CLASS 8 : ACID & Indexes

ACID

- Atomicity
- Consistency
- Replication
- Sharding

Atomicity

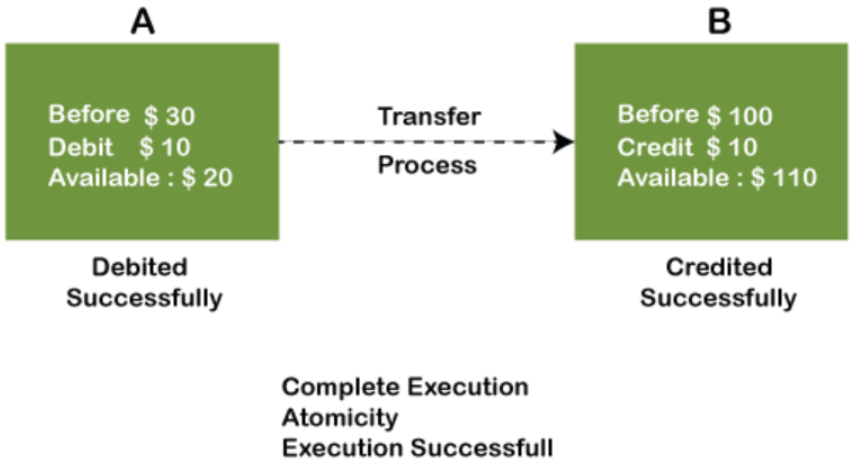
Atomicity in MongoDB ensures that operations are executed as a complete unit, preventing partial updates and maintaining data consistency. Single document operations are atomic by default, while multi-document transactions provide atomicity across multiple documents and collections.

1. Data Consistency:

- **Single Document Updates**: Ensures that updates to a single document are completed fully or not at all, preventing partial updates that can lead to inconsistent states.
- **Multi-Document Transactions**: Starting with MongoDB 4.0, multi-document transactions ensure that a series of operations across multiple documents and collections are either fully completed or fully rolled back, maintaining data consistency.

2. Complex Operations:

- **Banking Transactions**: In scenarios like transferring money between accounts, atomicity ensures that the debit from one account and the credit to another happen together. If either operation fails, neither account is updated.
- **Inventory Management**: Updating inventory levels and recording the transaction simultaneously ensures that the inventory data remains accurate and consistent.



3. Error Handling:

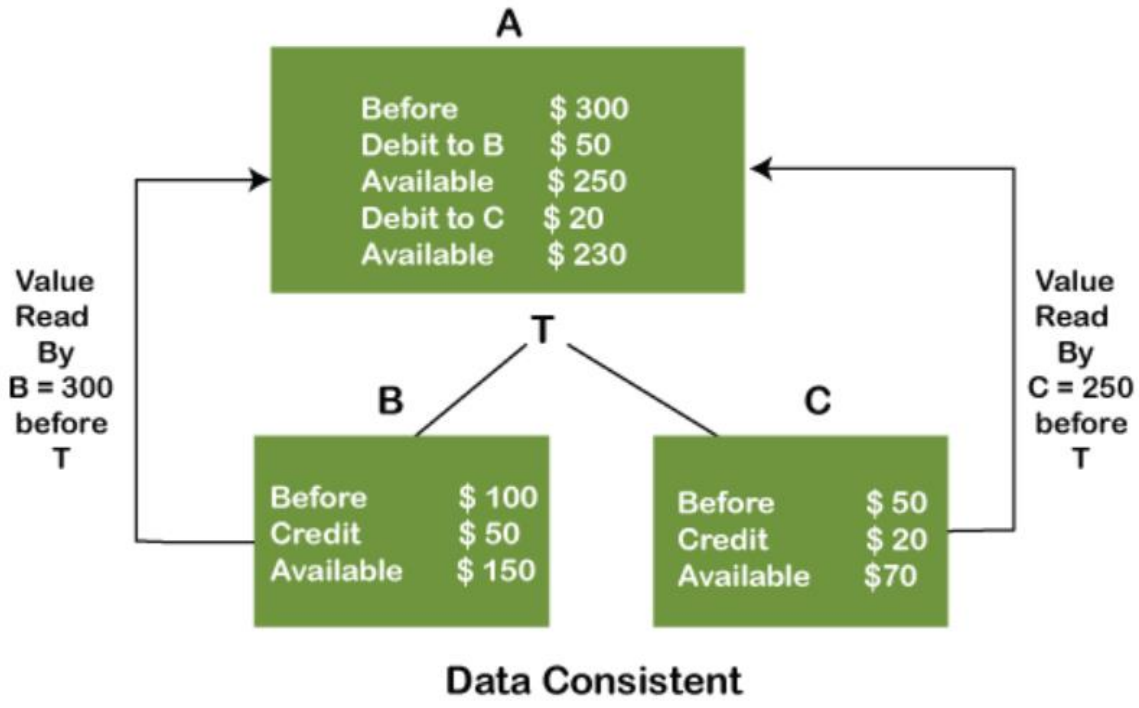
- Atomicity simplifies error handling in applications. Developers can handle errors by aborting transactions, knowing that no partial updates will remain in the database.

4. Data Integrity:

- Ensures that the database remains in a valid state, even in the event of system crashes or power failures. Unfinished transactions are not applied, preventing corrupt or incomplete data from being saved.

Consistency

Consistency ensures that the database remains in a valid and predictable state after any operation. This is achieved through schema validation, atomic operations, replica sets, indexes, and multi-document transactions. By maintaining consistency, MongoDB helps developers build reliable applications with accurate and trustworthy data.



1. Schema Validation:

- MongoDB allows for schema validation rules that enforce specific constraints on the structure and content of documents within a collection. This ensures that any document inserted or updated adheres to the defined schema, maintaining data integrity.

2. Atomic Operations:

- As mentioned earlier, single-document operations in MongoDB are atomic, which helps maintain consistency. Multi-document transactions also provide atomicity, ensuring that a series of operations are either fully completed or not applied at all.

3. Replica Sets:

- MongoDB uses replica sets to ensure data redundancy and availability. Consistency is maintained through replication, where secondary members replicate data from the primary member. MongoDB offers various read preferences that can affect consistency, such as reading from the primary for strong consistency or from secondaries for eventual consistency.

4. Indexes:

- Indexes in MongoDB ensure that data remains consistent and queries return accurate results efficiently. Unique indexes, for instance, enforce that no two documents have the same value for a specific field, maintaining data consistency.

5. Transactions:

- Starting with MongoDB 4.0, support for multi-document transactions ensures that operations across multiple documents and collections are consistent. If any part of a transaction fails, the entire transaction is rolled back, preserving the consistent state of the database.

Eventual Consistency

Eventual consistency in MongoDB refers to a consistency model where updates to a distributed database are propagated to all nodes asynchronously, ensuring that all nodes will eventually become consistent, given enough time. This model is particularly relevant in distributed systems and can be seen in MongoDB replica sets when read preferences are set to read from secondary members.

Key Concepts:

1. **Primary and Secondary Nodes:**
    - In a MongoDB replica set, one node is the primary (which receives all write operations), and the others are secondary nodes (which replicate the data from the primary).
2. **Replication Lag:**
    - There can be a delay (replication lag) between when the data is written to the primary and when it is replicated to

the secondaries. During this time, the secondary nodes may not have the most recent data.

3. **Read Preferences:**
    - MongoDB allows applications to specify read preferences. If the read preference is set to secondary, secondaryPreferred, or nearest, clients may read stale data because secondaries might not have the latest updates.

Example:

User A posts a photo:

- The photo is written to the primary node.
- The write operation will eventually be replicated to the secondary nodes.

User B views User A's profile:

- User B reads from one of the secondary nodes.
- Due to replication lag, the new post might not be immediately visible to User B.

In an Instagram-like application, eventual consistency allows the system to be highly available and responsive. Users can continue reading from secondary nodes even if they might temporarily see slightly outdated information. This is acceptable in scenarios where seeing the most up-to-date information is not critical for every user interaction.

Isolation

Isolation refers to how the database handles concurrent operations, ensuring data consistency and integrity. MongoDB uses a form of isolation called multi-granularity locking (MGL) that allows operations to lock at different levels of granularity, such as global, database, collection, or document levels. This allows MongoDB to provide better performance and concurrency compared to databases that only support row-level locking or table-level locking.

1.Document-level Locking: MongoDB uses document-level locking, meaning each document in a collection can be locked independently. This improves concurrency since operations on different documents can proceed simultaneously without waiting for locks to be released on other documents.

2.Multi-granularity Locking (MGL): MongoDB employs MGL, which allows it to lock at various levels:

Global: Affects the entire MongoDB instance.

Database: Affects a specific database.

Collection: Affects a specific collection.

Document: Affects a specific document.

3.Write Concern and Read Concern:

Write Concern: Controls the acknowledgment of write operations. It specifies the level of acknowledgment requested from

MongoDB for write operations. Higher write concern levels ensure greater durability but might impact performance.

Read Concern: Controls the consistency and isolation properties of the data read from a replica set. It allows you to specify the level of isolation for read operations, ensuring that read operations return data that meets the specified consistency level.

4.Transactional Isolation:

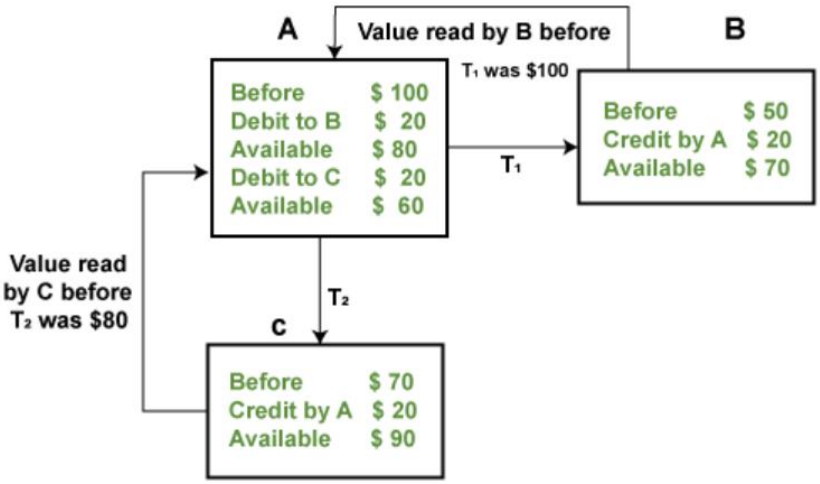
- MongoDB supports multi-document ACID transactions starting from version 4.0. Transactions provide snapshot isolation, ensuring that all reads within a transaction see a consistent snapshot of the data as it was at the start of the transaction.
- Transactions can span multiple documents, collections, databases, and even shards in a sharded cluster.

5.Isolation Levels:

- MongoDB provides snapshot isolation within transactions. This ensures that the data read during a transaction will not change until the transaction completes.
- Outside of transactions, MongoDB operations provide isolation at the document level, ensuring that operations on a single document are atomic and isolated from operations on other documents.

Example:

Suppose you have a banking application where you need to transfer money between two accounts. Using MongoDB transactions, you can ensure that the debit from one account and the credit to another account are performed atomically, preventing any inconsistencies that could arise from concurrent operations.



Benefits

1.Data Consistency: Ensures that operations on the database do not leave it in an inconsistent state.

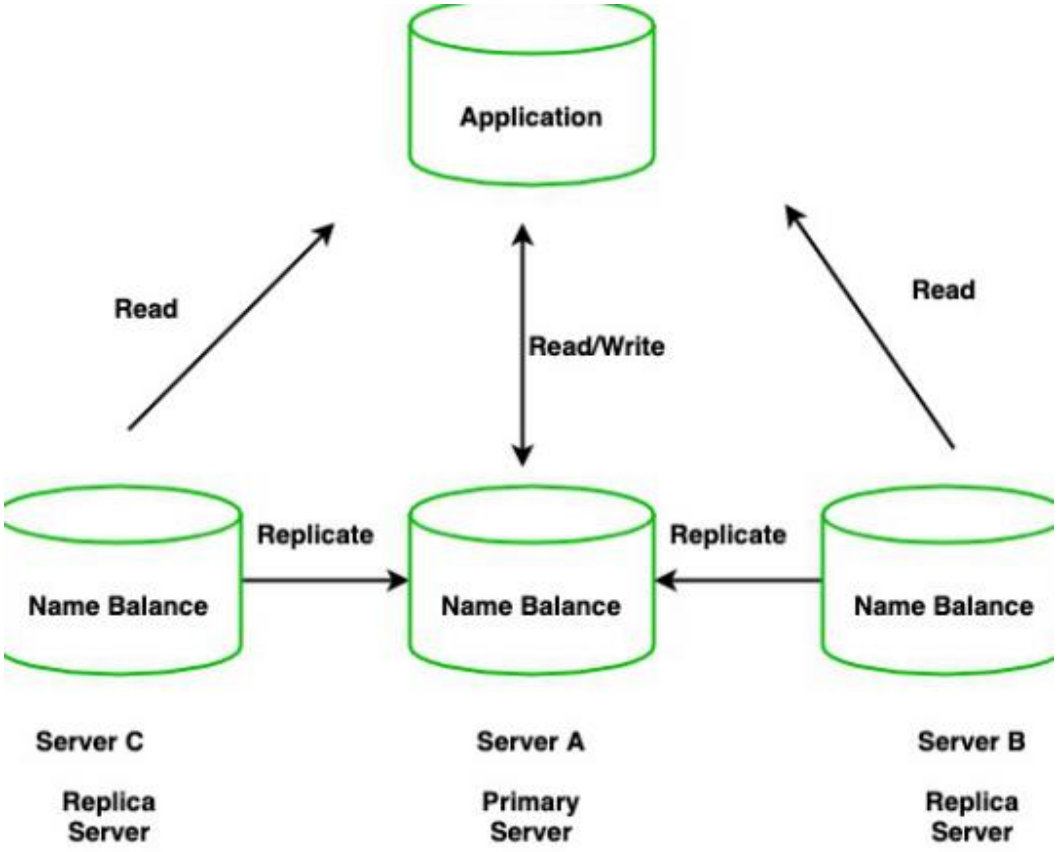
2.Atomic Operations: Guarantees that multiple operations either all succeed or all fail, maintaining atomicity.

3.Concurrency Control: Allows multiple operations to be performed concurrently without interference, improving performance.

4.Error Handling: Simplifies error handling by allowing transactions to be rolled back in case of failures.

Replication (Master-Salve)

Replication in MongoDB is a process that allows data to be copied and distributed across multiple servers, ensuring high availability, redundancy, and disaster recovery. This mechanism is crucial for maintaining data reliability and ensuring that the system remains operational even in the event of hardware failures or other issues.



Benefits of Replication:

1. **High Availability**:
    - Replication ensures that data remains available even if one or more nodes fail.
    - Automatic failover allows secondary nodes to take over as the primary node without manual intervention.
2. **Data Redundancy**:
    - Multiple copies of data are maintained across different nodes.
    - This redundancy protects against data loss due to hardware failures.
3. **Load Balancing**:
    - Read operations can be distributed across secondary nodes, reducing the load on the primary node.
    - This improves performance and responsiveness for read-heavy applications.
4. **Disaster Recovery**:
    - Replica sets can span multiple data centers or geographic regions.
    - This geographic distribution ensures that data is protected against regional failures or disasters.

Sharding

Sharding is a method for distributing data across multiple servers, or shards, to ensure scalability and manage large data sets. It helps to distribute the load and handle large volumes of data by dividing it into smaller, more manageable pieces. Each shard holds a subset of the sharded data, and collectively the shards form a single logical database.

1.Shard:

- A shard is a single MongoDB instance or replica set that holds a portion of the sharded data.

2.**Shard Key**:

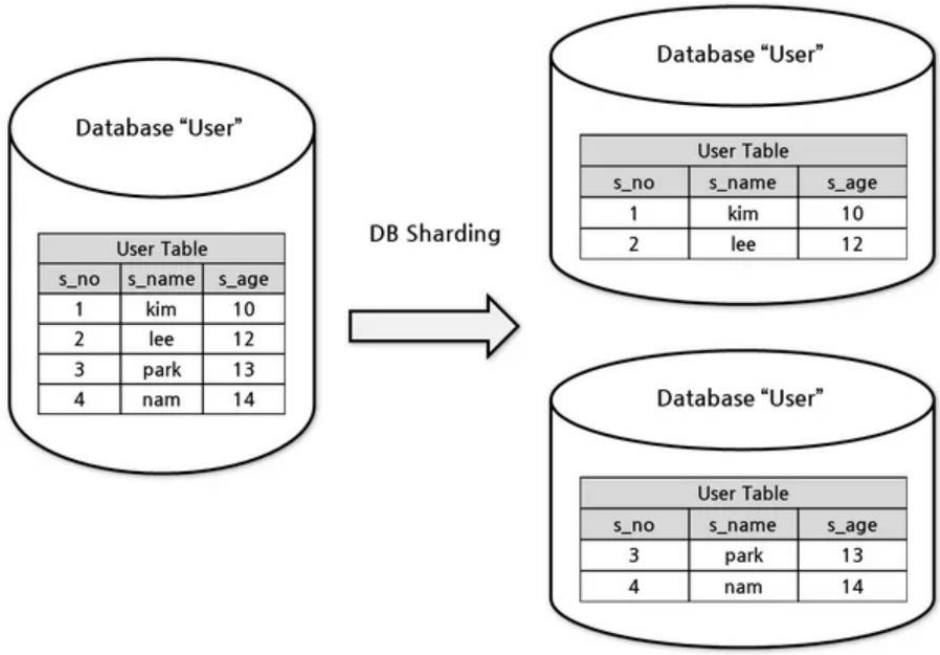
- The shard key is a field or set of fields that determines how data is distributed across the shards.
- It is crucial to choose an appropriate shard key to ensure even data distribution and avoid bottlenecks.

3.**Config Server**:

- Config servers store metadata and configuration settings for the sharded cluster.
- They keep track of the distribution of data and the state of the cluster.

4.Query Router (mongos):

- The query router, or mongos, directs queries from client applications to the appropriate shards.
- It uses the metadata stored in the config servers to determine which shards to query.



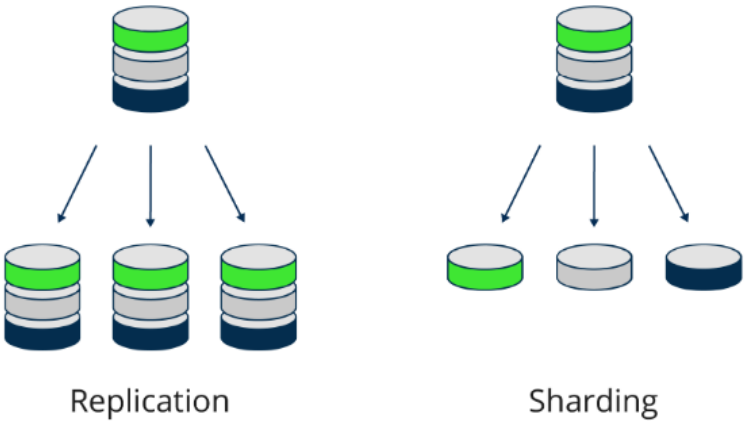
Benefits of Sharding:

1. **Horizontal Scalability**:
    - Sharding allows a database to scale out horizontally by adding more servers to handle increased load and data volume.
    - This helps in managing very large data sets efficiently.

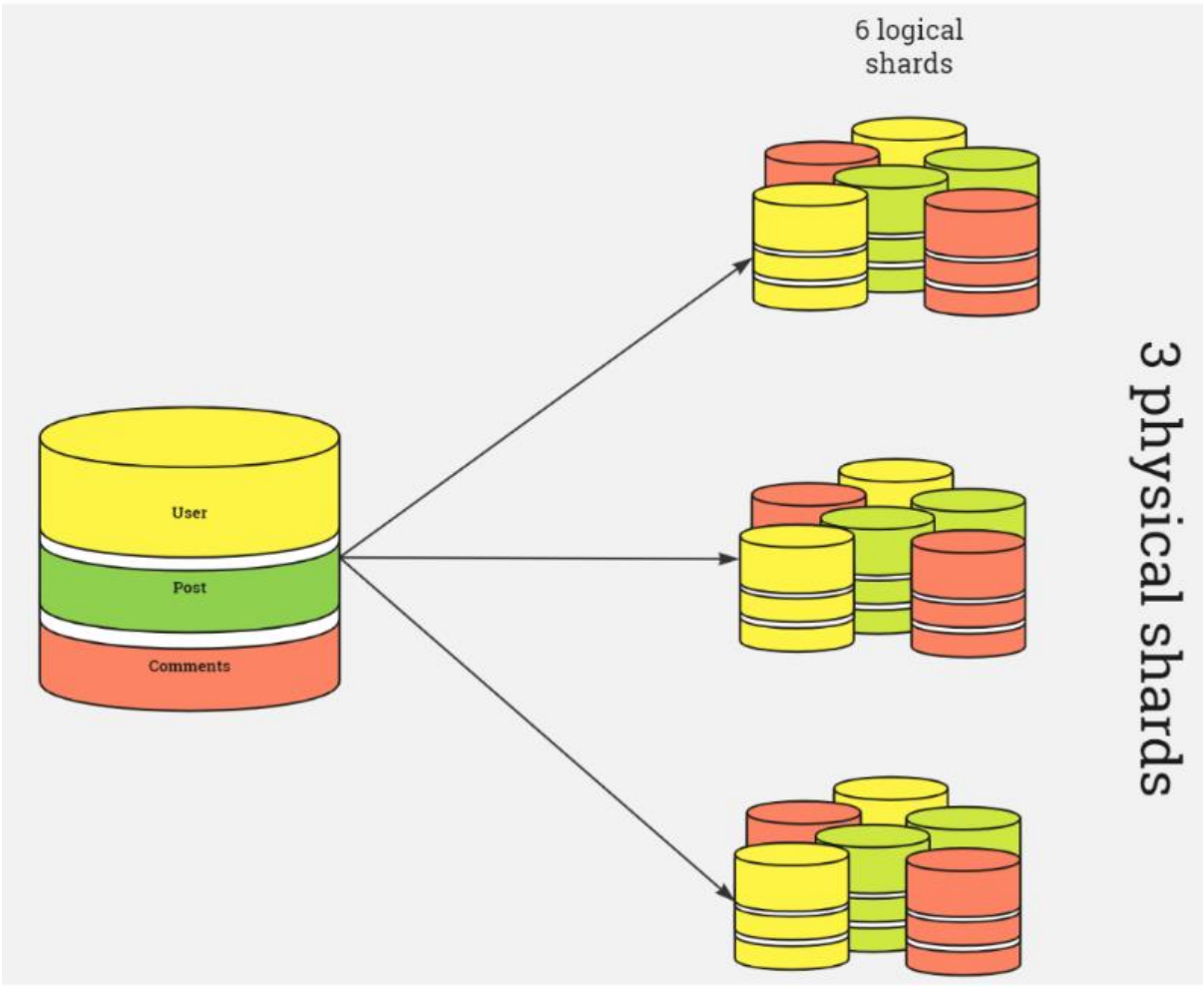
2. **Improved Performance**:
    - By distributing data and queries across multiple shards, sharding can significantly improve read and write performance.
    - It helps to avoid performance bottlenecks that occur when a single server handles all operations.
3. **High Availability**:
    - Sharded clusters often use replica sets for each shard, providing high availability and redundancy.
    - This ensures that data remains available even if some nodes fail.

Replication v/s Sharding

Feature	Replication	Sharding
Purpose	High availability and redundancy	Scalability and distribution of data
Data Distribution	Copies of the same data across multiple nodes	Different subsets of data across multiple nodes
Primary Component	Replica Set	Sharded Cluster
Nodes	Primary and secondary nodes	Shards, Config Servers, and Query Routers (mongos)
Write Operations	Only primary node accepts writes	Writes distributed based on shard key
Read Operations	Can be served by any node (primary or secondary)	Directed by mongos to appropriate shards
Fault Tolerance	High, due to data redundancy	Moderate, relies on replication within shards
Automatic Failover	Yes, automatic election of new primary	Depends on replica sets within shards
Data Consistency	Strong consistency by default, tunable with read and write concerns	Ensured within shards, requires additional configuration across shards
Load Balancing	Not inherently load balanced, can read from secondaries to distribute read load	Yes, data is distributed across shards
Use Case	High availability and disaster recovery	Large datasets and high throughput operations
Setup Complexity	Moderate	High, requires careful planning and configuration
Maintenance	Easier, mostly involves monitoring replication status	More complex, involves monitoring data distribution and balancing
Query Complexity	Simple, queries directed to primary or secondaries	More complex, queries routed to appropriate shards by mongos
Scalability	Limited to vertical scaling (more powerful servers)	Horizontal scaling (adding more shards)



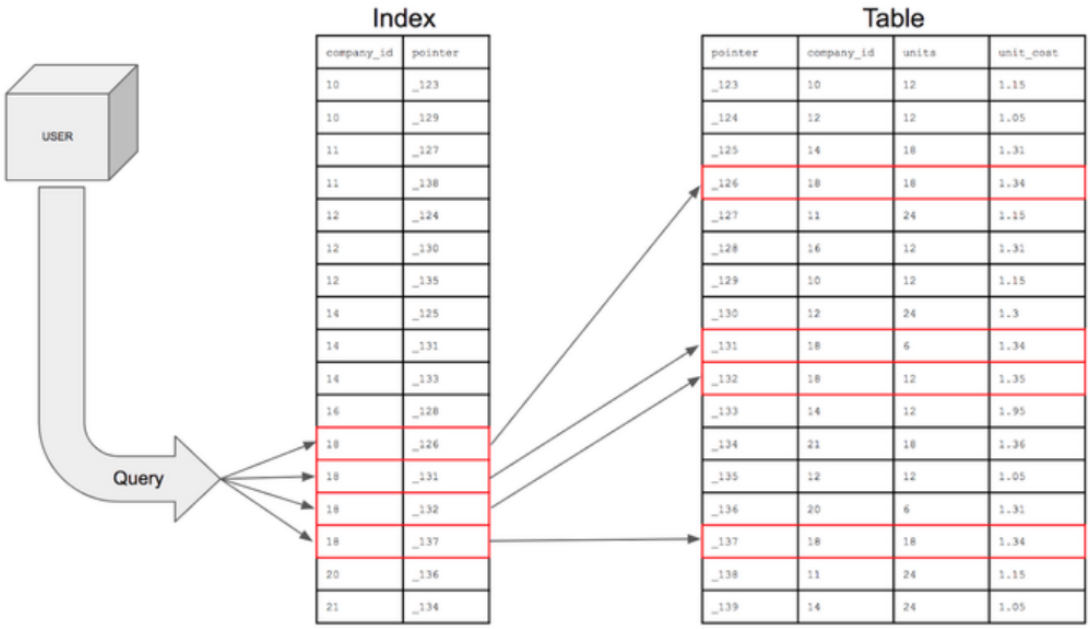
Replication + Sharding



Aspect	Replication + Sharding
Purpose	Combines data distribution with fault tolerance.
Components	Sharded Cluster: Multiple shards, each being a replica set.
Data Distribution	Data is distributed across shards, and each shard is replicated.
Failover	Failover occurs both at the shard level and within individual shards.
Query Routing	Queries are routed by the mongos instances, which direct them to the appropriate shard.
Configuration	Requires configuration of both replication (replica sets) and sharding.

Indexes

Indexes in MongoDB are crucial for optimizing query performance and improving data retrieval speed. They help MongoDB efficiently find and sort data, which is especially important for large datasets and complex queries.



An index is a data structure that improves the speed of data retrieval operations on a collection.

MongoDB indexes work similarly to indexes in traditional databases by allowing the database engine to quickly locate and access the desired data.

Types

1.Single Field Index: Indexes a single field of a document. It is the simplest form of index.

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

3.Multikey Index: Indexes fields that contain arrays. This allows MongoDB to index array elements.

4.Text Index: Indexes text content for text search queries. Supports searching for words or phrases.

5.Geospatial Index: Indexes geographic location data to support geospatial queries.

6.Hashed Index: Indexes a hashed value of the field. Useful for sharding.

Index Options:

- **Unique**: Ensures that all values for a given field or set of fields are unique across documents.
- **Sparse**: Indexes only documents that contain the indexed field, ignoring documents that do not contain it.
- **TTL (Time-To-Live)**: Automatically removes documents from the collection after a certain period.

Uses

1. Speed Up Queries
2. Improve Performance of Aggregation Pipelines
3. Enable Efficient Full-Text Search
4. Support Geospatial Queries
5. Optimize Uniqueness Constraints