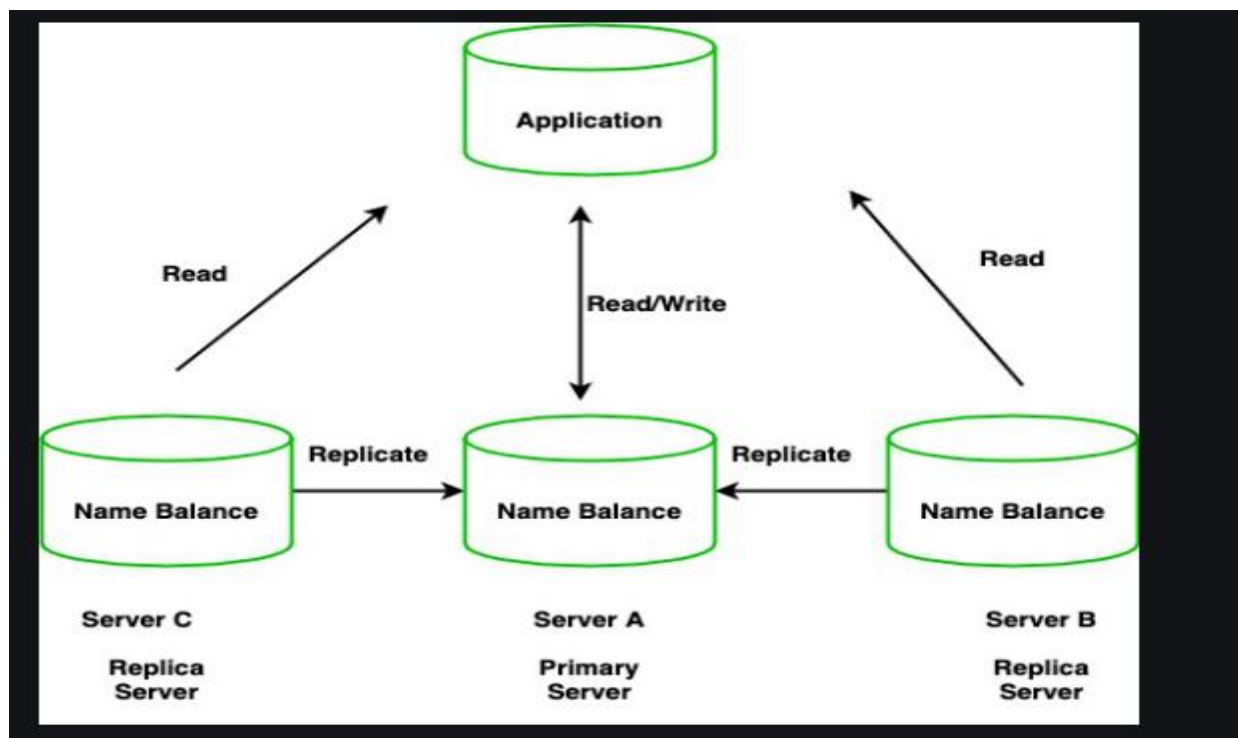
Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently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



Isolation - Independent execution of T1 & T2 by A

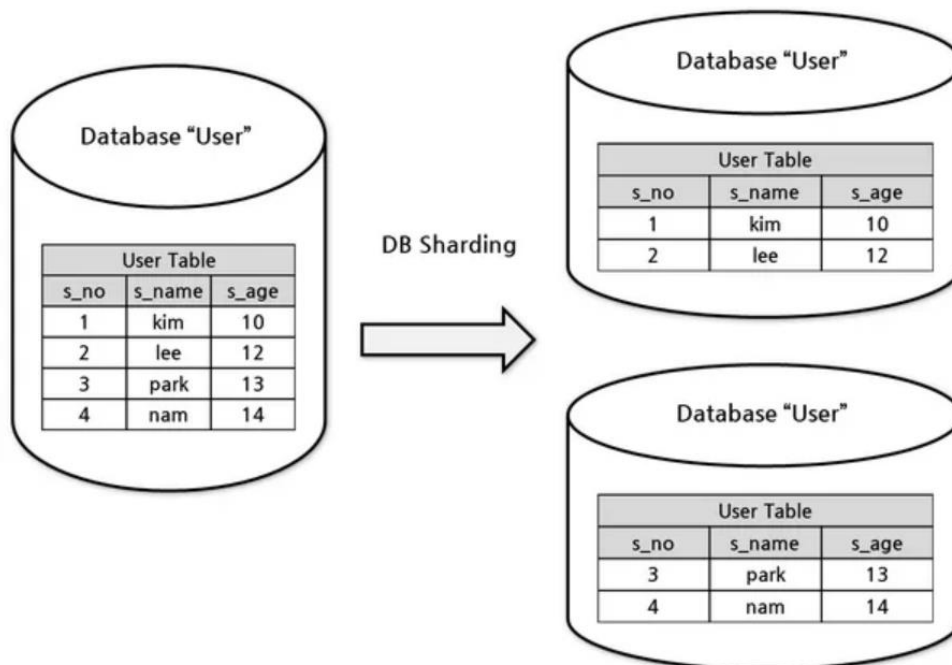
REPLICATION(MASTER-SLAVE):

In MongoDB, replication with master-slave architecture refers to the process where data from a primary node (master) is asynchronously copied to one or more secondary nodes (slaves). The primary node handles all write operations and propagates these changes to the secondary nodes, ensuring data redundancy and fault tolerance. This replication mechanism enhances availability and scalability by allowing secondary nodes to serve read operations independently, thereby distributing the workload. In case of a primary node failure, MongoDB can automatically elect a new primary from the available secondary nodes, maintaining continuous database operations. MongoDB's replication also supports features like delayed replication, where data changes can be applied to secondary nodes with a specified delay, providing protection against data corruption or accidental deletions. Overall, MongoDB's master-slave replication architecture ensures data durability, high availability, and efficient scaling of read operations in distributed database environments.



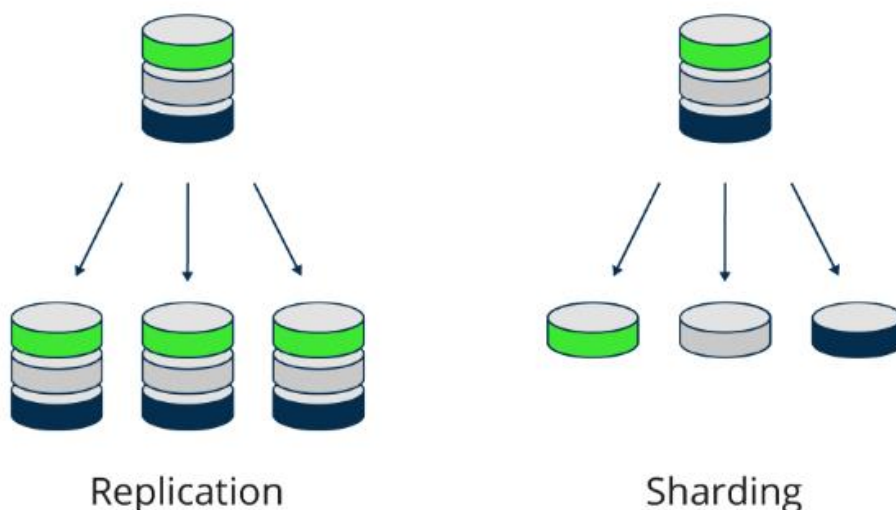
SHARDING:

In MongoDB, sharding is a technique used for horizontal sealing of data across multiple machines or nodes in a distributed database system. It involves partitioning data into smaller chunks called shards, which are distributed across different servers or clusters. Each shard contains a subset of the dataset, allowing MongoDB to manage larger datasets and handle increased throughput by distributing read and write operations across multiple shards. MongoDB uses a sharded cluster architecture where data distribution and balancing are managed by the mongos routers, which direct queries to the appropriate shards based on a shard key. This approach improves performance and scalability, as queries can be executed in parallel across multiple shards, reducing response times and increasing overall system capacity. Sharding in MongoDB also supports automatic data migration and rebalancing, ensuring even distribution of data and efficient utilization of resources as the dataset grows or workload changes.



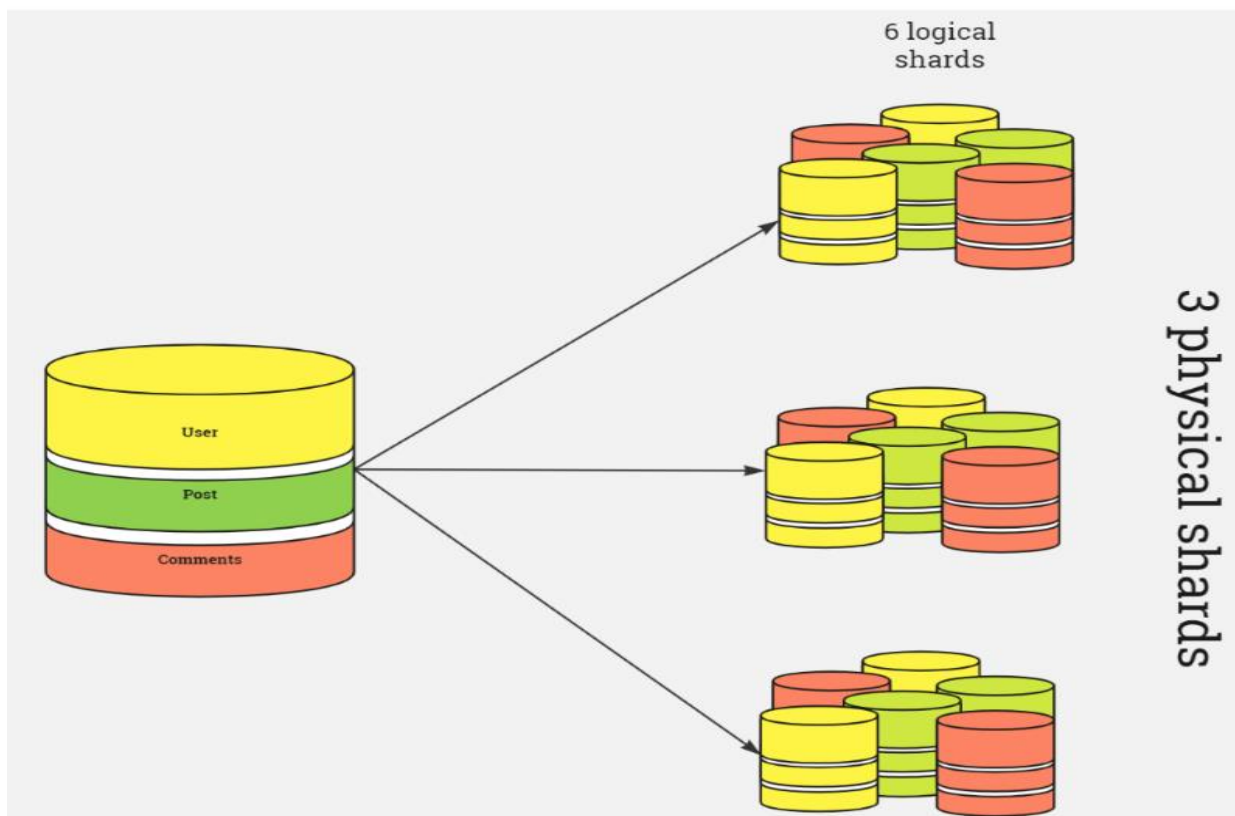
REPLICATION VS SHARDING:

In MongoDB, replication and sharding are two distinct strategies for scaling and managing data in distributed database environments. Replication involves copying data from a primary node to one or more secondary nodes (slaves) to ensure data redundancy, fault tolerance, and high availability. It enhances reliability by allowing secondary nodes to serve read operations independently and enabling automatic failover if the primary node fails. On the other hand, sharding focuses on horizontal scaling by partitioning data into smaller subsets called shards, which are distributed across different servers or clusters. Each shard contains a portion of the dataset, and MongoDB's sharded cluster architecture ensures efficient distribution of read and write operations across shards based on a shard key. While replication ensures data redundancy and availability, sharding enhances scalability and performance by enabling parallel execution of queries across multiple shards. Together, replication and sharding in MongoDB provide robust solutions for managing large-scale data and supporting demanding workloads in distributed database environments.



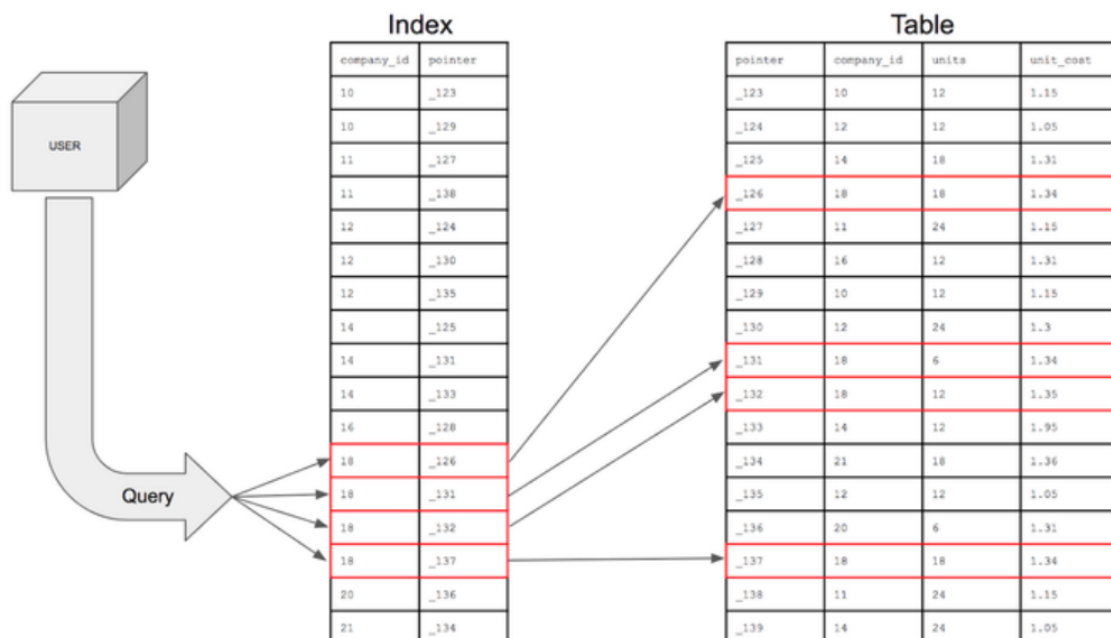
REPLICATION+SHARDING:

In MongoDB, replication and sharding are complementary strategies used together to achieve both high availability and scalability in distributed database systems. Replication involves copying data from a primary node to multiple secondary nodes (slaves), ensuring data redundancy and fault tolerance. This setup enhances reliability by allowing secondary nodes to serve read operations independently and providing automatic failover in case of primary node failure. Sharding, on the other hand, focuses on horizontal scaling by partitioning data into smaller chunks called shards, which are distributed across different servers or clusters based on a shard key. This allows MongoDB to manage large datasets and handle increased throughput by distributing read and write operations across multiple shards. Together, replication and sharding in MongoDB enable systems to achieve both high availability through data redundancy and scalability through distributed data management, ensuring robust performance and reliability for demanding applications.



INDEXES:

In MongoDB, indexes are data structures that store a small portion of the collection's data in an easy-to-traverse form. They improve the efficiency of read operations by allowing MongoDB to quickly locate documents based on the indexed fields. MongoDB supports various types of indexes, including single field, compound, multikey, geospatial, text, hashed, and wildcard indexes, each optimized for different query patterns. Indexes can significantly speed up queries by reducing the number of documents MongoDB needs to examine during query execution. However, they also impose overhead on write operations because MongoDB must update indexes whenever documents are added, modified, or removed. Properly designed and maintained indexes are crucial for optimizing performance in MongoDB databases, balancing query speed with the impact on write operations based on the specific workload and access patterns of the application.



TYPES OF INDEXES:

Basic Index Types

- **Single Field Index:**

- Indexes a single field within a document.
- Example: `db.collection.createIndex({ field1: 1 })`

- **Compound Index:**

- Indexes multiple fields in a specified order.
- Useful for range-based queries involving multiple fields.
- Example: `db.collection.createIndex({ field1: 1, field2: -1 })`

- **Multikey Index:**

- Indexes array elements individually.
- Enables efficient queries on array elements.
- Example: `db.collection.createIndex({ arrayField: 1 })`

Specialized Index Types

- **Text Index:**

- Indexes text content for full-text search capabilities.
- Supports text search operators like `$text` and `$search`.
- Example: `db.collection.createIndex({ text: "text" })`

- **Geospatial Index:**

- Indexes geospatial data (coordinates) for efficient proximity-based queries.
- Supports 2dsphere and 2d indexes for different use cases.
- Example: `db.collection.createIndex({ location: "2dsphere" })`

- **Hashed Index:**

- Creates a hashed index for the specified field.
- Primarily used for the `_id` field for performance optimization.
- Example: `db.collection.createIndex({ _id: "hashed" })`

Additional Considerations

- **Sparse Indexes:**

- Only index documents where the indexed field exists.
- Can improve performance for sparse datasets.

- **Unique Indexes:**

- Ensure that the indexed field has unique values across all documents.

- **TTL Indexes:**

- Automatically expire documents after a specified time.

Choosing the right index type depends on your specific data structure, query patterns, and performance requirements. Careful index design can significantly improve query performance, but excessive indexing can impact write performance.

Would you like to delve deeper into a specific index type or discuss index creation strategies for a particular use case?