Class 8: ACID & Indexes

ACID is an acronym that refers to the set of 4 key properties that define a transaction Atomicity, Consistency ,isolation and Durability. If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems.

- Atomicity: each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed. This property prevents data loss and corruption from occurring if, for example, if your streaming data source fails mid-stream.
- Consistency: ensures that transactions only make changes to tables in predefined, predictable ways. Transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table.
- Isolation: when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously.
- Durability: ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

Replication:

In simple terms, MongoDB replication is the process of creating a copy of the same data set in more than one MongoDB server. This can be achieved by using a Replica Set. A replica set is a group of MongoDB instances that maintain the same data set and pertain to any mongod process.

Replication enables database administrators to provide:

- Data redundancy
- High availability of data

Maintaining multiple MongoDB servers with the same data provides distributed access to the data while increasing the fault tolerance of the database by providing backups.
Additionally, replication can also be used as a part of load balancing,
where **read** and **write** operations can be distributed across all the instances depending on the use case.

Features of Replication:

1. Data Redundancy:

Multiple copies of the data ensure that data is not lost if a server fails.

2. High Availability:

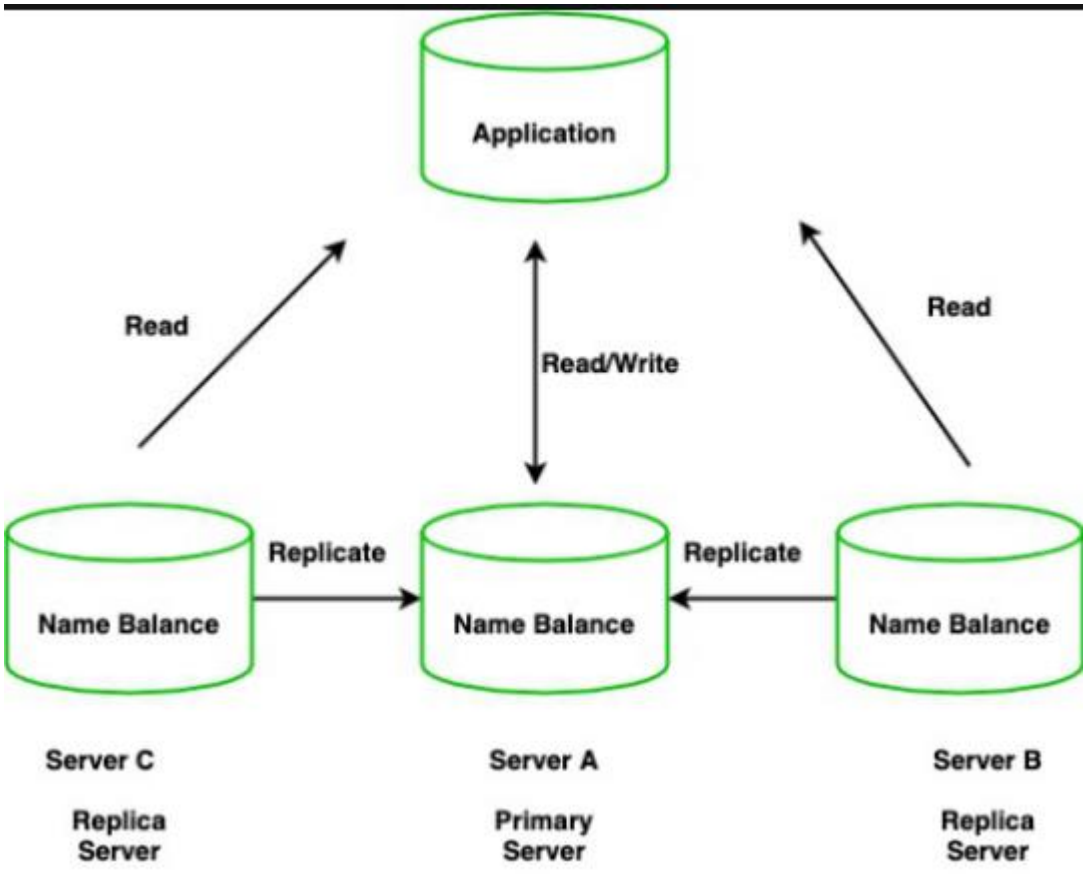
If the primary server fails, one of the secondary servers can be promoted to the primary, ensuring the database remains available.

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. MongoDB 6
- 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?