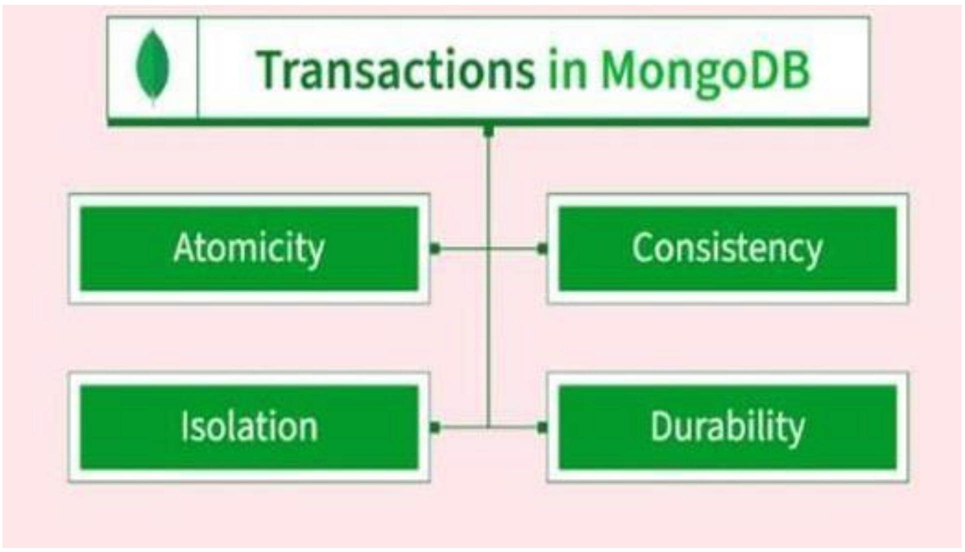
MONGO DB

CLASS 8: ACID & INDEXES.

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

ACID IN MONGODB:

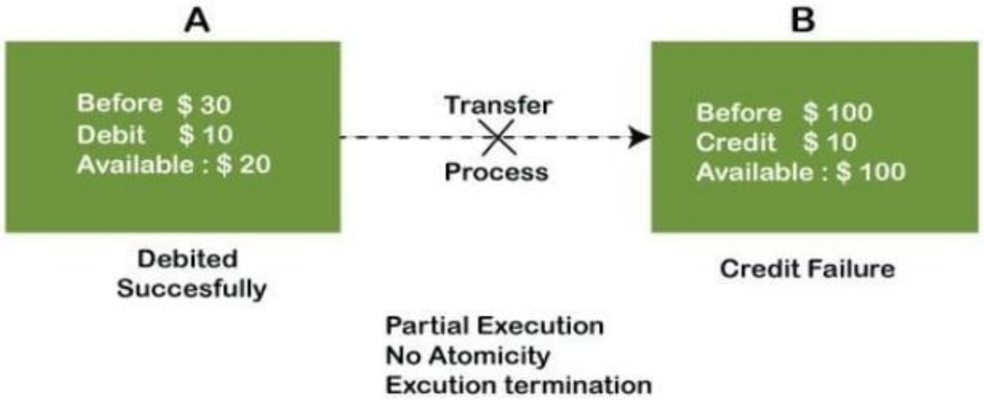
ACID stands for Atomicity, Consistency, Isolation, and Durability which are properties that ensure reliable transaction processing in databases.

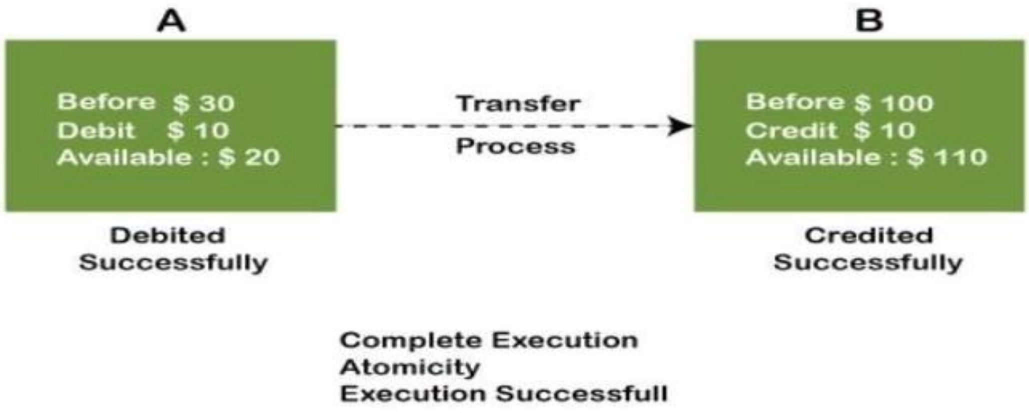


ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable transactions in a database system. Understanding how MongoDB adheres to ACID principles helps in designing robust applications that require transactional integrity.

1.Atomicity:

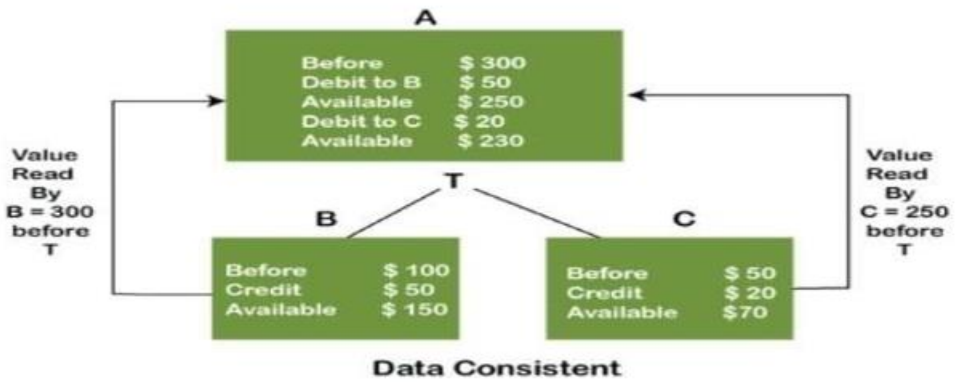
Atomicity guarantees that all of the commands that make up a transaction are treated as a single unit and either succeed or fail together. This is important as in the case of an unwanted event, like a crash or power outage, we can be sure of the state of the database. The transaction would have either completed successfully or been rolled back if any part of the transaction failed. If we continue with the above example, money is deducted from the source and if any anomaly occurs, the changes are discarded and the transaction fails.





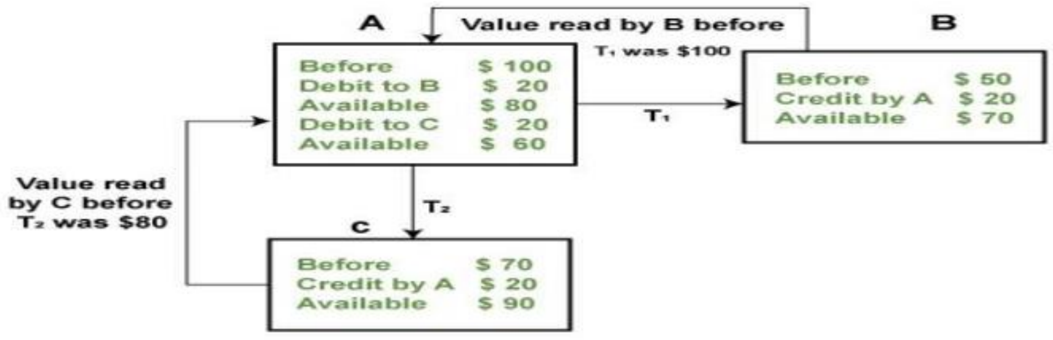
2.Consistency:

Consistency guarantees that changes made within a transaction are consistent with database constraints. This includes all rules, constraints, and triggers. If the data gets into an illegal state, the whole transaction fails. Going back to the money transfer example, let's say there is a constraint that the balance should be a positive integer. If we try to overdraw money, then the balance won't meet the constraint. Because of that, the consistency of the ACID transaction will be violated and the transaction will fail.



3.Isolation:

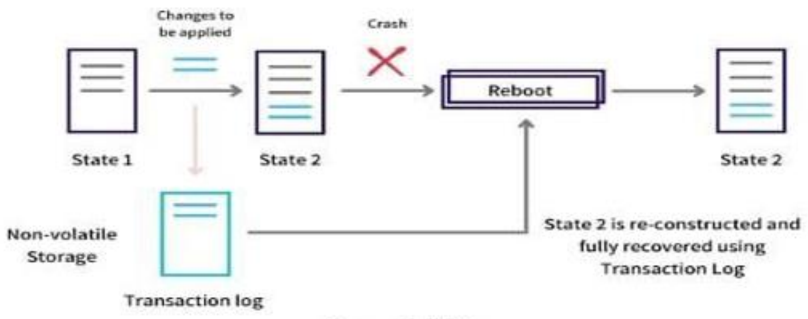
Isolation ensures that all transactions run in an isolated environment. That enables running transactions concurrently because transactions don’t interfere with each other. For example, let’s say that our account balance is $200. Two transactions for a $100 withdrawal start at the same time. The transactions run in isolation which guarantees that when they both complete, we’ll have a balance of $0 instead of $100.



Isolation - Independent execution of T₁ & T₂ by A

4.Durability:

Durability guarantees that once the transaction completes and changes are written to the database, they are persisted. This ensures that data within the system will persist even in the case of system failures like crashes or power outages. The ACID characteristics of transactions are what allow developers to perform complex, coordinated updates and sleep well at night knowing that their data is consistent and safely stored.



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.

Database
Indexing
Replication
Sharding

1.INDEXING:-

Indexes are special data structures that store a small portion of the data set in an easy-to traverse form. They improve query performance and enable efficient execution of various operations.

Types of Indexes:

1.Single Field Indexes: Indexes on a single field.

2. Compound Indexes: Indexes on multiple fields.

3.Multikey Indexes: Indexes on array fields, creating an index key for each array element.

4.Geospatial Indexes: Indexes supporting geospatial queries.

5.Text Indexes: Indexes supporting text search on string content.

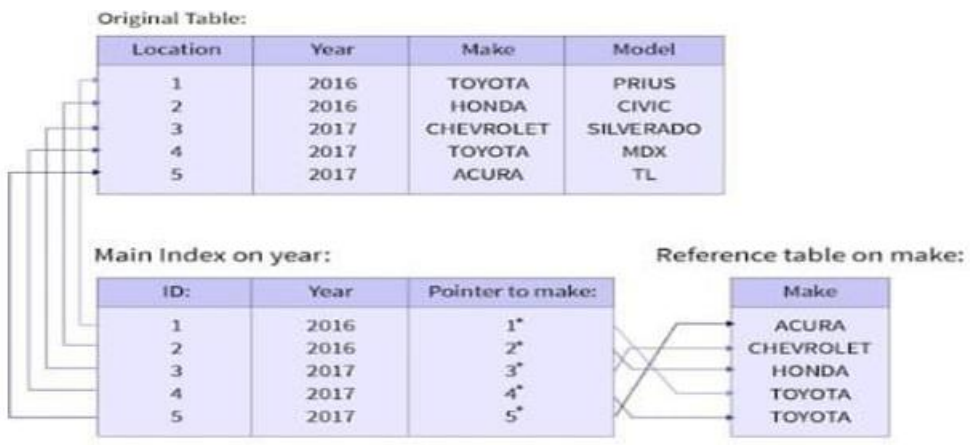
6.Hashed Indexes: Indexes with hashed values of the indexed field.

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.



Original Table:

Location	Year	Make	Model
1	2016	TOYOTA	PRIUS
2	2016	HONDA	CIVIC
3	2017	CHEVROLET	SILVERADO
4	2017	TOYOTA	MDX
5	2017	ACURA	TL

Main Index on year:

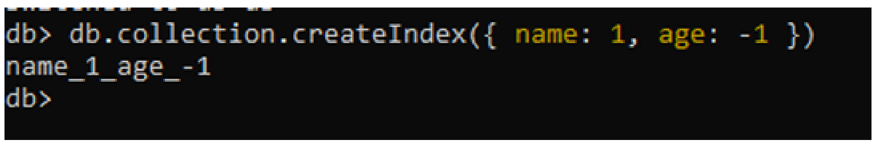
ID:	Year	Pointer to make:
1	2016	1*
2	2016	2*
3	2017	3*
4	2017	4*
5	2017	5*

Reference table on make:

Make
ACURA
CHEVROLET
HONDA
TOYOTA
TOYOTA

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

The default name for an index is the concatenation of the indexed keys and each key's direction in the index (1 or -1) using underscores as a separator. For example, an index created on.



```
db> db.collection.createIndex({ name: 1, age: -1 })
name_1_age_-1
db>
```

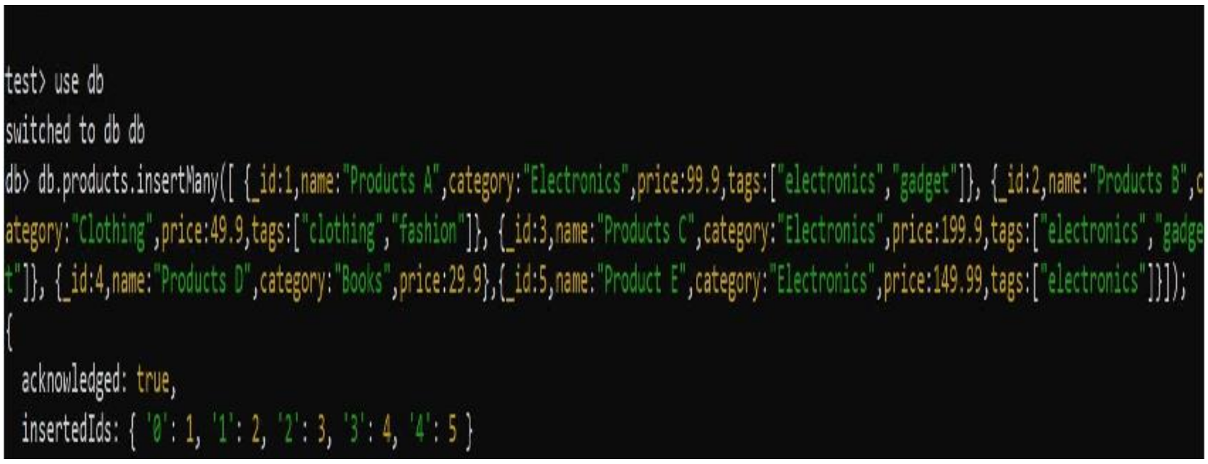
You cannot rename an index once created. Instead, you must with a new name. drop and recreate the index.

Incorporating indexes into your MongoDB collections is a fundamental practice for improving query efficiency and overall database performance. By carefully designing and maintaining indexes, developers can ensure that their applications remain responsive and capable of handling large volumes of data and complex queries. Understanding the different types of indexes and their appropriate use cases is essential for making informed decisions about index implementation and optimization.

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"] }]);

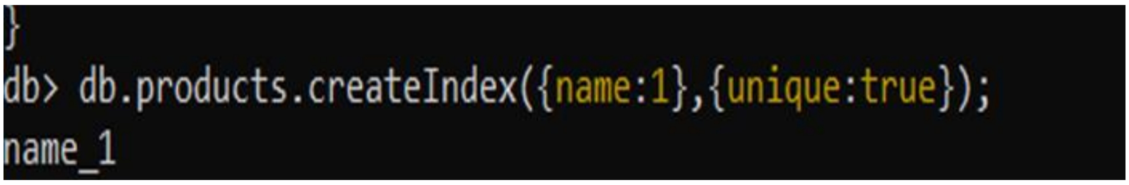


```
test> use db
switched to db db
db> db.products.insertMany([ {_id:1,name:"Products A",category:"Electronics",price:99.9,tags:["electronics","gadget"]}, {_id:2,name:"Products B",c
ategory:"Clothing",price:49.9,tags:["clothing","fashion"]}, {_id:3,name:"Products C",category:"Electronics",price:199.9,tags:["electronics","gadge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

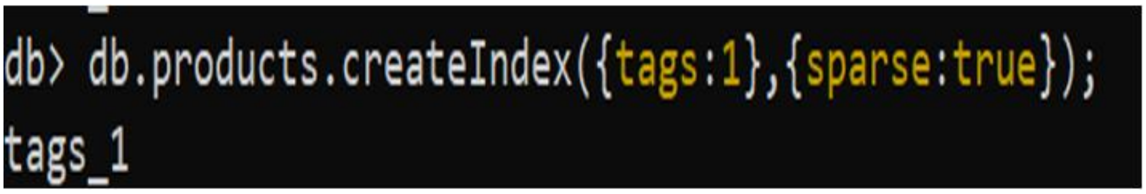


```
}
db> db.products.createIndex({name:1},{unique:true});
name_1
```

2.Sparse index:

Indexes only documents where the specified field exists.

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

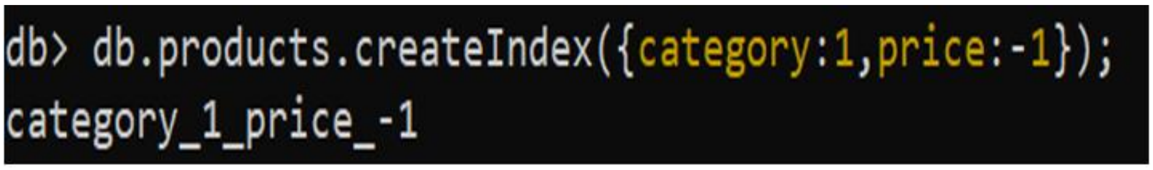


```
db> db.products.createIndex({tags:1},{sparse:true});
tags_1
```

3.Compound index:

Indexes multiple fields in a specified order.

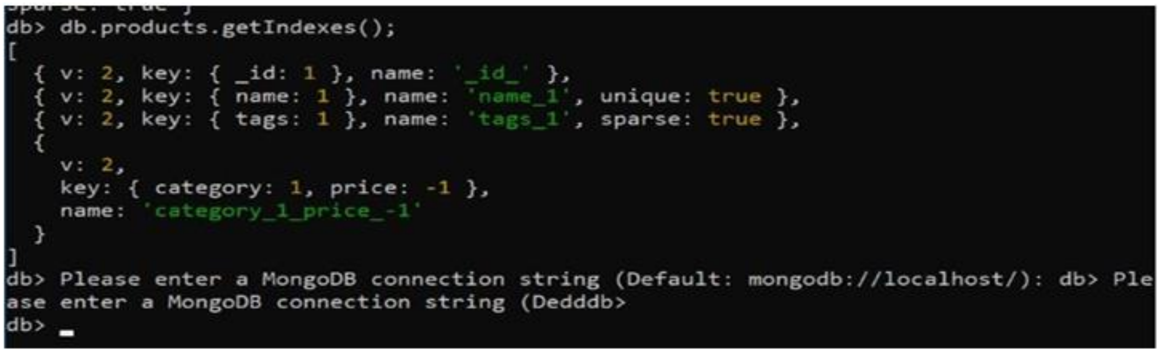
db.products.createIndex({ category: 1, price: -1 });



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

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

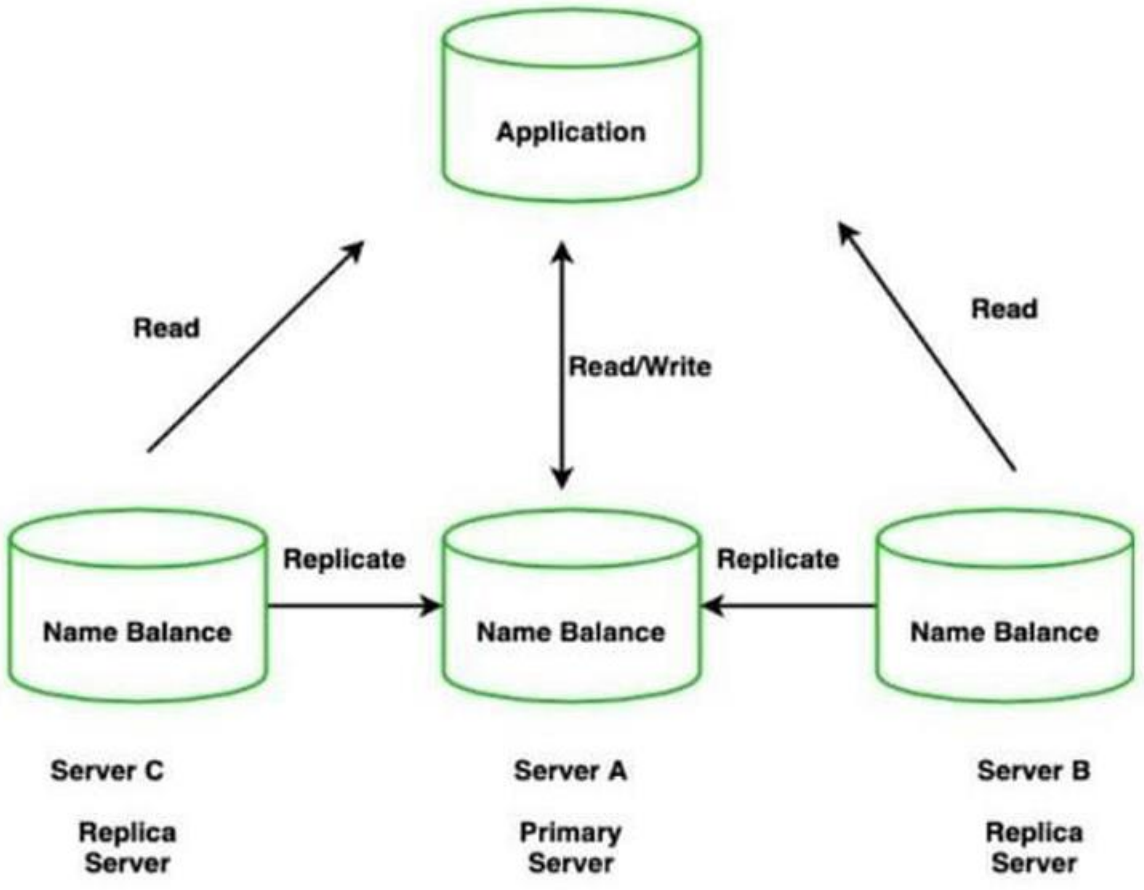
db.products.getIndexes();



```
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> _
```

2.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.

Benefits of Replication:

High Availability: If the primary server fails, an automatic election takes place to select a new primary from the secondaries, ensuring continuous availability.

Data Redundancy: Multiple copies of data are maintained across different servers, protecting against data loss.

Read Scalability: Secondary servers can serve read operations, distributing the read load and improving performance.

3. SHARDING:

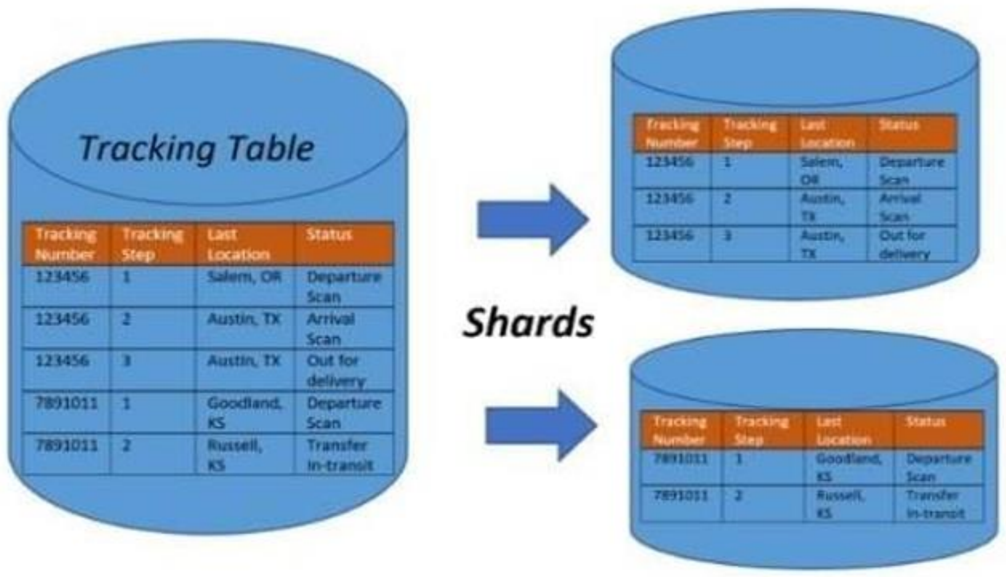
In MongoDB, the concept you might be referring to is "sharding." Sharding is a method used to distribute data across multiple servers to support deployments with very large data sets and high throughput operations.

Benefits of Sharding:

Scalability: By adding more shards, the system can handle increased load and larger datasets.

High Throughput: Distributing data across multiple shards allows for parallel processing of queries, improving performance.

Fault Tolerance: Even if one shard fails, the rest of the system can continue to operate, provided the application can tolerate partial data unavailability.



Tracking Number	Tracking Step	Last Location	Status
123456	1	Salem, OR	Departure Scan
123456	2	Austin, TX	Arrival Scan
123456	3	Austin, TX	Out for delivery
7891011	1	Goodland, KS	Departure Scan
7891011	2	Russell, KS	Transfer In-transit

Tracking Number	Tracking Step	Last Location	Status
123456	1	Salem, OR	Departure Scan
123456	2	Austin, TX	Arrival Scan
123456	3	Austin, TX	Out for delivery

Tracking Number	Tracking Step	Last Location	Status
7891011	1	Goodland, KS	Departure Scan
7891011	2	Russell, KS	Transfer In-transit

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:

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.

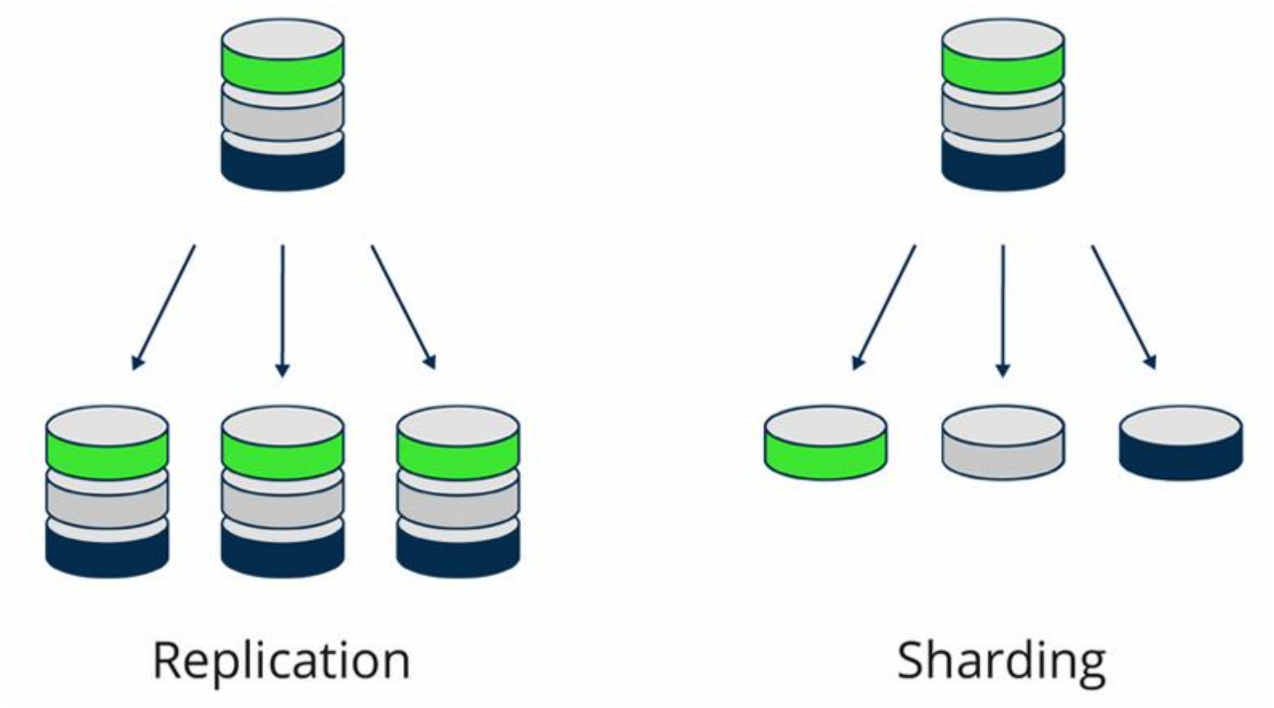
Improves performance and scalability by distributing the data and load across multiple servers.

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.

Sharding and replication are critical components of MongoDB's architecture, enabling it to handle large-scale, high-performance, and highly available applications. Sharding distributes data across multiple servers for horizontal scaling, while replication ensures data redundancy and high availability. Properly implementing and managing these features allows MongoDB to efficiently support demanding applications and workloads.