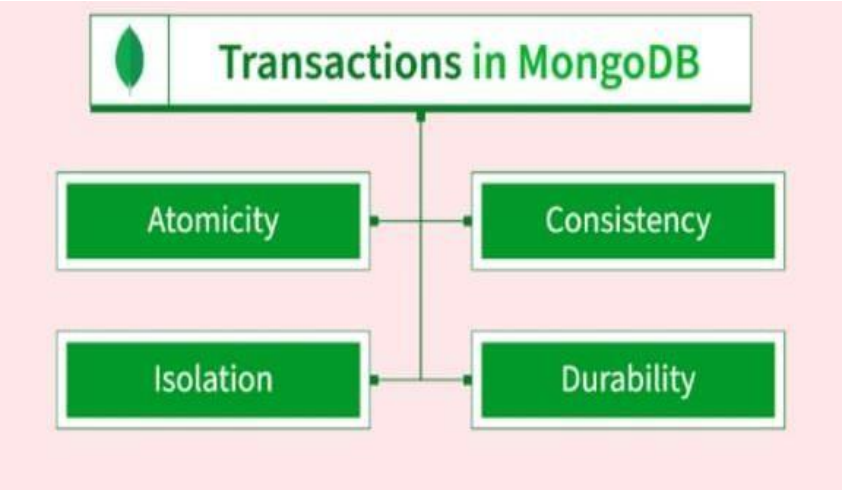
ACID AND INDEXES

In MongoDB, **ACID** and **Indexes** play important roles in managing data.

ACID IN MONGODB:

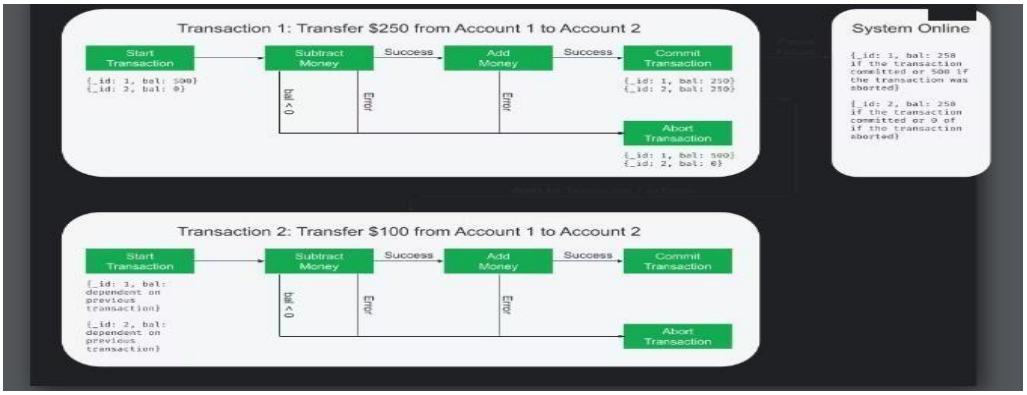
ACID stands for **Atomicity, Consistency, Isolation, and Durability** which are properties that ensure reliable transaction processing in databases.



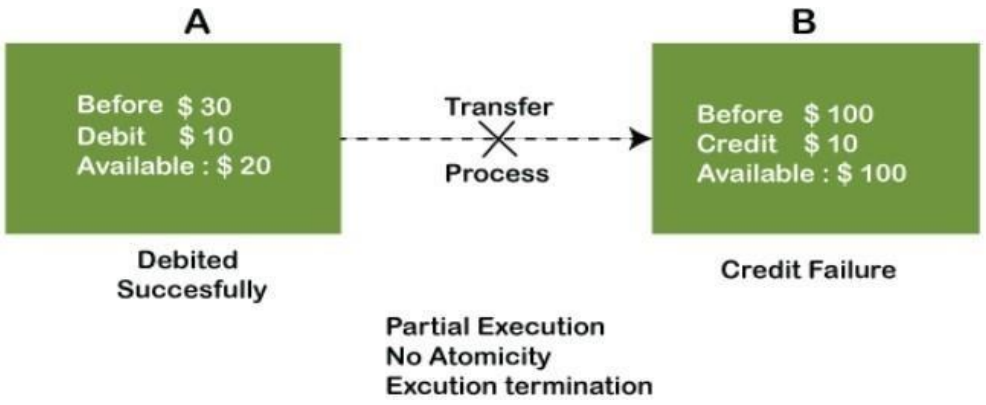
1.Atomicity:

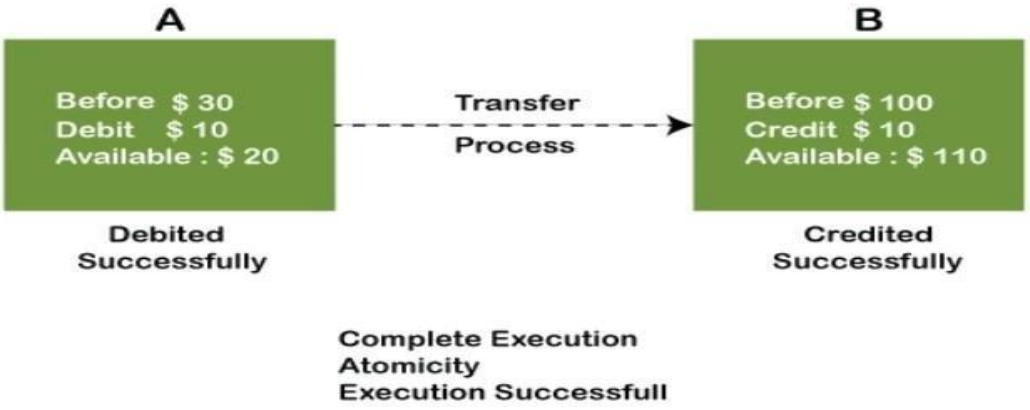
It Ensures that a series of operations within a transaction are completed entirely or not at all. MongoDB provides atomicity on single document operations (e.g., insert, update, delete).

Example 1:-



Example 2:-

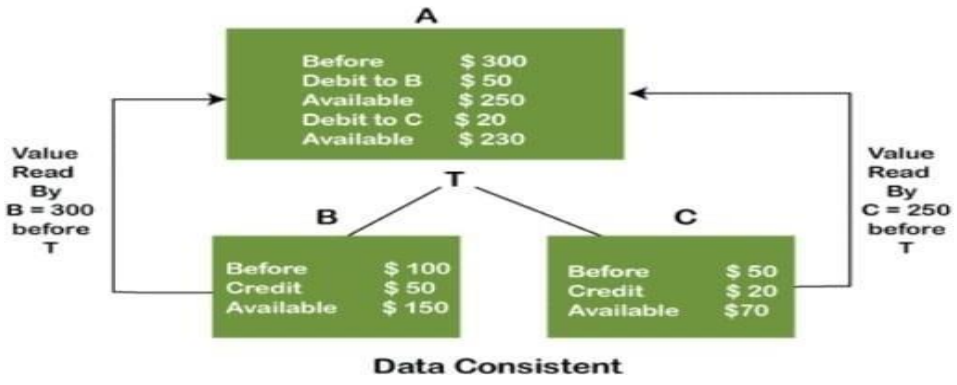




2.Consistency:

It Ensures that a transaction brings the database from one valid state to another, maintaining database invariants. MongoDB enforces schema validation rules to ensure consistency.

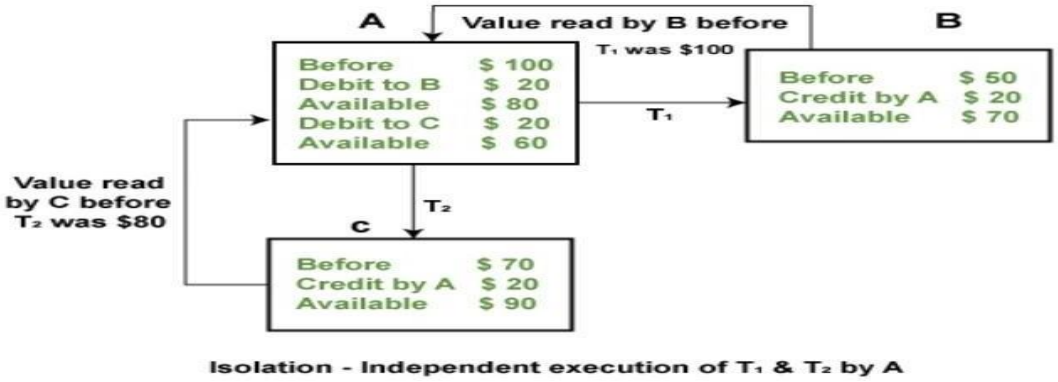
Example:-



3.Isolation:

It Ensures that transactions are securely and independently processed. MongoDB offers isolation at the document level, ensuring operations on a document do not interfere with each other.

Example:-

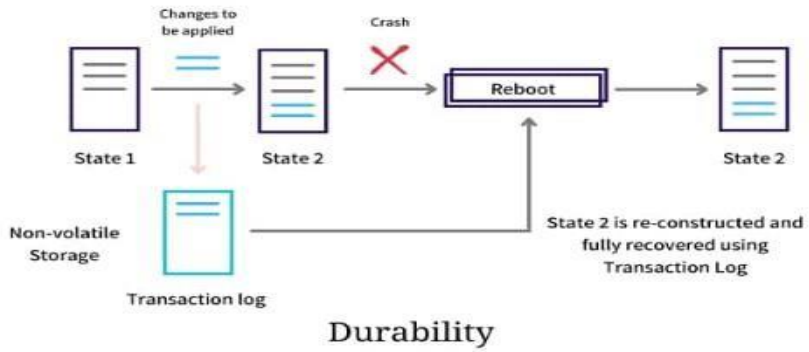


Isolation - Independent execution of T_1 & T_2 by A

4.Durability:

It Ensures that the results of a transaction are permanently stored, even in the case of a system failure. MongoDB uses journaling to provide durability.

Example:-



Durability

Why are ACID transactions important?

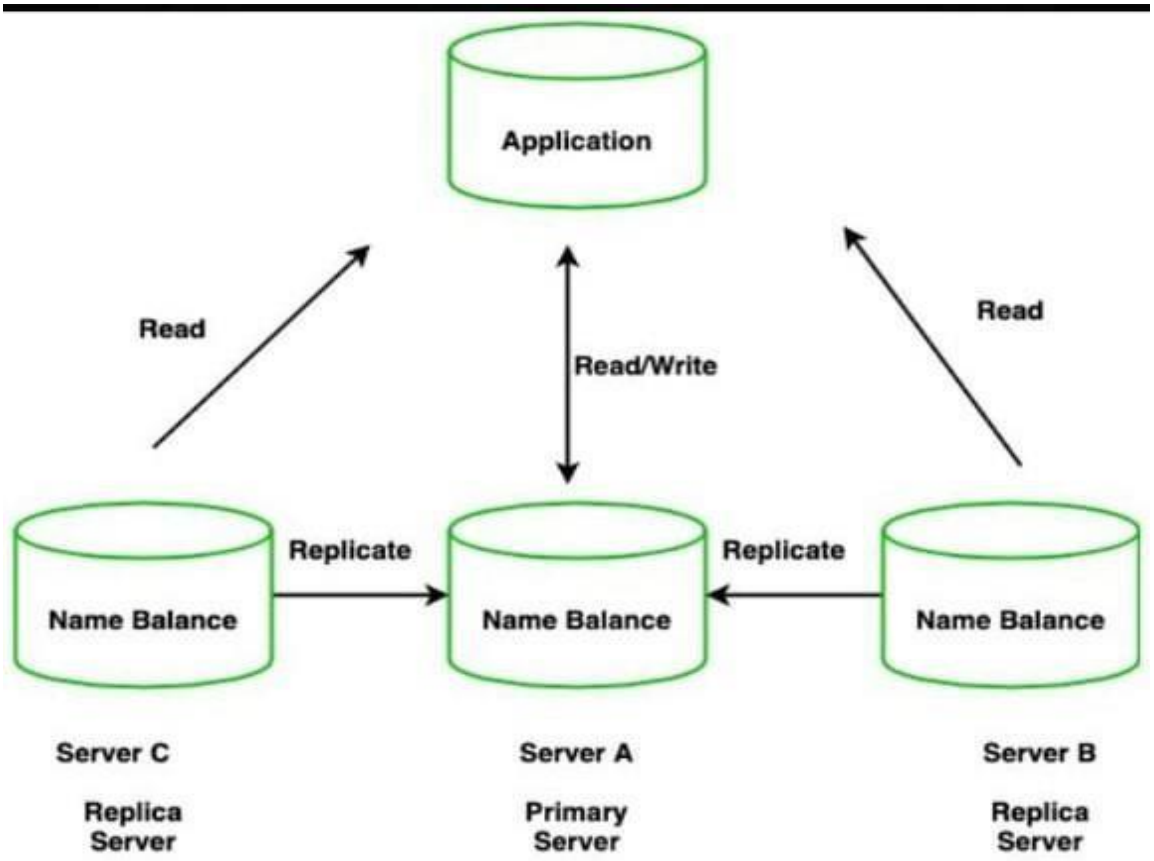
ACID transactions ensure data remains consistent in a database. In data models where related data is split between multiple records or documents, multi-record or multi-document ACID transactions can be critical to an application's success.

How do ACID transactions work in MongoDB?

MongoDB's document model allows related data to be stored together in a single document. The document model, combined with atomic document updates, the need for transactions in a majority of use cases.

1.REPLICATION:

In MongoDB, replication refers to the process of synchronizing data across multiple servers to ensure high availability, redundancy, and data durability. It involves maintaining identical copies of a dataset across multiple MongoDB servers, which are organized in a replica set.



A replica set in MongoDB consists of:

1. Primary: The main server that handles all write operations. All changes to the data are recorded here first.

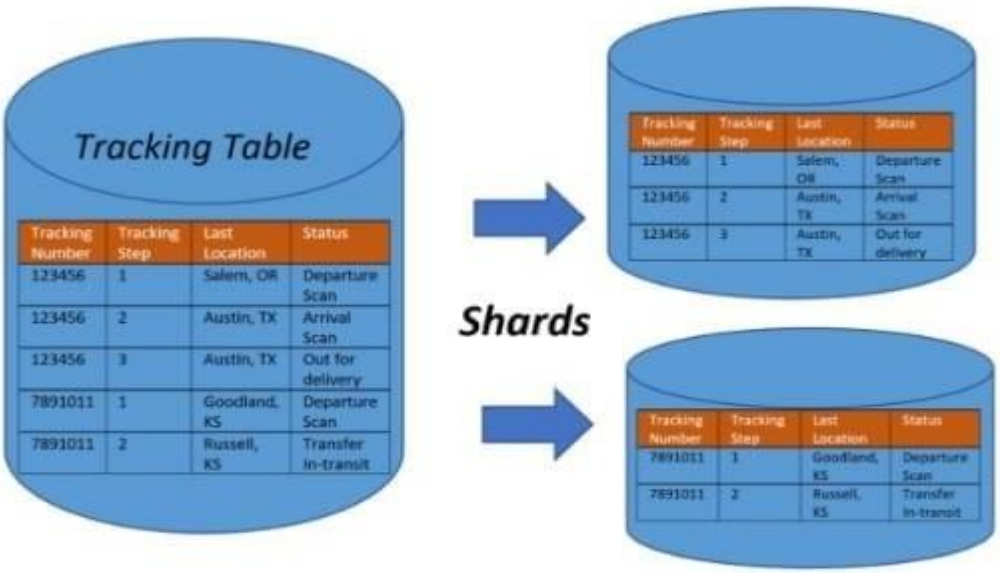
2. Secondary: One or more servers that replicate the data from the primary server. They can handle read operations and can be promoted to primary if the current primary fails.

3. Arbiter: A server that does not hold data but participates to help in deciding the new primary when needed.

MongoDB automatically initiates a failover process to promote a secondary to primary, ensuring minimal downtime and continuous availability.

2,SHARDING:-

In MongoDB, the concept you might be referring to is "sharding." Sharding is a method used to distribute data across multiple servers to support deployments with very large data sets and high throughput operations.



Key concepts of Sharding in MongoDB:

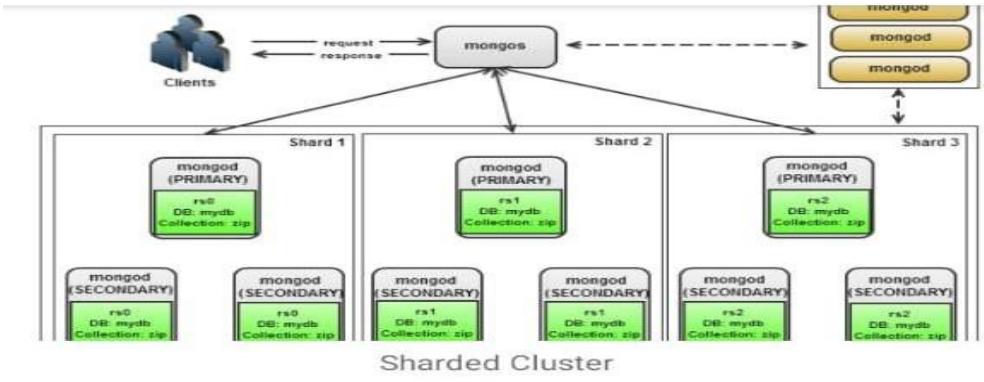
1.Shard: A single MongoDB instance that holds a subset of the sharded data. Each shard can be a replica set to provide high availability.

2.Sharded Cluster: A collection of shards, each holding a subset of the data. Together, they make up the entire dataset.

3.Shard Key: A field or combination of fields that determines how data is distributed across the shards. The choice of shard key can significantly impact the performance and efficiency of the sharded cluster.

REPLICATION VERSUS SHARDING:

Replication and sharding are two different strategies used to manage and scale databases:



Replication vs Sharding in MongoDB

1.Replication:

It Involves creating copies of the same database across multiple servers.

Enhances data availability and fault. If one server fails, the data is still accessible from another server.

It suitable for read-heavy workloads where data consistency is critical. Improves read performance since multiple copies can handle read requests simultaneously.

A primary database that handles writes and multiple secondary databases that handle reads.

2.Sharding:

Involves splitting a large database into smaller, more manageable pieces called shards, each hosted on a separate server.

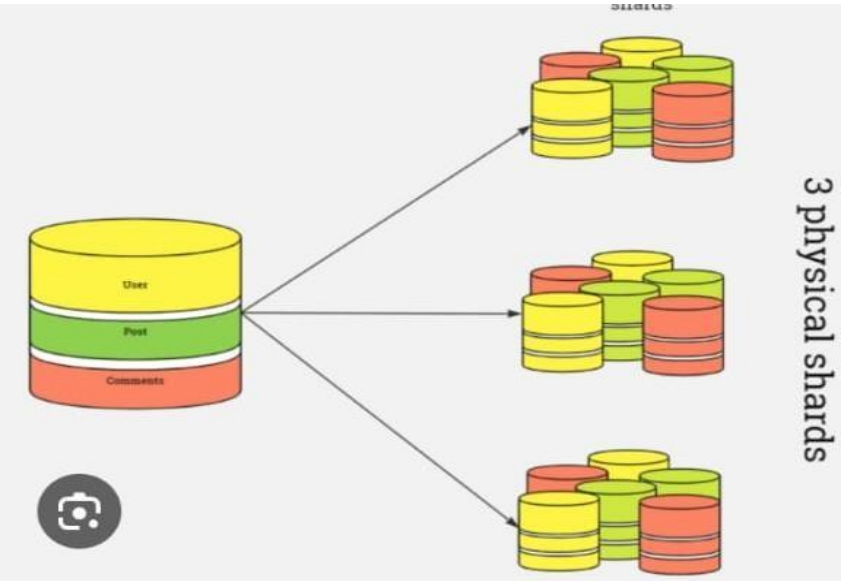
Improves performance and scalability by distributing the data and load across multiple servers.

A user database where each shard contains data for users from specific geographical regions.

Both strategies can be combined to the strengths of each approach, depending on the specific requirements of the application

REPLICATION AND SHARDING:

Using replication and sharding together can optimize database performance and reliability.



Sharding:

Distributes data across multiple servers or nodes.

Improves performance by parallelizing queries and reducing the load on individual servers.

Example:

A social media platform with user data spread across multiple shards based on user ID.

Replication:

Creates copies of the data across multiple servers or nodes.

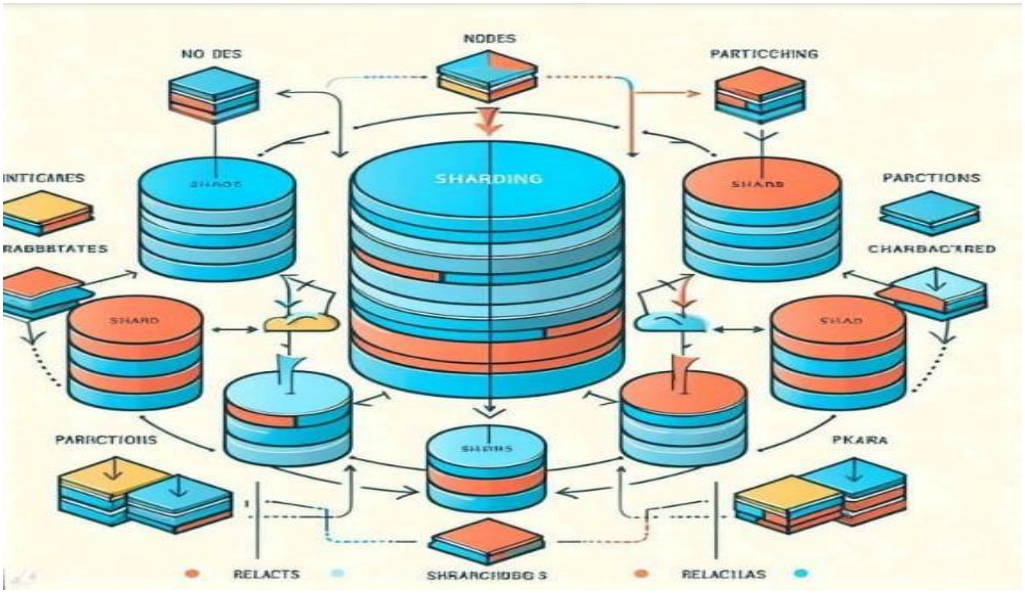
Enhances data availability, fault tolerance, and disaster recovery.

Provides load balancing for read operations.

Example:

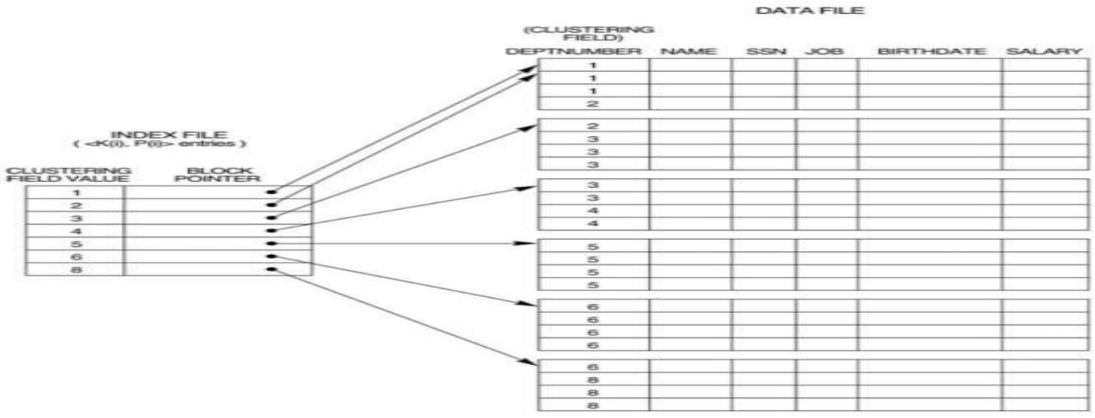
An e-commerce website with replicated data across different geographic regions to ensure availability.

Using both techniques together the strengths of each, providing a robust, scalable, and reliable database solution.



3.INDEXES:

Indexes in MongoDB improve the efficiency of search operations. They work similarly to indexes in relational databases.



BASIC INDEXES TYPES:

1.Single Field Index: Indexes a single field in a document. This is the most common type of index.

db.collection.createIndex({ field: 1 })

The 1 specifies ascending order and -1 would specify descending order.

2.Compound Index: Indexes multiple fields within a document. Useful for queries that sort or filter on multiple fields.

db.collection.createIndex({ field1: 1, field2: -1 })

3.Multikey Index: Indexes array fields. Each element of the array is indexed as a separate entry.

db.collection.createIndex({ arrayField: 1 })

4.Text Index: Supports text search queries on string content. You can index multiple fields for text search.

db.collection.createIndex({ field: "text" })

5.Geospatial Index: Supports geospatial queries on coordinates. Usefulfor location-based searches.

db.collection.createIndex({ location: "2dsphere" })

6.Hashed Index: Hashes the value of a field to support sharding and equality checks.

db.collection.createIndex({ field: "hashed" })

7.Unique Indexes: Ensure that all values in the index are unique. These are often used for fields that must contain unique values, such as primary keys.

8.Partial Indexes: Indexes that only include documents that meet a specified filter criteria. They can be more efficient than full indexes for specific queries.

9.Sparse Indexes: Indexes that only include documents that have the indexed field. This can save space and improve performance when the indexed field is not present in all documents.

10.Wildcard Indexes: Indexes that automatically include all fields in a document. Useful for indexing documents with dynamic schema.

Indexes improve query performance but come with overhead for storage and maintenance.

CREATING DIFFERENT TYPES OF INDEXES IN MONGODB:

Let's define a sample collection ,To insert many of the _id's we use a command

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 }, { _id: 5, name: "Product E", category: "Electronics", price: 149.99, tags: ["electronics"] }]);

```
test> use db
switched to db db
db> db.products.insertMany([ {_id:1,name:"Products A",category:"Electronics",price:99.9,tags:["electronics","gadget"]}, {_id:2,name:"Products B",c
ategory:"Clothing",price:49.9,tags:["clothing","fashion"]}, {_id:3,name:"Products C",category:"Electronics",price:199.9,tags:["electronics","gadge
t"]}, {_id:4,name:"Products D",category:"Books",price:29.9},{_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 }
```

1.Unique index:

Ensures that each value in the indexed field is unique across all documents.

db.products.createIndex({ name: 1 }, { unique: true });

```
}
db> db.products.createIndex({name:1},{unique:true});
name_1
```

2.Sparse index:

Indexes only documents where the specified field exists.

db.products.createIndex({ tags: 1 }, { sparse: true });

```
db> db.products.createIndex({tags:1},{sparse:true});
tags_1
```

3.Compound index:

Indexes multiple fields in a specified order.

db.products.createIndex({ category: 1, price: -1 });

```
db> db.products.createIndex({category:1,price:-1});
category_1_price_-1
```

To verify if the indexes have been created we use a command:

db.products.getIndexes();

```
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> Please enter a MongoDB connection string (Default: mongodb://localhost/): db> Ple
ase enter a MongoDB connection string (Dedddb>
db>
```