Develop a Reliable Backend with Node and Express

Schema, Shell, and Server in MongoDB

A Day in the Life of a MERN Stack Developer

Trevor, a MERN stack developer, is working on a transport and construction services project. For this, he will have to deal with large volumes of commercial data in a large database.

In this lesson, you will get a brief idea of the kind of database paradigm that will be used if the end users want to extract and retrieve data without loading large files into the memory.

Learning Objectives

By the end of this lesson, you will be able to:

- Imbibe the knowledge of schemas to understand how data is stored and accessed within a database
- Analyze the concepts of the MongoDB server for managing databases that contain collections of documents
- Assess the concepts of memory management to ensure the database utilizes available memory effectively to enhance performance
- Describe the concepts of relationships in MongoDB to aid in designing the data models that suit specific application

Schemas

What Is a Schema?

A schema is a framework or plan that defines the structure of the database, which includes the tables, fields, relationships, procedures, and constraints.

It can be seen as a set of regulations or guidelines.

It can also be used to define access control policies.

Why Use Schemas?

Schema can be used for the following reasons:

01

Organize data in a structured and logical way

02

Manage and manipulate data easily

03

Standardize the structure and content of documents

Why Use Schemas?

Schema can be used for the following reasons:

04

Facilitate communication between different groups or individuals

05

Validate data, conforming it to a particular structure or set of rule

06

Define reusable templates or patterns to use across multiple applications or systems

Structuring Documents

It is important to structure documents to make them more organized, readable, and understandable. Here are some ways to structure documents:

Lists:

Highlight key points, summarize information, or present step-by-step instructions

Tables and diagrams:

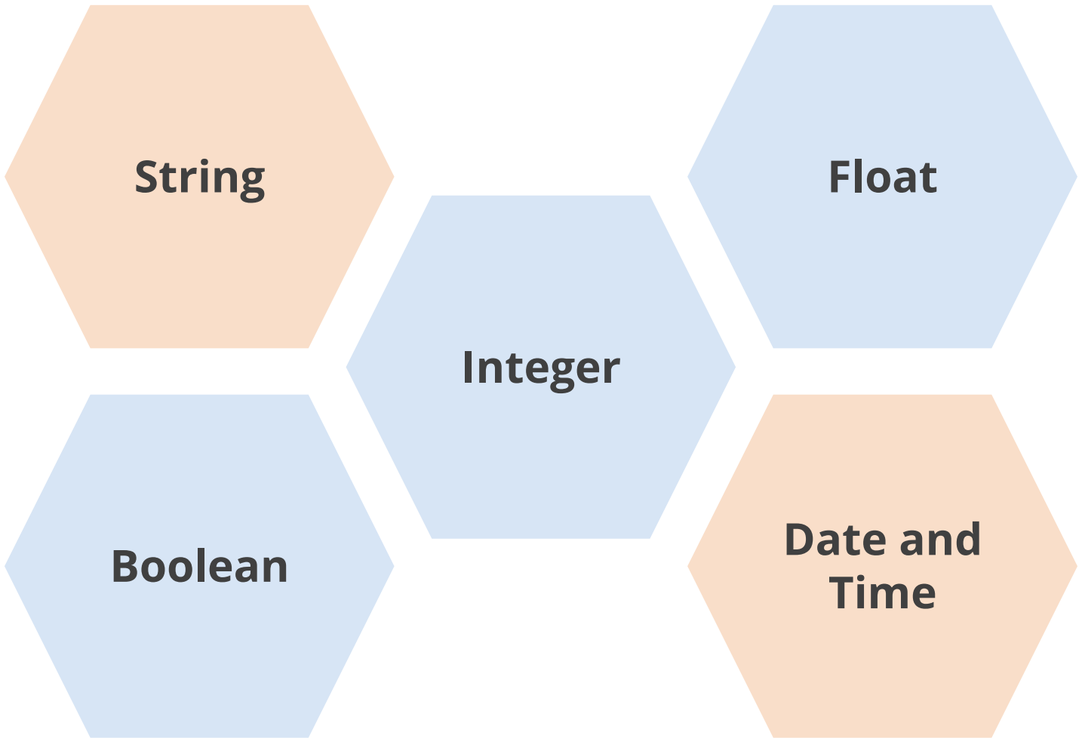
Present complex information in a visual format

Formatting:

Highlight important information and make it stand out

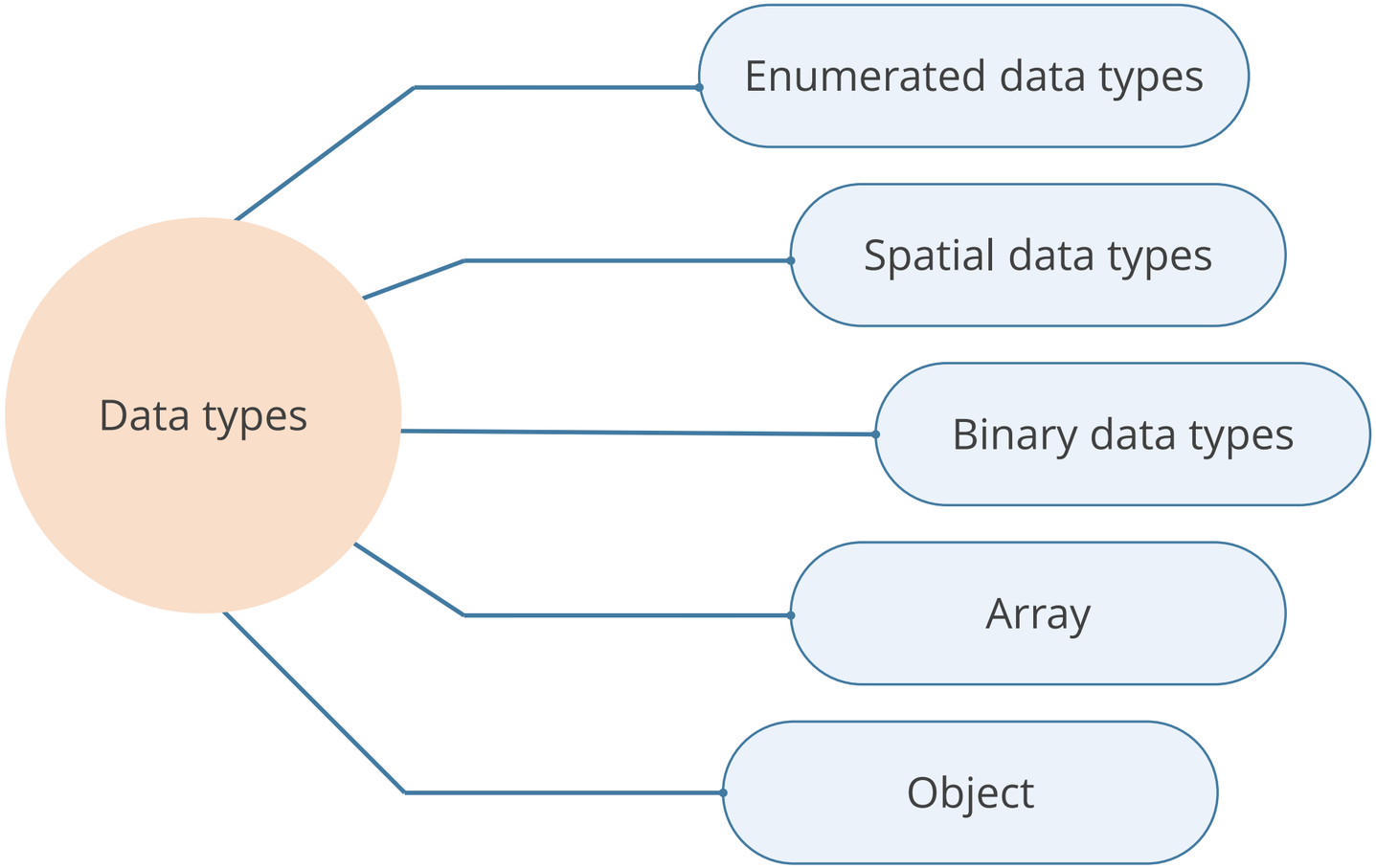
Data Types: An Overview

The schemas of different database management systems support varying data types. Some common data types used in database schemas include:



Data Types: An Overview

Data types used in database schemas include the following:



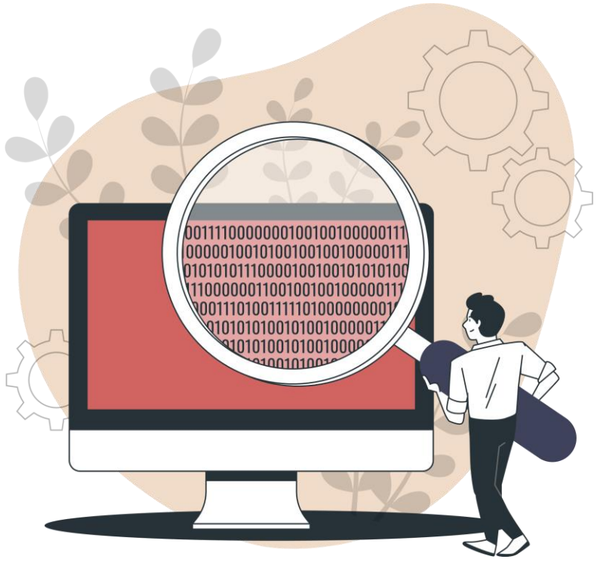
Data Types in Action

This refers to the practical implementation of data types in database management systems.

It ensures that databases are accurate, consistent, and performant.

Data Types and Limits

Data types and limits encompass the various kinds of data that a database can store and impose constraints on that data.



Limits are employed to confine the quantity of data held in each field or table.

Validations

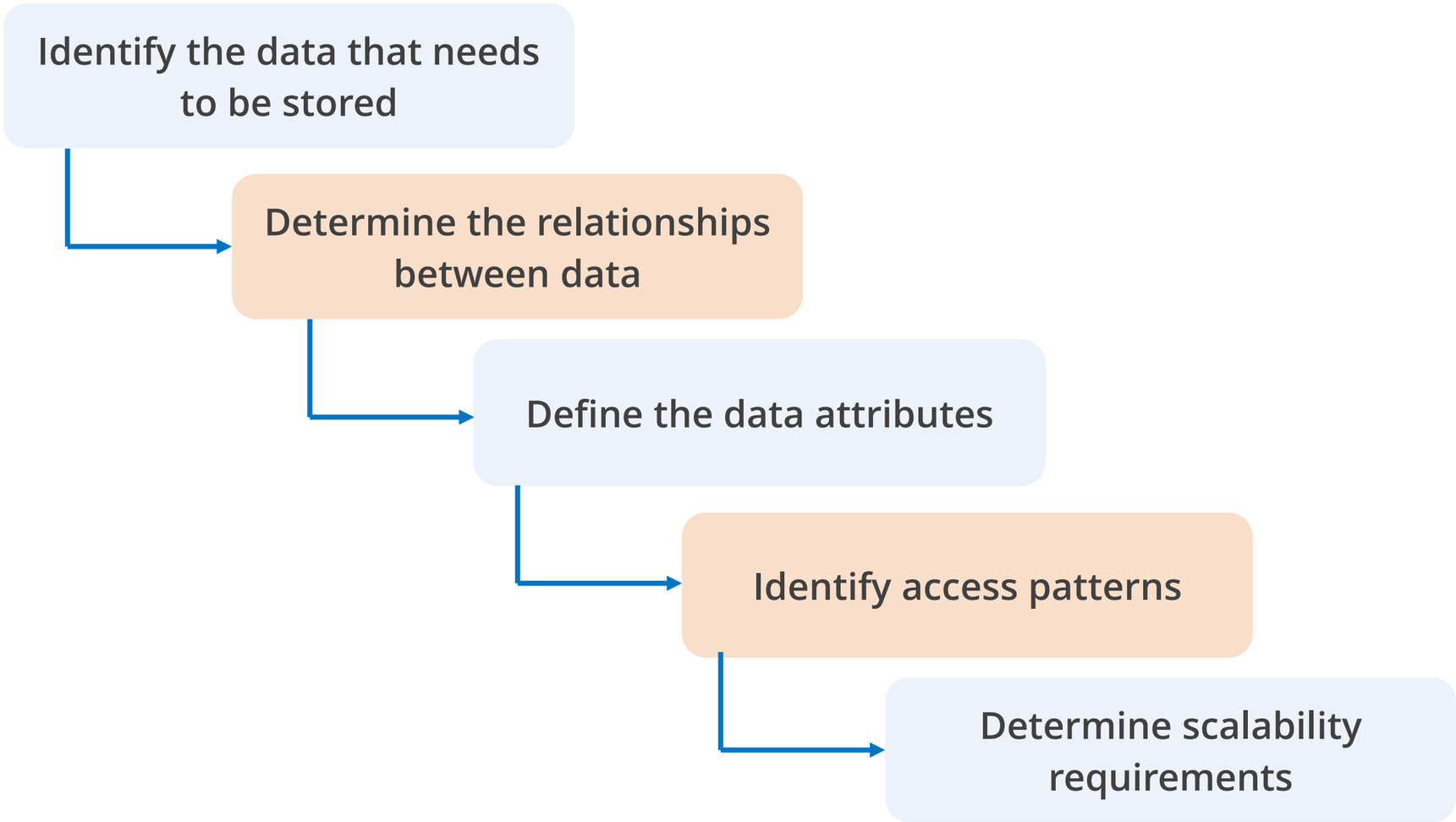
Validations

The schema is essential for ensuring the database is accurate, secure, and efficient.

It allows developers to design databases that meet the specific needs of a project. It also ensures that the data is organized logically and coherently.

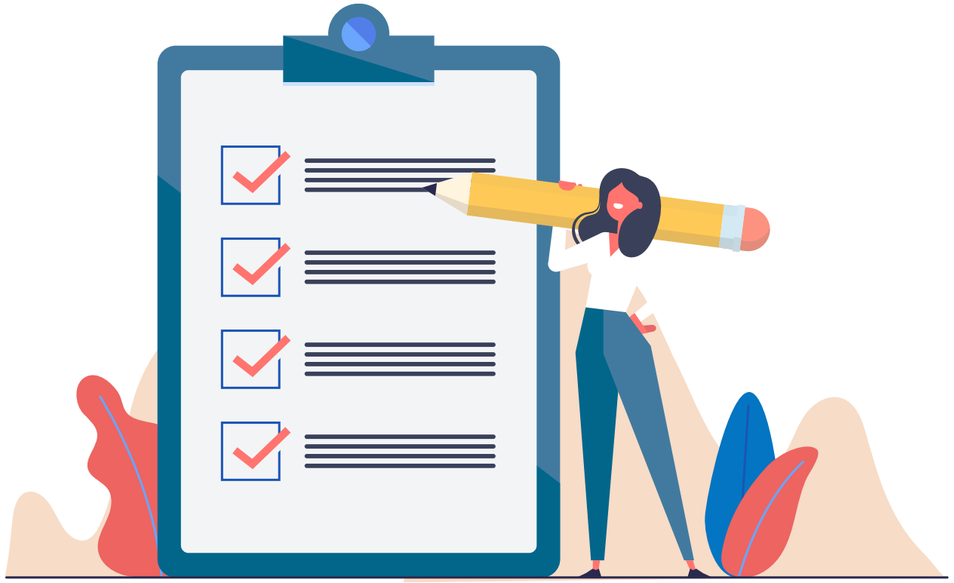
Derive Data Structure Requirements

The steps to derive the data structure requirements are as follows:



Schema Validation

Schema validation is a process of checking whether the data in a database conforms to the defined schema.



Schema Validation

The following are the features of schema validation:

It ensures that the data is accurate, consistent, and complete, and meets the schema requirements.

It helps ensure data quality within a database.

Schema Validation

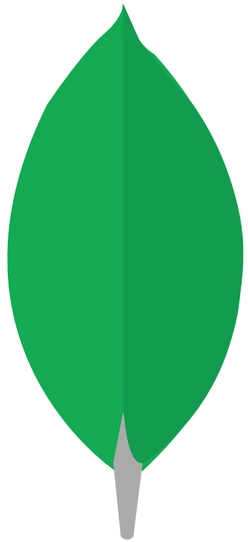
The following are the features of schema validation:

It verifies that each field's data types, size, and constraints are correct and that the relationships between tables are valid.

It can also help to improve system performance and security.

Collection Document Validation

It is a feature in MongoDB that allows developers to enforce rules and constraints on the data inserted or updated in a collection.



MongoDB

Collection Document Validation

One can add collection document validation by using the **$jsonSchema** operator to define a JSON schema. It specifies the rules and constraints for the collection.

Schema :: JSON, draft-06 Format

```
{
   "title": "Person",
   "description": "A person",
   "type": "object",
   "properties":
   {
      "name":
      {
         "description": "A person's name",
         "type": "string"
      },
      "age":
      {
         "description": "A person's age",
         "type": "number",
         "minimum": 18,
         "maximum": 64
```

Message

Schema is valid according to draft-06.

Document :: JSON Format

```
{
   "name": "John Doe",
   "age": 35
}
```

Message

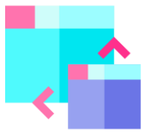
Document validates against the schema, spec version draft-06.

Collection Document Validation

The basic steps to add collection document validation are as follows:

Define the rules and constraints for the collection in a JSON schema

Create the collection and specify the JSON schema using the $jsonSchema operator



Insert or update documents in the collection

Change the Validation Action

In MongoDB, one can change the validation action to control what happens when a document fails to pass validation against the specified schema.

Change the Validation Action

The steps to change the validation action are as follows:

Connect to the MongoDB instance using the appropriate client or shell

Select the collection that needs to be modified

Change the Validation Action

The steps to change the validation action are as follows:

Step 1: Use the **collMod** command to modify the validation action for the collection

For example, to change the action to log validation failures, use the following command:

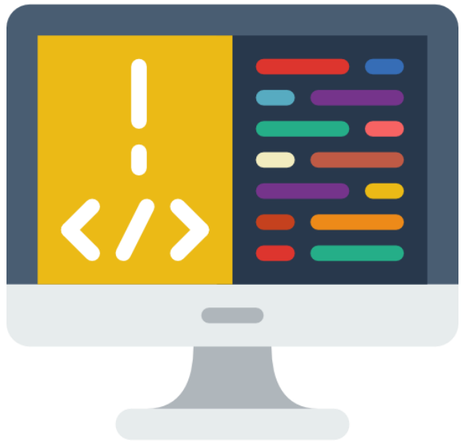
Demo

```
db.runCommand({
collMod: "collection_name",
 validator: { $jsonSchema: { ... } },
validationAction: "log" })
```

Change the Validation Action

Step 2: Test the validation action

Check the logs to verify that the validation action is working as expected. By changing the validation action, customize the behavior of the database



Schema Guidelines

Problem Statement: **Duration: 30 min.**

You have been assigned the task to access schema guidelines on a remote desktop.

ASSISTED PRACTICE

Assisted Practice: Guidelines

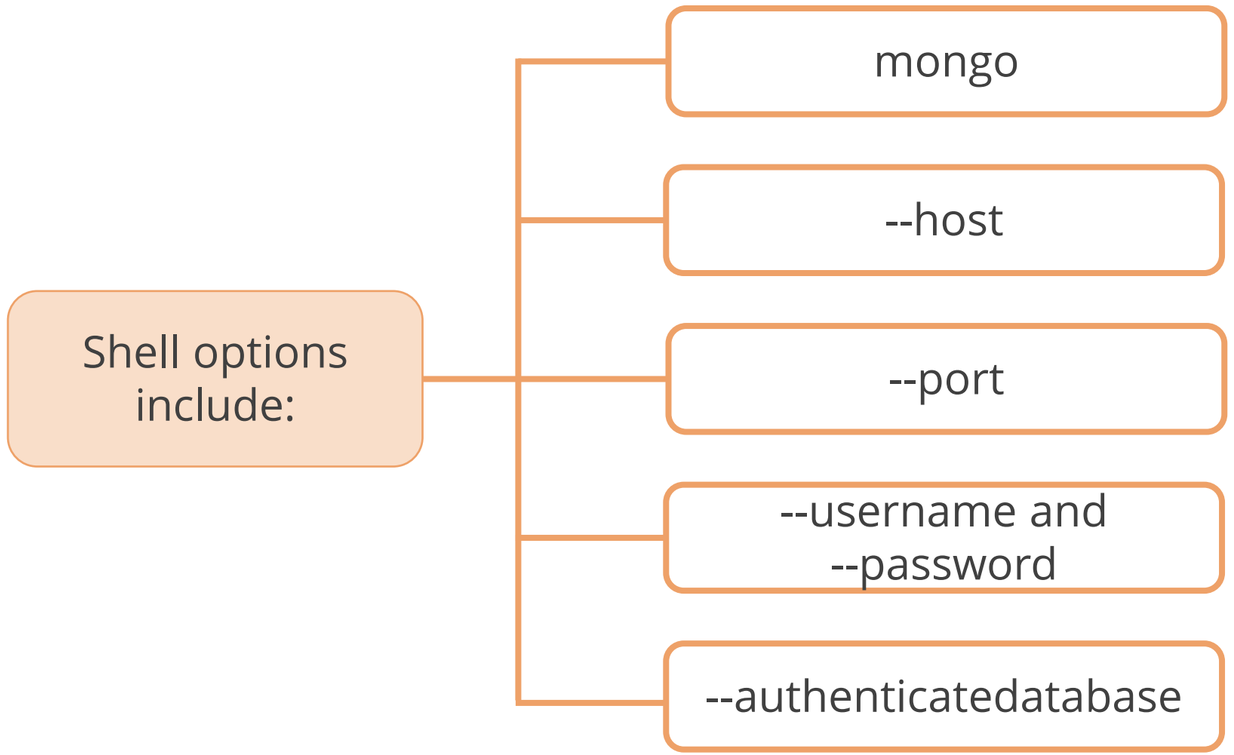
Steps to be followed:

1. Connect to MongoDB Compass using virtual machine extensions
2. Create databases in MongoDB Compass
3. Create collections in a database
4. Validate schema guidelines

Shell and Server in MongoDB

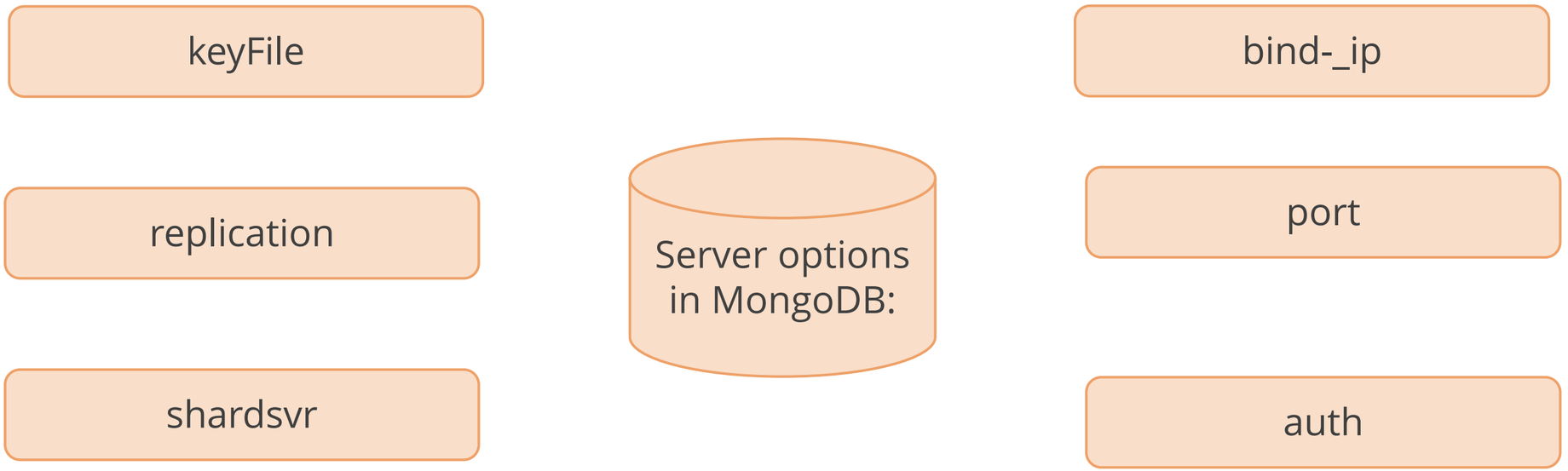
Shell and Options

Shell is a JavaScript-based interface that allows users to interact with the database using the command-line interface.



Server and Options

The MongoDB server provides access to the MongoDB database. It processes requests and responds according to the client's needs.



dbpath

The **dbpath** configuration option indicates the directory path where the data files of the MongoDB instance are stored.

```
systemLog:
  destination: file
  path: /var/log/mongodb/mongod.log
  logAppend: true
storage:
  dbPath: /data/db
net:
  bindIp: 127.0.0.1
  port: 27017
```

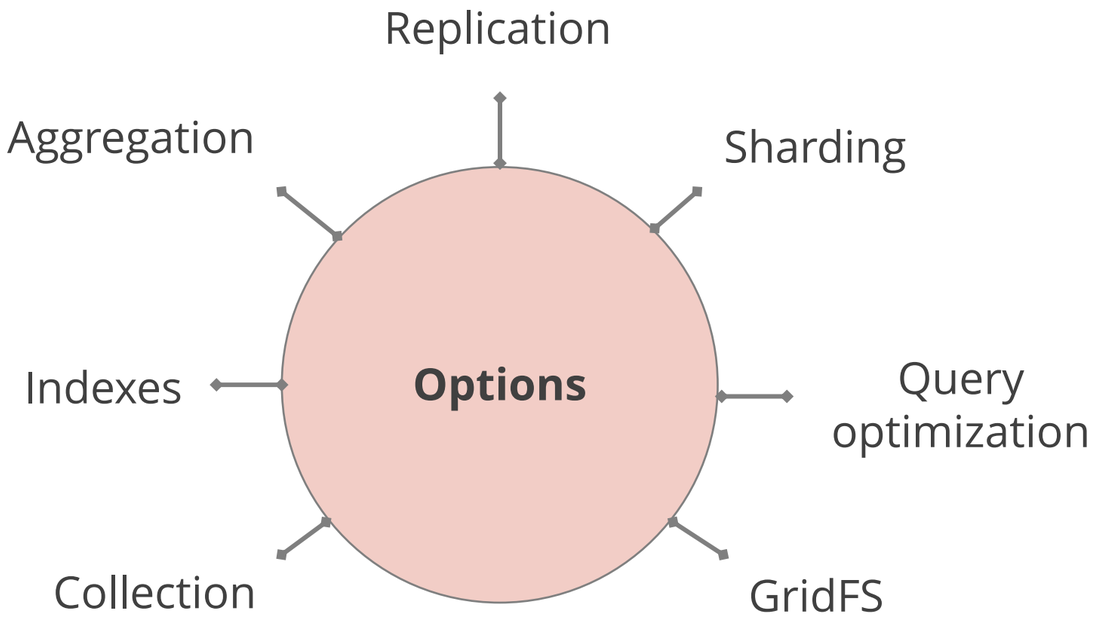
logpath

The **logpath** configuration option in MongoDB specifies the file where the log output of the MongoDB instance should be written.

```
systemLog:
  destination: file
  path: /var/log/mongodb/mongod.log
  logAppend: true
storage:
  dbPath: /data/db
net:
  bindIp: 127.0.0.1
  port: 27017
```

MongoDB Options

Below are some common options that users can explore while using MongoDB:



MongoDB as a Background Service

1. Runs automatically when the computer is started
2. Handles large volumes of data and scales up or down as needed
3. Includes replication and sharding for high availability and data durability

Use a Config file

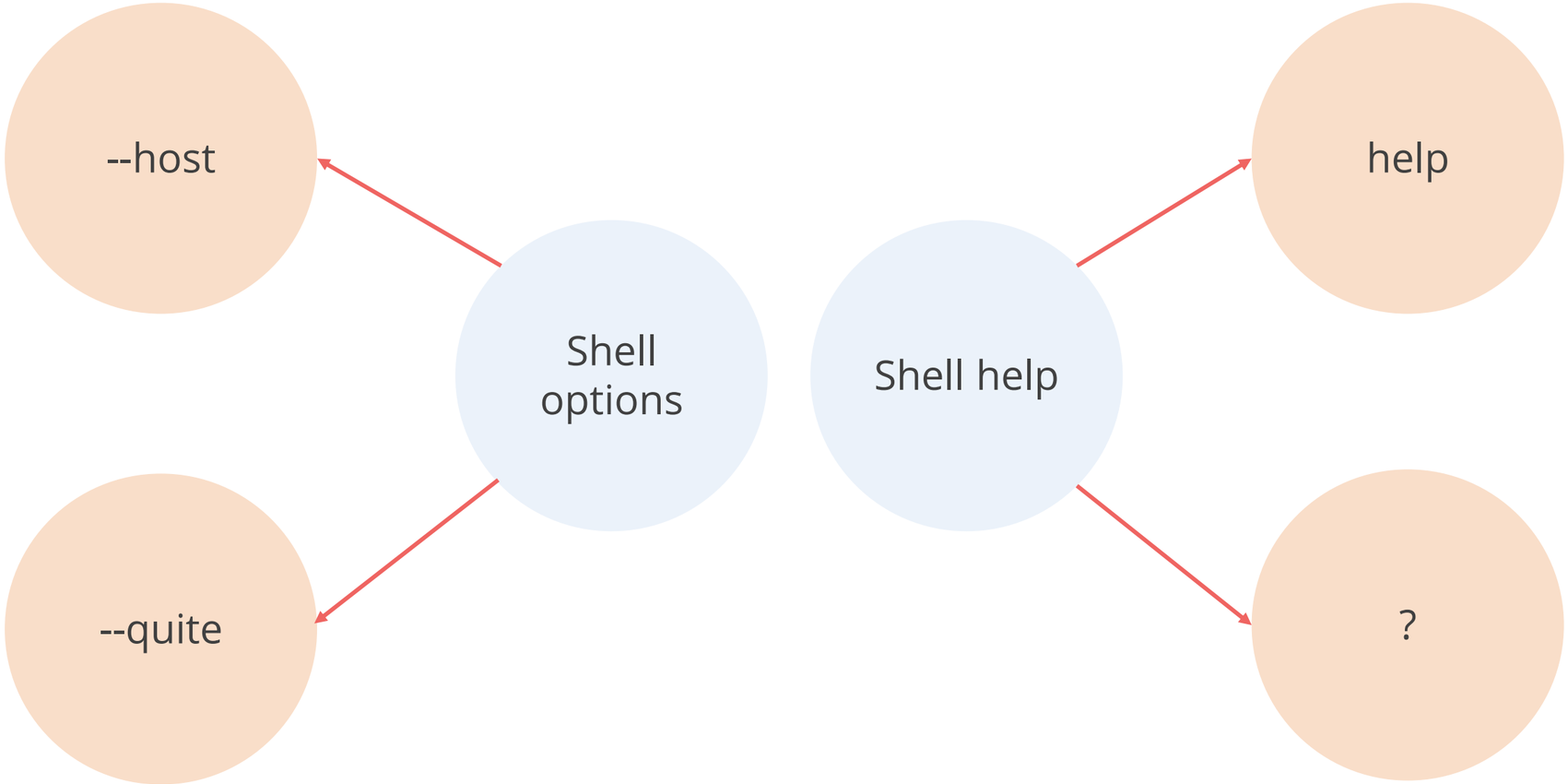
A configuration file can specify various settings and parameters that control the behavior of the database server.

- To configure mongod or mongos using a config file, specify the config file with the --config option or the -f option

```
mongod --config /path/to/mongod.conf
```

Shell Options and Help

The shell in MongoDB offers several commands and options to customize its behavior and aid the user. Here are some of the significant shell options and help features in MongoDB:



Exploring the Shell and the Server

Problem Statement: **Duration: 20 min.**

You have been assigned the task to gain hands-on experience with MongoDB Compass and Shell.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Connect to the MongoDB server
2. Create a collection and add a document
3. Explore the MongoDB Shell

Scaling and Replicating

Scaling Up vs. Scaling Out

Examine the concepts of scaling up and scaling out:

Scale up

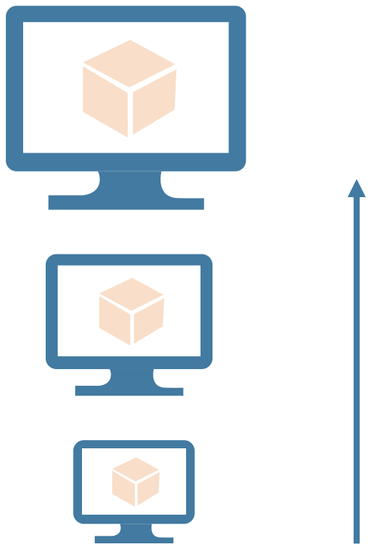
Scaling up involves increasing the resources of an individual server, such as by adding more memory or upgrading the CPU.

Scale out

Scaling out involves adding more servers to a system, such as a database cluster.

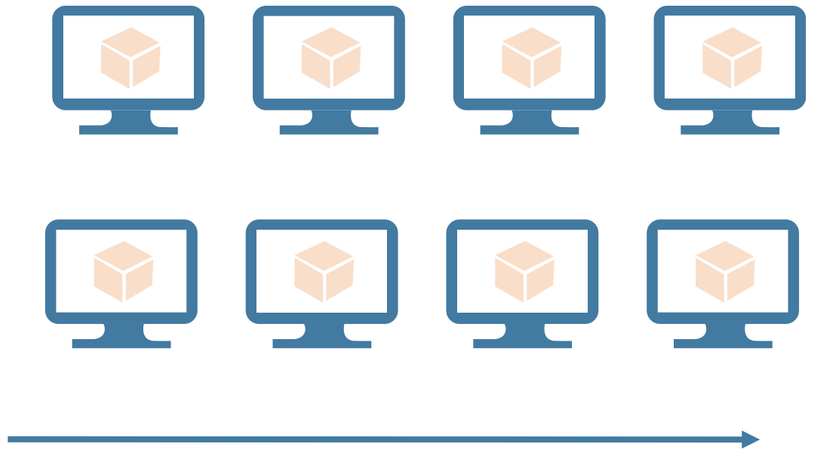
Vertical and Horizontal Scaling

Vertical Scaling



- Scaling up is also known as **vertical scaling.**
- Vertical scaling involves scaling a single system by adding more resources.

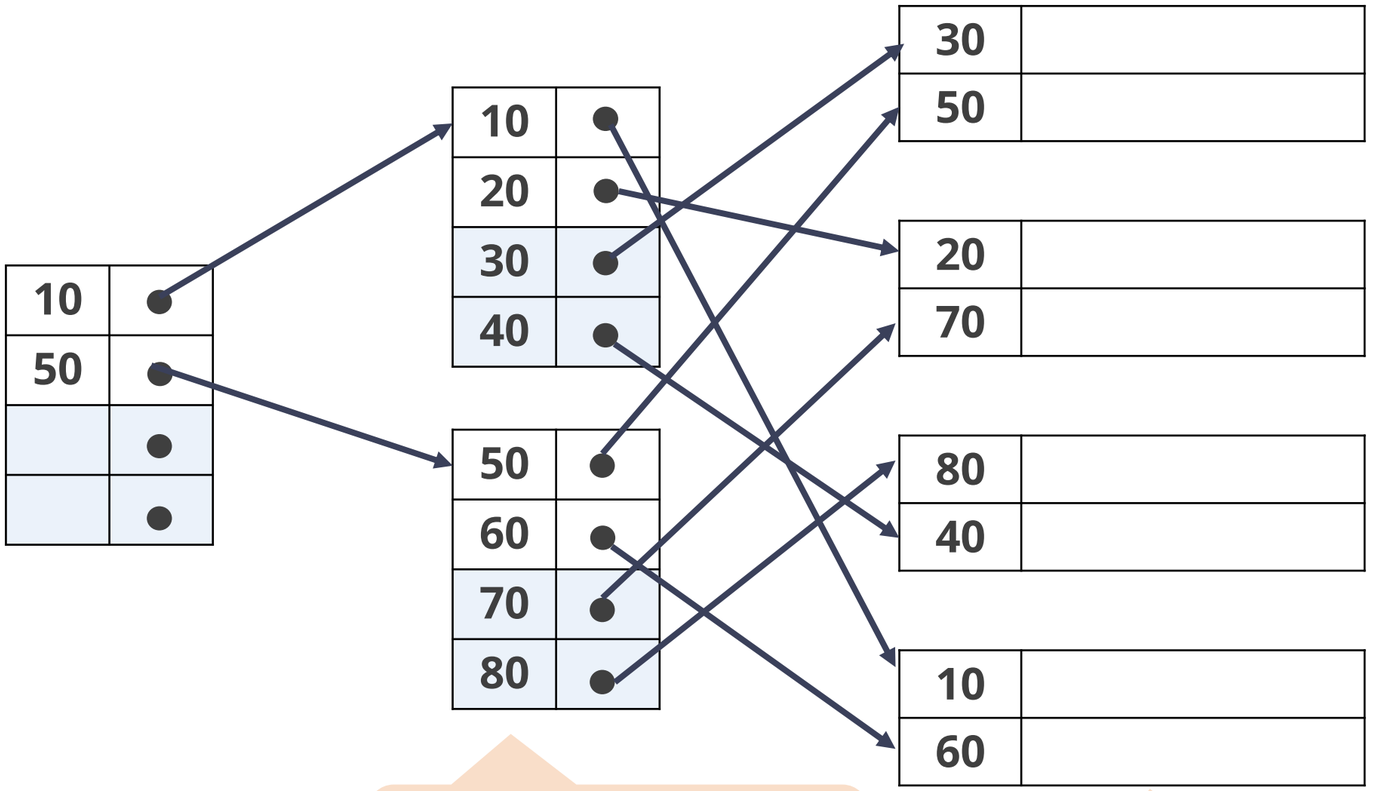
Horizontal Scaling



- Scaling out is also referred to as **horizontal scaling.**
- Scaling out involves adding more instances or systems to distribute the load on the server.

Secondary Index

A secondary index is an additional index created on a collection to improve the efficiency of queries.

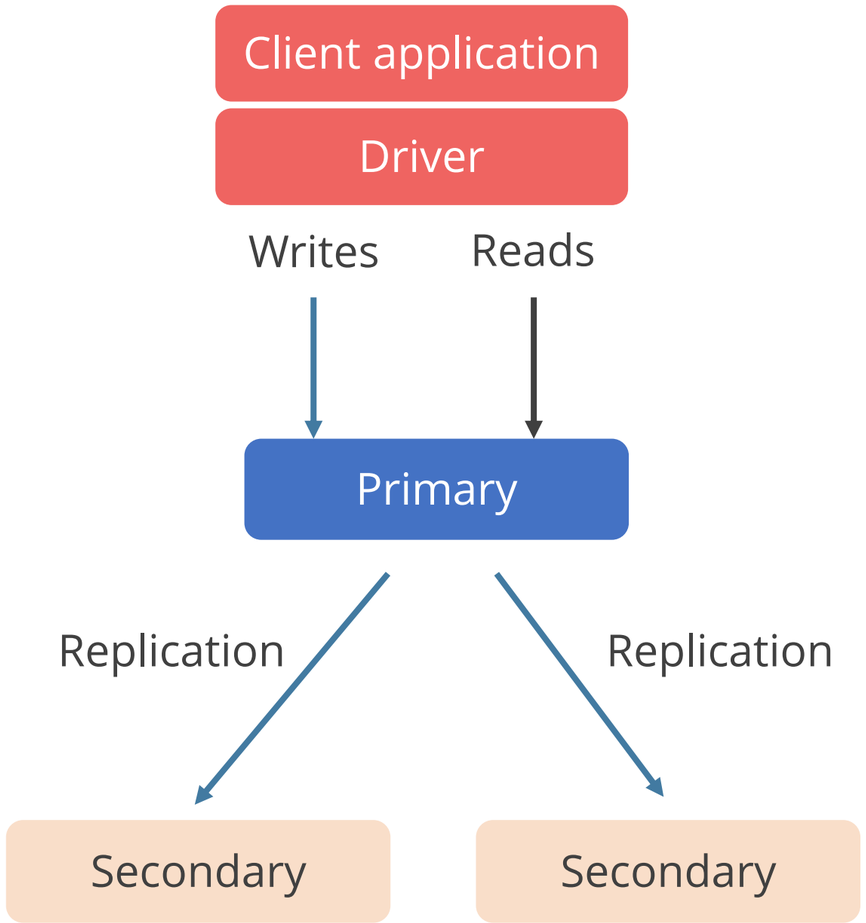


Dense index: Sorted on the secondary key

Blocks as they are

Replication

The process of creating and maintaining multiple copies of the same data across several servers is known as replication.



Memory Management

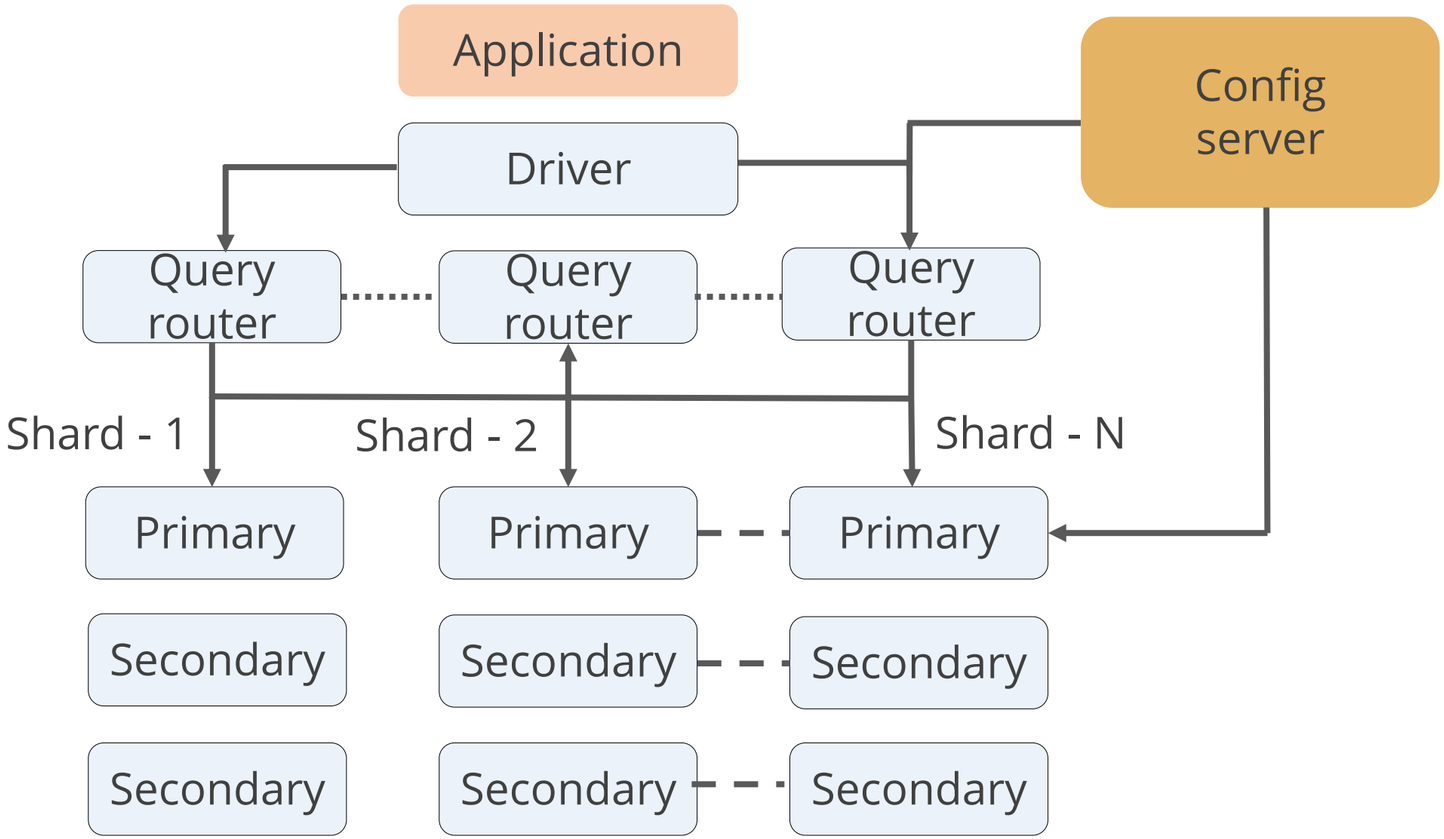
Memory Management

MongoDB uses a memory-mapped file approach and a two-tiered approach for memory management.

This approach comprises the WiredTiger cache and the operating system file system cache.

Auto-Sharding

Auto-sharding is a memory management technique that automatically distributes data across multiple machines in a cluster.



Auto-Sharding

The following are the features of auto-sharding:

01

It automates the process of creating and managing shards.

02

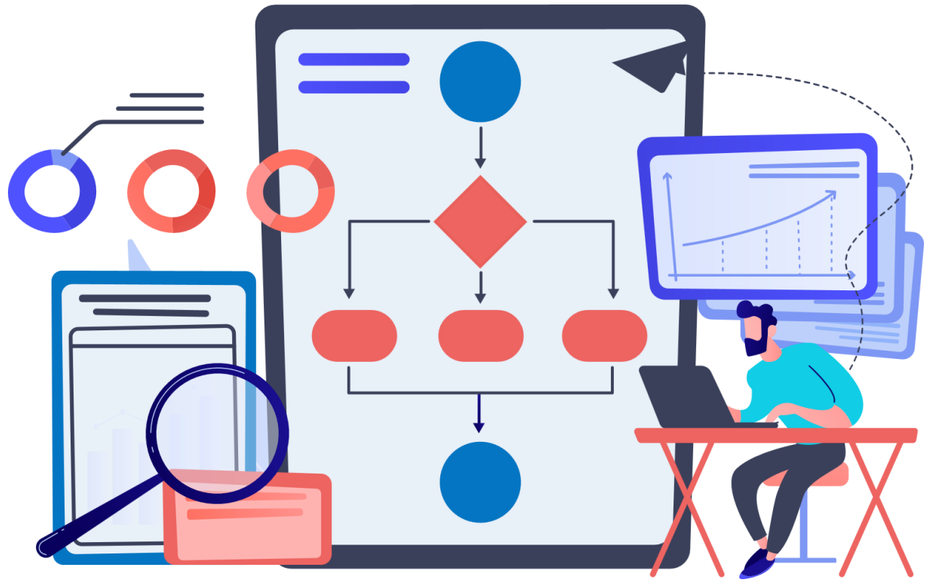
It enables MongoDB to scale out horizontally.

Aggregation

Aggregation is a data processing framework. It is a sequence of stages that are used to process and transform data.

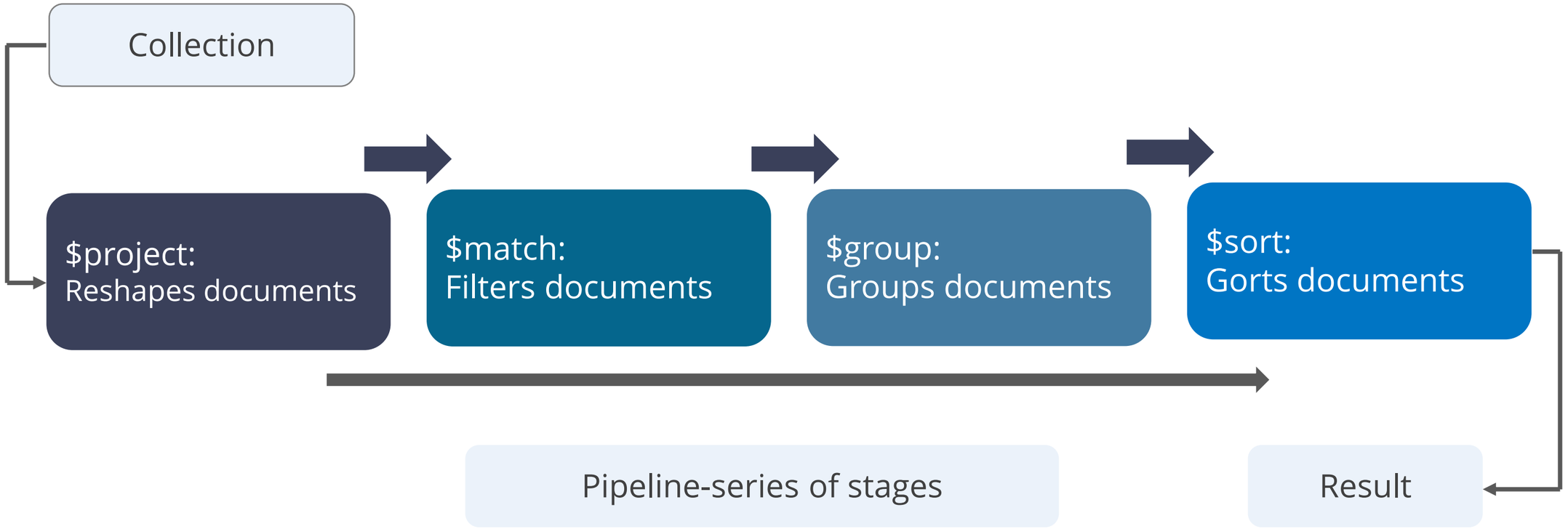
MapReduce

MapReduce is a data processing framework that allows users to process large datasets in parallel.



Stages of Aggregation

Following is a representation of stages in the aggregation pipeline:



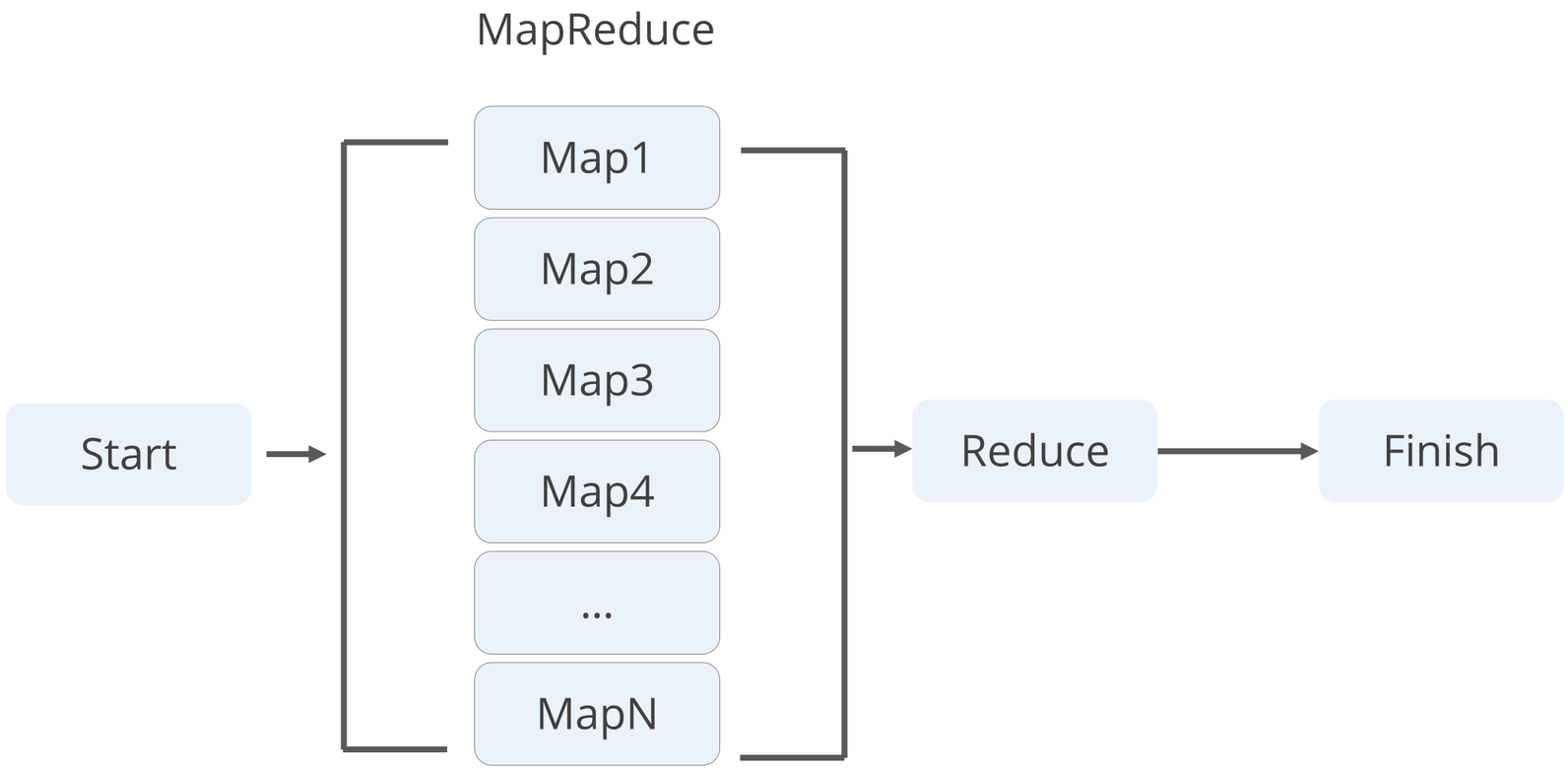
Stages of MapReduce

Following are the phases of MapReduce:

Map phase	Reduce phase
Applies a user-defined function to each document in the collection	Applies a user-defined function to the intermediate key-value pairs

Stages of MapReduce

Diagram for the phases of MapReduce:



Core Servers

Several types of servers work together to support a distributed database system, such as:

Mongod

The mongod server handles data storage, indexing, and querying. It enables database access through the MongoDB query language.

Configsvr

The configsvr server is accountable for storing metadata related to sharded clusters, such as the mapping between data shards and their corresponding servers.

Mongos

The mongos server functions as a query router in a sharded cluster, receiving incoming queries and directing them to the appropriate data shards.

MongoDB Tools

Following are the common MongoDB tools:

- mongostat
- mongotop
- Database profiler
- MongoDB management service
- MongoDB cloud manager
- Indexes
- Sharding

Managing Memory

Problem Statement:

Duration: 20 min.

You have been assigned the task to manage the memory of the databases.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Check memory usage, analyze, and optimize queries
2. Use the aggregation pipeline to perform aggregation operations

Relationships in MongoDB

What Is a Relationship?

A **relationship** refers to the way data from various collections are linked to each other.

It enables the organization to query related data in a structured and efficient manner.

Data Models to Create Relationships

Two common data models to build relationships in MongoDB are:

01

Reference data model

02

Embedded data model

Reference Data Model

A **reference data model** stores the related data in separate collections and links them using references.

Reference Data Model: Advantages

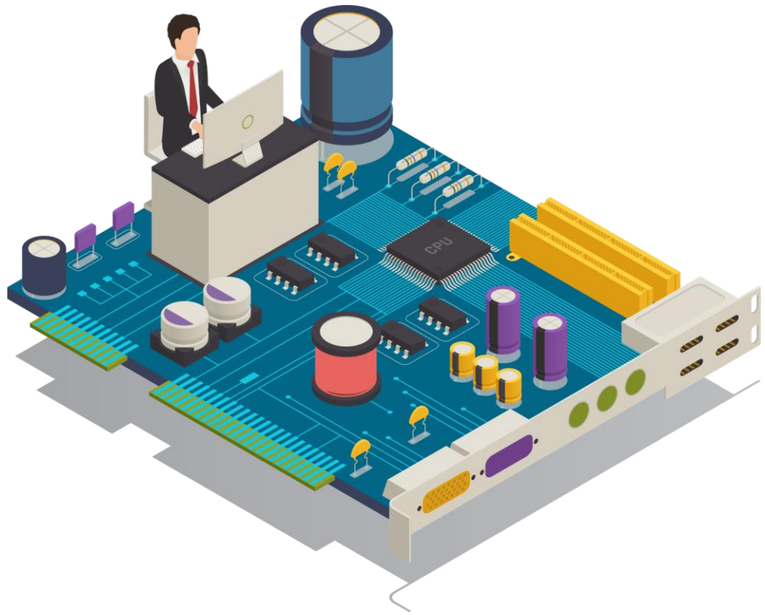
The following are the advantages of the reference data model:

It can be more flexible and scalable for write-heavy workloads.

Related data can be updated separately without impacting the performance of other operations.

Embedded Data Model

An **embedded data model** is a way of storing related data within a single document.



It is useful for datasets with smaller and simpler relationships, as it can avoid the need for complex queries and joins.

Embedded Data Model: Advantages

The following are the advantages of the embedded data model:

An embedded data model is easier to manage because all the related data is stored in a single document.

It can result in lower storage requirements than other data models.

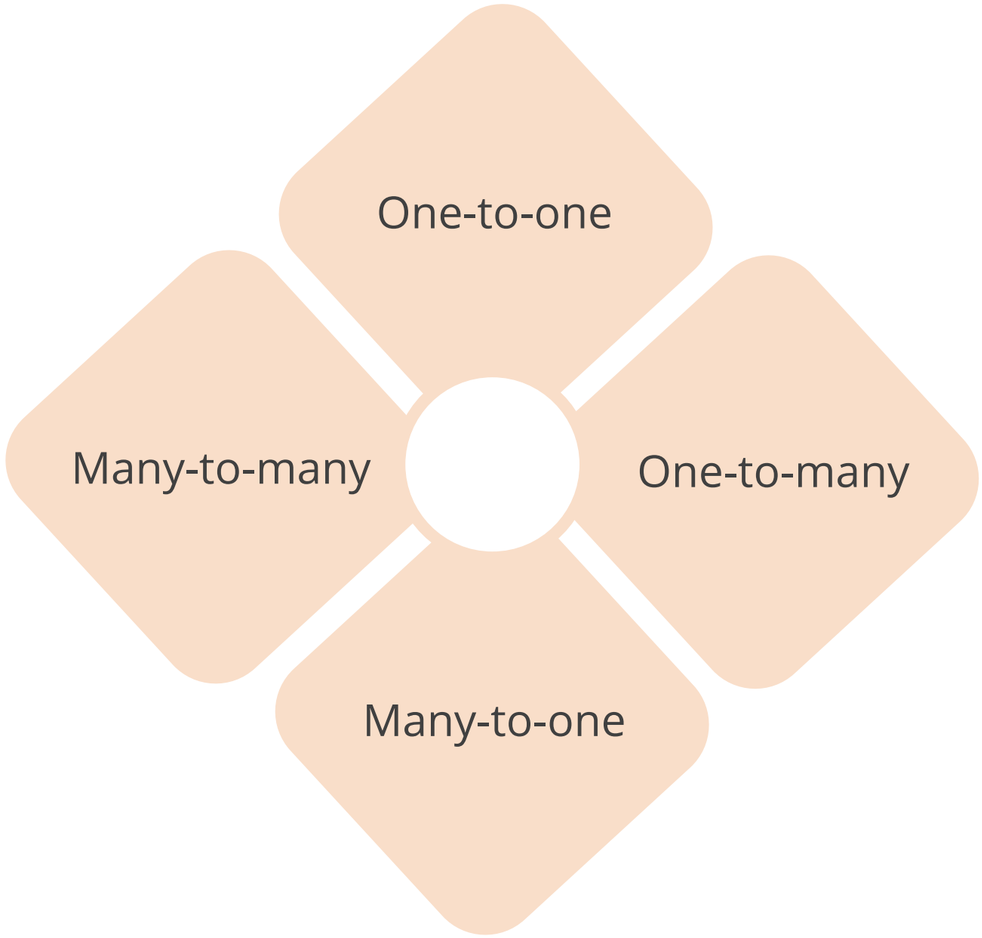
Relationships in MongoDB

Relationships refer to how data is connected or related to each other within the database.

The data can be organized into collections, and documents within those collections can be related in various ways.

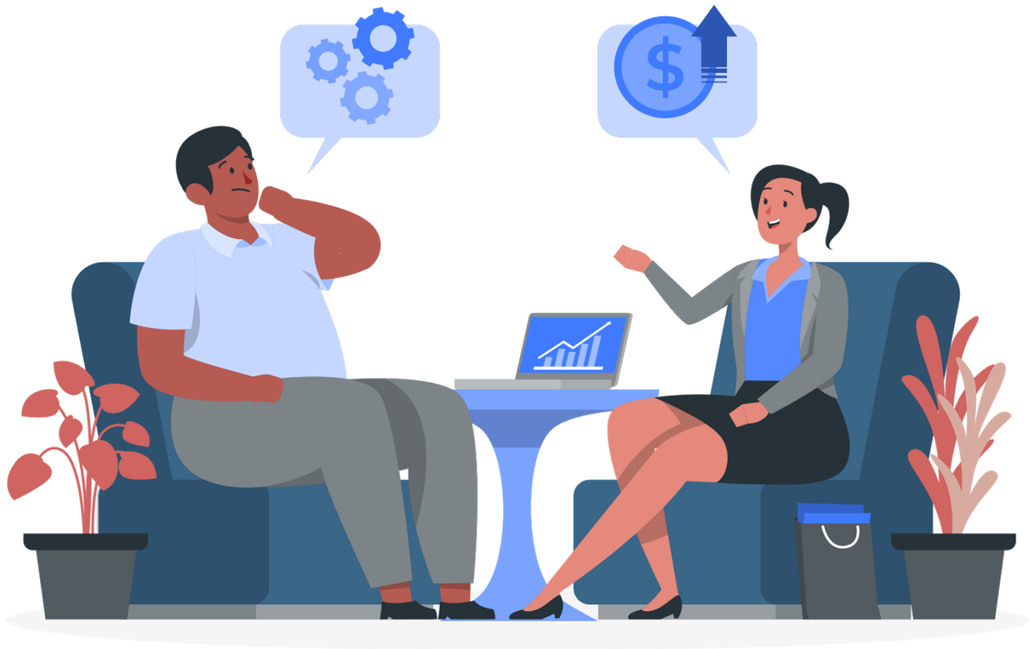
Relationships in MongoDB: Types

Following are the types of relationships in MongoDB:



One-to-One Relationship

A **one-to-one** relationship can be implemented using either the embedded data model or the reference data model, depending on the application's specific requirements.



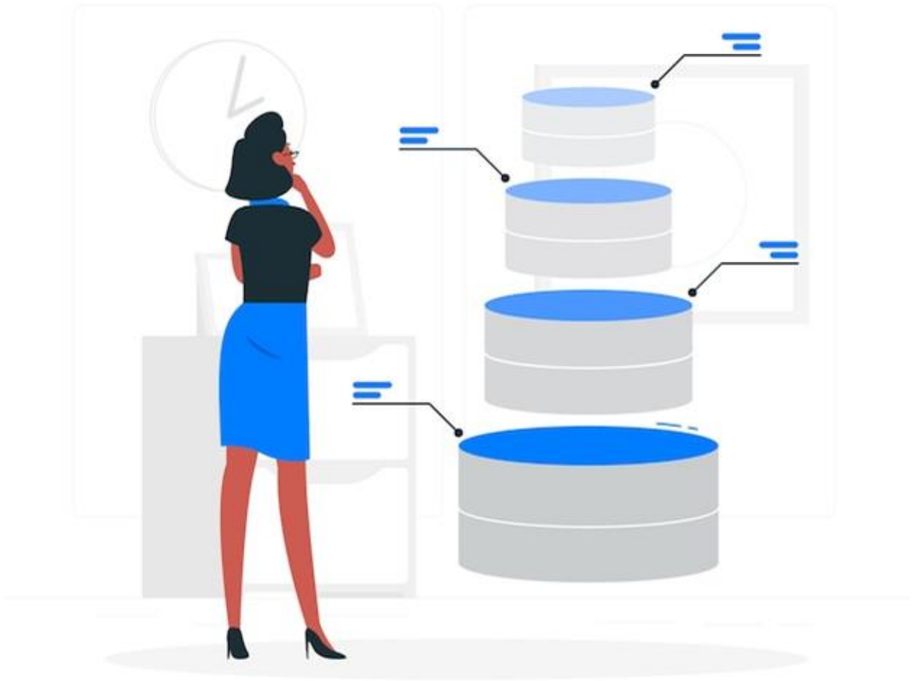
One-to-Many Relationship

A **one-to-many** relationship can be implemented using either the embedded data model or the reference data model.

Related data is stored within a single document as sub-documents or arrays.

Many-to-One Relationships

A **many-to-one** relationship can be implemented using the reference data model.



Many-to-Many Relationships

A **many-to-many** relationship can be implemented using the reference data model with an additional join collection.

This involves storing the related data in separate collections and linking them using references through a join collection.

Relationships in MongoDB

Problem Statement: **Duration: 15 min.**

You have been assigned the task to perform relationships in MongoDB on a remote desktop.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Connect to MongoDB Compass by using virtual machine extensions
2. Create a database in MongoDB Compass
3. Create collections in the database
4. Perform relationships between documents

Key Takeaways

- Learning schema involves comprehending how to design and organize databases in a safe, scalable, and effective manner.
- Collection document validation is a feature in MongoDB that allows developers to enforce rules and constraints on the data inserted or updated in a collection.
- The MongoDB server is the component that stores and manages the data in a database.
- A relationship refers to how data from different collections are connected.

Thank You