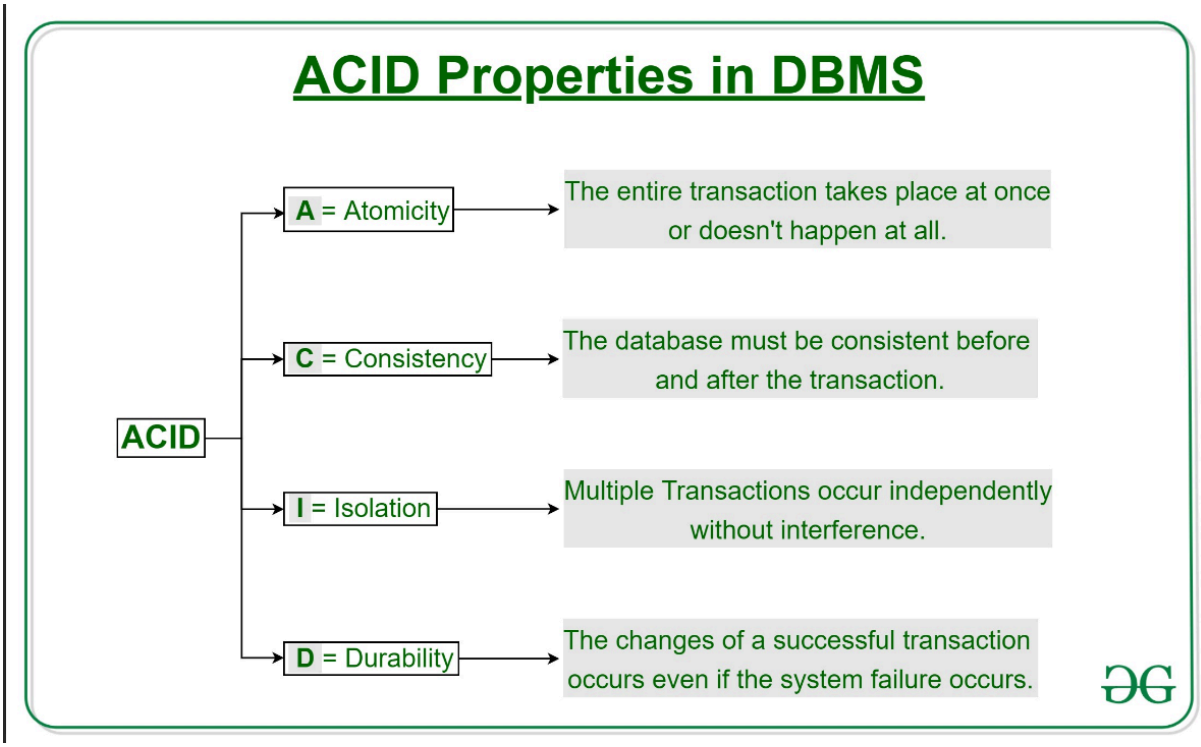
ACID & INDEXES



ACID in MongoDB:

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that ensure database transactions are processed reliably. MongoDB supports a subset of ACID properties, which are:

1. **Atomicity:** MongoDB ensures that database operations are executed as a single, all-or-nothing unit of work. If any part of the operation fails, the entire operation is rolled back.
2. **Consistency:** MongoDB ensures that the database remains in a consistent state, even in the event of a failure. This means that the database will always be in a valid state, even if an operation fails.
3. **Isolation**: MongoDB uses a document-level locking mechanism to ensure that concurrent operations on the same document are isolated from each other.
4. **Durability:** MongoDB ensures that once a write operation is confirmed, the data is persisted to disk and will not be lost in the event of a failure.

Indexing in MongoDB:

Indexing is a crucial aspect of MongoDB performance optimization. An index is a data structure that improves the speed of query operations by providing a quick way to locate specific data. MongoDB supports several types of indexes, including:

1. **Single-field indexes:** An index on a single field, such as a username or email address.
2. **Compound indexes**: An index on multiple fields, such as a combination of username and email address.
3. **Multi-key indexes:** An index on an array field, such as a list of tags or categories.
4. **Text indexes**: An index on a text field, optimized for full-text search queries.
5. **Hashed indexes**: An index on a field that uses a hash function to map the field values to a fixed-size string.

Benefits of Indexing in MongoDB:

Indexing provides several benefits, including:

1. **Improved query performance:** Indexes allow MongoDB to quickly locate specific data, reducing the time it takes to execute queries.
2. **Faster data retrieval:** Indexes enable MongoDB to retrieve data more efficiently, reducing the amount of data that needs to be scanned.
3. **Reduced CPU usage:** By using an index, MongoDB can reduce the CPU resources required to execute queries.
4. **Improved data integrity**: Indexes can help ensure data consistency and prevent duplicate values.

How to Create an Index in MongoDB:

To create an index in MongoDB, you can use the createIndex() method in the MongoDB shell or through a MongoDB driver. For example:

```
db.collection.createIndex({ username: 1 })
```

This creates a single-field index on the username field in the collection collection.

ATOMICITY:

Atomicity in MongoDB refers to the ability of the database to ensure that database operations are executed as a single, all-or-nothing unit of work. This means that if any part of the operation fails, the entire operation is rolled back, and the database is returned to its previous state.

Importance of Atomicity in MongoDB

Atomicity is crucial in MongoDB because it ensures:

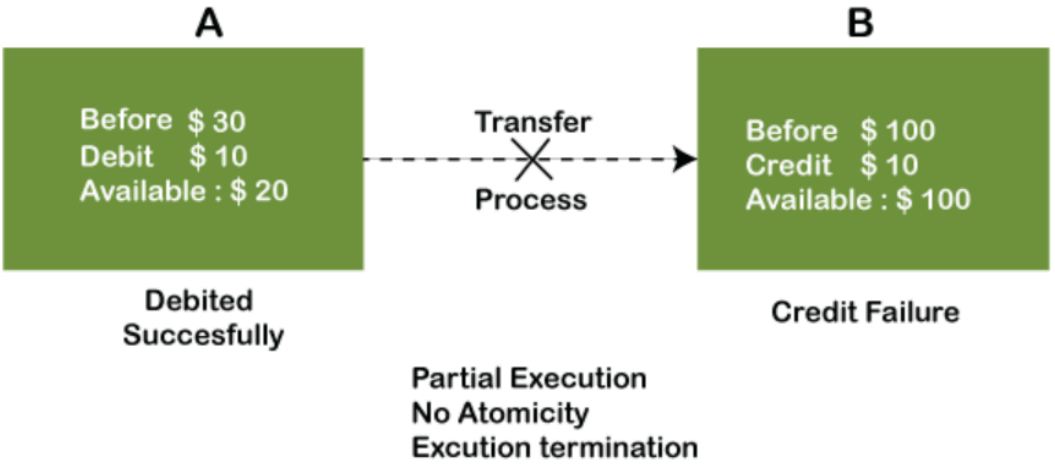
1. **Data Consistency:** Atomicity ensures that the database remains in a consistent state, even in the event of a failure.
2. **Data Integrity:** Atomicity ensures that data is not corrupted or partially updated, which can lead to data inconsistencies.
3. **Reliability**: Atomicity ensures that MongoDB operations are reliable and can be trusted to execute correctly, even in the presence of failures.

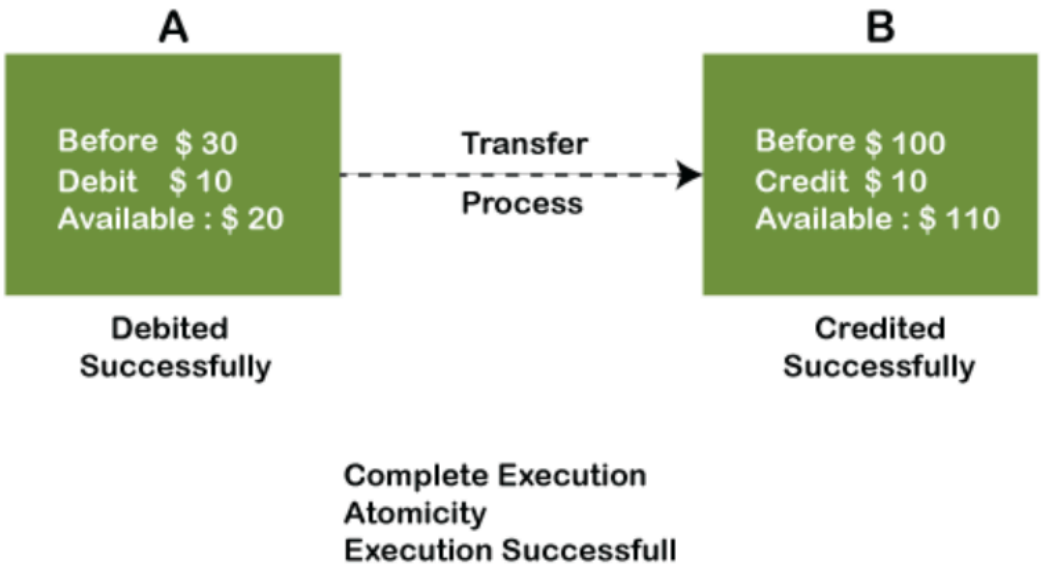
Types of Atomicity in MongoDB

MongoDB supports two types of atomicity:

1. **Document-level atomicity:** MongoDB ensures that operations on a single document are atomic. This means that if an update operation fails, the entire document is rolled back to its previous state.

2. **Collection-level atomicity**: MongoDB ensures that operations on a single collection are atomic. This means that if an operation fails, the entire collection is rolled back to its previous state.

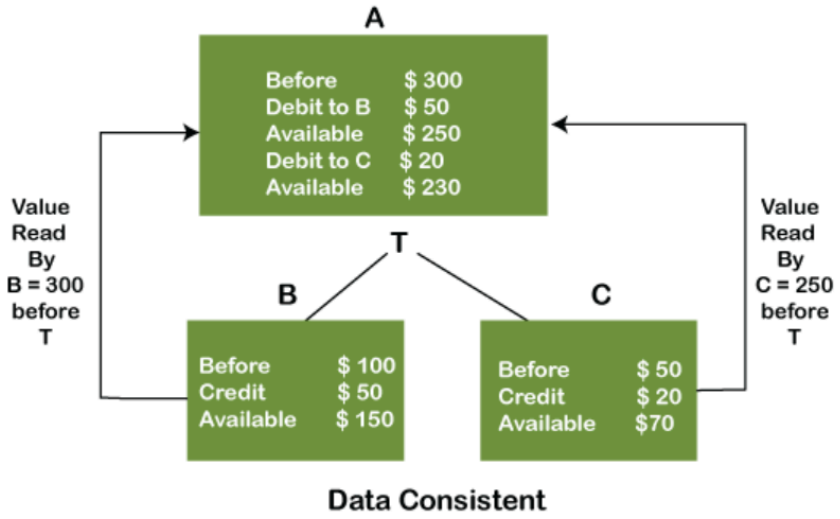




CONSISTENCY:

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:



Types of Consistency in MongoDB

MongoDB supports several types of consistency:

1. **Strong Consistency:** Strong consistency ensures that all nodes in the replica set have the same data values for a given document. This is achieved through the use of a primary node that accepts writes and replicates them to secondary nodes.
2. **Weak Consistency:** Weak consistency allows for temporary inconsistencies between nodes in the replica set. This can occur during network partitions or when a node is unavailable.

3. **Eventual Consistency**: Eventual consistency ensures that all nodes in the replica set will eventually converge to the same state, even if there are temporary inconsistencies.

Benefits of Consistency in MongoDB:

Consistency in MongoDB provides several benefits, including:

1. **Data Integrity:** Consistency ensures that data is accurate and reliable.
2. **High Availability:** Consistency ensures that the database remains available even in the presence of failures or network partitions.
3. **Scalability**: Consistency enables MongoDB to scale horizontally, allowing for high performance and throughput.

REPLICATION:

Replication in MongoDB is the process of maintaining multiple copies of data in different locations, typically across multiple nodes in a cluster. This ensures that data is available and accessible even in the event of node failures, network partitions, or other disruptions.

Types of Replication in MongoDB:

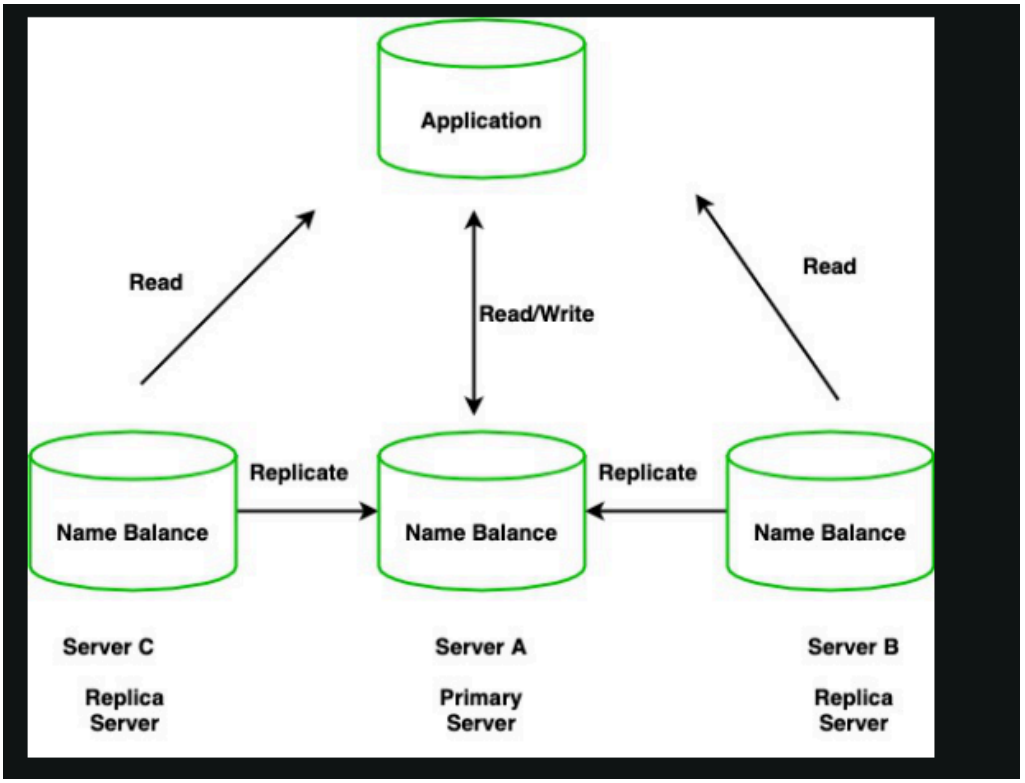
MongoDB supports several types of replication:

1. **Master-Slave Replication**: In this type of replication, one node (the master) accepts writes and replicates them to one or more slave nodes. The master node is responsible for handling writes, while the slave nodes are read-only.
2. **Master-Master Replication**: In this type of replication, all nodes in the cluster are equal peers, and each node can accept writes and replicate them to other nodes.
3. **Multi-Source Replication:** In this type of replication, multiple nodes can accept writes and replicate them to other nodes, providing high availability and fault tolerance.

Benefits of Replication in MongoDB:

Replication in MongoDB provides several benefits, including:

1. **High Availability:** Replication ensures that data is available even in the event of node failures or network partitions.
2. **Fault Tolerance:** Replication provides fault tolerance, allowing the cluster to continue operating even if one or more nodes fail.
3. **Improved Read Performance:** Replication allows reads to be distributed across multiple nodes, improving read performance and reducing latency.
4. **Disaster Recovery:** Replication provides a mechanism for disaster recovery, allowing data to be restored from a secondary node in the event of a primary node failure.



SHARDING:

Sharding in MongoDB is a method of horizontal partitioning of data across multiple machines, called shards, to improve the scalability and performance of a MongoDB cluster. Sharding allows MongoDB to distribute data across multiple nodes, making it possible to store large amounts of data and handle high traffic.

How Sharding Works in MongoDB

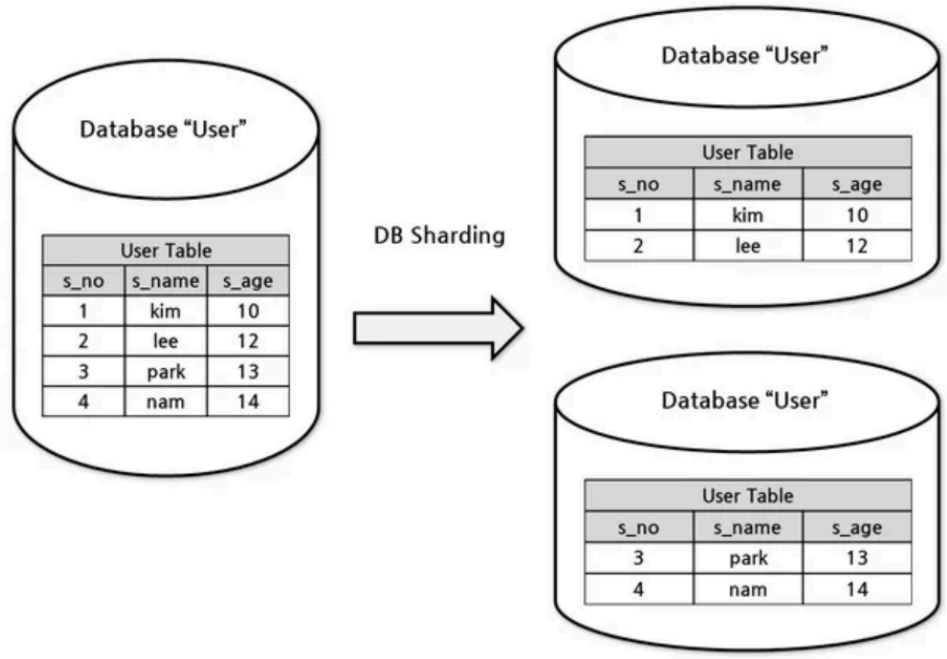
Here's a high-level overview of how sharding works in MongoDB:

1. **Shard Key**: A shard key is a field or set of fields in a document that determines which shard a document belongs to.
2. **Shard**: A shard is a single node or a group of nodes that stores a portion of the data.
3. **Chunk**: A chunk is a contiguous range of shard key values that are stored on a single shard.
4. **Shard Split**: When a chunk grows beyond a certain size, MongoDB splits it into two smaller chunks, and the data is rebalanced across the shards.
5. **Shard Migration**: When a shard becomes too large or too small, MongoDB migrates chunks to other shards to maintain a balanced distribution of data.

Types of Sharding in MongoDB

MongoDB supports two types of sharding:

1. **Range-Based Sharding**: In range-based sharding, MongoDB divides the data into chunks based on a range of values in the shard key.
2. **Hash-Based Sharding**: In hash-based sharding, MongoDB uses a hash function to map the shard key to a specific shard.

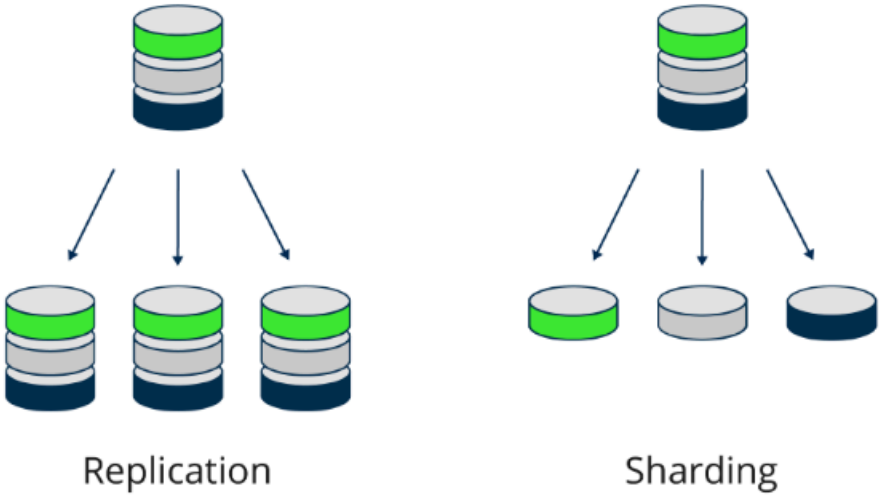


Benefits of Sharding in MongoDB

Sharding in MongoDB provides several benefits, including:

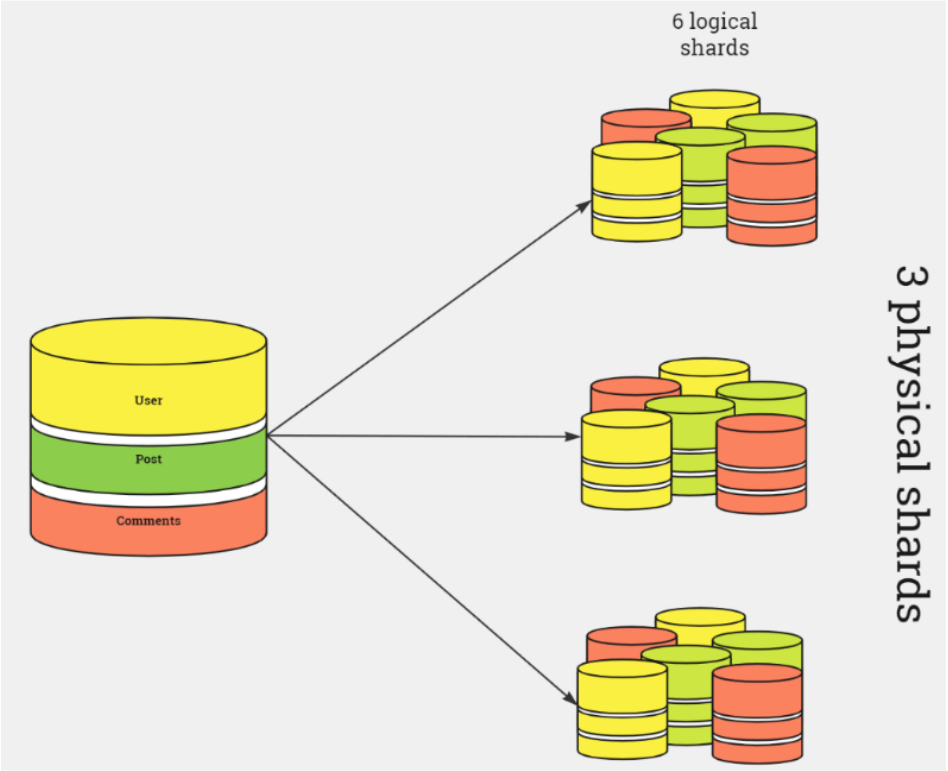
1. **Horizontal Scaling**: Sharding allows MongoDB to scale horizontally, making it possible to handle large amounts of data and high traffic.
2. **Improved Performance**: Sharding improves performance by distributing the data across multiple nodes, reducing the load on individual nodes.
3. **High Availability:** Sharding provides high availability by ensuring that data is available even if one or more nodes fail.
4. **Flexible Data Distribution:** Sharding allows for flexible data distribution, making it possible to distribute data based on specific criteria, such as location or usage patterns.

Replication VS Sharding:



	Replication	Sharding
Purpose	Ensure high availability and fault tolerance by maintaining multiple copies of data	Scale horizontally by distributing data across multiple nodes
Data Distribution	Data is replicated across multiple nodes, but each node has a complete copy of the data	Data is split into smaller chunks and distributed across multiple nodes, each node has a portion of the data
Node Roles	Primary node accepts writes, secondary nodes replicate data from primary	Each node is a shard, responsible for a portion of the data
Write Operations	Writes are accepted by the primary node and replicated to secondary nodes	Writes are distributed across multiple shards, each shard is responsible for a portion of the write
Read Operations	Reads can be directed to any node, primary or secondary	Reads are directed to the shard that contains the requested data
Scalability	Limited scalability, as each node must maintain a complete copy of the data	Highly scalable, as data is distributed across multiple nodes
Performance	Improved read performance, as reads can be directed to any node	Improved write performance, as writes are distributed across multiple nodes
Data Consistency	Strong consistency, as all nodes have a complete copy of the data	Eventual consistency, as data is distributed across multiple nodes
Complexity	Relatively simple to implement and manage	More complex to implement and manage, requires careful planning and configuration

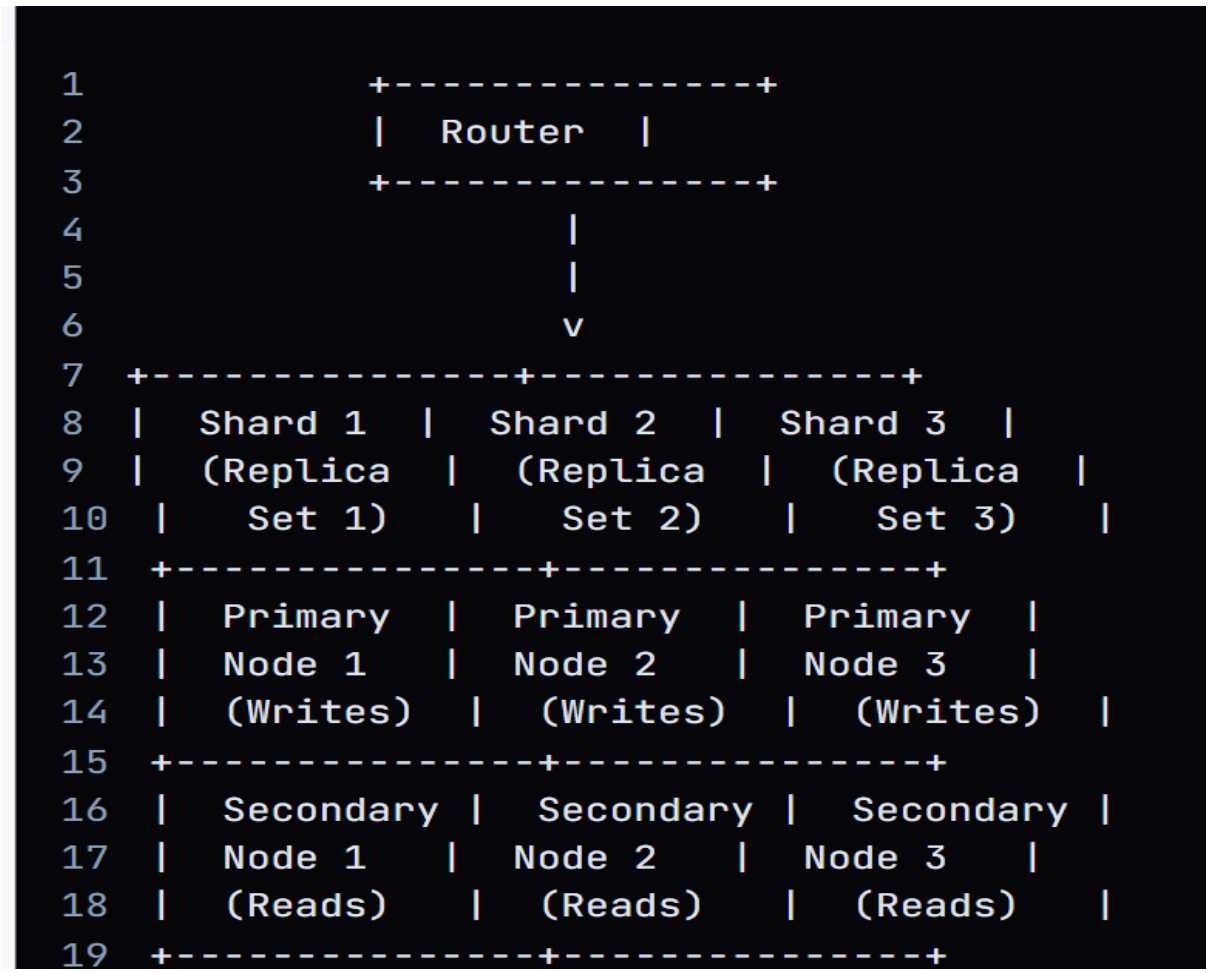
REPLICATION + SHARDING



How it Works

Here's a high-level overview of how replication and sharding work together in MongoDB:

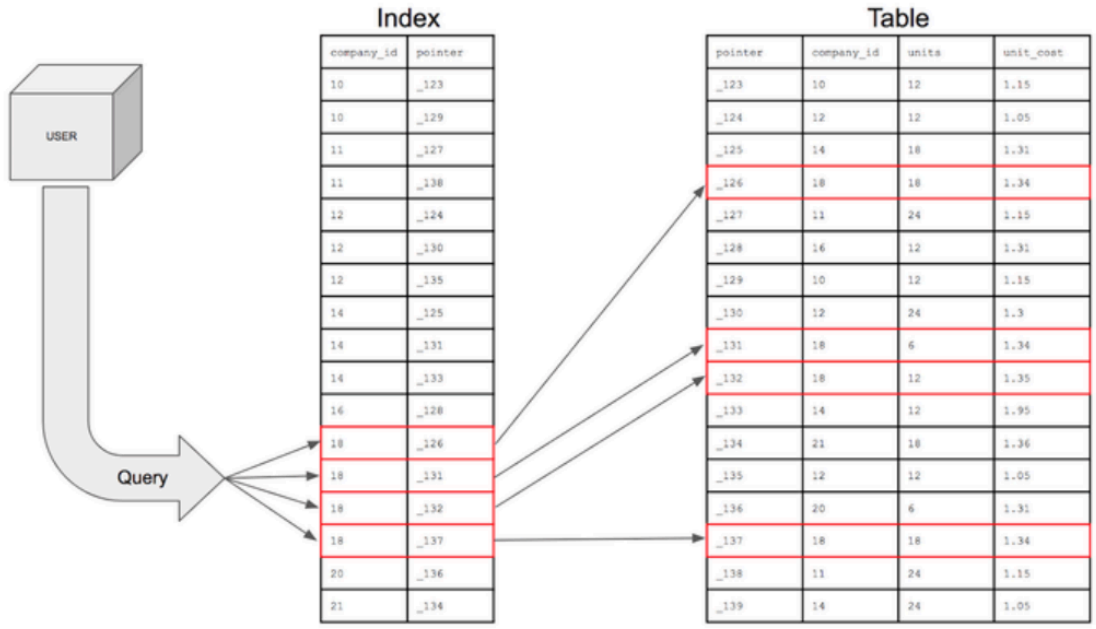
1. Sharding: Data is split into smaller chunks and distributed across multiple shards, each responsible for a portion of the data.
2. Replication: Each shard is replicated across multiple nodes, with one primary node and one or more secondary nodes.
3. Primary Node: The primary node in each shard accepts writes and replicates data to secondary nodes.
4. Secondary Nodes: Secondary nodes in each shard replicate data from the primary node and can serve reads.
5. Shard Key: A shard key is used to determine which shard a document belongs to.
6. Chunk: A chunk is a contiguous range of shard key values that are stored on a single shard.



Here's an example architecture that combines replication and sharding in MongoDB in the above figure.

INDEXES

In MongoDB, an index is a data structure that improves the speed of query operations by providing a quick way to locate specific data. Indexes are similar to the indexes in a book, which allow you to quickly find specific pages or topics.



Types of Indexes in MongoDB

MongoDB supports several types of indexes, including:

1. **Single Field Index:** An index on a single field in a document.
2. **Compound Index**: An index on multiple fields in a document.
3. **Multi-Key Index**: An index on an array field, which allows MongoDB to efficiently query arrays.
4. **Text Index:** An index on a text field, which allows MongoDB to efficiently query text data.
5. **Hashed Index:** An index on a field that uses a hash function to map the field's values to a unique index key.

How Indexes Work in MongoDB

Here's how indexes work in MongoDB:

1. **Index Creation:** When you create an index, MongoDB builds a data structure that contains the indexed field(s) and a pointer to the corresponding document.
2. **Query Optimization:** When you execute a query, MongoDB's query optimizer uses the index to quickly locate the required data.
3. **Index Scanning:** MongoDB scans the index to find the required data, which reduces the number of documents that need to be scanned.
4. **Document Retrieval**: Once the required data is located, MongoDB retrieves the corresponding document(s) from the collection.

Common Indexing Use Cases in MongoDB

Here are some common indexing use cases in MongoDB:

1. **Primary Key Index:** Create an index on the primary key field to improve query performance.
2. **Unique Index**: Create a unique index on a field to ensure that each value is unique.
3. **Text Search Index:** Create a text index on a text field to enable efficient text search queries.
4. **Range Query Index:** Create an index on a field that is frequently used in range queries to improve query performance.

Indexing Strategies in MongoDB

Here are some indexing strategies to keep in mind:

1. **Index Frequently Queried Fields:** Index fields that are frequently used in queries to improve query performance.
2. **Use Compound Indexes**: Use compound indexes to improve query performance when querying multiple fields.

3. **Avoid Over-Indexing**: Avoid creating too many indexes, as this can increase storage overhead and slow down write operations.
4. **Monitor Index Performance**: Monitor index performance regularly to identify bottlenecks and optimize indexing strategies.

Benefits of Indexes in MongoDB

Indexes provide several benefits, including:

1. **Improved Query Performance**: Indexes can significantly improve query performance by reducing the number of documents that need to be scanned.
2. **Faster Data Retrieval:** Indexes allow MongoDB to quickly locate specific data, which reduces the time it takes to retrieve data.
3. **Efficient Query Execution:** Indexes enable MongoDB to execute queries more efficiently, which reduces the load on the database.
4. **Reduced Disk I/O**: Indexes can reduce disk I/O operations, which improves overall system performance.

Inserting documents into collection:

```
test> use db
switched to db db
db> db.products.insertMany([
...   { _id: 1, name: "Product A", category: "Electronics", price: 99.99, tags: ["electronics", "gadget"] },
...   { _id: 2, name: "Product B", category: "Clothing", price: 49.99, tags: ["clothing", "fashion"] },
...   { _id: 3, name: "Product C", category: "Electronics", price: 199.99, tags: ["electronics", "gadget"] },
...   { _id: 4, name: "Product D", category: "Books", price: 29.99 }, // No tags
...   { _id: 5, name: "Product E", category: "Electronics", price: 149.99, tags: ["electronics"] }
... ]);
{
  acknowledged: true,
  insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }
}
```

Creating Different Types of Indexes:

By using unique,sparse and compound index we use the following codes:

```
db> db.products.createIndex({ name: 1 }, { unique: true });
name_1
db> db.products.createIndex({ tags: 1 }, { sparse: true });
tags_1
db> db.products.createIndex({ category: 1, price: -1 });
category_1_price_-1
```

To get the indexes we use the below code:

```
db> db.products.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1', unique: true },
  { v: 2, key: { tags: 1 }, name: 'tags_1', sparse: true },
  {
    v: 2,
    key: { category: 1, price: -1 },
    name: 'category_1_price_-1'
  }
]
db>
```