CLASS-8

ACID and Indexes

ACID Properties in MongoDB

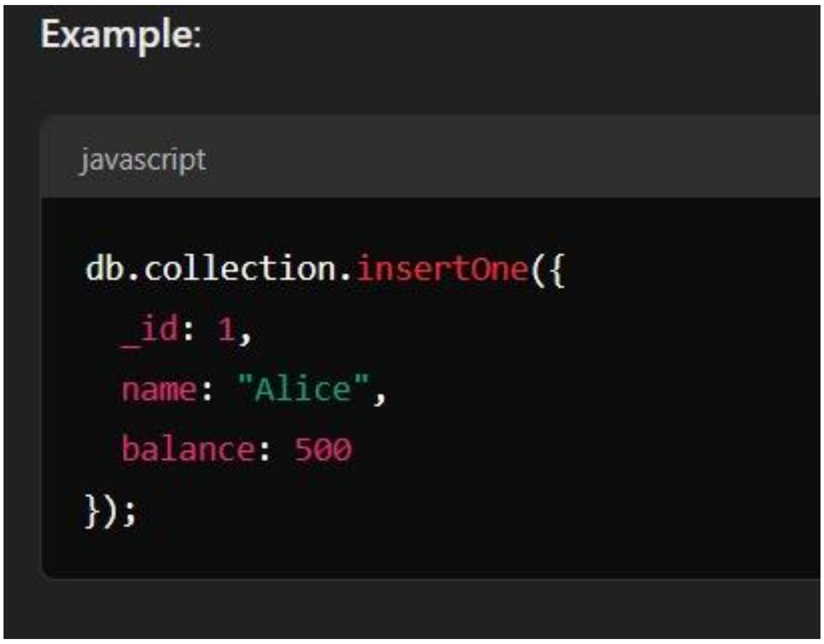
ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure the reliable processing of database transactions. MongoDB provides ACID properties at the document level.

Atomicity:

It ensures that a series of operations within a single transaction are completed entirely or not at all.

In MongoDB, write operations on a single document (inserts, updates, deletes) are atomic, which means they either fully succeed or fully fail, that is, 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 been completed successfully or rolled back if any part of the transaction failed..


Example:

javascript

```javascript
db.collection.insertOne({
  _id: 1,
  name: "Alice",
  balance: 500
});
```

Consistency:

It ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants.

MongoDB guarantees consistency within a single document write operation.

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.

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.

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.

In MongoDB, single-document updates have always been atomic. The document model lends itself to storing related data that is accessed together in a single document (compared to the relational model, where related data may be normalized and split between multiple tables). In most cases, a well-designed document schema allows you to work without the need for multi-document transactions.

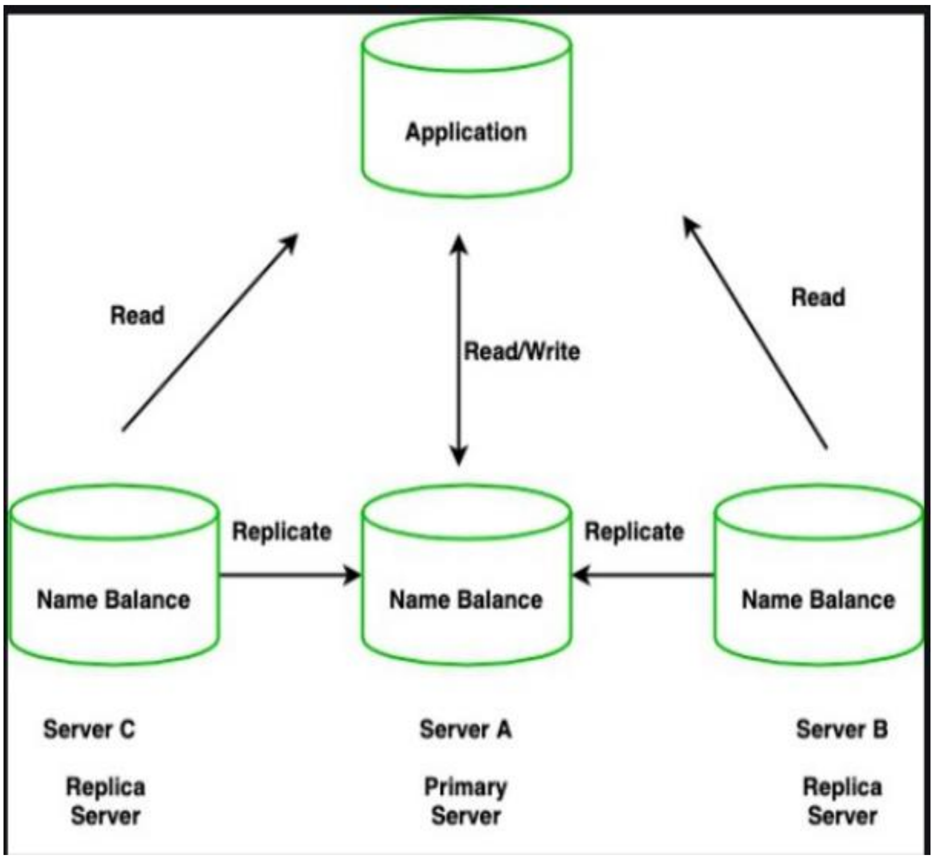
When you do need multi-document ACID compliance, MongoDB transactions work across your cluster and operate the way you'd expect. There is performance overhead to using transactions in a distributed system, so you'll want to be mindful of your resource constraints and performance goals.

Replication

Replication involves copying data from one MongoDB server (primary) to one or more MongoDB servers (secondary). It provides data redundancy and increases data availability.

Features of Replication:

1. **Data Redundancy**:
   - Multiple copies of the data ensure that data is not lost if a server fails.
2. **High Availability**:
   - If the primary server fails, one of the secondary servers can be promoted to the primary, ensuring the database remains available.
3. **Automatic Failover**:
   - In a replica set, if the primary server goes down, the secondary servers hold an election to choose a new primary automatically.
4. **Read Scaling**:
   - Read operations can be distributed across multiple secondaries to improve read performance.

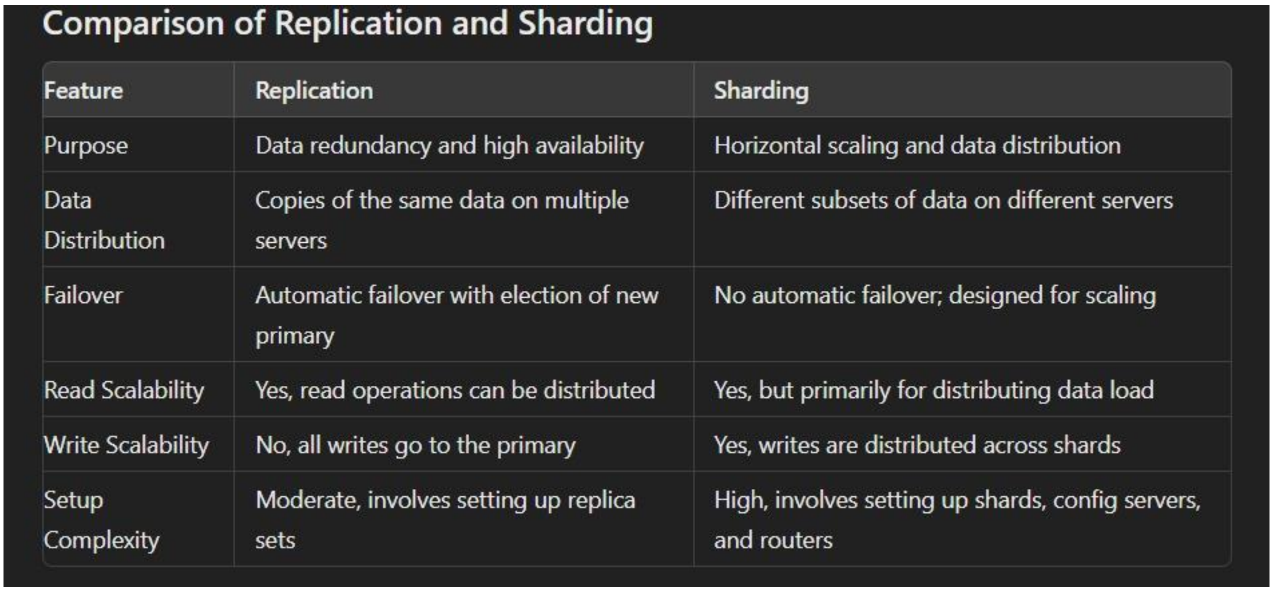


Sharding

Sharding is a method for distributing data across multiple machines. It is used to support deployments with very large data sets and high throughput operations.

Features of Sharding:

1. **Horizontal Scaling**:
   - Data is distributed across multiple servers, each holding a subset of the data (shards).
2. **Data Distribution**:
   - Sharding allows MongoDB to distribute data across multiple servers, balancing the load and improving performance.
3. **Query Routing**:
   - MongoDB automatically routes queries to the appropriate shard(s) based on the data distribution.
4. **Large Data Set Handling**:
   - Enables MongoDB to handle very large datasets that exceed the capacity of a single server.



Comparison of Replication and Sharding

Feature	Replication	Sharding
Purpose	Data redundancy and high availability	Horizontal scaling and data distribution
Data Distribution	Copies of the same data on multiple servers	Different subsets of data on different servers
Failover	Automatic failover with election of new primary	No automatic failover; designed for scaling
Read Scalability	Yes, read operations can be distributed	Yes, but primarily for distributing data load
Write Scalability	No, all writes go to the primary	Yes, writes are distributed across shards
Setup Complexity	Moderate, involves setting up replica sets	High, involves setting up shards, config servers, and routers

Indexes in MongoDB

Indexes in MongoDB are special data structures that store a small portion of the collection's data set in an easy-to-traverse form. They improve the efficiency of read operations by enabling MongoDB to quickly locate data, instead of scanning the entire collection.

Types of Indexes in MongoDB

Single Field Index:

- Indexes a single field of a document.
- The simplest and most common type of index.

Compound Index:

- Indexes multiple fields within a document.
- Useful for queries that filter on multiple fields.

Multikey Index:

- Indexes fields that contain arrays.
- Creates an index entry for each element of the array.

Text Index:

- Indexes string content within documents for text search.
- Supports text search queries and allows for indexing on multiple fields.

Geospatial Index:

- Indexes geographic location data.
- Supports geospatial queries (e.g., finding documents within a certain radius)

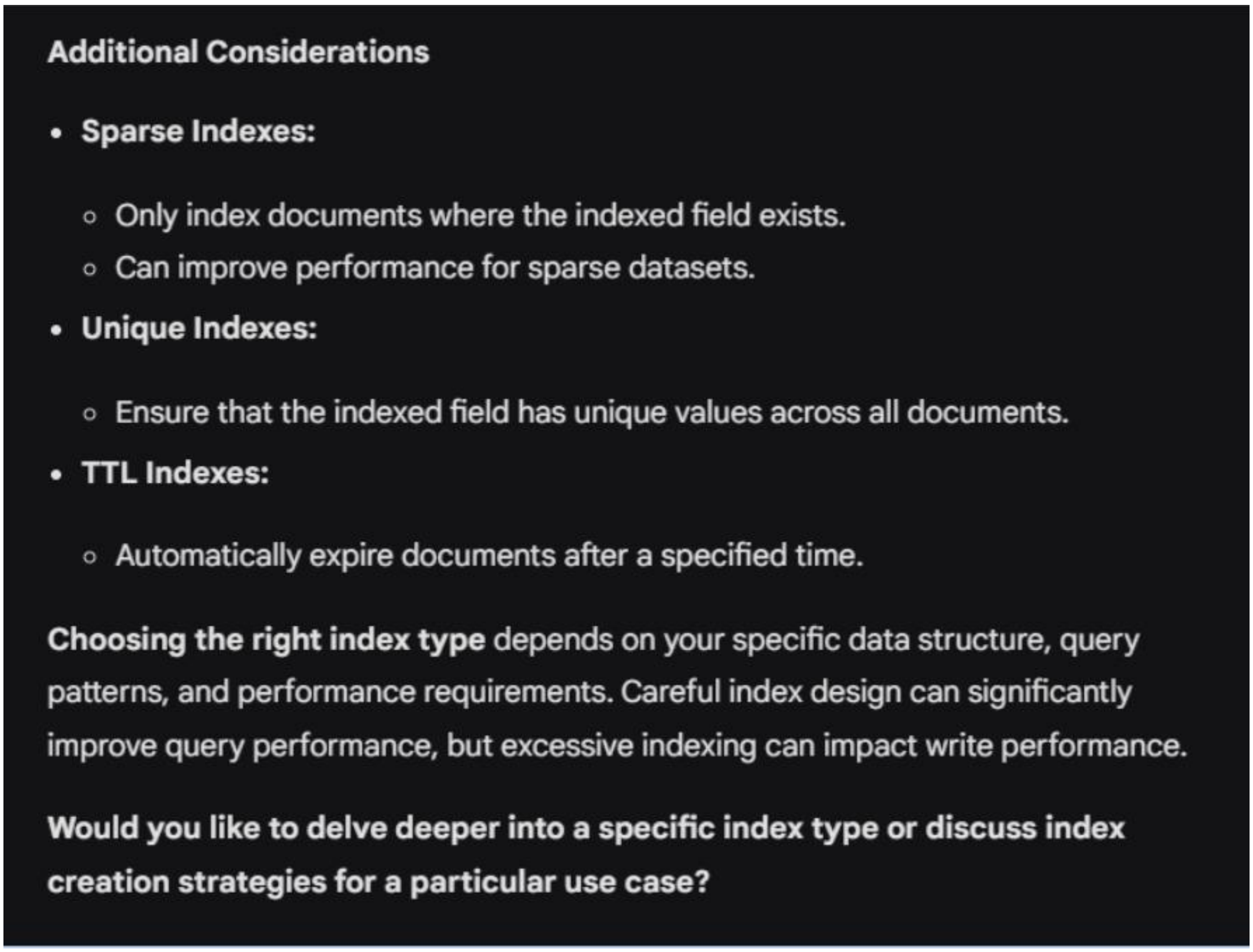
Hashed Index:

- Indexes the hash of the value of a field.
- Ensures even distribution of data across the shards in a sharded cluster.

Wildcard Index:

- Indexes fields with dynamic or unknown schema.

- Can index all fields in a document.



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?