Develop a Reliable Backend with Node and Express

Getting Started with MongoDB

A Day in the Life of a MERN Stack Developer

Trevor, a MERN stack developer, is working on a financial and commercial services project. He must work on large volumes of data in a database. However, the end users want to save and retrieve data without loading large files into memory.

In this lesson, Trevor 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:

- Define the basic concepts of MongoDB to provide a scalable, high-performance platform for handling diverse and unstructured data efficiently
- Elaborate the importance of MongoDB making it suitable for applications requiring fast iterations and scalability
- Analyze the concept of databases to provide a structured and efficient way to access, manage, and manipulate information
- Describe how to handle data storage to ensure data integrity, accessibility, and scalability while adhering to performance and security requirements

Introduction to MongoDB

Enhancing and Organizing

JSON: JSON i.e. JavaScript Object Notation is a human-readable data interchange format.

BSON: BSON’s binary structure encodes type and length information, which allows it to be traversed much more quickly compared to JSON.

What Is MongoDB?

MongoDB is a free, open-source document-oriented database developed by **MongoDB Inc.**

It is designed to store large amounts of data that allows the users to work efficiently.

MongoDB: Features

01 Is categorized under NoSQL database management system

02 Stores unstructured and semi-structured data

03 Improves scalability, performance, and availability

04 Supports indexing and querying capabilities

Importance of MongoDB

Reasons why MongoDB can be a good choice for the application:

01 High availability

02 Better querying and indexing

03 Easy application

04 Good performance

05 Open-source database

06 Large community and ecosystem

MongoDB Structure

MongoDB structure contains the following components:

Database: Provides easily scalable and high-performance data access

Fields: Represents complex data structures

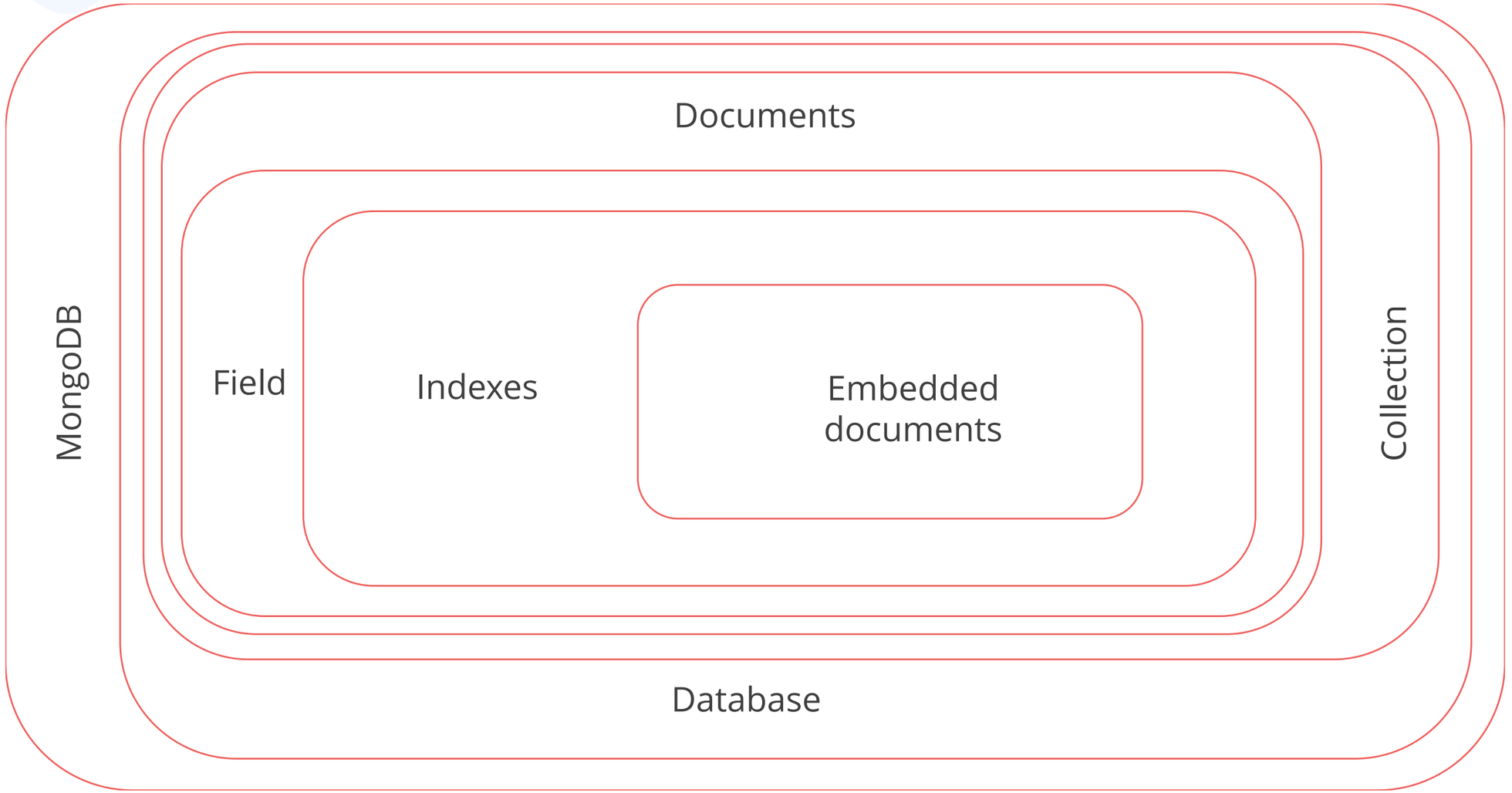
Collections: Stores query data in a flexible manner

Indexes: Stores a small subset of data from a collection

Documents: Creates a collection of key-value pairs

Embedded documents: Organizes related data within a single document

MongoDB Structure



MongoDB structure

MongoDB: Advantages

The following are the advantages of MongoDB:

01 Flexible data model

02 Scalability

03 High performance

04 Rich querying language

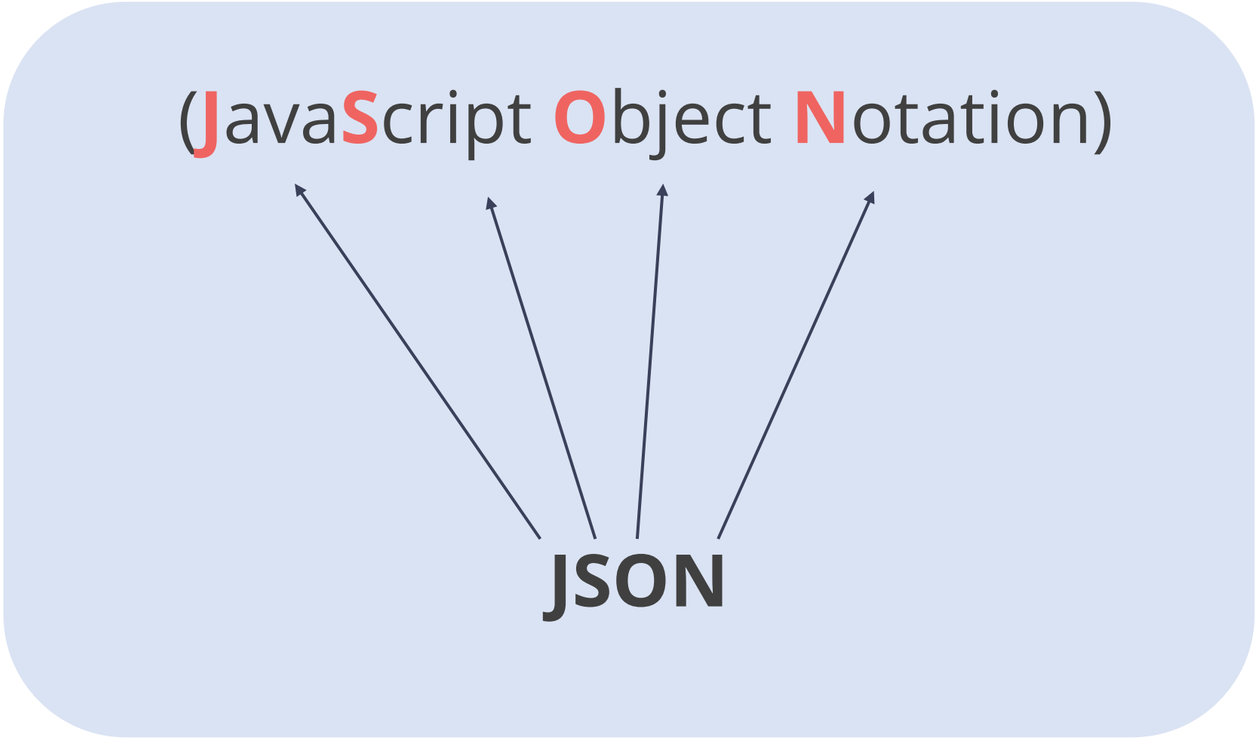
05 Low overhead

06 Flexible deployment options

07 Easy integration with other technologies

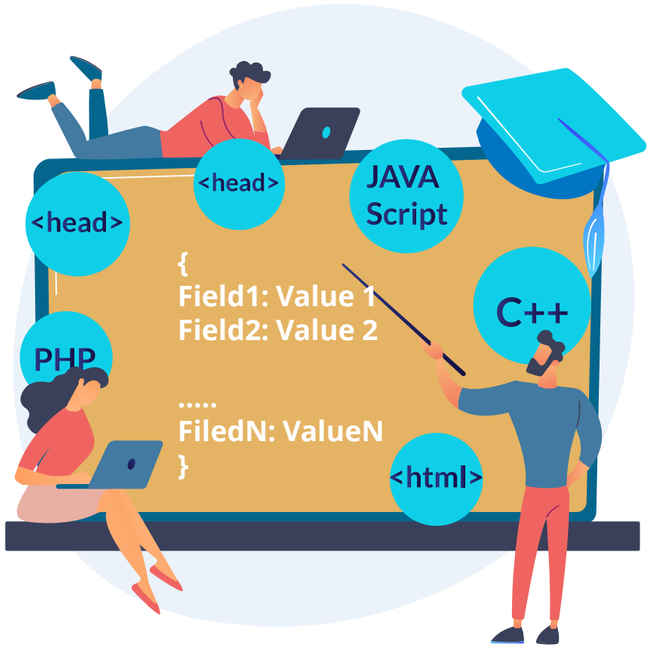
Introduction to JSON

JavaScript Object Notation, also known as JSON, is one of the most popular data transition formats.



It is a text-based and lightweight composition.

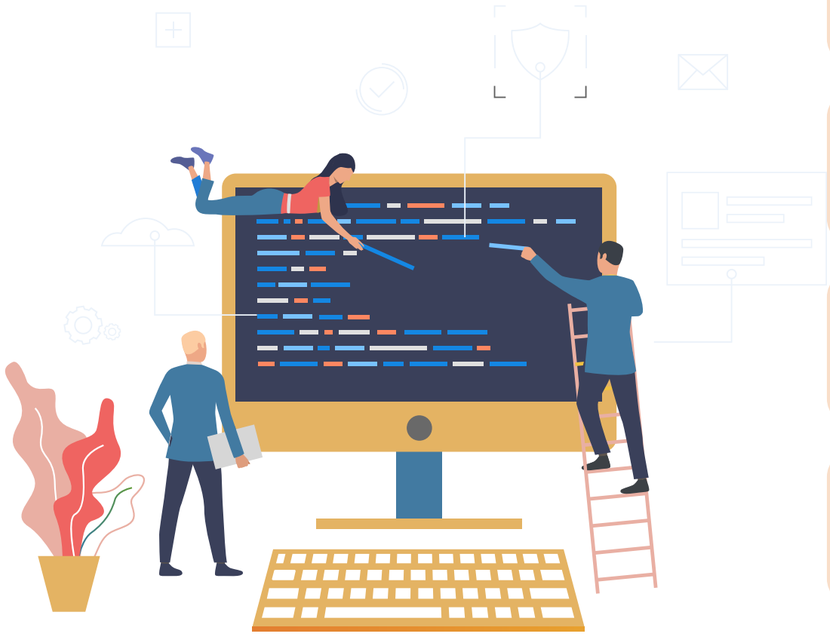
Properties of JSON



- It is a **data interchange format** that is easy to read and write for humans and machines.
- It is based on a subset of the JavaScript programming language standard.
- It is used for transmitting data between a client and a server in a web application.

Importance of JSON

Following are the reasons to use JSON:



- Transmits data between a server and web applications
- Helps web services and APIs to provide public data
- Transmits data between a client and a server in web applications
- Consists of extremely lightweight and language independent structure
- Serializes and transmits structured data over a network

JSON: Format

Following are the basic rules for JSON format:

- JSON represents data objects in a structured format.
- The object is surrounded by curly braces {}.
- A key-value pair consists of a key and a value, separated by a colon (:).

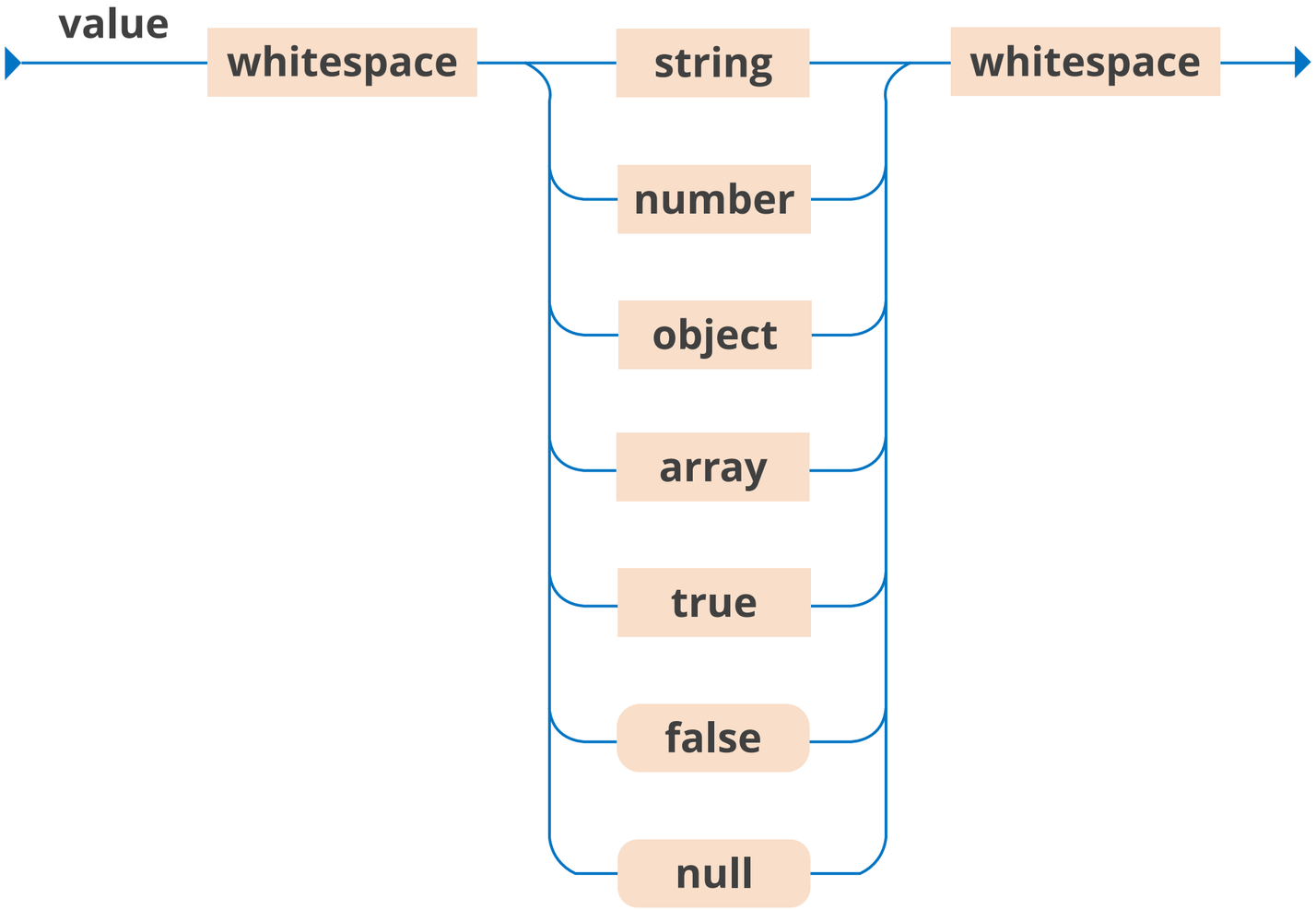
```
{

 Field1: Value1
 Field2: Value2
 .....
 FieldN: ValueN

 }
```

JSON: Structure

The following is a structural diagram of JSON:



JSON: Example

Here is an example of a simple JSON object:

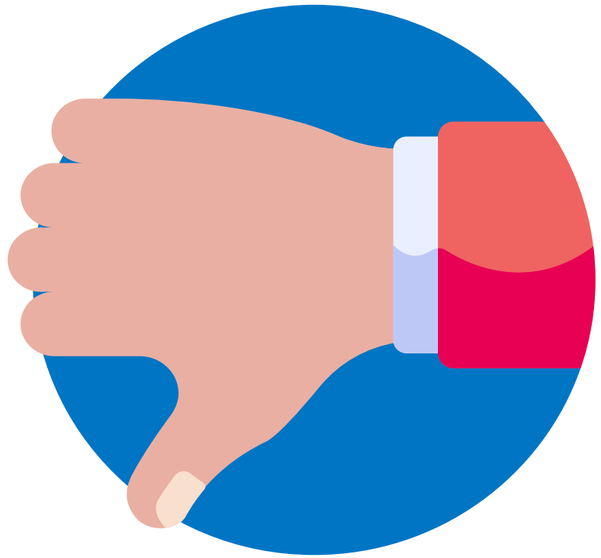
```
{
  "name": "John Smith",
  "age": 30,
  "isStudent": true,
  "hobbies": ["reading", "hiking", "photography"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zip": "12345"
  }
}
```

Drawbacks of JSON

Supports a limited number of basic data types

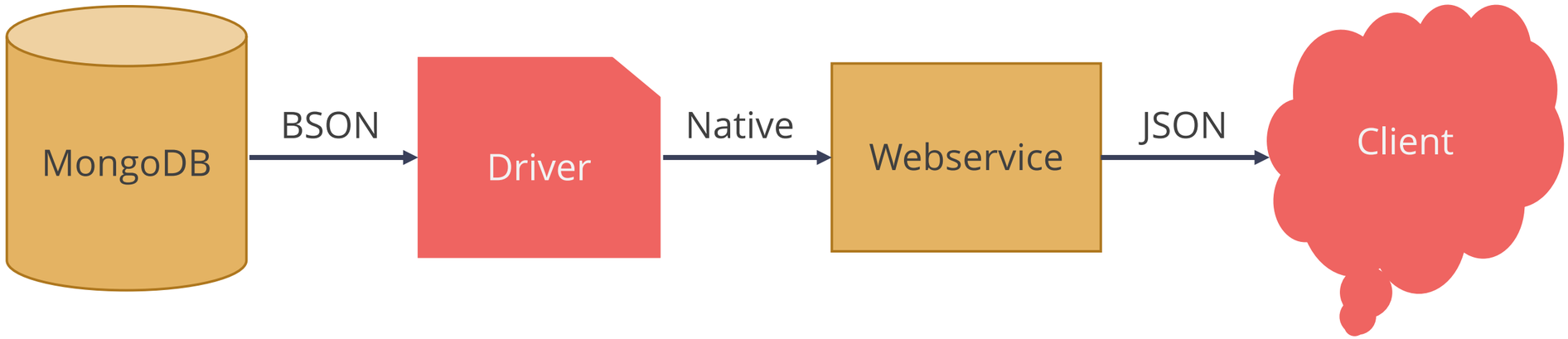
Lacks support for dates and binary data

Slows down the traversal as the objects and properties don’t have a fixed length



Introduction to BSON

BSON (Binary JavaScript Object Notation) is a computer data interchange format. It is a binary file format that stores serialized JSON documents in a binary-encoded format.



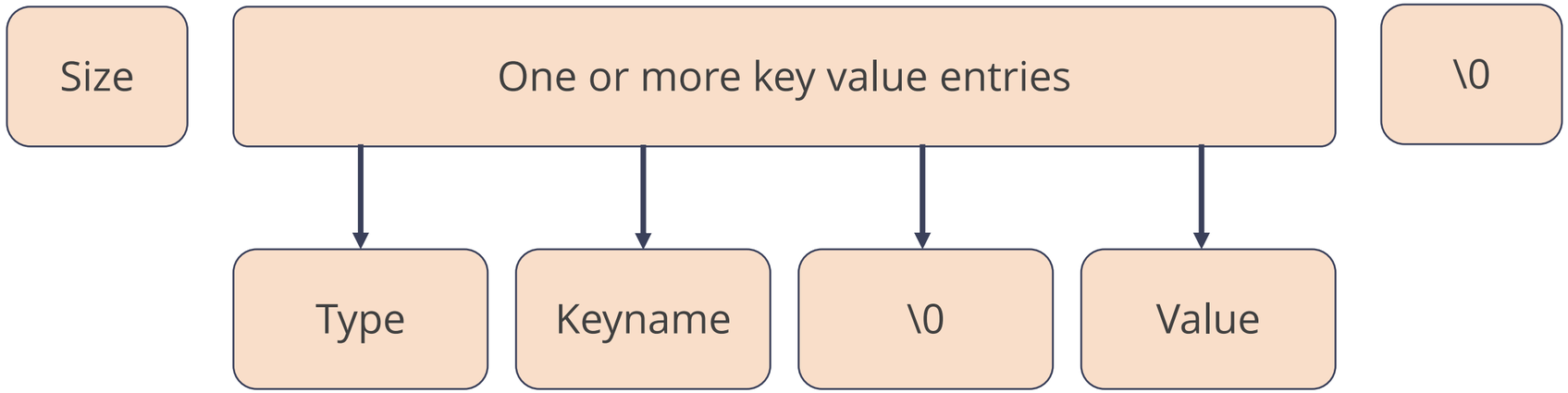
Importance of BSON

The reasons to use BSON are:

- More compact and efficient than JSON
- Lightweight
- Highly traversable
- Less spacious and faster scan speed
- Easily and quickly parsed
- Wide range of data types

BSON Format to Store Data

BSON documents consist of a sequence of fields, where each field is a key-value pair.



The key is a string that identifies the field, and the value can be of any BSON data type.

BSON

The following are the datatypes supported by BSON (Binary JSON):

Double	Boolean	JavaScript code with scope
String	Date	32-bit integer
Object	Null	Timestamp
Array	Regular expression	64-bit integer
Binary data	JavaScript code	Decimal128
ObjectID	Symbol	

BSON Document: Example

Here is an example of a BSON document that represents a simple employee record:

```
\x2A\x00\x00\x00          // length of document
\x08                      // type of field (string)
\x5F\x69\x64\x00          // field name (_id)
\xB7\x1D\x14\xF8\x53\x2E // ObjectId value
\x10                      // type of field (int32)
\x61\x67\x65\x00          // field name (age)
\x1E\x00\x00\x00          // int32 value (30)
\x02                      // type of field (string)
\x66\x69\x72\x73\x74\x4E // field name (first_name)
\x05\x00\x00\x00\x4A\x6F\x68\x6E\x00 // string value (John)
\x02                      // type of field (string)
\x6C\x61\x73\x74\x5F\x6E\x61\x6D\x65 // field name (last_name)
\x04\x00\x00\x00\x44\x6F\x65\x00 // string value (Doe)
```

JSON vs. BSON

Following are the differences between JSON and BSON:

JSON	BSON
• It is a text-based format.	• It is a binary-encoded format.
• It supports strings, numbers, objects, arrays, and boolean values.	• It supports the same data types as JSON, plus additional types.
• It is easily parsed and serialized with standard libraries and tools.	• It requires specialized libraries and tools.
• It is suitable for data interchange between web applications and APIs.	• It is suitable for storage and retrieval of data in MongoDB.
• It is human-readable.	• It is not human-readable.

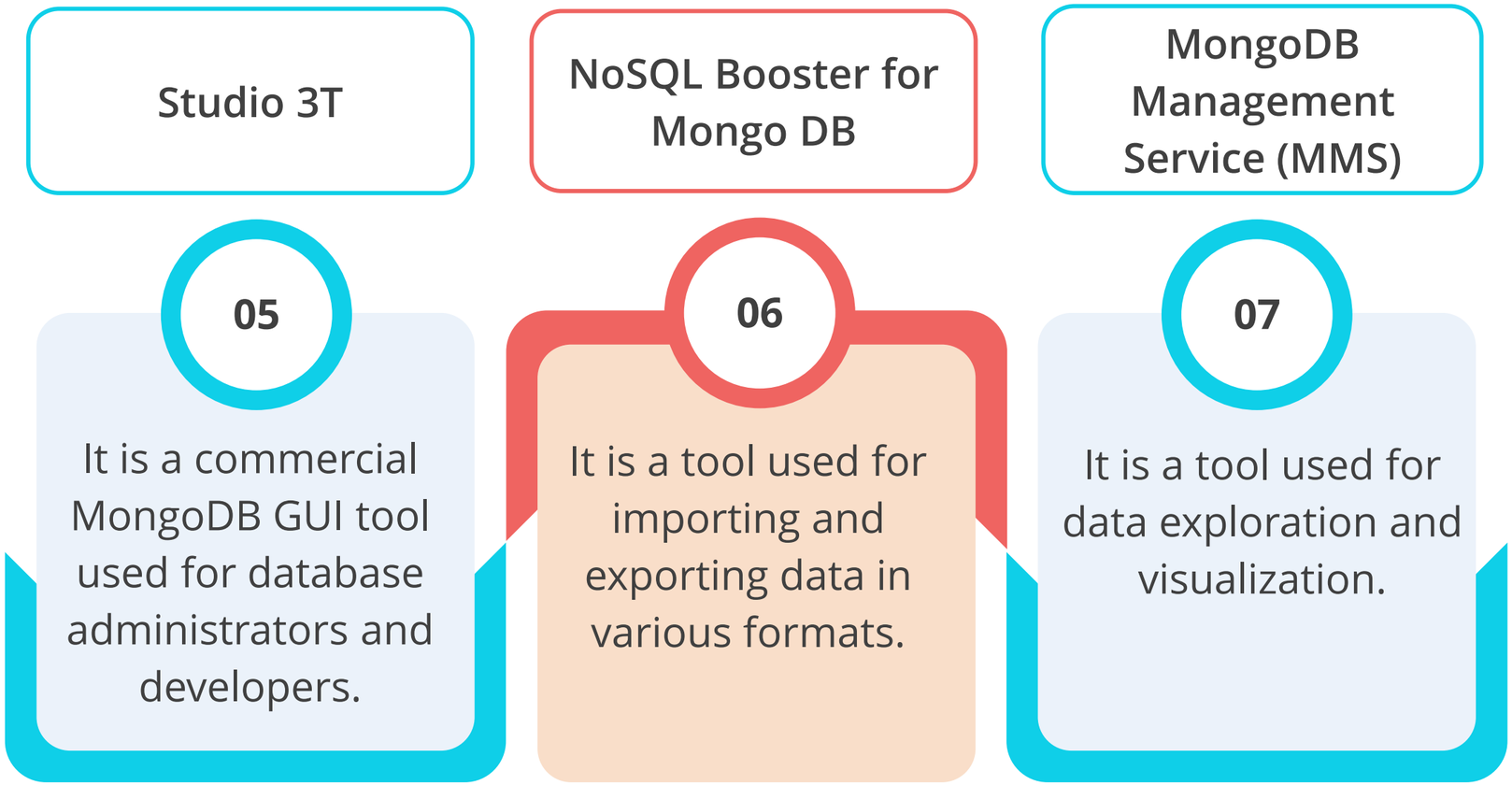
MongoDB GUI Tools

MongoDB offers several graphical user interface (GUI) tools that provide a user-friendly interface for managing MongoDB databases. Some of the MongoDB GUI tools are:

MongoDB Compass	Robo 3T	Robo Mongo	HumongouS.io
01	02	03	04
It visualizes the schema, creates and edits documents, and runs ad-hoc queries.	It builds the query, edits documents, and monitors the server.	It provides support for multiple tabs and connections.	It performs as a cloud-based MongoDB GUI tool.

MongoDB GUI Tools

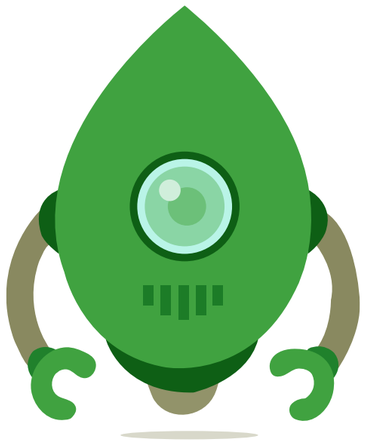
Following are some of the MongoDB GUI tools:



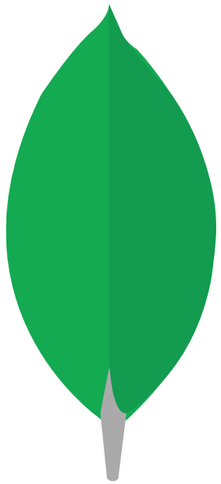
Major MongoDB GUI Tools

Some of the important MongoDB GUI tools are:

Studio 3T



Robo 3T



MongoDB Compass

Studio 3T

Studio 3T is made for growing professional teams. It offers a variety of ways to view and interrogate data collections, including sophisticated aggregations.

It enables teams to create, manage, and visualize their MongoDB databases with ease.

Studio 3T: Features

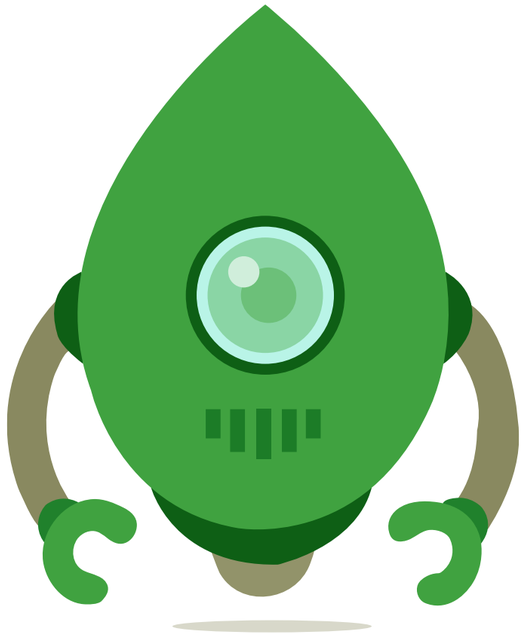
Some of the key features of Studio 3T are:

- Visual query builder
- Aggregation editor
- Schema explorer

- SQL importer/exporter
- IntelliShell
- SQL query
- Data compare and sync

Robo 3T

Robo 3T is a lightweight, open-source MongoDB GUI with an embedded Mongo shell and a real auto-completion.

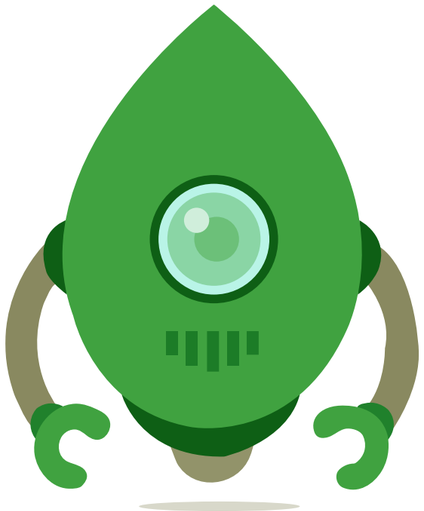


It helps developers and database administrators to work with MongoDB databases easily.

Robo 3T: Features

Some of the key features of Robo 3T are:

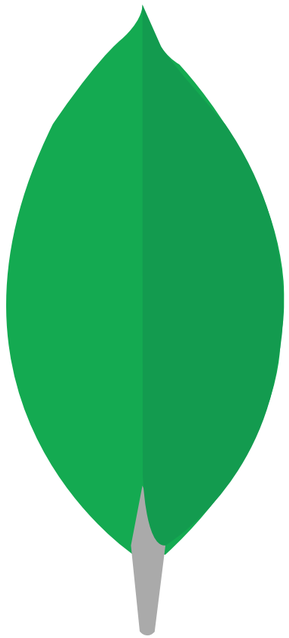
- Intuitive user interface
- Visual query builder
- Shell integration



- JSON editor
- MongoDB Atlas integration
- Export and import
- Schema visualization

MongoDB Compass

MongoDB Compass is a graphical user interface (GUI) for working with MongoDB databases.

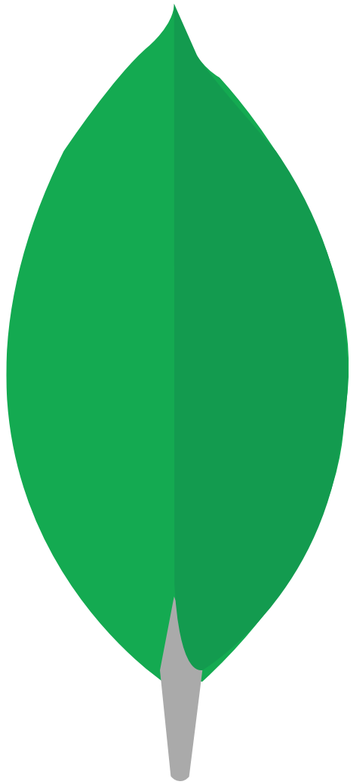


It allows users to easily visualize and manipulate data and perform administrative tasks on their databases.

MongoDB Compass: Features

Some of the key features of MongoDB Compass are:

- Data visualization
- Query building
- Schema analysis
- Index management
- Data manipulation



JSON and BSON Structure

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

You have been assigned a task to gain basic understanding of JSON and BSON structure for application in MongoDB.

ASSISTED PRACTICE

Assisted Practice: Guidelines

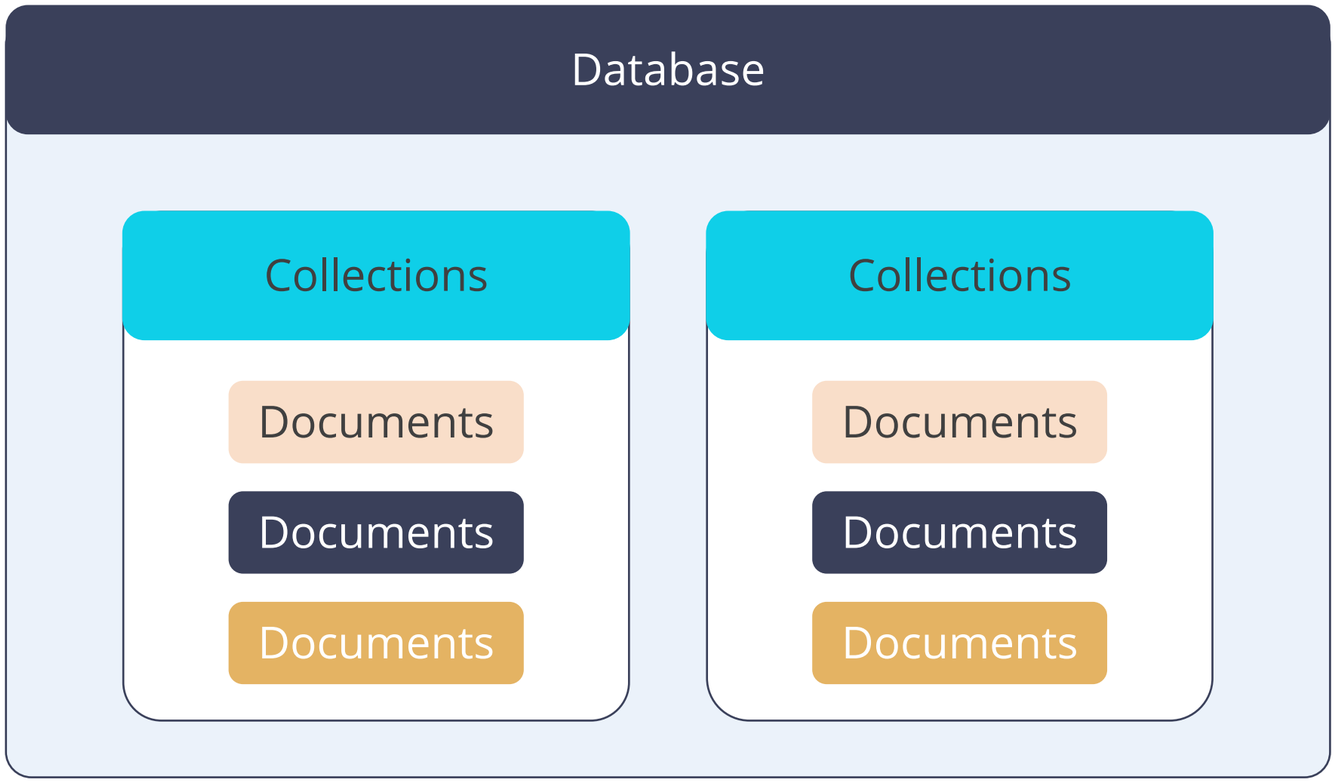
Steps to be followed:

1. Open Visual Studio code using virtual machine extensions
2. Use the cJSON library
3. Define a JSON object using a JSON file
4. Convert the JSON object to a string
5. Use the BSON library
6. Define a BSON object
7. Append the data to the BSON Document

Databases, Collections, and Documents

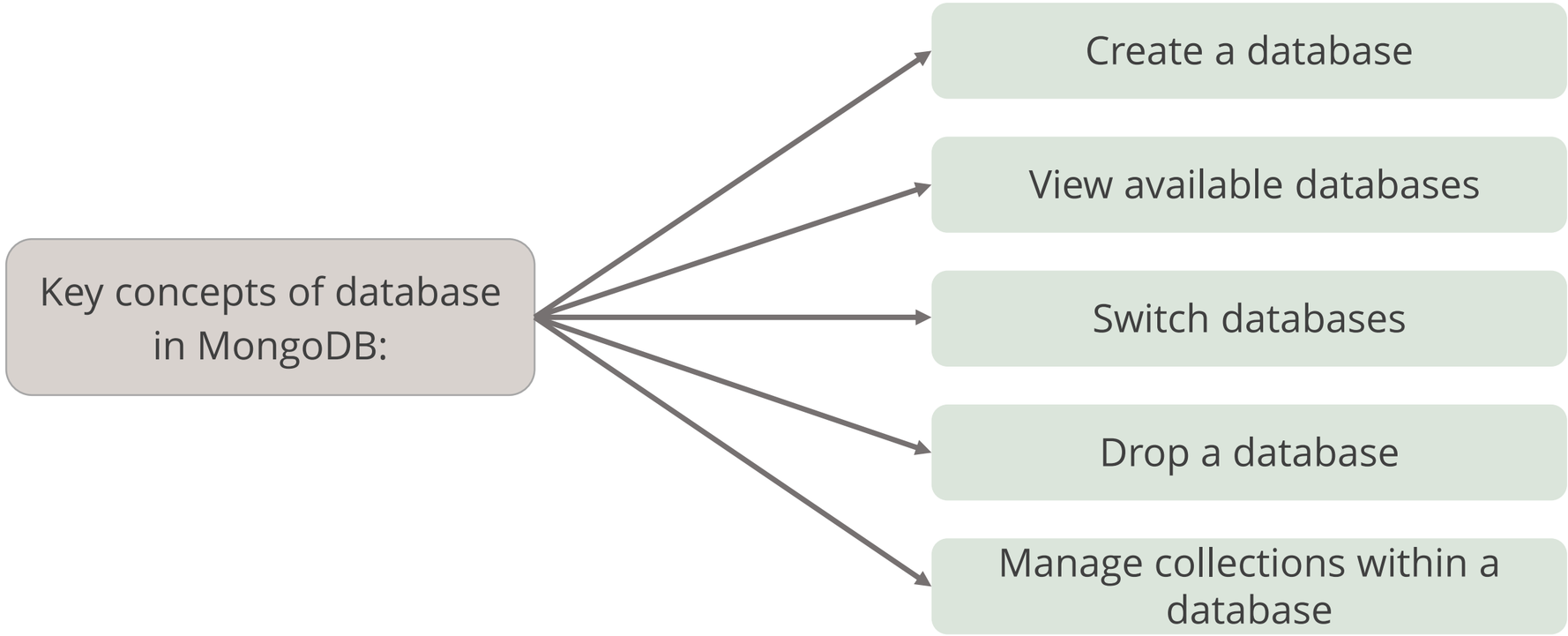
Introduction

The fundamental building blocks for storing data in MongoDB are databases, collections, and documents.



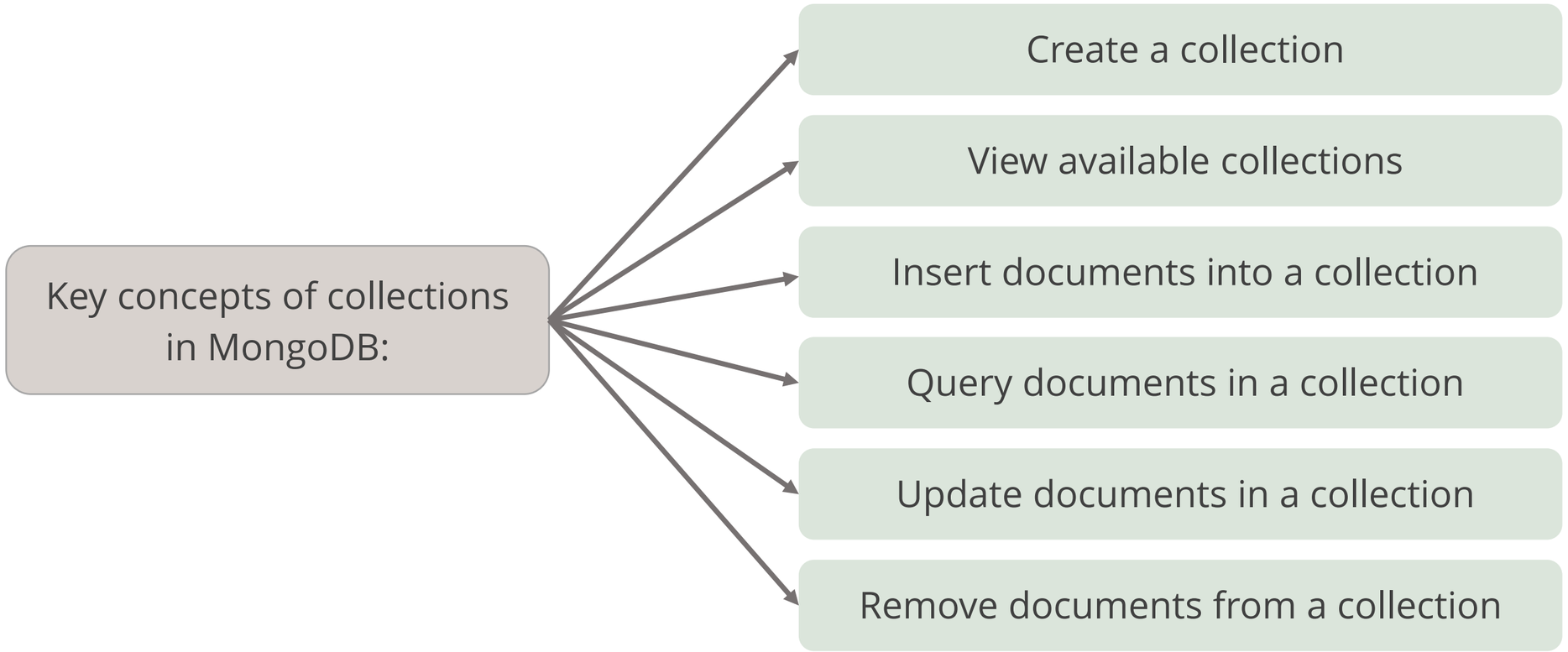
Databases in MongoDB

A **database** is a container for the collection of documents.



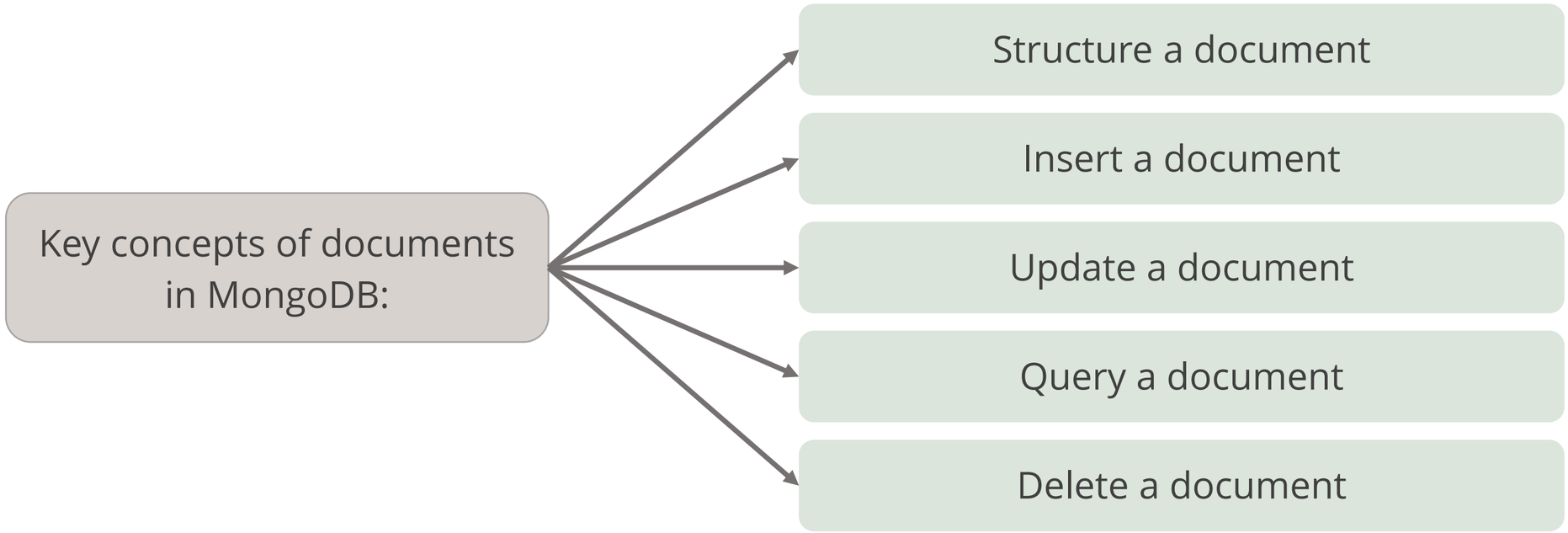
Collections in MongoDB

A **collection** is a group of MongoDB documents equivalent to a table in a relational database.



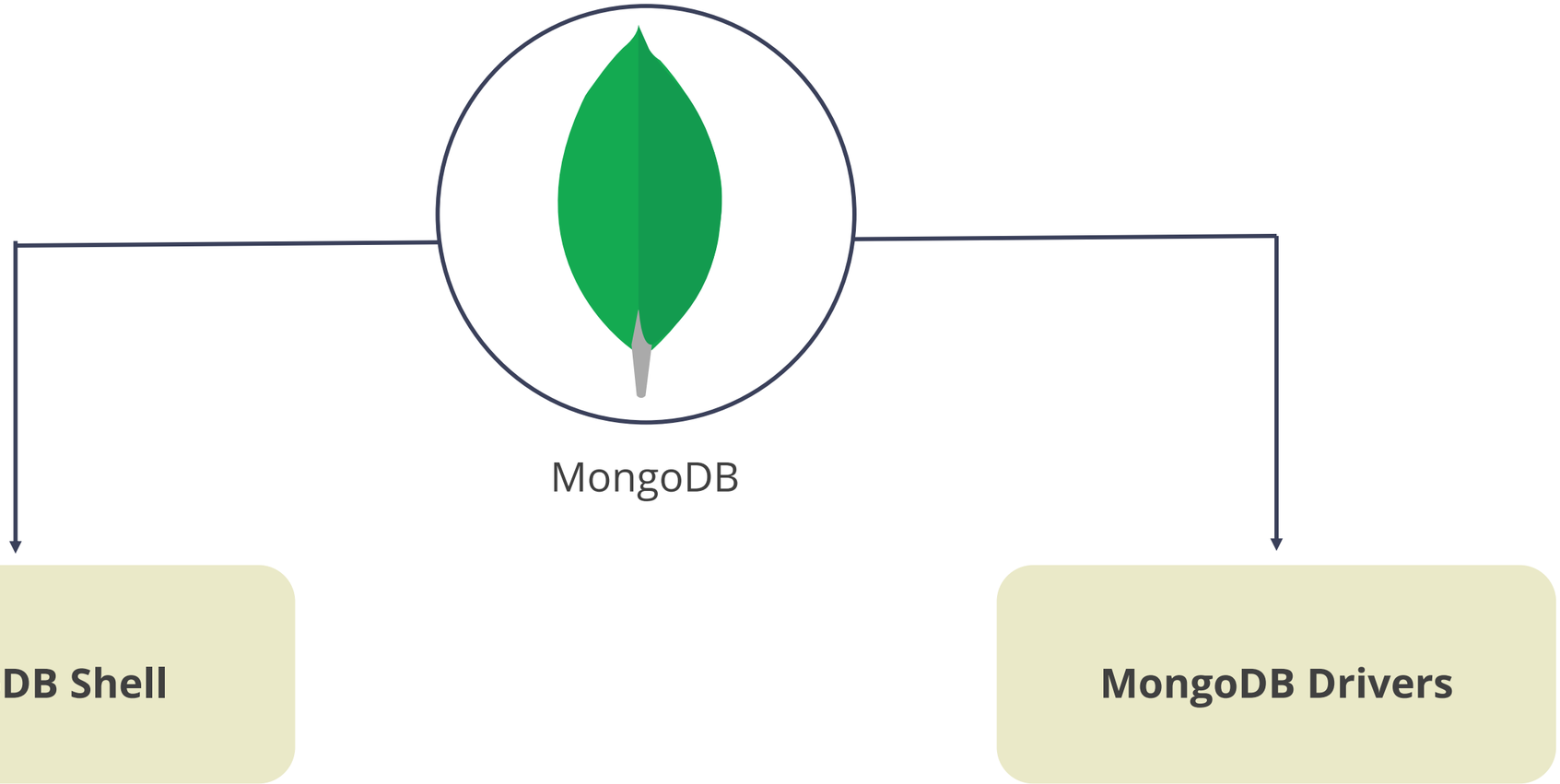
Documents in MongoDB

A **document** is a set of key-value pairs representing a single object in a collection.



MongoDB Shell and Drivers

The following are the essential tools for working with MongoDB:



MongoDB Shell and Drivers

MongoDB Shell is a command-line interface that allows users to interact with MongoDB databases.

It supports powerful tools for developers and database administrators.

MongoDB Shell and Drivers

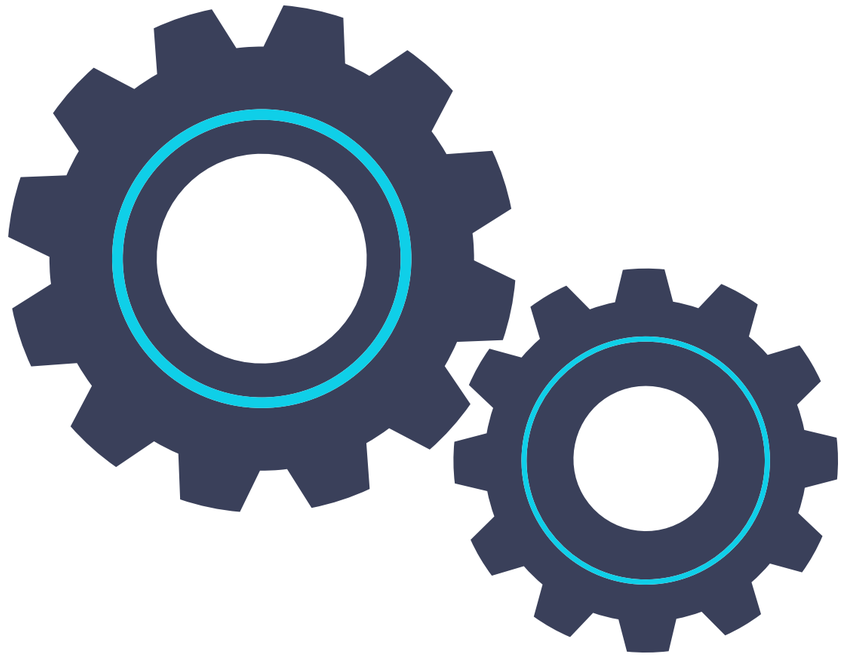
MongoDB shell is available for the following operating systems:

macOS

- **Windows**
- **Linux**
- **macOS**

MongoDB Shell and Drivers

MongoDB drivers are software components that enable applications to communicate with MongoDB databases.



To interact with MongoDB, users can use different programming languages and drivers.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

Java

MongoDB allows developers to connect to MongoDB databases from Java applications. The driver is compatible with all major Java frameworks and supports asynchronous programming.

Python

MongoDB provides a Python driver that allows developers to connect to MongoDB databases from Python applications. PyMongo is the official MongoDB driver for Python.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

C++

MongoDB provides a C++ driver that allows developers to connect to MongoDB databases from C++ applications. The driver is compatible with all major C++ compilers and supports synchronous and asynchronous programming.

C#

MongoDB provides a C# driver that allows developers to connect to MongoDB databases from .NET applications. The driver is compatible with all major .NET frameworks.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

Ruby

MongoDB provides a Ruby driver that allows developers to connect to MongoDB databases from Ruby applications. The driver is compatible with Ruby 1.9 and later versions and supports asynchronous programming.

Node.js

MongoDB provides a Node.js driver that allows developers to connect to MongoDB databases from Node.js applications. The driver is compatible with Node.js 8.x and later versions and supports asynchronous programming.

Create Databases and Collections

The steps to create databases and collections include:

- Use the PyMongo library to create databases and collections in MongoDB using Python

```
import pymongo

# Connect to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create a database called "mydatabase"
mydb = client["mydatabase"]

# Create a collection called "customers"
mycol = mydb["customers"]
```

Create Databases and Collections

- Note that the database and collection are not created until the user inserts data.
- Use the **insert_one()** method as shown below:

```
# Insert a document into the "customers" collection
mydict = { "name": "John", "address": "Highway 37" }
x = mycol.insert_one(mydict)
```

Understand Projection

Projection in MongoDB refers to the process of selecting or filtering specific fields from a document.

Importance of Projection

Reasons to use projection are:

It allows a user to select only the necessary data rather than selecting the whole data from the document.

It also allows them to retrieve only the required data.

It helps improve query performance and reduce the amount of data sent over the network.

Example

Suppose a document has four fields, and the user needs to show only three of them and then select only two from them:

```
{
  "_id": ObjectId("6096c8f71d57b24f98d0f11e"),
  "name": "John",
  "age": 25,
  "email": "john@example.com"
}
```

Example

To project the name and age fields:

- Use the **find()** method with a projection

Syntax of **db.collection.find ({}, {KEY:1})** is as follows:

```
db.users.find({}, {name: 1, age: 1, _id: 0})
```

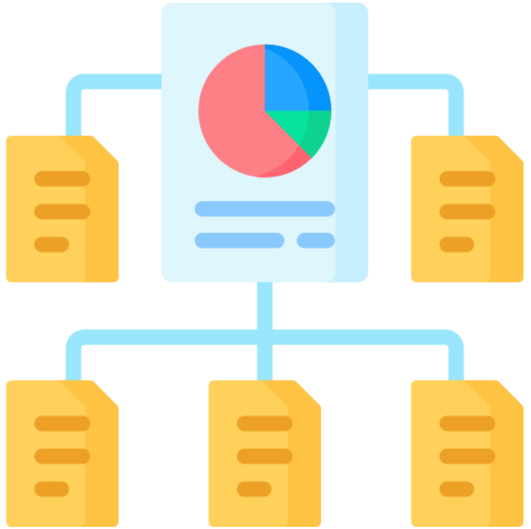
Output:

```
{ "name" : "John", "age" : 25 }
```

Embedded Documents and Arrays

The benefits of using embedded documents and arrays in MongoDB are as follows:

- In MongoDB, documents can contain embedded fields or arrays of documents.
- It allows modeling complex data structures flexibly.
- An embedded document is a document that is nested within another document.
- Embedded documents can contain any number of fields and can be nested to any depth.



Benefits of Embedded Documents and Arrays

The benefits of using embedded documents and arrays in MongoDB are as follows:

01 Improved performance

02 Easier data management

03 Atomic updates

04 Flexibility

05 Reduced storage space

Work with Embedded Documents

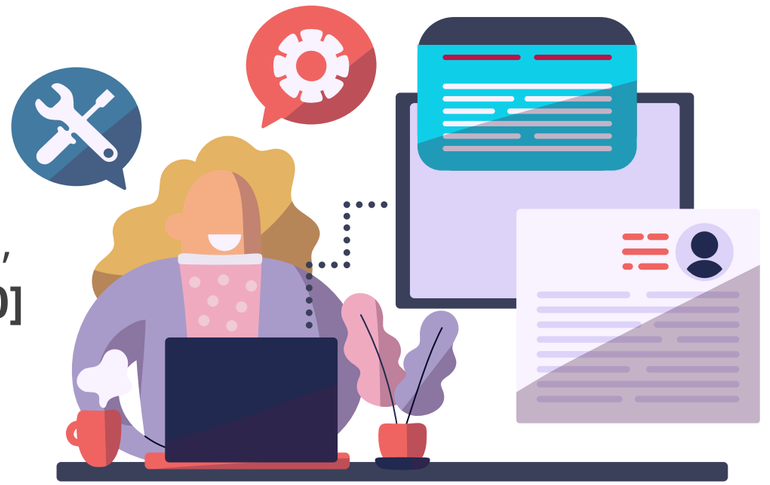
There are several ways to work with embedded documents in MongoDB.

Use dot notation

Example: If a document contains an embedded document for address, use the following syntax: **document.address.city**

Use index notation

Example: If a document contains an array of embedded documents, it can be accessed by the following syntax: **document.arrayName[0]**



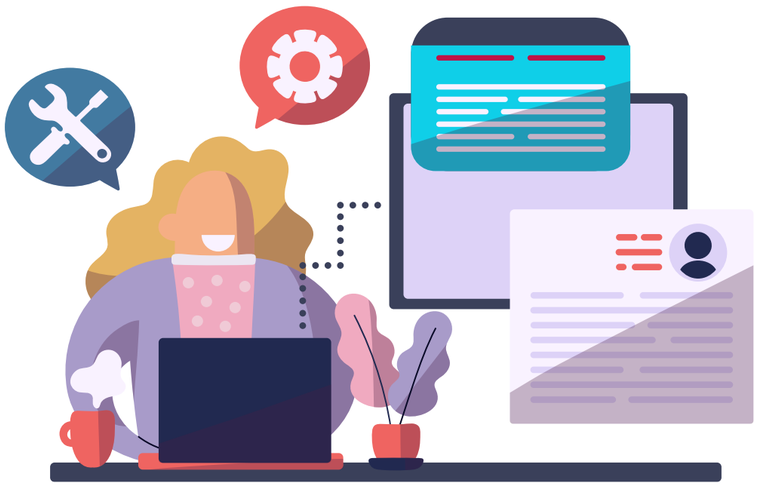
Work with Embedded Documents

Query

A query can be performed using dot notation or index notation.

Example: To query all the documents where the city field in the embedded address document is "New York," use the following syntax:

db.collection.find({ "address.city": "New York" })



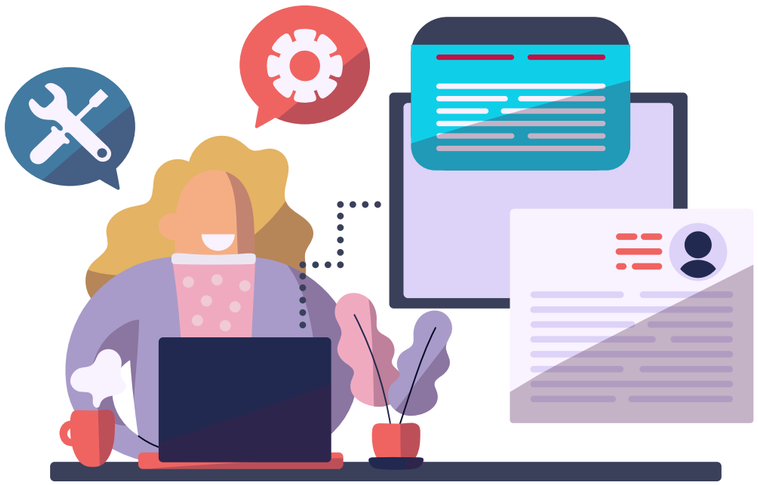
Work with Embedded Documents

Update

The update can be done using the **$set** operator.

Example: To update the city field in the embedded address document, use the following syntax:

db.collection.updateOne({ "_id": ObjectId("123") }, { "$set": { "address.city": "San Francisco" } })



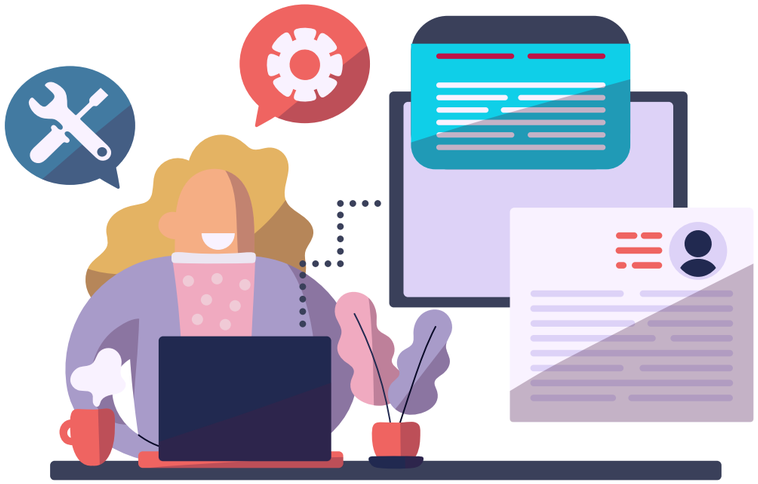
Work with Embedded Documents

Delete

The embedded document can be deleted by setting its value to null using the **$unset** operator.

Example: To delete the address field in a document, use the following syntax:

```
db.collection.updateOne({ "_id": ObjectId("123") }, { "$unset": { "address": "" } })
```



Accessing Structured Data

The structured data accessing in MongoDB involves using queries to retrieve data from collections.

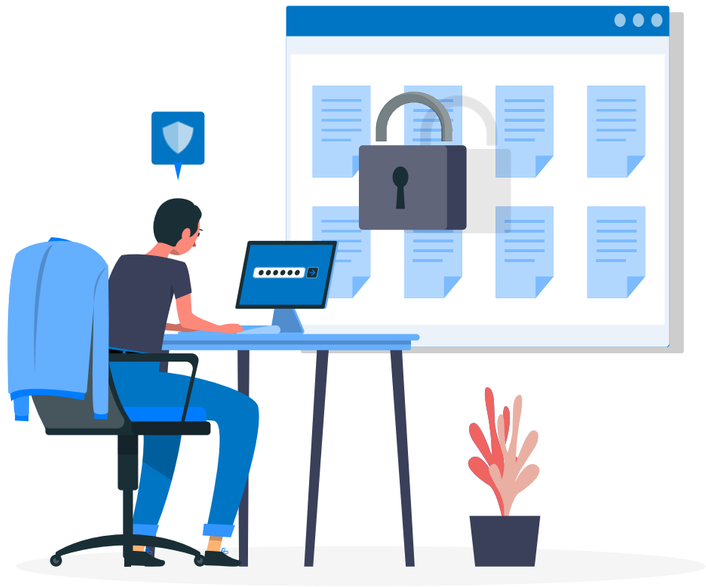
Query for all documents:

Examples of how to access structured data in MongoDB:

- Use the **find()** method without any parameters

Example:

db.collection.find() will return all documents in the **collection.**



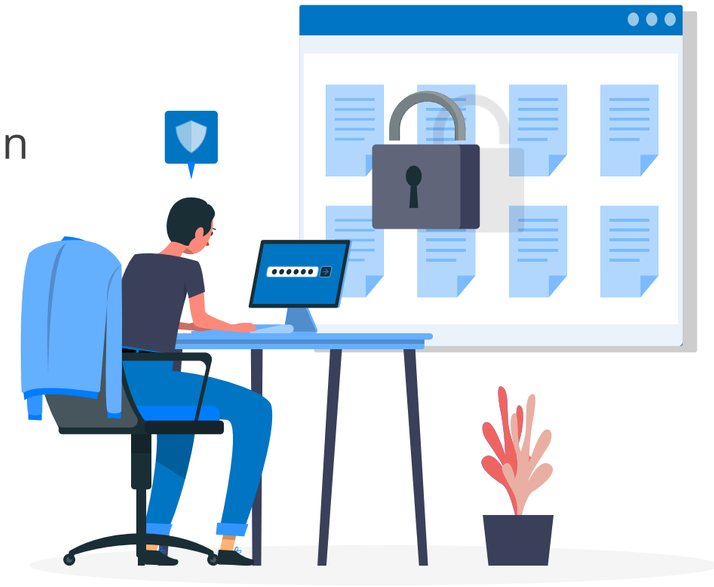
Accessing Structured Data

Query for documents with a specific value:

- Use the **find()** method with a query parameter

Example:

db.collection.find({ "field": "value" }) will return all documents in the **collection** where the **"field"** value is **"value.”**



Accessing Structured Data

Query for documents with multiple criteria:

- Use logical operators such as **$and** and **$or** to specify multiple criteria in a single query

Example:

db.collection.find({ "$or": [{ "field1": "value1" }, { "field2": "value2" }] }) will return all documents in the **collection** where **"field1"** value is **"value1“** or **"field2"** value is **"value2.“**

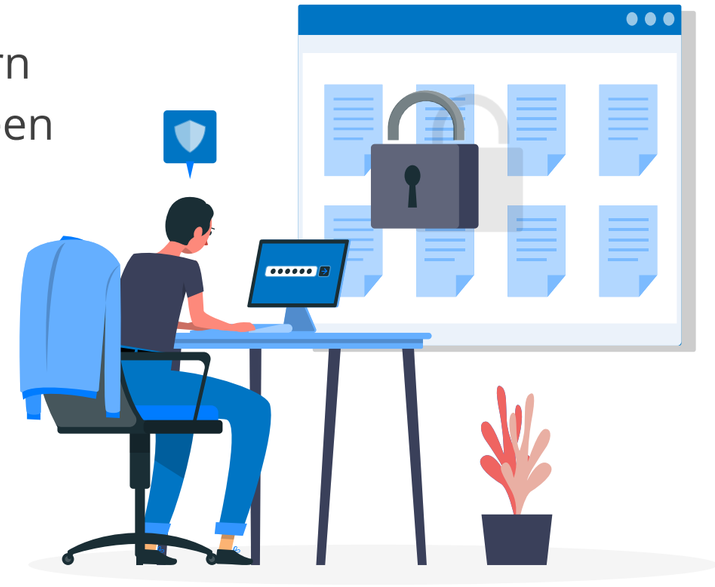
Accessing Structured Data

Query for documents with a range of values:

- Use comparison operators such as **$gt**, **$gte**, **$lt**, and **$lte** to specify a range of values in a query

Example:

db.collection.find({ "field": { "$gte": 10, "$lte": 20 } }) will return all documents in the **collection** where the **"field"** value is between 10 and 20.



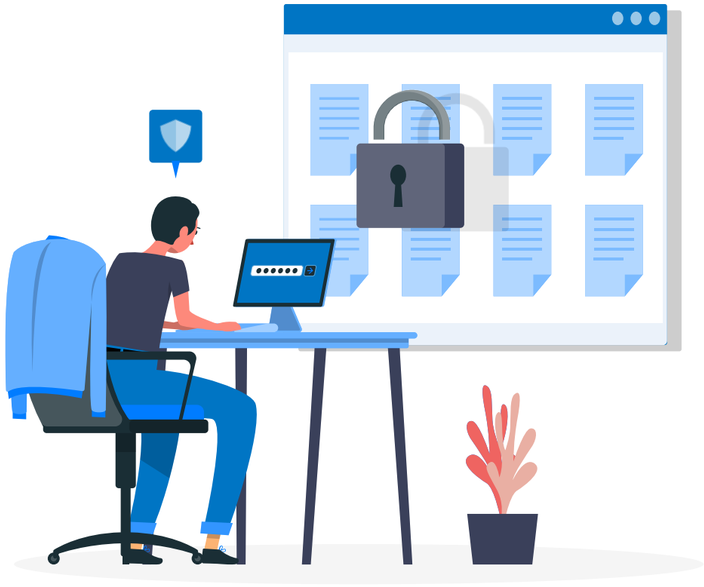
Accessing Structured Data

Query for documents with embedded documents:

- Use dot notation to query for embedded documents within a document

Example:

db.collection.find({ "embeddedDocument.field": "value" }) will return all documents in the **collection** where the **"embeddedDocument.field"** value is **"value.“**



Accessing Structured Data with arrays

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

You have been assigned a task to access Structured Data with arrays on remote desktop.

ASSISTED PRACTICE

Assisted Practice: Guidelines

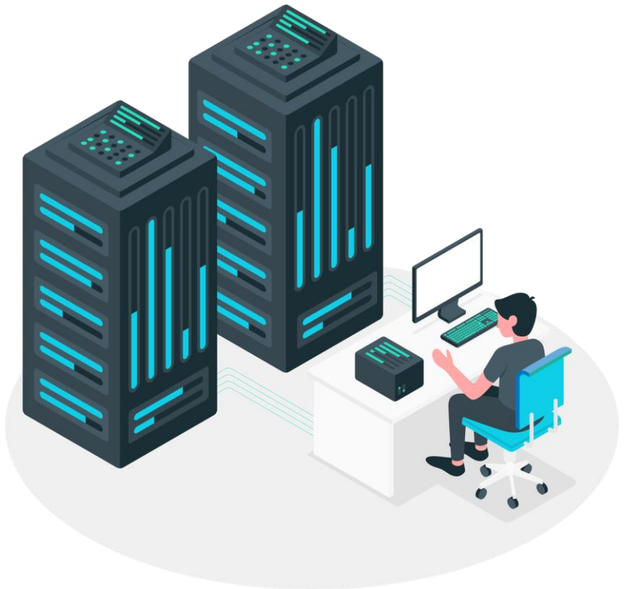
Steps to be followed:

1. Connect to MongoDB Compass using virtual machine extensions
2. Create databases in MongoDB Compass
3. Create a collection in the database
4. Insert data into the collection
5. Access the document from the collection

Introduction to NoSQL Database

What Is NoSQL?

NoSQL refers to a database that stores and manages data using a different relational paradigm than the common Structured Query Language (SQL).



It uses real-time web applications for operation and huge data processing.

Features of NoSQL Databases

01

Big data capability

02

Easy replication

03

Fault tolerance

RDBMS vs. NoSQL

Some of the main differences are:

RDBMS	NoSQL
• Structured data model based on tables and relation	• Various data models (document, key-value, column-family, and graph)
• Vertically scalable	• Horizontally scalable
• ACID-compliant	• Eventually consistent
• SQL	• Different query languages
• Excellent performance for complex queries and transactions	• Excellent performance for high read or write workloads and large datasets

Benefits of NoSQL

In comparison to traditional relational databases, NoSQL databases offer the following advantages:

Scalability

It helps in easily scaling the growing amount of data and traffic.

Flexibility

It is highly flexible and can accommodate various data types, structures, and relationships.

Benefits of NoSQL

Some of the other benefits of NoSQL databases are:

Performance

It provides superior performance for large -data queries.

Availability

It has a high availability rate and fault tolerance as it replicates data over multiple servers.

Benefits of NoSQL

Here are some more advantages of NoSQL over traditional relational databases:

Cost

NoSQL databases are often open-source and have lower licensing fees than traditional RDBMS.

NoSQL Database Types

Some of the most common types are:

Document-based

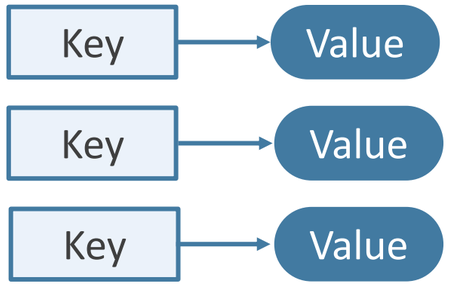
In document-based databases, data is typically stored as documents in JSON or XML format.

Document-based

Key-value

Key-value stores data as key-value pairs that require fast **read** and **write** operations and high scalability.

Key-value



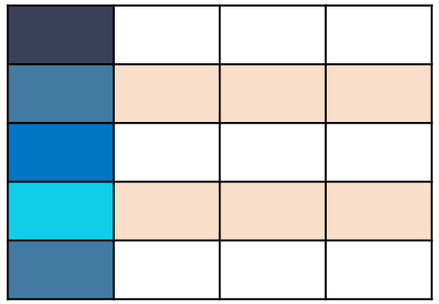
NoSQL Database Types

Following are some more common types of NoSQL databases:

Column-family

They are well suited for applications that require complex querying and analysis of large datasets.

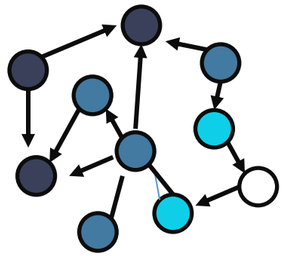
Column-family



Graph-based

They use edges and nodes to represent data and connections between data points.

Graph-based



NoSQL Commands

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

You have been assigned a task to perform NoSQL commands on a remote desktop.

ASSISTED PRACTICE

Assisted Practice: Guidelines

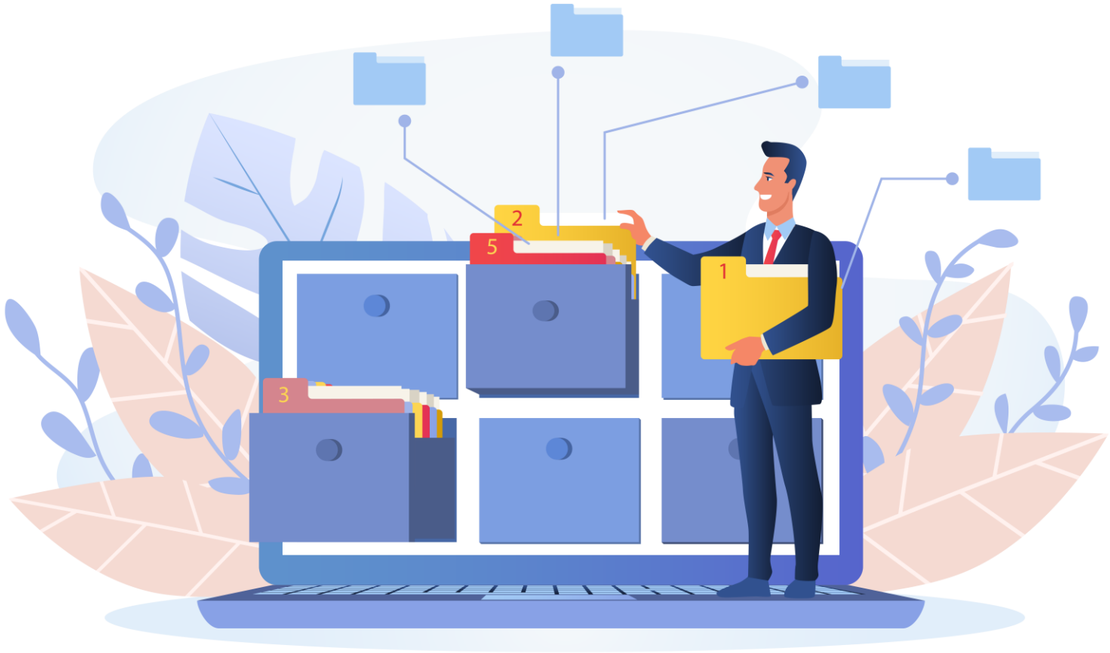
Steps to be followed:

1. Connect to MongoDB Compass by using virtual machine extensions
2. Create databases in the MongoDB Compass
3. Create a collection in the database
4. Insert document in the collection
5. Access document using NoSQL commands

MongoDB as Document Database

MongoDB as Document Database

MongoDB is a non-relational document database that supports JSON-like storage.



The MongoDB database has full indexing support, replication, and extensive, user-friendly APIs. Its flexible data model makes it possible to store unstructured data.

MongoDB as a Document Database Document Store: Example

Problem Statement:

Duration: 20 min.

You have been assigned a task to access MongoDB as a document database.

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 a collection in the database
4. Open the collection document in MongoDB Compass

Key Takeaways

- MongoDB is an open-source, document-oriented NoSQL database management system.
- Instead of using tables and rows as in traditional relational databases, MongoDB uses collections and documents.
- MongoDB offers high scalability, allowing users to distribute data across multiple servers.
- A document is a collection of key-value pairs that are stored.
- MongoDB is a document-oriented database available on various platforms and offers high performance, high availability, and easy scalability.

Key Takeaways

- A database uses a different relational model from the common structured query language (SQL) paradigm to store and manage data.
- NoSQL database offers outstanding performance for reading or writing loads of large datasets.
- They manage a massive quantity of unstructured or semi-structured data and can handle dynamic changes in the data model.

Thank You