WEEK 8

Q: What is Sharding?

A: Sharding in MongoDB is a technique used to divide a large dataset into smaller pieces and distributing those pieces across different machines.

Imagine you have a really big bookshelf filled with books. It becomes difficult to manage and search for a specific book when the bookshelf gets too crowded. Sharding in MongoDB is like having multiple bookshelves where each shelf contains a subset of books.

When you shard a MongoDB database, you split your data into smaller chunks called "shards." Each shard is stored on a separate server or machine. The MongoDB system intelligently distributes the data across these shards based on a sharding key, which is a field or set of fields in your data.

By distributing the data across multiple machines, sharding allows you to store and process large amounts of data more efficiently. It helps improve performance and allows your database to handle more read and write operations simultaneously.

When you perform a query on a sharded MongoDB database, the system automatically routes the query to the appropriate shard or shards that contain the relevant data. This way, you can distribute the workload across multiple machines, making the queries faster and more scalable.

Q: What is Capped Collection?

A: A capped collection in MongoDB is like a fixed-size container that holds a limited number of items. When its full, new items replace the oldest ones, making it ideal for storing time-based where you only need the most recent information.

Imagine you have a drawer with a limited space for storing your favourite books. You decide that you can only keep a maximum of 10 books in the drawer. As you add new book, the oldest book gets pushed out to make room for the new one. This way, you always have the most recent books in the drawer, but the total number is limited.

In a capped collection, it's similar. You define a maximum number of documents it can hold. Once it reaches that limit, any new document you add will replace the oldest document in the collection. So, you have a set size, and new data simply "circulates" in and out.

The main benefit of a capped collection is that it's useful for storing time-based data or logs, where you only want to keep the most recent information. It saves space and makes it easy to access the latest entries without worrying about deleting old data manually.

Here are a few key characteristics of capped collections:

- Size Limit: Capped collections have a maximum size. When it reaches the limit, new data replaces the old data.

- Insertion Order: Documents in a capped collection are stored based on when they were added. The most recent data is always at the end.
- No Updates: Once a document is added to a capped collection, it cannot be updated. You have to remove and replace it with a new document.
- No Deletion: By default, old documents in a capped collection are not automatically deleted. However, you can set a time limit to remove them using a "time-to-live" (TTL) index.

Q: What is GridFS?

A: GridFS is a specification and feature in MongoDB that allows you to store and retrieve large files (such as images, videos, or documents) that exceed the BSON document size limit of 16 megabytes. Instead of storing the entire file in a single document, it enables you to break down these large files into smaller chunks, or "chunks," and store them as separate documents in the database. When you want to retrieve the file, GridFS automatically combines the chunks and gives you the complete file. It handles the process of breaking down and reassembling the file for you.

Q: What is Transaction?

A: A transaction is a concept in database management that ensures a group of database operations (such as reading, writing, or modifying data) is executed as a single, atomic unit. Transactions guarantee that all changes are either committed (made permanent) or rolled back (undone) if an error occurs, maintaining data integrity.

Think of it like you need to move money from one bank account to another. With a transaction, you can ensure that the money is deducted from one account and added to the other account as a single step. If something goes wrong during the transaction, like an error or interruption, the money is not moved, and everything goes back to how it was before.

Transactions make sure that changes to the database happen together or not at all. It keeps things consistent, isolated from other actions happening at the same time, and durable even if the system crashes.

Q: What is Trigger?

A: A trigger is a special type of procedure or code that is automatically executed in response to a specific event or action performed on a database table. It's like a set of instructions that say, "When this happens, do that automatically."

For example, imagine you have a database table for customer orders. You can set up a trigger that says, "Whenever a new order is placed, automatically update the inventory to reflect the items sold."

Q: What is Cluster?

A: A cluster is like a team of servers or machines that work together to handle a big task. They make the database more reliable, handle more data, and keep things running smoothly even if there are problems.

Think of it like a group project in school. If you have a big assignment, you can form a cluster with your classmates. Each person can work on a part of the project, making it faster and more reliable. If one person gets sick or can't work, others can step in to keep things going.

In a database cluster, multiple computers (servers) work together to handle data storage and processing. If one server has a problem, the others can take over to keep the database running smoothly. This makes the database more reliable and allows it to handle a larger amount of data and user requests.

Clusters help ensure that the database is always available, even if there are hardware issues or a lot of people using it at the same time. They also make it easier to expand the database as needed by adding more computers to the cluster.

Q: What is Journaling?

A: Journaling is like keeping a backup copy of your work before making it official. It's a way to protect your data in case something goes wrong.

Imagine you're writing an important essay on your computer. Before you save it, you make a backup copy just in case the computer crashes. Journaling works similarly for databases like MongoDB.

With journaling, whenever changes are made to the database, they are first recorded in a special log file called a journal. This log file acts as a backup of the changes. Only after the changes are safely recorded in the journal, they are permanently written to the database.

If there is a system crash or unexpected shutdown, the database can recover by using the journal file. It reads the recorded changes from the journal and applies them to the database, bringing it back to a consistent state.

Journaling helps ensure that your data is protected and can be recovered even if something bad happens to the database. It adds a layer of security and helps maintain the integrity of your data.

Q: What is Replica Set?

A: A replica set is like having multiple copies of your database working together as a team. It's a way to make sure your data is safe and always available, even if something goes wrong.

Imagine you have an important document that you want to keep safe. You make several copies and give them to different friends for safekeeping. If you lose one copy or it gets damaged, you can always rely on the other copies to have the same information.

In a replica set, you have multiple MongoDB servers (computers) that store the same data. One server is the leader, called the primary server, and the others are followers, called secondary servers. The primary server handles all read and write operations, while the secondary servers keep copies of the data.

If the primary server fails for any reason, one of the secondary servers automatically takes over as the new primary. This ensures that the database keeps running without any interruption.

Replica sets also make your database faster. Since the secondary servers have copies of the data, they can handle read requests, reducing the load on the primary server.

Q: What is Horizontal Scaling?

A: Horizontal scaling, also known as scale-out, refers to the process of adding more machines or servers to a system to handle increased workload and accommodate growth. It involves distributing the load across multiple machines rather than relying on a single, larger machine.

Imagine you have a big pile of tasks to complete. Instead of doing everything on your own, you invite your friends to help. Each friend takes a portion of the tasks and works on them independently. By doing this, you can finish the entire pile much faster.

In the same way, horizontal scaling involves adding more machines or servers to handle a larger workload. Each machine takes care of a part of the work, sharing the load and making things faster. By having multiple machines working together, you can handle more requests, process data quicker, and improve the performance of your system.

Horizontal scaling also helps with reliability. If one machine has a problem or gets overloaded, the other machines can still handle the workload. It's like having backup teammates ready to step in if someone needs a break.

Q: What is Vertical Scaling?

A: Vertical scaling, also known as scaling up or scaling vertically, refers to increasing the capacity or resources of a single server or machine to handle a greater workload or accommodate more users. It involves upgrading the existing hardware or infrastructure to support increased demands.

Imagine you have a computer that can handle a certain amount of work. If you need to handle more work or accommodate more users, you can upgrade your computer's resources, such as adding more RAM, increasing the processing power with a faster CPU, or expanding storage capacity. This allows your computer to handle a greater workload and support more users.

In the context of databases or servers, vertical scaling involves upgrading the hardware of a single machine. For example, you might increase the memory, CPU cores, or disk space of a server to improve its performance and capacity. This allows the server to handle more requests, process more data, or support a larger number of concurrent users.

Vertical scaling is typically easier to implement because it involves upgrading a single machine. However, there are limits to how much you can scale vertically since there's a maximum capacity for each hardware component. Eventually, you may reach the limits of what a single machine can handle, at which point you might need to consider horizontal scaling, which involves adding more machines or servers to distribute the workload.

Q: What is ACID Theorem?

A: The ACID theory is a set of properties that guarantee reliability and consistency in database transactions. ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure that database transactions are executed reliably and maintain the integrity of the data. ACID ensures that transactions are treated as a single unit (atomicity), maintain data integrity (consistency), work independently (isolation), and endure system failures (ensuring data is not lost) (durability).

Q: Different types of NoSQL Database?

A:

- Document Databases: Store data in flexible, document-oriented formats (e.g., JSON or XML). Examples: MongoDB, CouchDB.
- Key-Value Stores: Store data as key-value pairs, enabling fast retrieval using unique keys. Examples: Redis, Amazon DynamoDB.
- Column-Family Databases: Store data in columns rather than rows, suitable for handling large structured and semi-structured data. Examples: Apache Cassandra, HBase.
- Graph Databases: Store data in a graph-like structure of nodes and edges, efficient for managing complex relationships. Examples: Neo4j, Amazon Neptune.
- Wide-Column Stores: Similar to column-family databases, store and retrieve large amounts of data with dynamic column structures. Examples: Apache HBase, Google Bigtable.

Q: What is cursor?

A: A cursor is like a pointer that helps you go through rows of data in a database result set, one at a time. It keeps track of where you are in the data so that you can fetch and work with each row individually. Cursors give you control over navigating and processing large sets of data, allowing you to do things like reading values, updating data, or deleting rows as you move through them. (Just remember that using cursors can affect performance, so it's important to use them wisely and consider other methods when possible).

Imagine you have a large table with thousands of rows of data. To work with the data, you can use a cursor as a pointer or a reference to navigate through the rows one at a time. The cursor keeps track of the current position, allowing you to fetch and process each row individually.

Q: What is Shell?

A: A shell is like a command-line tool that lets you interact with a database using text-based commands. It's like typing instructions to the database in a special language. You can do things like create or modify objects, run queries, manage users, and monitor the database. The shell takes your commands, communicates with the database, and shows you the results. It's a flexible way to work with the database using a text-based interface. Just remember that each database system has its own shell with its own set of commands.

Q: What is Data Modelling?

A: Data modelling is like creating a plan for how data will be stored in a database. It involves deciding what information will be stored, how it will be organized, and how different pieces of data relate to each other. The goal is to make sure the data is structured in a way that makes it easy to use and retrieve. It's like designing a blueprint for a database to store and organize information effectively.

Q: What is TTL?

A: TTL (Time To Live) in MongoDB is a feature that automatically deletes documents from a collection after a certain amount of time. It's like setting an expiration date for data. Once the time has passed, MongoDB removes the documents for you, saving you from having to manually delete them. It's useful for managing temporary or time-sensitive data that you don't need to keep indefinitely.

Q: What is Covered Query?

A: A covered query in MongoDB is like finding the answer to your question by looking at an index card instead of reading the entire book. It means MongoDB can give you the result of a query just by using the index, without needing to retrieve and process the actual data. Covered queries are faster and use less network bandwidth because they only work with the index information. To make a query covered, you need to create an index that includes all the necessary fields.

Q: What is Backup & Restore?

A:

- Backup: Backup refers to the process of creating a copy of the database or specific data within the database to protect against data loss or corruption. It involves taking snapshots of the data and storing it in a separate location, such as another disk, server, or cloud storage. Backups can be performed regularly to ensure up-to-date copies of the data are available in case of any unexpected events or disasters.

- Restore: Restore is the process of recovering data from a backup and returning it to its original state. It involves retrieving the backup files and using them to replace the existing data, either partially or entirely. The restore process is typically performed when data is lost, corrupted, or needs to be rolled back to a previous state. It helps to recover the database to a known good state using the backed-up data.

Q: Types of Indexing?

A:

- Single Field Index
- Compound Index
- Multikey Index
- Text Index
- Hashed Index

Q: What is CAP Theorem?

A: The CAP theorem also called as Brewer's theorem states that in a distributed computer system, you can't have all three of the following at the same time: consistency, availability, and partition tolerance.

- Consistency means that all nodes see the same data at the same time.
- Availability means that the system remains operational and accessible to users, even if some nodes fail.
- Partition tolerance means the system continues to work even if there are communication failures or nodes become unreachable.

Q: What is MongoDB Aggregation? How does it differ from regular queries?

A: MongoDB Aggregation is a framework for data processing and analysis in MongoDB. It allows for complex data transformations, grouping, filtering, and computations on documents in a collection. Unlike regular queries (such as find), which retrieve documents, aggregation operations process and reshape data, returning calculated results.

Q: Explain the concept of the aggregation pipeline in MongoDB.

A: The aggregation pipeline in MongoDB is a sequence of stages that allows for data transformation and analysis. Each stage takes the output of the previous stage and passes it to the next stage. It enables multi-step processing of data, including filtering, grouping, projecting, sorting, and more.