

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
BELAGAVI – 5900 18**



A SEMINAR REPORT ON

“MongoDB: Cross Platform Document Oriented Database”

**Submitted in partial fulfillment of the requirements of the award of degree
of**

**BACHELOR OF ENGINEERING
IN
INFORMATION SCIENCE & ENGINEERING**

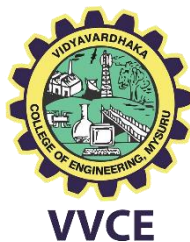
Submitted By:

Manoj M [4VV19IS045]

UNDER THE GUIDANCE OF

Prof. VIDYASHREE K P

**Assistant Professor
Dept of IS&E
VVCE, Mysuru**



2022 -2023

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
VIDYAVARDHAKA COLLEGE OF ENGINEERING
MYSURU-570002**

Vidyavardhaka College of Engineering
Gokulam III Stage, Mysuru-570002

Department of Information Science and Engineering



CERTIFICATE

This is to certify that the seminar report entitled “**MongoDB: Cross Platform Document Oriented Database**” is carried out by **Manoj M (4VV19IS045)**, student of VIII Semester Information Science and Engineering, **Vidyavardhaka College of Engineering, Mysuru** in partial fulfillment for the award of the degree of **Bachelor of Engineering in Information Science & Engineering** of the **Visvesvaraya Technological University, Belagavi**, during the academic year **2022-2023**. It is certified that all the suggestions and corrections indicated and have been incorporated in the report. The report has been approved as it satisfies the requirements in respect of seminar work prescribed for the said degree.

Signature of the Guide

(Prof. Vidyashree K P)

Signature of the HOD

(Dr. A B Rajendra)

ACKNOWLEDGEMENT

The Seminar would not have been possible without the guidance, assistance and suggestions of many individuals. I would like to express our deep sense of gratitude and indebtedness to each and every one who has helped me to make this project a success.

I heartily thank our beloved Principal, **Dr. B Sadashive Gowda** for his whole hearted support and for his kind permission to undergo the seminar.

I wish to express my deepest gratitude to **Dr. A B Rajendra**, Head of Department, Information Science and Engineering, VVCE, for his constant encouragement and inspiration in taking up this seminar.

I gracefully thank our seminar guide, **Prof. Vidyashree K P**, Designation, Dept. of Information Science and Engineering for her encouragement and advice throughout the course of the seminar work.

In the end, I anxious to offer our sincere thanks to my family members and friends for their valuable suggestions and encouragement.

Manoj M
(4VV19IS045)

ABSTRACT

MongoDB was developed as a NoSQL database that could provide a more flexible and scalable alternative to traditional relational databases. Instead of storing data in tables, MongoDB stores data in documents that can be nested within each other. This document-oriented approach allows for a more flexible schema and can handle a wide variety of data types, including unstructured data.

MongoDB's flexibility is further enhanced by its powerful query language, which enables developers to perform complex queries on the data. This flexibility and power make MongoDB an ideal choice for big data applications where the structure of the data may not be fully known in advance.

One of the key advantages of MongoDB is its ability to scale horizontally. This means that additional servers can be added to handle increased traffic or data volume, providing a highly scalable platform for cloud-based applications. Additionally, MongoDB's distributed architecture ensures high availability, with data replicated across multiple servers to ensure that it is always accessible.

Overall, MongoDB has become a critical tool for businesses that rely on big data. Its document-oriented approach and powerful query language provide developers with the flexibility and control they need to work with a variety of data types. Additionally, its scalability and high availability make it an ideal choice for cloud-based applications that need to handle large amounts of data and traffic.

CONTENTS

CHAPTER 1: INTRODUCTION	1
1.1 What is MongoDB.....	1
1.2 Evolution of MongoDB	1
1.3 Why Use MongoDB and When to Use It?	2
1.4 Data Types in MongoDB	3
CHAPTER 2: MONGODB COMPONENTS.....	6
2.1 Databases	6
2.2 Collections	7
2.3 Documents	8
CHAPTER 3: MONGODB ARCHITECTURE.....	9
3.1 Core Processes	9
3.2 Database architecture.....	10
3.3 Types of database architecture	10
3.4 Three levels of database architecture in MongoDB Atlas	12
CHAPTER 4: MONGODB REPLICATION	14
4.1 Replication.....	14
4.2 Redundancy and Data Availability.....	14
4.3 Replication in MongoDB	14
4.4 Automatic Failover.....	15
4.5 Asynchronous Replication	16
CHAPTER 5: MONGODB DEVELOPER TOOLS.....	17
5.1 MongoDB Shell.....	17
5.2 MongoDB Compass	19
5.3 MongoDB Atlas	20
CHAPTER 6: MONGODB CRUD OPERATIONS	22
6.1 Insert and Find Documents.....	22
6.2 Update and Delete Documents	23

6.3 Modifying Query Results	24
CHAPTER 7: MONGODB DATA MODELING	25
7.1 Data Modeling	25
7.2 Flexible Schema	26
7.3 Document Structure.....	26
7.4 Use of Data Modeling	27
7.5 Advantages of Data Modeling.....	28
7.6 Types of Data Modeling	28
7.7 Flexible Data Modeling with MongoDB Atlas	28
CHAPTER 8: MONGODB INDEXING	29
8.1 Create an Index.....	29
8.2 Index Types	29
CHAPTER 9: MONGODB AGGREGATION.....	32
9.1 Aggregation Pipelines	32
9.2 Single Purpose Aggregation Methods	33
CHAPTER 10: COMPARISONS WITH MONGODB	34
10.1 MongoDB v/s MySQL	34
10.2 MongoDB v/s Cassandra.....	34
10.3 MongoDB v/s Redis	36
CHAPTER 11: ADVANTAGES AND DISADVANTAGES OF MONGODB	37
11.1 Advantages of MongoDB.....	37
11.2 Disadvantages of MongoDB	39
CHAPTER 12: APPLICATIONS OF MONGODB.....	42
12.1 MongoDB Content Management System.....	42
12.2 Internet of Things Applications.....	43
12.3 MongoDB in Big Data Analytics	45
12.4 E-Commerce.....	46
12.5 Operational Intelligence	46
CONCLUSION	48
REFERENCES	49

LIST OF FIGURES

Fig 1: MongoDB Logo.....	1
Fig 2:Evolution of MongoDB	2
Fig 3: Components of MongoDB.....	6
Fig 4:Collection.....	7
Fig 5: Document.....	8
Fig 6: 1-Tire Architecture	10
Fig 7: 2-Tire Architecture	11
Fig 8: 3-Tire Architecture	12
Fig 9: MongoDB Architecture	12
Fig 10: Replica Set.....	14
Fig 11: MongoDB Replication.....	15
Fig 12: Replication with Arbiter Node.....	15
Fig 13: Automatic Failover	16
Fig 14: MongoDB Developer Tools	17
Fig 15: Mongo Shell UI	18
Fig 16: MongoDB Compass UI	19
Fig 17: MongoDB Atlas UI	20
Fig 18: Find Document	23
Fig 19:Update Document	23
Fig 20: Delete Document	24
Fig 21: Embedded Data.....	26
Fig 22: Reference Document	27
Fig 23: MongoDB CMS.....	42
Fig 24: IoT architecture in MongoDB Atlas	44

LIST OF TABLES

Table 1: Single Purpore Aggrigation Methods	33
Table 2: MongoDB v/s MySQL.....	34
Table 3: MongoDB v/s Cassandra	34
Table 4: MongoDB v/s Redis.....	36
Table 5: Advantage of MongoDB in CMS	43

CHAPTER 1

INTRODUCTION

1.1 What is MongoDB

MongoDB is an open-source NoSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.



Fig 1: MongoDB Logo

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases. SQL normalizes data as schemas and tables, and every table has a fixed structure.

Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents. Documents are made up of key-value pairs -- MongoDB's basic unit of data. Collections, the equivalent of SQL tables, contain document sets. MongoDB offers support for many programming languages, such as C, C++, C#, Go, Java, Python, Ruby and Swift.

1.2 Evolution of MongoDB

MongoDB was first introduced in 2009 as a NoSQL database designed to provide a more flexible and scalable alternative to traditional relational databases. Since then, it has undergone significant evolution to become one of the most popular databases for big data applications. In its early years, MongoDB was primarily used by developers who wanted a database that could handle unstructured data and provide better performance and scalability than relational databases. As it gained popularity, MongoDB evolved to include more advanced features, such as automatic sharding, which allows for data to be distributed across multiple servers for improved performance and scalability. It also added support for ACID transactions, which provide data consistency and reliability, and further enhanced its security features to meet the growing demands of enterprise customers.

In recent years, MongoDB has continued to evolve to keep up with the demands of the modern data landscape. It has added support for graph data and JSON schema validation, which enable developers to

work with even more types of data. Additionally, it has introduced new cloud-based services and expanded its partnerships with major cloud providers to make it easier to use MongoDB in cloud environments. With these continued enhancements, MongoDB is well-positioned to remain a leading database platform for many years to come.

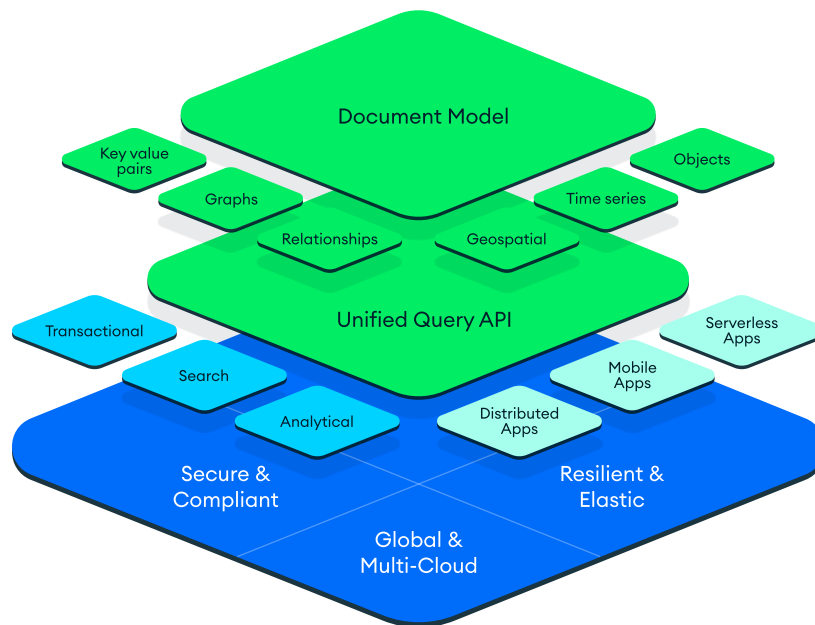


Fig 2:Evolution of MongoDB

1.3 Why Use MongoDB and When to Use It?

1.3.1 Why Use MongoDB

MongoDB is built on a scale-out architecture that has become popular with developers of all kinds for developing scalable applications with evolving data schemas. As a document database, MongoDB makes it easy for developers to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in most modern programming languages, making it a natural choice for developers, as they do not need to think about normalizing data. MongoDB can also handle high volume and can scale both vertically or horizontally to accommodate large data loads.

MongoDB was built for people building internet and business applications who need to evolve quickly and scale elegantly. Companies and development teams of all sizes use MongoDB for a wide variety of reasons. MongoDB is an excellent choice if you need to: Support rapid iterative development. Enable collaboration of a large number of teams. Scale to high levels of read and write traffic. Scale your data repository to a massive size. Evolve the type of deployment as the business changes. Store, manage, and search data with text, geospatial, or time-series dimensions.

1.3.2 When to Use MongoDB

MongoDB is a general-purpose database used in various ways to support applications in many different industries (e.g., telecommunications, gaming, finances, healthcare, and retail). MongoDB has found a home in many different businesses and functions because it solves long-standing problems in data

management and software development.

Typical use cases for MongoDB include:

- Integrating large amounts of diverse data
- Describing complex data structures that evolve
- Delivering data in high-performance applications
- Supporting hybrid and multi-cloud applications

1.4 Data Types in MongoDB

In MongoDB, the documents are stores in BSON, which is the binary encoded format of JSON and using BSON we can make remote procedure calls in MongoDB. BSON data format supports various data-types. Below are the enlisted MongoDB data types:

- **String:** This is the most commonly used data type in MongoDB to store data, BSON strings are of UTF-8. So, the drivers for each programming language convert from the string format of the language to UTF-8 while serializing and de-serializing BSON. The string must be a valid UTF-8.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom", //String
  age: 21,
  pass: true
}
```

- **Integer:** In MongoDB, the integer data type is used to store an integer value. We can store integer data type in two forms 32 -bit signed integer and 64 – bit signed integer.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  age: 21, //Integer
  pass: true
}
```

- **Double:** The double data type is used to store the floating-point values.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  age: 21, //Integer
  pass: true,
  marks: 211.5
}
```

- **Boolean:** The boolean data type is used to store either true or false.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  marks: 211.5,
  pass: true //boolean
}
```

- **Null:** The null data type is used to store the null value.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  marks: 211.5,
  mobile: null //Null Value
}
```

- **Array:** The Array is the set of values. It can store the same or different data types values in it. In MongoDB, the array is created using square brackets([]).

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  marks: 211.5,
  skills: [ //Array
    'c',
    'cpp',
    'java'
  ]
}
```

- **Object:** Object data type stores embedded documents. Embedded documents are also known as nested documents. Embedded document or nested documents are those types of documents which contain a document inside another document.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),
  name: "Tom",
  marks: 211.5,
  address: { //Objects
    'city': "Mysore",
    'state': "Karnataka",
    'zipcode': 270001
  }
}
```

- **Object Id:** Whenever we create a new document in the collection MongoDB automatically creates a unique object id for that document(if the document does not have it). There is an `_id` field in MongoDB for each document. The data which is stored in Id is of hexadecimal format and the length of the id is 12 bytes which consist:
 - 4-bytes for Timestamp value.
 - 5-bytes for Random values. i.e., 3-bytes for machine Id and 2-bytes for process Id.
 - 3- bytes for Counter

You can also create your own id field, but make sure that the value of that id field must be unique.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'), //Object ID
  name: "Tom",
  marks: 211.5,
}
```

- **Undefined:** This data type stores the undefined values.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),    //Object ID
  name: "Tom",
  marks: 211.5,
  duration: undefined
}
```

- **Date:** Date data type stores date. It is a 64-bit integer which represents the number of milliseconds. BSON data type generally supports UTC datetime and it is signed. If the value of the date data type is negative then it represents the dates before 1970. There are various methods to return date, it can be returned either as a string or as a date object. Some method for the date:

- Date(): It returns the current date in string format.
- new Date(): Returns a date object. Uses the ISODate() wrapper.
- new ISODate(): It also returns a date object. Uses the ISODate() wrapper.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),    //Object ID
  name: "Tom",
  marks: 211.5,
  duration: undefined,
  dob: "Thu Feb 04 2001 01:00:34 GMT+0530 (India Standard Time)",
  admission-date: ISODate("2021-04-01T19:30:56:454Z"), //ISODate()
}
```

- **Timestamp:** In MongoDB, this data type is used to store a timestamp. It is useful when we modify our data to keep a record and the value of this data type is 64-bit. The value of the timestamp data type is always unique.

```
{
  _id: ObjectId('62cc1eee2f8e0917ee8ab9af'),    //Object ID
  name: "Tom",
  marks: 211.5,
  duration: undefined,
  timestamp: Timestamp(1612380924,1)
}
```

There are many other types of datatypes which make MongoDB to store the data.

CHAPTER 2

MONGODB COMPONENTS

MongoDB is a document-oriented NoSQL database that provides a flexible and scalable solution for storing and managing large volumes of data. MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. It uses a JSON-like data model, which allows for the storage of data in semi-structured documents that can be easily queried and updated. MongoDB's document-oriented approach provides a more natural and efficient way of handling complex data structures, making it a popular choice for many modern web applications

There are mainly 3 components present in the MongoDB

- Databases
- Collections
- Documents

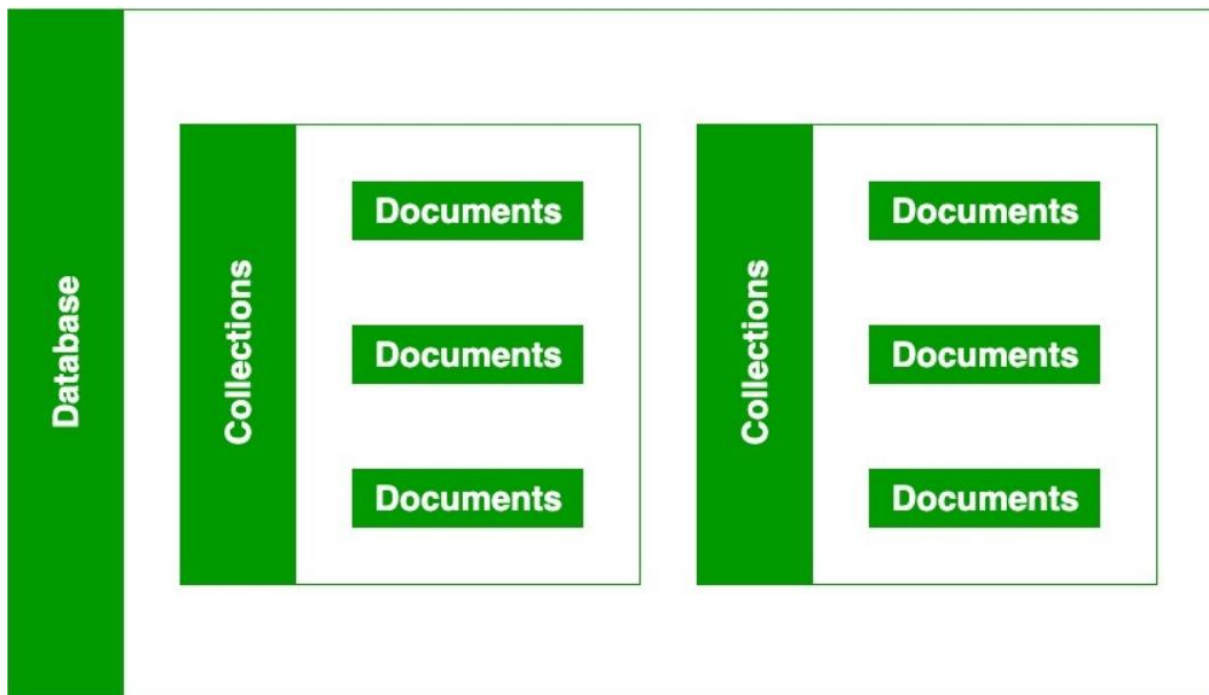


Fig 3: Components of MongoDB

2.1 Databases

In MongoDB, databases hold one or more collections of documents. To list a database, we can use command,

```
show databases
```

2.1.1 Naming Restrictions for Database

Before creating a database you should first learn about the naming restrictions for databases:

- In MongoDB, the names of the database are case insensitive, but you must always remember that the database names cannot differ only by the case of the characters.
- For windows user, MongoDB database names cannot contain any of these following characters:

/\ . "\$* : | ?

- For Unix and Linux users, MongoDB database names cannot contain any of these following characters:

/\ . "\$

- MongoDB database names cannot contain null characters (in windows, Unix, and Linux systems).
- MongoDB database names cannot be empty and must contain less than 64 characters.

2.1.2 Creating Database

If a database does not exist, MongoDB creates the database when you first store data for that database. As such, you can switch to a non-existent database and perform the following,

```
use database_name
```

This command actually switches you to the new database if the given name does not exist and if the given name exists, then it will switch you to the existing database. Now at this stage, if you use the show command to see the database list where you will find that your new database is not present in that database list because, in MongoDB, the database is actually created when you start entering data in that database

2.2 Collections

Collections are just like tables in relational databases, they also store data, but in the form of documents. A single database is allowed to store multiple collections.

MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

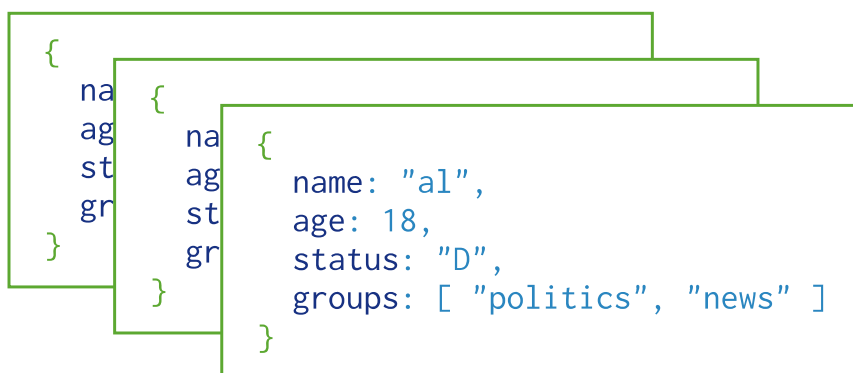


Fig 4:Collection

2.2.1 Naming Restrictions for Collections

Before creating a collection, you should first learn about the naming restrictions for collections:

- Collection name must start with an underscore or a character.
- Collection name does not contain \$, empty string, null character and does not begin with system. prefix.
- The maximum length of the collection name is 120 bytes(including the database name, dot separator, and the collection name).

2.2.2 Creating Collections

If a collection does not exist, MongoDB creates the collection when you first store data for that collection.

```
db.myNewCollection2.insertOne( { x: 1 } )
db.myNewCollection3.createIndex( { y: 1 } )
```

Both the insertOne() and the createIndex() operations create their respective collection if they do not already exist. Be sure that the collection name follows MongoDB Naming Restrictions.

2.3 Documents

MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents, though it contains more data types than JSON.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```



field: value
field: value
field: value
field: value

Fig 5: Document

In MongoDB, the data records are stored as BSON documents. Here, BSON stands for binary representation of JSON documents, although BSON contains more data types as compared to JSON. The document is created using field-value pairs or key-value pairs and the value of the field can be of any BSON type.

2.3.1 Naming Restrictions for Collections

Before moving further first you should learn about the naming restrictions for fields:

- The field names are of strings.
- The `_id` field name is reserved to use as a primary key. And the value of this field must be unique, immutable, and can be of any type other than an array.
- The field name cannot contain null characters.
- The top-level field names should not start with a dollar sign (\$).

The maximum size of the BSON document is 16MB. It ensures that the single document does not use too much amount of RAM or bandwidth (during transmission). If a document contains more data than the specified size, then MongoDB provides a GridFS API to store such type of documents.

CHAPTER 3

MONGODB ARCHITECTURE

3.1 Core Processes

The core components in the MongoDB package are

- **mongod** , which is the core database process
- **mongos** , which is the controller and query router for sharded clusters
- **mongo** , which is the interactive MongoDB shell

These components are available as applications under the bin folder. Let's discuss these components in detail.

3.1.1 mongod

The primary daemon in a MongoDB system is known as mongod . This daemon handles all the data requests, manages the data format, and performs operations for background management. When a mongod is run without any arguments, it connects to the default data directory, which is C:\data\db or /data/db , and default port 27017, where it listens for socket connections. It's important to ensure that the data directory exists and you have write permissions to the directory before the mongod process is started. If the directory doesn't exist or you don't have write permissions on the directory, the start of this process will fail. If the default port 27017 is not available, the server will fail to start.

mongod also has a HTTP server which listens on a port 1000 higher than the default port, so if you started the mongod with the default port 27017, in this case the HTTP server will be on port 28017 and will be accessible using the URL <http://localhost:28017> . This basic HTTP server provides administrative information about the database.

3.1.2 mongo

mongo provides an interactive JavaScript interface for the developer to test queries and operations directly on the database and for the system administrators to manage the database. This is all done via the command line. When the mongo shell is started, it will connect to the default database called test. This database connection value is assigned to global variable db. As a developer or administrator, you need to change the database from test to your database post the first connection is made. You can do this by using <database>.

3.1.3 mongos

mongos is used in MongoDB sharding. It acts as a routing service that processes queries from the application layer and determines where in the sharded cluster the requested data is located. We will discuss mongos in more detail in the sharding section. Right now you can think of mongos as the process that routes the queries to the correct server holding the data.

For a sharded cluster, the mongos instances provide the interface between the client applications

and the sharded cluster. The mongos instances route queries and write operations to the shards. From the perspective of the application, a mongos instance behaves identically to any other MongoDB instance.

3.2 Database architecture

When designing a modern application, chances are that you will need a database to store data. There are many ways to architect software solutions that use a database, depending on how your application will use this data. In this article, we will cover the different types of database architecture and describe in greater detail a three-tier application architecture, which is extensively used in modern web applications.

Database architecture describes how a database management system (DBMS) will be integrated with your application. When designing a database architecture, you must make decisions that will change how your applications are created.

First, decide on the type of database you would like to use. The database could be centralized or decentralized. Centralized databases are typically used for regular web applications and will be the focus of this article. Decentralized databases, such as blockchain databases, might require a different architecture. Once you've decided the type of database you want to use, you can determine the type of architecture you want to use. Typically, these are categorized into single-tier or multi-tier applications.

3.3 Types of database architecture

When we talk about types there are number of tiers an application can have so for simplicity, we talk on 3 types of database architecture

- 1-tier architecture
- 2-tier architecture
- 3-tier architecture

3.3.1 1-Tier Architecture

In 1-tier architecture, the database and any application interfacing with the database are kept on a single server or device. Because there are no network delays involved, this is generally a fast way to access data without requirement of network.

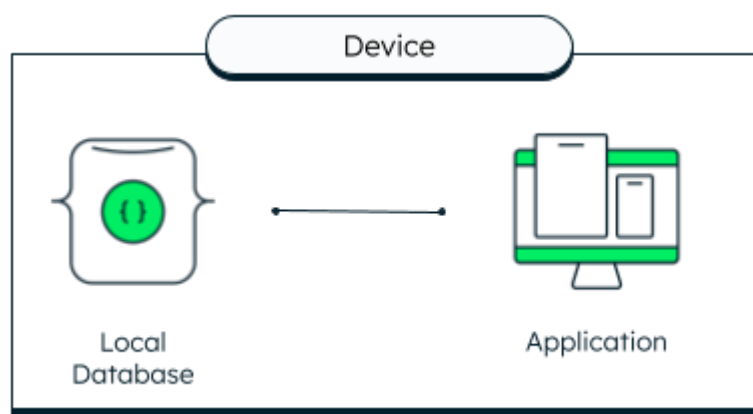


Fig 6: 1-Tire Architecture

An example of a 1-tier architecture would be a mobile application that uses Realm, the open-source

mobile database by MongoDB, as a local database. In that case, both the application and the database are running on the user's mobile device.

3.3.2 2-Tier Architecture

2-tier architectures consist of multiple clients connecting directly to the database. This architecture is also known as client-server architecture.

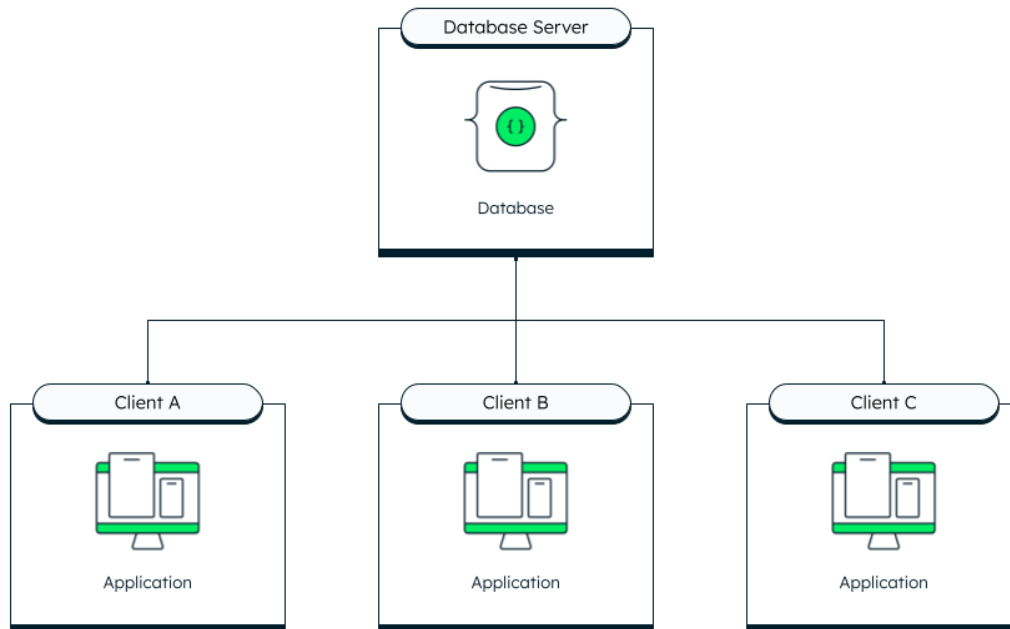


Fig 7: 2-Tire Architecture

This architecture used to be more common when a desktop application would connect to a single database hosted on an on-premise database server—for example, an in-house customer relationship management (CRM) that connects to an Access database.

3.3.3 3-Tier Architecture

Most modern web applications use a 3-tier architecture. In this architecture, the clients connect to a back end, which in turn connects to the database. Using this approach has many benefits:

- **Security:** Keeping the database connection open to a single back end reduces the risks of being hacked.
- **Scalability:** Because each layer operates independently, it is easier to scale parts of the application.
- **Faster deployment:** Having multiple tiers makes it easier to have a separation of concerns and to follow cloud-native best practices, including better continuous delivery processes.

An example of this type of architecture would be a React application that connects to a Node.js back end. The Node.js back end processes the requests and fetches the necessary information from a database such as MongoDB Atlas, using the native driver.

As the above the MongoDB also contain same number of tires like data or database layer, application layer and presentation layer it would be discussed below.

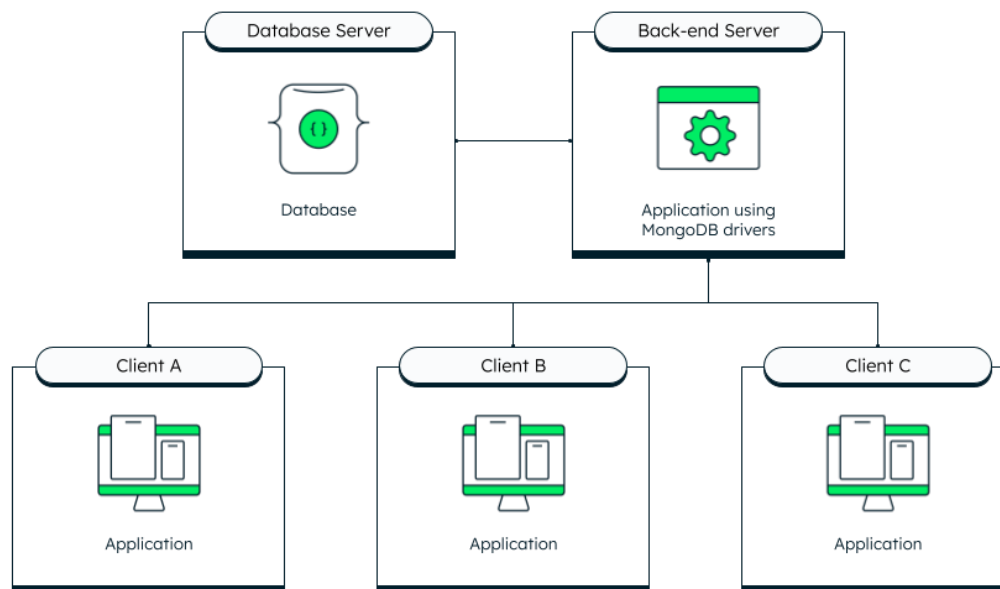


Fig 8: 3-Tire Architecture

3.4 Three levels of database architecture in MongoDB Atlas

The most common DBMS architecture used in modern application development is the 3-tier model. Since it's so popular, let's look at what this architecture looks like with MongoDB Atlas.

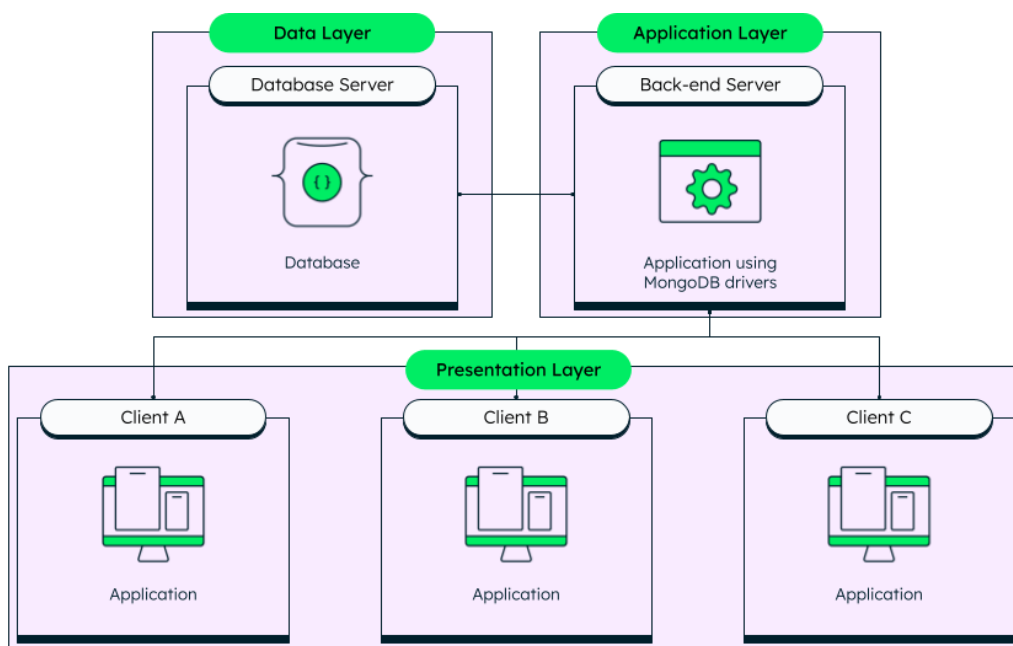


Fig 9: MongoDB Architecture

As you can see in this diagram, the 3-tier architecture comprises of,

- Data Layer
- Application Layer
- Presentation Layer

3.4.1 Data (Database) Layer

As the name suggests, the data layer is where the data resides. In the scenario above, the data is stored in a MongoDB Atlas database hosted on any public cloud—or across multiple clouds, if needed. The

only responsibility of this layer is to keep the data accessible for the application layer and run the queries efficiently.

3.4.2 Application (Middle) Layer

The application tier is in charge of communicating with the database. To ensure secure access to the data, requests are initiated from this tier. In a modern web application, this would be your API. A back-end application built with Node.js (or any other programming language with a native driver) makes requests to the database and relays the information back to the clients.

3.4.3 Presentation (User) Layer

The final layer is the presentation layer. This is usually the UI of the application with which the users will interact. In the case of a MERN or MEAN stack application, this would be the JavaScript front end built with React or Angular.

CHAPTER 4

REPLICATION IN MONGODB

4.1 Replication

A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. This section introduces replication in MongoDB as well as the components and architecture of replica sets. The section also provides tutorials for common tasks related to replica sets.

4.2 Redundancy and Data Availability

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

In some cases, replication can provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data in different data centers can increase data locality and availability for distributed applications. You can also maintain additional copies for dedicated purposes, such as disaster recovery, reporting, or backup.

4.3 Replication in MongoDB

A replica set is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes.

The primary node receives all write operations. A replica set can have only one primary capable of confirming writes with { w: "majority" } write concern; although in some circumstances, another mongod instance may transiently believe itself to also be primary. The primary records all changes to its data sets in its operation log, i.e. oplog.

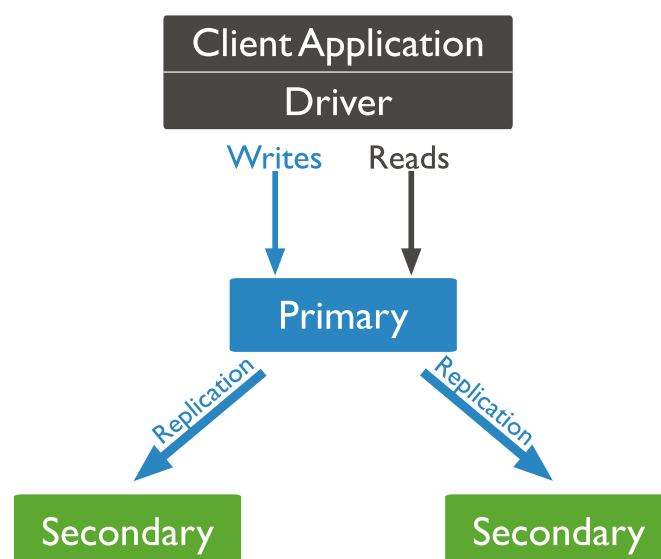


Fig 10: Replica Set

The secondaries replicate the primary's oplog and apply the operations to their data sets such that the secondaries' data sets reflect the primary's data set. If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.

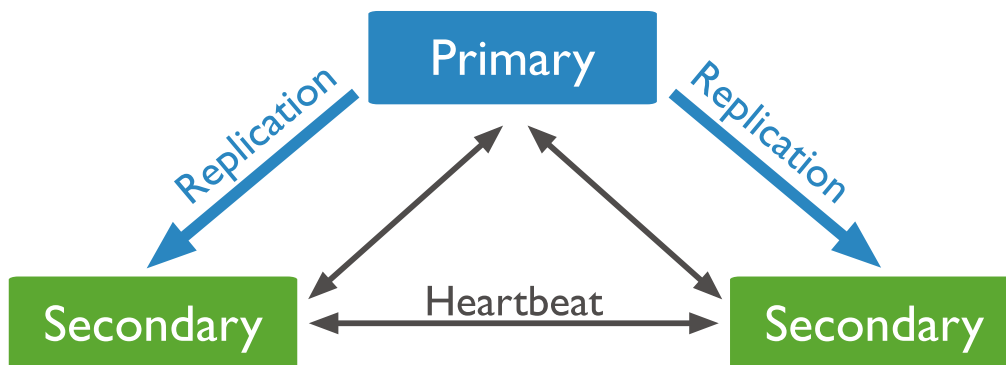


Fig 11: MongoDB Replication

In some circumstances (such as you have a primary and a secondary but cost constraints prohibit adding another secondary), you may choose to add a mongod instance to a replica set as an arbiter. An arbiter participates in elections but does not hold data (i.e. does not provide data redundancy). An arbiter will always be an arbiter whereas a primary may step down and become a secondary and a secondary may become the primary during an election.

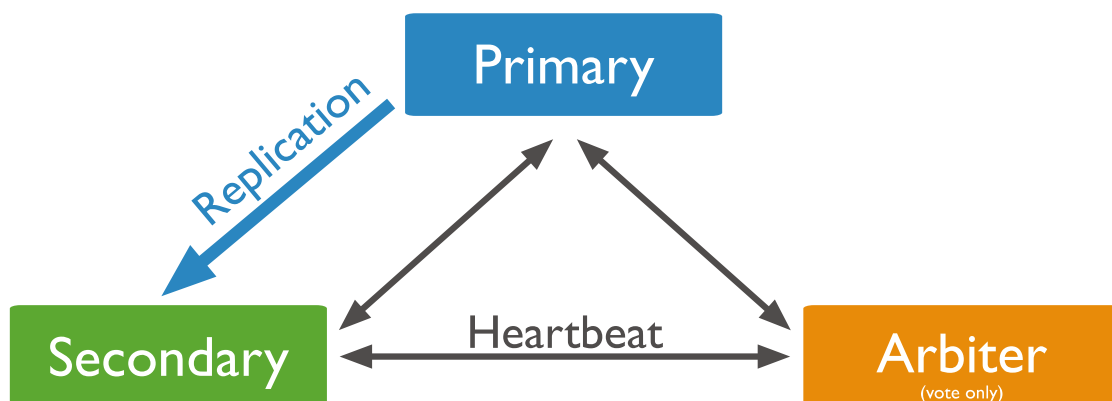


Fig 12: Replication with Arbiter Node

An arbiter will always be an arbiter whereas a primary may step down and become a secondary and a secondary may become the primary during an election.

4.4 Automatic Failover

When a primary does not communicate with the other members of the set for more than the configured `electionTimeoutMillis` period (10 seconds by default), an eligible secondary calls for an election to nominate itself as the new primary. The cluster attempts to complete the election of a new primary and resume normal operations.

The replica set cannot process write operations until the election completes successfully. The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary

is offline. The median time before a cluster elects a new primary should not typically exceed 12 seconds, assuming default replica configuration settings. This includes time required to mark the primary as unavailable and call and complete an election. You can tune this time period by modifying the `settings.electionTimeoutMillis` replication configuration option. Factors such as network latency may extend the time required for replica set elections to complete, which in turn affects the amount of time your cluster may operate without a primary.

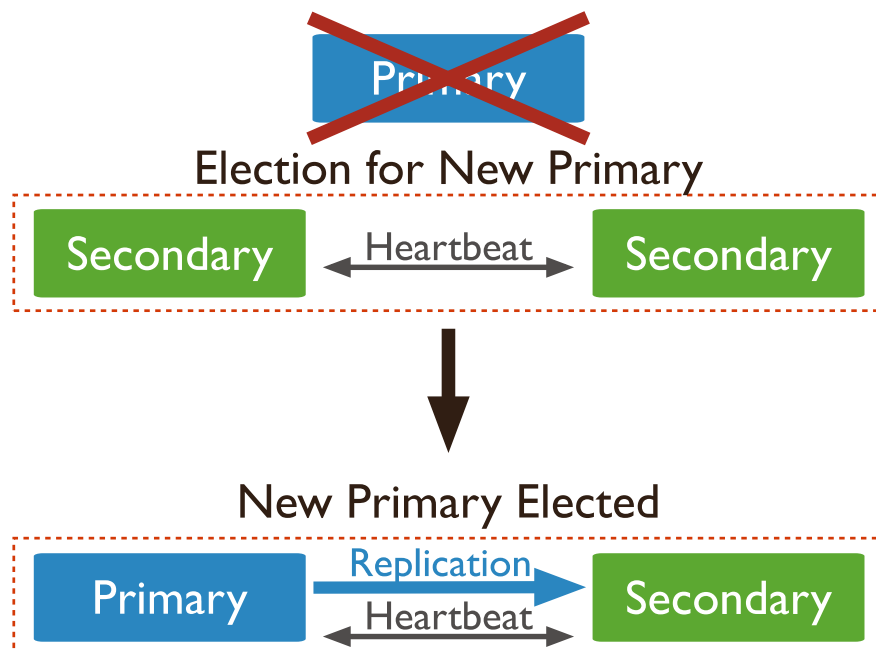


Fig 13: Automatic Failover

4.5 Asynchronous Replication

Secondaries replicate the primary's oplog and apply the operations to their data sets asynchronously. By having the secondaries' data sets reflect the primary's data set, the replica set can continue to function despite the failure of one or more members.

CHAPTER 5

MONGODB DEVELOPER TOOLS

MongoDB Developer Tools provide the easiest way for you to connect and work with your MongoDB data from an interface that you are most comfortable and familiar with. Tools provided by MongoDB are:

- MongoDB Shell
- MongoDB Compass
- MongoDB Atlas

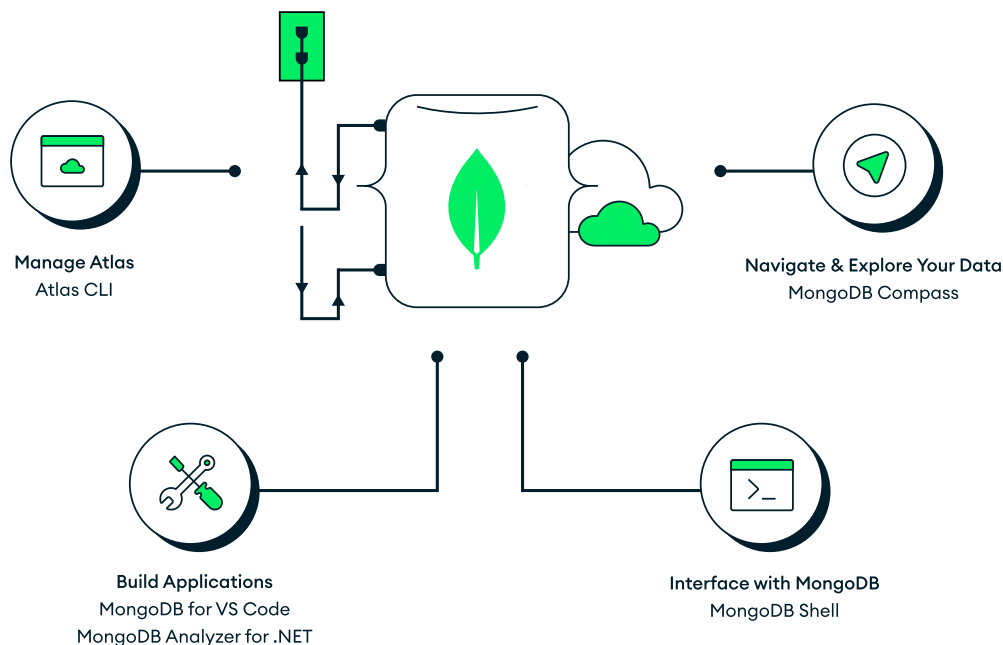


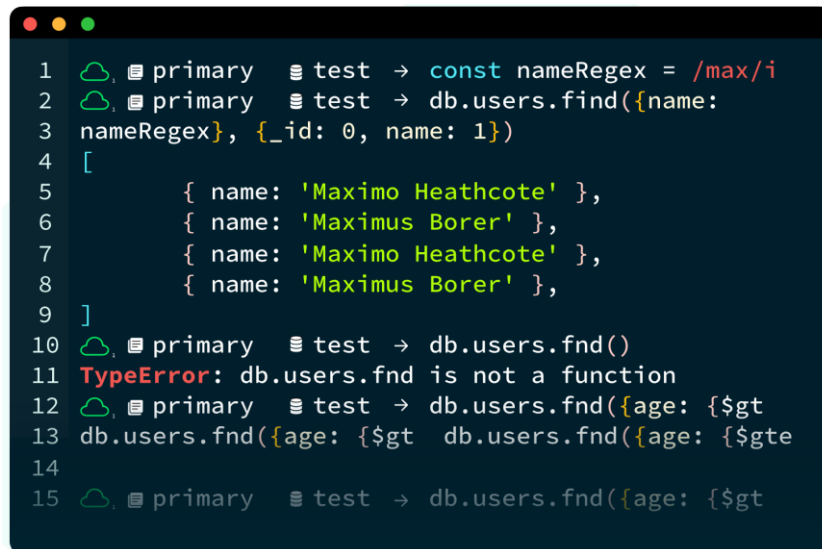
Fig 14: MongoDB Developer Tools

5.1 MongoDB Shell

The MongoDB shell is a JavaScript-based interface that provides a powerful command-line interface for managing MongoDB databases. The shell allows users to perform a variety of operations, including database administration, data retrieval, and data manipulation. The MongoDB shell is a critical tool for developers and database administrators who work with MongoDB because it allows them to interact with databases in real-time. MongoDB shell provides a range of commands for working with databases. These commands include CRUD operations (create, read, update, delete), aggregation, indexing, and replication. The shell also allows users to write complex queries using the MongoDB Query Language (MQL). MQL is a rich query language that provides a range of operators for querying and manipulating data.

The shell also provides a range of administrative commands that allow users to manage databases, collections, and users. Administrative commands include creating and dropping databases and collections, adding and removing users, and configuring database settings. These commands are essential for managing MongoDB databases in a production environment. It is a flexible tool that can be used in a variety of ways.

Some developers prefer to use the shell for ad-hoc queries and data manipulation, while others use it to create and manage scripts that automate database tasks. The shell can be run from the command line or from within a script, making it easy to integrate with other tools and workflows. Overall, the MongoDB shell is a powerful and versatile tool that provides a flexible and interactive way to work with MongoDB databases.



```

1  ☁️ primary test → const nameRegex = /max/i
2  ☁️ primary test → db.users.find({name:
3  nameRegex}, {_id: 0, name: 1})
4  [
5      { name: 'Maximo Heathcote' },
6      { name: 'Maximus Borer' },
7      { name: 'Maximo Heathcote' },
8      { name: 'Maximus Borer' },
9  ]
10 ☁️ primary test → db.users.fnd()
11 TypeError: db.users.fnd is not a function
12 ☁️ primary test → db.users.fnd({age: {$gt
13 db.users.fnd({age: {$gt db.users.fnd({age: {$gte
14
15 ☁️ primary test → db.users.fnd({age: {$gt
  
```

Fig 15: Mongo Shell UI

Key features of MongoDB shell are:

- **Syntax Highlighting:** MongoDB shell has syntax highlighting feature which helps in identifying different elements in the code. For example, keywords, strings, variables, comments, and operators are all highlighted in different colors making the code more readable and easy to understand.
- **Error Messages:** When there is an error in the code, MongoDB shell provides detailed error messages which help in identifying the problem. This feature is particularly useful for debugging and fixing errors in the code.
- **Intelligent Autocomplete:** MongoDB shell has an intelligent autocomplete feature which suggests possible commands or collections based on the context of the code. This feature saves time and reduces errors while typing the commands.
- **Contextual Help:** MongoDB shell provides contextual help feature which provides information about the commands, collections, and other MongoDB concepts. This feature helps in understanding the MongoDB concepts and using the commands effectively.
- **Scripting:** MongoDB shell allows users to write scripts in JavaScript which can be used for various purposes like querying the database, inserting data, updating data, and deleting data. This feature is useful for automating repetitive tasks and creating complex queries.
- **Extensible with Snippets:** MongoDB shell can be extended with snippets which are small pieces of code that can be reused in the shell. This feature helps in writing complex queries and reduces the need for typing the same code repeatedly.

5.2 MongoDB Compass

MongoDB Compass provides a visual interface for managing MongoDB databases. The GUI allows users to view the structure of databases, collections, and documents, and provides a range of tools for querying and manipulating data. The interface is designed to be intuitive and user-friendly, making it easy for developers and database administrators to work with MongoDB. One of the key features of MongoDB Compass is its ability to visualize data. The tool provides a range of data visualization options, including charts, tables, and maps, that allow users to explore and analyze data in new ways. The visualization options are highly customizable, making it easy to create the perfect visualization for any dataset.

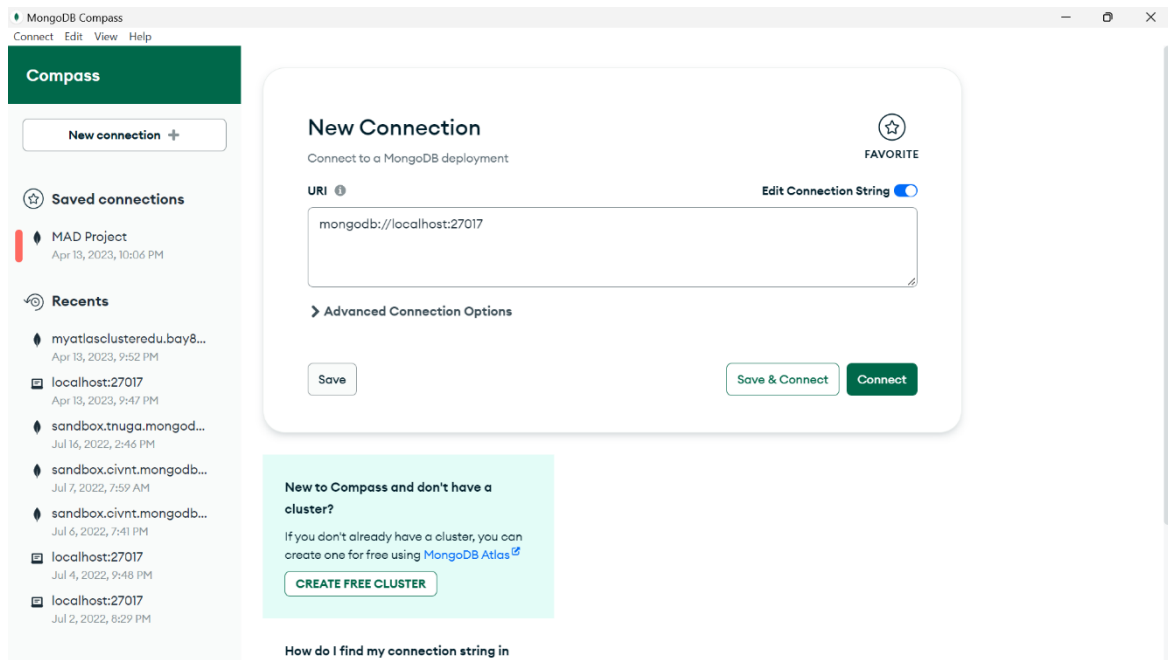


Fig 16: MongoDB Compass UI

Compass also provides a range of tools for managing MongoDB databases. These tools include the ability to create and edit databases, collections, and indexes, as well as the ability to manage users and roles. The tool provides a simple and intuitive interface for managing databases, making it easy to perform complex administrative tasks. It is also highly extensible, with a range of plugins and integrations available that can extend the tool's functionality. The tool integrates seamlessly with other MongoDB tools and services, as well as with third-party tools like Tableau and Power BI. This makes it easy to integrate MongoDB data into existing workflows and toolchains.

Key features of MongoDB Compass

- **Interact with data using our versatile GUI:** MongoDB Compass provides a graphical user interface (GUI) that allows users to interact with MongoDB data visually. Users can navigate through the data, view and edit documents, and perform CRUD (create, read, update, delete) operations on the data.
- **Get the data you need – fast:** MongoDB Compass has a powerful search feature that allows users to quickly find the data they need based on different criteria, such as field values or text searches. This feature is particularly useful for working with large databases or datasets.

- **Drag and drop aggregation pipeline builder:** Compass allows users to build aggregation pipelines using a drag-and-drop interface. This feature makes it easier for users to create complex queries and analysis of the data.
- **Access the MongoDB Shell directly in Compass:** Compass provides access to the MongoDB shell directly within the GUI, allowing users to perform advanced operations and run scripts without leaving the application.
- **Visualize, validate, and analyze schema:** Compass allows users to view and analyze the schema of their MongoDB collections. This feature helps in understanding the structure of the data and identifying potential issues or inconsistencies.
- **Assess query performance in granular detail:** Compass provides a detailed view of query performance, including execution time, index usage, and query plan. This feature helps users optimize their queries for better performance and efficiency.

5.3 MongoDB Atlas

MongoDB Atlas provides a fully managed cloud database service for MongoDB. The service allows users to easily deploy, manage, and scale MongoDB databases in the cloud. MongoDB Atlas is built on the same technology as MongoDB, providing users with a consistent and reliable database experience in the cloud. One of the key features of MongoDB Atlas is its scalability. The service allows users to scale databases up and down as needed, without the need for complex manual processes. MongoDB Atlas also provides automatic sharding and load balancing, making it easy to scale databases horizontally.

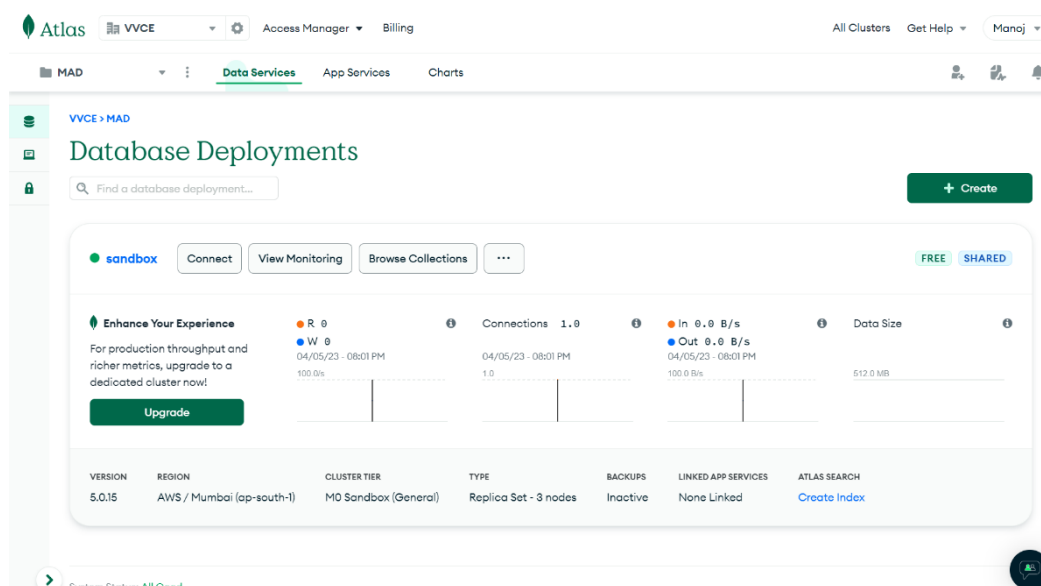


Fig 17: MongoDB Atlas UI

MongoDB Atlas also provides a range of security features to help protect data in the cloud. These features include data encryption at rest and in transit, network isolation, and role-based access control. MongoDB Atlas also provides a range of compliance certifications, including SOC 2, PCI DSS, and HIPAA. MongoDB Atlas also provides a range of tools and services to help developers and database administrators work with MongoDB in the cloud. These tools include MongoDB Compass, a GUI tool for

managing MongoDB databases, and MongoDB Stitch, a serverless platform for building and deploying MongoDB-powered applications. MongoDB Atlas also provides integrations with popular cloud services like AWS, Azure, and Google Cloud Platform.

Key features of MongoDB Atlas:

- **Multi-region, multi-cloud:** MongoDB Atlas is a cloud-based database service that allows users to deploy their databases across multiple regions and multiple cloud providers. This feature provides flexibility and helps in ensuring high availability and disaster recovery.
- **Serverless and elastic:** MongoDB Atlas provides a serverless architecture which automatically scales up or down based on the usage and workload. This feature helps in reducing costs and improving the performance of the database.
- **Always-on security:** MongoDB Atlas provides a comprehensive security model that includes encryption at rest and in transit, network isolation, access control, and audit logs. This feature ensures that the data is always protected and compliant with industry standards.
- **Continuous backups:** MongoDB Atlas provides continuous backup and point-in-time recovery feature which helps in recovering the data in case of any disaster or corruption. This feature provides peace of mind to users and ensures that their data is always protected and available.

CHAPTER 6

MONGODB CRUD OPERATIONS

6.1 Insert and Find Documents

6.1.1 Insert Documents

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection.

```
db.myNewCollection.insertOne( ) // New in version 3.2
db.myNewCollection.insertMany( ) // New in version 3.2
```

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

```
db.user.insertOne(
{
  name: "Tom",
  age: 21,
  status: "pending"
})
```

The below code describes the insertion of many document to the collection in a single throw using insertMany() method.

```
db.user.insertMany(
[
  {
    name: "Tom",
    age: 21,
    status: "pending"
  },
  {
    name: "Jerry",
    age: 25,
    status: "aproved"
  },
])
```

6.1.2 Find Documents

Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection.

Below code explains find method we need to mention collection name and in the method we can add query criteria. To project specified column we mention key with value and we can limit the number of documents by cursor.

```
db.users.find(
  { age: { $gt: 18 } },
  { name: 1, address: 1 }
).limit(5)
```



Fig 18: Find Document

6.2 Update and Delete Documents

6.2.1 Update Documents

Update operations modify existing documents in a collection. In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

MongoDB provides the following methods to update documents of a collection:

```
db.collection.updateOne() //New in version 3.2
db.collection.updateMany() //New in version 3.2
db.collection.replaceOne() //New in version 3.2
```

As in the above code the updateOne() method used to updated only one document which matches the mentioned filter, updateMany() method also same as updateOne() but it is used to update many documents which matches the condition. replaceOne() method which is used to replace the existing document according to the condition mentioned.

There is also another option in the update called upsert which takes Boolean values if the value is true then if the document does not exists it will insert the document.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

```
db.users.updateMany(
  { age: { $lt: 18 } },
  { $set: { status: "reject" } }
```



Fig 19:Update Document

6.2.2 Delete Document

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

```
db.collection.deleteOne() //New in version 3.2
db.collection.deleteMany() //New in version 3.2
```

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

```
db.users.deleteMany(  
  { status: "reject" }  
)
```

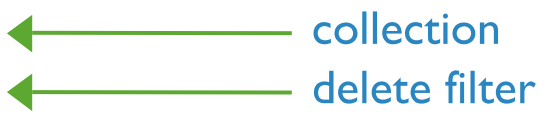


Fig 20: Delete Document

6.3 Modifying Query Results

In MongoDB we can modify the documents that are returned from the query by using something called cursors. Cursors in MongoDB means results that get from the query.

There are so many cursor methods some of them are mentioned below.

- Sorting query results in MongoDB

```
cursor.sort()
```

- Limiting query results in MongoDB

```
cursor.limit(<no of documents>)
```

- Display results in a format that is easy to read.

```
db.collection.find(<query>).pretty()
```

- Count the number of documents referred by cursor

```
cursor.count()
```

There are many of these cursor methods to make the query easy and to provide easy interaction between the application.

CHAPTER 7

MONGODB DATA MODELING

Any modern application needs data to run, but how you model your data will have a drastic impact on the performance of your application, as well as the speed of development. In this article, you will learn what data modeling is, why you need it, and some efficient data modeling techniques to be used with MongoDB.

7.1 Data Modeling

Data modeling is the process of creating a clean data model of how you will store data in a database. Data models also describe how the data is related. The goal of data modeling is to identify all the data components of a system, how they are connected, and what are the best ways to represent these relationships.

Data modeling is done at the application level. Data models consist of the following components:

- **Entity:** An independent object that is also a logical component in the system. Entities can be categorized into tangible and intangible. Tangible entities, such as books, exist in the real world, while intangible entities, such as book loans, don't have a physical form. In document databases, each document is an entity. In tabular databases, each row is an entity.
- **Entity type:** The categories used to group entities. For example, the book entity with the title "Alice in Wonderland" belongs to the entity type "book."
- **Attribute:** The characteristics of an entity. For example, the entity "book" has the attributes ISBN (String) and title (String).
- **Relationships:** Define the connections between the entities. For example, one user can borrow many books at a time. The relationship between the entities "users" and "books" is one to many.

Although MongoDB has a flexible schema, you'd need data modeling or schema design. A good data model means that you will establish a strong foundation for an ever-evolving data model that will adapt as your requirements change. By creating a solid data model right from the start, you can ensure that your application will perform better and be more future-proof.

MongoDB document based database does supports multiple ways of connection (relationship) between entities:

- **One to one (1-1):** In this type, one value is associated with only one document—for example, a book ISBN. Each book can have only one ISBN.
- **One to many (1-N):** Here, one value can be associated with more than one document or value. For example, a user can borrow more than one book at a time.
- **Many to many (N-N):** In this type of model, multiple documents can be associated with each other. For example, a book can have many authors, and one author can write many different books. The relationship between author and book is many to many.

7.2 Flexible Schema

Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema.

That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection.

7.3 Document Structure

The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document.

7.3.1 Embedded Data

Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.



Fig 21: Embedded Data

Advantages of Embedded Data in MongoDB:

- **Performance:** Embedded data helps to improve query performance by reducing the number of database operations required to retrieve related data. This is because all the related data is stored in a single document, and a single database query can retrieve all the data in one go.

- **Data Consistency:** Embedded data helps to ensure data consistency by allowing atomic updates. Since all the related data is stored in a single document, updating the data in one go ensures that all the data is updated atomically and consistently.
- **Simplified Data Model:** Embedding related data in a single document simplifies the data model and makes it easier to understand and maintain. It eliminates the need for complex joins and relationships between collections.
- **Flexibility:** Embedding data in a single document provides flexibility in designing the data model. It allows data to be organized in a way that best suits the application's needs, rather than being constrained by traditional relational database design principles.
- **Scalability:** Embedding data can improve database scalability by reducing the number of database operations and reducing the need for database joins. This makes it easier to scale the database horizontally by adding more servers.

7.3.2 Referenced Documents

References store the relationships between data by including links or references from one document to another. Applications can resolve these references to access the related data. Broadly, these are normalized data models.

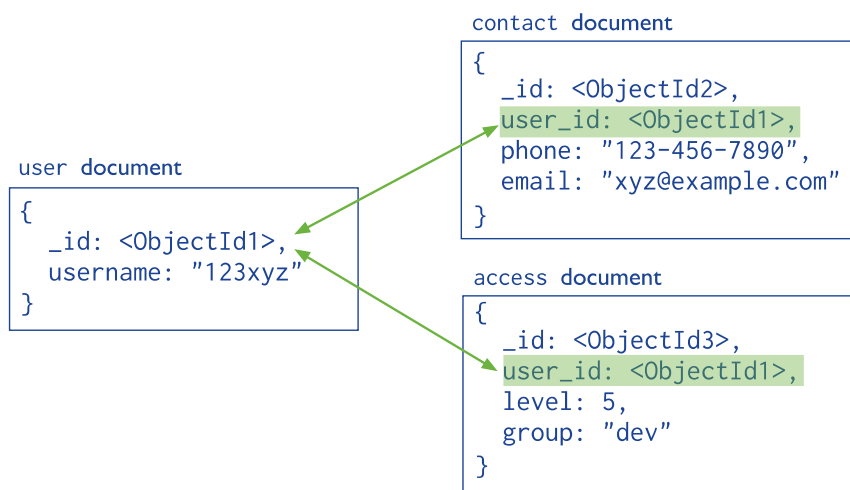


Fig 22: Reference Document

7.4 Use of Data Modeling

A data model provides you with the foundation for the data structure of your application. Creating the data model can help you to identify the business rules that your application will need to follow. The data model will also provide your development team with a consistent map of the data used by the application they are developing.

Thinking ahead of time about how you will access the information from the application will help you plan better. It will also help you understand the business processes that are sometimes hidden or not explicitly explained by your stakeholders.

While data modeling might seem like an additional step in the development planning cycle, it can

make the development cycles much faster.

7.5 Advantages of Data Modeling

There are several advantages of data modeling:

- Ensures better database planning, design, and implementation, leading to improved application performance.
- Promotes faster application development through easier object mapping.
- Better discovery, standardization, and documentation of multiple data sources.
- Allows organizations to think of long-term solutions and model data considering not only current projects, but also future requirements of the application.

7.6 Types of Data Modeling

As you start modeling your data, you will likely go through various steps of data analysis. Each step might produce different types of data models. Therefore, data models can be generally thought of as being one of the three following types.

- **Conceptual Data Model:** The conceptual data model explains what the system should contain with regard to data and how it is related. This model is usually built with the help of the stakeholders. It represents the application's business logic and is often used as the basis for one or more of the following models.
- **Logical Data Model:** The logical data model will describe how the data will be structured. In this model, the relationship between the entities is established at a high level. You will also list the attributes for the entities represented in the model.
- **Physical Data Model:** The physical data model represents how the data will be stored in a specific database management system (DBMS). With this model, you would establish your primary and secondary keys in a relational database or decide whether to embed or link your data in a document database such as MongoDB. You will also establish the data types for each of your fields. This will provide you with your database schema.

These models are created using entity-relationship diagrams (ERD).

7.7 Flexible Data Modeling with MongoDB Atlas

This flexible schema means that you can easily change your data structure as your application progresses and the requirements change. This doesn't mean that you shouldn't think about data modeling, though. A good data model will help you understand the business requirements for your application and structure your schema to optimize your queries.

As you model your data, think about how the data will be displayed and used in your application. The way you use the data should dictate the structure of your database, and not the other way around. This is why MongoDB is easier to use for software developers. MongoDB's fully managed database-as-a-service (DBaaS).

CHAPTER 8

MONGODB INDEXING

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan, i.e., scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index. MongoDB indexes use a B-tree data structure.

Default `_id` Index: MongoDB creates a unique index on the `_id` field during the creation of a collection. The `_id` index prevents clients from inserting two documents with the same value for the `_id` field. You cannot drop this index on the `_id` field.

8.1 Create an Index

To create an index in the Mongo Shell, use `db.collection.createIndex()`.

```
db.collection.createIndex( <key and index type specification>,  
                           <options> )
```

The following example creates a single key descending index on the name field:

```
db.collection.createIndex( { name: -1 } )
```

The `db.collection.createIndex()` method only creates an index if an index of the same specification does not already exist.

8.2 Index Types

MongoDB provides a number of different index types to support specific types of data and queries.

1. Single Field Index
2. Compound Field Index
3. Multi Key Index
4. Geospatial Index
5. Text Search Indexes
6. Hashed Indexes
7. Clustered Indexes

8.2.1 Single Field Index

In addition to the MongoDB-defined `_id` index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document. For a single-field index and sort operations,

the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.

8.2.2 Compound Index

MongoDB supports compound indexes, where a single index structure holds references to multiple fields within a collection's documents.

Create a Compound Index

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>,  
... } )
```

The value of the field in the index specification describes the kind of index for that field. For example, a value of 1 specifies an index that orders items in ascending order. A value of -1 specifies an index that orders items in descending order. For additional index types, see index types.

The order of the indexed fields has a strong impact on the effectiveness of a particular index for a given query. For most compound indexes, following the ESR (Equality, Sort, Range) rule helps to create efficient indexes.

8.2.3 Multi Key Index

MongoDB uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array. These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.

Create Multi key Index

To create a multikey index, use the `db.collection.createIndex()` method:

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>,  
... } )
```

MongoDB automatically creates a multikey index if any indexed field is an array; you do not need to explicitly specify the multikey type.

8.2.4 Geospatial Index

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2dsphere indexes that use spherical geometry to return results.

8.2.5 Text Search Indexes

To run text search queries on self-managed deployments, you must have a text index on your collection. MongoDB provides text indexes to support text search queries on string content. Text indexes can include any field whose value is a string or an array of string elements. A collection can only have one text search index, but that index can cover multiple fields.

Create Text Index

To create a text index, use the `db.collection.createIndex()` method:

```
db.collection.createIndex( { <field1>: <type>(String) } )
```

To index a field that contains a string or an array of string elements, include the field and specify the string literal in the index document.

8.2.6 Hashed Index

Hashed indexes maintain entries with hashes of the values of the indexed field. Hashed indexes support sharding using hashed shard keys. Hashed based sharding uses a hashed index of a field as the shard key to partition data across your sharded cluster. Using a hashed shard key to shard a collection results in a more even distribution of data.

Hashed indexes use a hashing function to compute the hash of the value of the index field. [1] The hashing function collapses embedded documents and computes the hash for the entire value but does not support multi-key (i.e. arrays) indexes. Specifically, creating a hashed index on a field that contains an array or attempting to insert an array into a hashed indexed field returns an error.

Create Hashed Index

MongoDB supports creating compound indexes that include a single hashed field. To create a compound hashed index, specify `hashed` as the value of any single index key when creating the index:

```
db.collection.createIndex( { _id: "hashed" } )
```

Using a hashed shard key to shard a collection results in a more even distribution of data.

CHAPTER 9

MONGODB AGGREGATION OPERATIONS

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyze data changes over time.

To perform aggregation operations, you can use:

1. Aggregation pipelines
2. Single purpose aggregation methods

9.1 Aggregation Pipelines

An aggregation pipeline consists of one or more stages that process documents:

- Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
- The documents that are output from a stage are passed to the next stage.
- An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

9.1.1 Aggregation Pipeline Example

The following aggregation pipeline example contains two stages and returns the total order quantity of medium size pizzas grouped by pizza name:

```
db.orders.aggregate( [

  // Stage 1: Filter pizza order documents by pizza size
  {
    $match: { size: "medium" }
  },

  // Stage 2: Group remaining documents by pizza name and calculate
  total quantity
  {
    $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
  }

] )
```

The **\$match** stage:

- Filters the pizza order documents to pizzas with a size of medium.
- Passes the remaining documents to the \$group stage.

The **\$group** stage:

- Groups the remaining documents by pizza name.

- Uses \$sum to calculate the total order quantity for each pizza name. The total is stored in the totalQuantity field returned by the aggregation pipeline.

9.1.2 Additional Aggregation Pipeline Stage Details

An aggregation pipeline consists of one or more stages that process documents:

- A stage does not have to output one document for every input document. For example, some stages may produce new documents or filter out documents.
- The same stage can appear multiple times in the pipeline with these stage exceptions: \$out, \$merge, and \$geoNear.
- To calculate averages and perform other calculations in a stage, use aggregation expressions that specify aggregation operators.

9.2 Single Purpose Aggregation Methods

The single purpose aggregation methods aggregate documents from a single collection. The methods are simple but lack the capabilities of an aggregation pipeline.

Table 1: Single Purpose Aggregation Methods

Method	Description
db.collection.estimatedDocumentCount()	Returns an approximate count of the documents in a collection or a view.
db.collection.count()	Returns a count of the number of documents in a collection or a view.
db.collection.distinct()	Returns an array of documents that have distinct values for the specified field.
db.collection.explain()	Returns information on the query plan for aggregate(), count(), find(), remove(), distinct(), findAndModify().

Not only above 4 methods there are so many single purpose aggregation method that you can refer the MongoDB Manual.

CHAPTER 10

COMPARISON WITH MONGODB

10.1 MongoDB v/s MySQL

Table 2: MongoDB v/s MySQL

MongoDB	MySQL
<ul style="list-style-type: none"> MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. It is a popular NoSQL database. 	<ul style="list-style-type: none"> MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation.
<ul style="list-style-type: none"> In MongoDB, each individual record are stored as 'documents'. 	<ul style="list-style-type: none"> In MySQL, each individual records are stored as 'rows' in a table.
<ul style="list-style-type: none"> Documents belonging to a particular class or group as stored in a 'collection'. 	<ul style="list-style-type: none"> A 'table' is used to store rows (records) of similar type.
<ul style="list-style-type: none"> MongoDB is what is called a NoSQL database. This means that pre-defined structure for the incoming data can be defined and adhered to but also, if required different documents in a collection can have different structures. It has a dynamic schema. 	<ul style="list-style-type: none"> MySQL as the name suggests uses Structured Query Language (SQL) for database access. The schema can not be changed. The inputs following the given schema are only entered.
<ul style="list-style-type: none"> MongoDB was designed with high availability and scalability in mind, and includes out-of-the-box replication and sharding. 	<ul style="list-style-type: none"> MySQL concept does not allow efficient replication and sharding but in MySQL one can access associated data using joins which minimizes duplication.

Ultimately, the choice between MySQL and MongoDB depends on the specific requirements of your application. If you need a highly reliable database with ACID compliance, MySQL may be the better choice. If you have large amounts of unstructured data and require scalability and flexibility, MongoDB may be a better fit.

10.2 MongoDB v/s Cassandra

Table 3: MongoDB v/s Cassandra

MongoDB	Cassandra
<ul style="list-style-type: none"> MongoDB stores documents in an optimized BSON format. Documents are grouped in databases and collections and returned as JSON 	<ul style="list-style-type: none"> Cassandra stores data in a wide column tabular store. Each row contains a unique identifier (or set of identifiers) as its primary key and a

documents with support for a large number of data types, including: Strings, numbers, geo data, dates, arrays, decimal, nested objects, and binary data.	following set of columns. The primary key serves as a partition hash key as data is distributed across the cluster.
<ul style="list-style-type: none"> • MongoDB supports many index types for various use cases. Secondary indexes on any field are available and supported in different types: Compound, Text, Geo, Wild Card, TTL, and Partial. 	<ul style="list-style-type: none"> • Cassandra offers standard built-in indexing as well as basic secondary indexes to index additional columns to allow queries filtering. Maintaining a large secondary index can potentially cause scalability issues as the cluster grows.
<ul style="list-style-type: none"> • MongoDB has a rich query language called MQL. It supports a wide variety of modern native drivers as well as a shell. • The queries can use complex operators. Additionally, the aggregation framework allows you to aggregate data with many stages. 	<ul style="list-style-type: none"> • Cassandra uses a proprietary language called CQL, which is similar to SQL. Applications can use different drivers provided mostly by third parties and a shell.
<ul style="list-style-type: none"> • MongoDB supports fully ACID compliant transactions 	<ul style="list-style-type: none"> • Cassandra does not support transactions.
<ul style="list-style-type: none"> • MongoDB allows multiple database users to concurrently access the same data by managing a well-defined concurrency control. • MongoDB uses document-level locking, so writes to a single document occur either in full or not at all, and clients always see consistent data. 	<ul style="list-style-type: none"> • Cassandra isolates on a row-level. This means that a specific row for a specific partition can be operated simultaneously by one client. Cassandra also offers tunable consistency for writing, when a client changes its consistency level (one, quorum, all).
<ul style="list-style-type: none"> • MongoDB was built from the ground up to support distribution of data using replication and sharding mechanisms. • Sharding allows you to easily scale your collections across multiple replica sets. With geo-zone sharding, you can also easily manage data sovereignty requirements. 	<ul style="list-style-type: none"> • Cassandra replicates its keyspace across the cluster nodes based on the keyspace replication factor. • Since it's a multi-master ring, each node is holding a range of the partition key per table, while also hosting parts of other nodes' replicas.

Both databases have their strengths and weaknesses, and the choice between them depends on specific requirements. MongoDB is ideal for applications that require flexible data models, while Cassandra is best for write-intensive and highly available applications that require scalability and fault tolerance.

10.3 MongoDB v/s Redis

Table 4: MongoDB v/s Redis

MongoDB	Redis
<ul style="list-style-type: none"> On-disk storage by default 	<ul style="list-style-type: none"> In-memory storage with on-disk persistence.
<ul style="list-style-type: none"> BSON (Binary JSON) documents Document size is up to 16MB. Supports multiple data types: String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Object, Date, Raw Binary, GeoJSON. 	<ul style="list-style-type: none"> Key-value store Keys are binary safe strings with length up to 512MB. Values: String and multiple data structures, such as List, Set, Bitmap, Hash.
<ul style="list-style-type: none"> MongoDB Query API Query documents by single or multiple keys, ranges, or text search. 	<ul style="list-style-type: none"> Key-value queries only Limited query functionality. By design only primary key access.
<ul style="list-style-type: none"> Rich and easy to create indexes Secondary indexes enable developers to build applications that can query and analyze the data in multiple ways. 	<ul style="list-style-type: none"> Hard to build secondary indexes Secondary indexes should be manually built and maintained.
<ul style="list-style-type: none"> Built-in Sharding Sharding allows you to scale out across multiple nodes and geographic regions. Multi-cloud support, consistent backups with point in time recovery by MongoDB Atlas. Partition data by range, hash, or zone. Perform cross-shard operations. 	<ul style="list-style-type: none"> Redis Cluster No multi-shard database operations. No consistent cross-shard backups and no strong consistency. Hash sharding only. Manual maintenance of the shards.
<ul style="list-style-type: none"> MongoDB was built from the ground up to support distribution of data using replication and sharding mechanisms. Sharding allows you to easily scale your collections across multiple replica sets. With geo-zone sharding, you can also easily manage data sovereignty requirements. 	<ul style="list-style-type: none"> Cassandra replicates its keyspace across the cluster nodes based on the keyspace replication factor. Since it's a multi-master ring, each node is holding a range of the partition key per table, while also hosting parts of other nodes' replicas.

MongoDB is a general-purpose persistent database used as a primary storage solution by businesses in various industries. Redis is an in-memory data store that can be configured to work in different ways depending on your system's needs.

CHAPTER 11

ADVANTAGES AND DISADVANTAGES OF MONGODB

11.1 Advantages of MongoDB

MongoDB offers many advantages over traditional relational databases:

- Full cloud-based developer data platform
- Flexible document schemas
- Widely supported and code-native data access
- Change-friendly design
- Powerful querying and analytics
- Easy horizontal scale-out with sharding
- High Performance
- Simple installation
- Cost-effective
- Full technical support and documentation

11.1.1 Flexible Document Schemas

MongoDB is a document-oriented database management system that is well-known for its ability to handle large-scale, complex data. One of the key features that sets MongoDB apart from traditional relational databases is that it is schemaless. This means that there is no predefined structure or schema that governs how data is organized and stored within MongoDB.

In a schemaless database like MongoDB, documents can be stored without any predefined structure, allowing for much greater flexibility and agility in data modeling. Instead of having to fit data into predefined tables with fixed column names and data types, developers can store data in a more natural and intuitive way. This allows for easier development and faster iteration times, as schema changes can be made more quickly and with less disruption.

MongoDB's document model allows virtually any data structure to be modeled and manipulated easily. MongoDB's BSON data format, inspired by JSON, allows you to have objects in one collection with different sets of fields (say, a middle name on a user only when applicable, or region-specific information that only applies to some records). MongoDB supports creating explicit schemas and validating data. This flexibility is an incredible asset when handling real-world data and changes in requirements or environment.

11.1.2 Full cloud-based developer data platform

MongoDB is much more than a database. It's a complete developer data platform. With MongoDB Atlas, the cloud offering by MongoDB, you have access to a collection of services that all integrate nicely with your database. Amongst other things, you will have:

- The Performance Advisor, which provides you with recommendations to optimize your database.
- Atlas Search, a full-text search engine that uses the same MongoDB Query API as other queries.
- MongoDB Charts, an easy-to-use interface to create stunning dashboards and visualizations.
- Multi-cloud deployment, which is offered out-of-the-box on any major cloud provider.

11.1.3 Widely supported and code native access

Most databases force you to use heavy wrappers, like ORMs (Object-Relational Mappers), to get data into Object form for use in programs. MongoDB's decision to store and represent data in a document format means that you can access it from any language, in data structures that are native to that language (e.g., dictionaries in Python, objects in JavaScript, Maps in Java, etc.).

11.1.4 Change-friendly design

If you're used to having to bring down your site or application in order to change the structure of your data, you're in luck: MongoDB is designed for change.

We spend a lot of time and effort designing efficient processes and learning from our mistakes, but typically the database is slowing the whole thing down. There's no downtime required to change schemas, and you can start writing new data to MongoDB at any time, without disrupting its operations.

11.1.5 Powerful querying and analytics

What good is a database if you can't find things inside it? MongoDB is designed to make data easy to access, and rarely to require joins or transactions, but when you need to do complex querying, it's more than up to the task.

The MongoDB Query API allows you to query deep into documents, and even perform complex analytics pipelines with just a few lines of declarative code.

11.1.6 Easy horizontal scale-out with sharding

Scalability is one of the biggest advantages of MongoDB, you can develop apps that are capable of handling traffic spikes smoothly, thanks to the scale-out architecture it features. This means that the work is distributed across numerous computers that are smaller and less expensive. Due to the many innovations by MongoDB, massive volumes of the read and write operations are supported.

Due to the manner of Sharding in MongoDB, information clusters can be stored in one place, though the information itself is stored on numerous computer clusters. This is in stark contrast to the relational database architecture which scales up in order to create computers that are speedier and more powerful and are therefore limited. Objects can be embedded within one another during data modeling in MongoDB. Instead of multiple transactions as in conventional relational databases, updating can be achieved here with just one transaction.

11.1.7 High performance (speed)

Performance issues may indicate that the database is operating at capacity and that it is time to add additional capacity to the database. In particular, the application's working set should fit in the available

physical memory. Thanks to the document model used in MongoDB, information can be embedded inside a single document rather than relying on expensive join operations from traditional relational databases. This makes queries much faster, and returns all the necessary information in a single call to the database. If needed, you can perform a left outer join with the \$lookup aggregation pipeline stage, which delivers similar performance to RDBMSs.

When it comes to write performance, MongoDB offers functionalities to insert and update multiple records at once with insertMany and updateMany. These two functions offer a significant performance boost when compared to batched writes in traditional databases.

11.1.8 Simple installation

With MongoDB Atlas, creating and setting up a MongoDB cluster is easier than ever. With just a few clicks in the intuitive UI, you can deploy a new forever-free instance. Within minutes, you will be able to connect to your database using the provided connection string.

If you want to run MongoDB on your own hardware, there are many ways to get started. You can install the community or enterprise version directly on a server. You can also create your own MongoDB container, or use a pre-built community one.

11.1.9 Cost-effective

MongoDB offers multiple flexible approaches. When using the cloud-based MongoDB Atlas, you can choose an instance size that fits your current needs. You can also adjust your cluster to automatically scale when needed. This way, you keep your costs at a minimum, while still having the flexibility to handle sudden traffic bursts.

In addition to the flexible cost for dedicated clusters, you can now create Serverless Databases. For these databases, you will only be charged for the actual usage, making it very flexible and perfect for many lower-usage use cases.

11.1.10 Full technical support and documentation

When it comes to finding help, MongoDB's got your back. MongoDB has an extensive documentation available, as well as a large collection of getting started tutorials on the documentation website. A community forum is also available for you to ask your questions.

If you want to learn more about how to use MongoDB, take a look at MongoDB University. MDBU offers a large collection of free courses that will teach you everything you need to know about MongoDB. If you still can't find an answer to your problem, MongoDB offers many support plans with MongoDB Enterprise and MongoDB Atlas paid tiers on a subscription model.

11.2 Disadvantages of MongoDB

MongoDB also have some disadvantages:

- Limited Transaction Support
- Limited Joins

- Limited Indexing
- Limited Data Size and Nesting
- Duplication of Data
- High Memory usage

11.2.1 Limited Transaction Support

Transactions refer to the process of reviewing and eliminating unwanted data. MongoDB uses multi-document ACID (Atomicity, Consistency, Isolation, and Durability) transactions.

Most of the application does not require transactions, although there are a few that may need it to update multiple documents and collections. This is one of the major limitations with MongoDB as it may lead to corruption of data.

11.2.2 Limited Joins

Joining documents in MongoDB can be a very tedious task. It fails to support joins as a relational database. Although there are teams deployed to fix this disadvantage, it is still in the initial stages and would take time to mature. Users can utilize joins functionality by manually adding the code. But acquiring data from multiple collections requires multiple queries and this may lead to scattered codes and consume time.

11.2.3 Limited Indexing

MongoDB offers high-speed performance with the right indexes. In case if the indexing is implemented incorrectly or has any discrepancies, MongoDB will perform at a very low speed. Fixing the errors in the indexes would also consume time. This is another one of the major limitations of MongoDB.

11.2.4 Limited Data Size and Nesting

While it allows for a high degree of nesting within documents, there are practical limitations to the amount of data that can be stored in a single document. MongoDB imposes a maximum document size of 16 megabytes, which means that very large datasets may need to be split across multiple documents or collections. Additionally, overly nested data structures can increase the size of individual documents and negatively impact performance, so it's important to carefully consider the level of nesting required for a particular use case.

11.2.5 Duplication of Data

Duplication of data in MongoDB is that it can lead to increased storage requirements and higher maintenance costs. When data is duplicated, it needs to be updated in multiple places whenever changes are made, which can lead to inconsistencies and errors if updates are not properly managed. Additionally, duplicate data can make it harder to enforce data integrity constraints and can result in slower query performance due to larger indexes and increased disk I/O. In some cases, it may be necessary to duplicate data in order to optimize query performance or support certain types of data access patterns, but this should be done judiciously and with an understanding of the trade-offs involved.

11.2.6 High Memory Usage

High memory usage in MongoDB is that it can lead to increased operational costs. MongoDB stores data in memory to improve performance and reduce disk I/O, but as the amount of data stored grows, so does the amount of memory required to maintain acceptable performance. When the amount of memory required exceeds the available physical memory on a system, MongoDB may start swapping data to disk, which can significantly slow down performance and increase disk I/O. This can lead to higher operational costs due to increased hardware requirements, as well as increased downtime and maintenance costs. Additionally, high memory usage can limit the scalability of MongoDB, as adding more nodes to a cluster may not be feasible if the amount of memory required by each node is too high.

CHAPTER 12

APPLICATIONS OF MONGODB

12.1 MongoDB Content Management System

Building a content management system (CMS) that stores and serves content to a variety of applications requires the latest technology and development approaches. For the best user experience, the CMS has to handle a large volume of data along with a great variety of unstructured data all in real-time. Typical relational database technology falls short in this area because this kind of database has difficulty incorporating new content, attributes, or features without negatively impacting performance. To address the requirements of modern content management systems you need to turn to the latest database technology that can handle a large volume of unstructured data. MongoDB, the leading database of a new generation of technology called NoSQL, lets you store and serve up any type of content within a single database. You can build these systems with MongoDB quickly and at much less expense than with decades old relational technology.

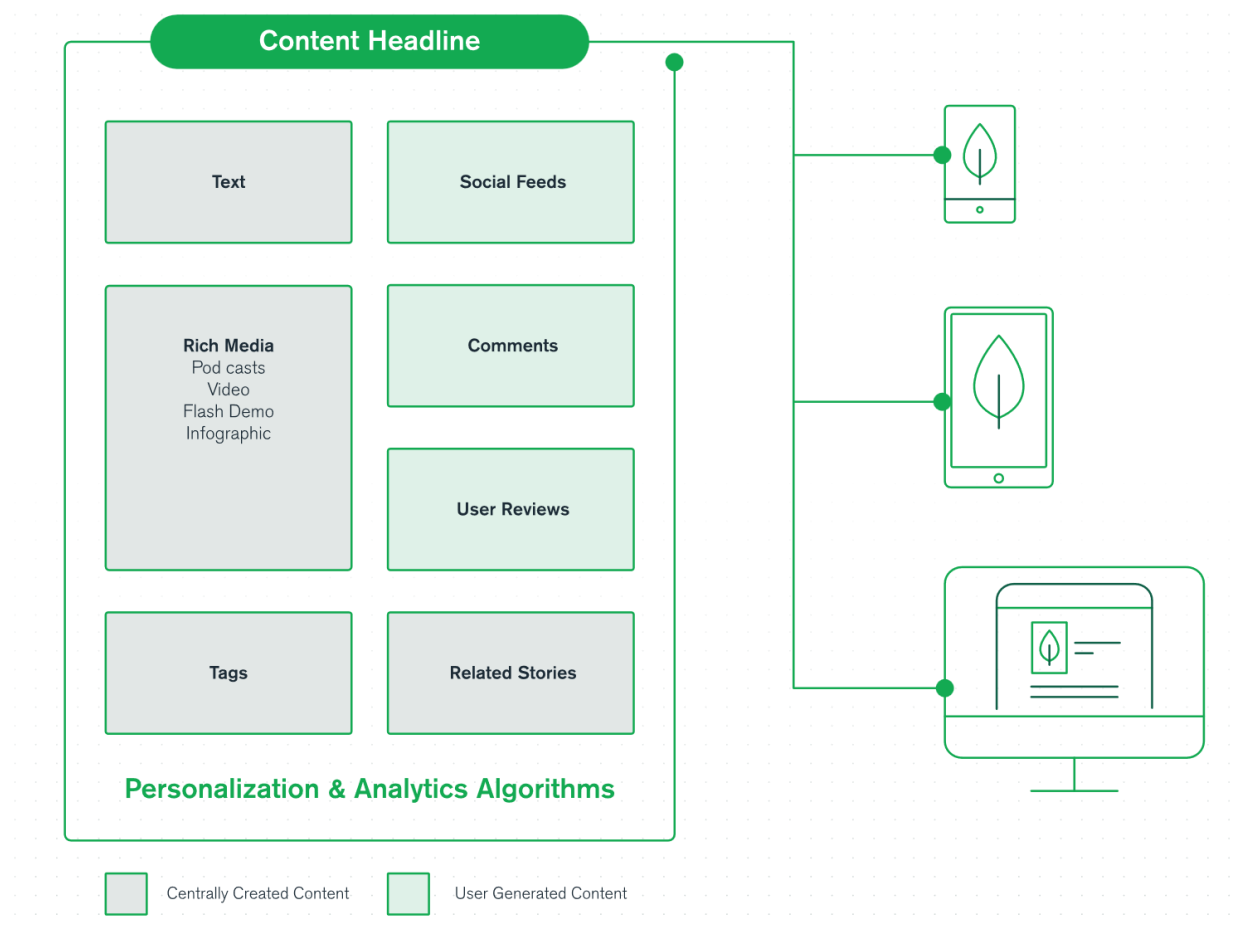


Fig 23: MongoDB CMS

MongoDB is particularly well-suited to support your CMS efforts because the software offers:

- A **flexible data model** means that you can incorporate any kind of data into your CMS, regardless of the source. This flexible model also lets you make frequent updates to the database without downtime to your application.

- **Scalable to millions of users** as MongoDB has a native horizontal scale out architecture that easily lets you accommodate additional demand as your audience grows.
- **Much lower cost** to complete your CMS project as your teams are more productive and you end up spending much less on commodity hardware to scale the system. By some estimates, it costs only 10% of what it would cost with a relational database to build a CMS on MongoDB.

Table 5: Advantage of MongoDB in CMS

Content Management is Hard	MongoDB Makes it Easy
Stuck. Data is more than text and pictures. Now it includes rich data types – tweets, videos, podcasts, animated gifs – which are hard, if not impossible, to store in a relational database. Development slows to a crawl, and ops is caught playing whack-a-mole.	Do the Impossible. MongoDB can incorporate any type of data, no matter what it looks like or where it comes from, while providing all the features needed to build content-rich apps.
Can't Scale. Your audience is global, in many countries, speaking many languages, accessing content on many devices. Scaling a relational database is not trivial. And it isn't cheap.	Scale Big. Scaling is built into the database. It is automatic and transparent. You can scale as your audience grows, both within a datacenter and across regions. This shouldn't be hard, and with MongoDB, it isn't.
Large teams tied up for long periods of time make these applications expensive to build and maintain. Proprietary software and hardware, plus separate databases and file systems needed to manage your content, add to the cost.	More productive teams, plus commodity hardware, make your projects cost 10% what they would with a relational database.

MongoDB unites all your content assets into a single database and makes for an overall better user experience. This helps your enterprise stay competitive as customers today expect a seamless experience from your business. mobile site in just a month. MongoDB helped the publisher gain more insight into the social sharing of their articles so that they were able to capitalize in real-time on content that was going viral. And there are countless other enterprises who have leveraged MongoDB technology for their CMS: eBay used it to build a media metadata storage for their web properties, Pearson employed MongoDB technology to develop a cloud-based learning management system, and Carfax migrated their vehicle history database into a MongoDB CMS and found that they could service 10x more customers as a result. These are just a few examples of companies who take advantage of the database to build powerful MongoDB CMS.

12.2 Internet of Things Applications

Technology companies are doing some awe-inspiring things with Internet of Things (IoT)

applications. IoT describes technology which connects physical assets and devices together to share information and make life easier and more convenient. IoT devices are processing volumes of data previously unimagined. Bosch, for example, is harnessing data to use in a range of industrial internet of things applications including manufacturing, automotive, retail, and energy. With these sensor-enabled objects, futuristic scenarios have come alive in the ways previously thought impossible. New revenue opportunities abound but only if companies can wrangle that data into something meaningful. Enter MongoDB, the world's most popular NoSQL database, to help you make sense of sensor data, building internet of things applications never before possible. All this in less time and with less cost than with alternative technologies.

The advantages of creating an internet of things application with MongoDB:

- **Document data model:** With MongoDB, you can manage and incorporate data in any structure. This allows for you to launch and iterate on your application without having to start from scratch to meet evolving requirements.
- **Inexpensively scale:** IoT applications process great volumes of data through sensors so your system will need to scale quickly and cheaply. One of the advantages of MongoDB is the ability to scale out on inexpensive commodity hardware in your data center or in the cloud.
- **Analyze any kind of data with MongoDB:** Real-time analysis within the database means you don't get the time delay you normally would be processing data through an expensive data warehouse system.

12.2.1 IoT architecture in MongoDB Atlas

The key to a successful IoT architecture is to find the right way to handle this large amount of data produced by the Perception layer. MongoDB Atlas, the Database-as-a-Service offering from MongoDB, provides you with a collection of tools that can be used at the various layers of your IoT solution.

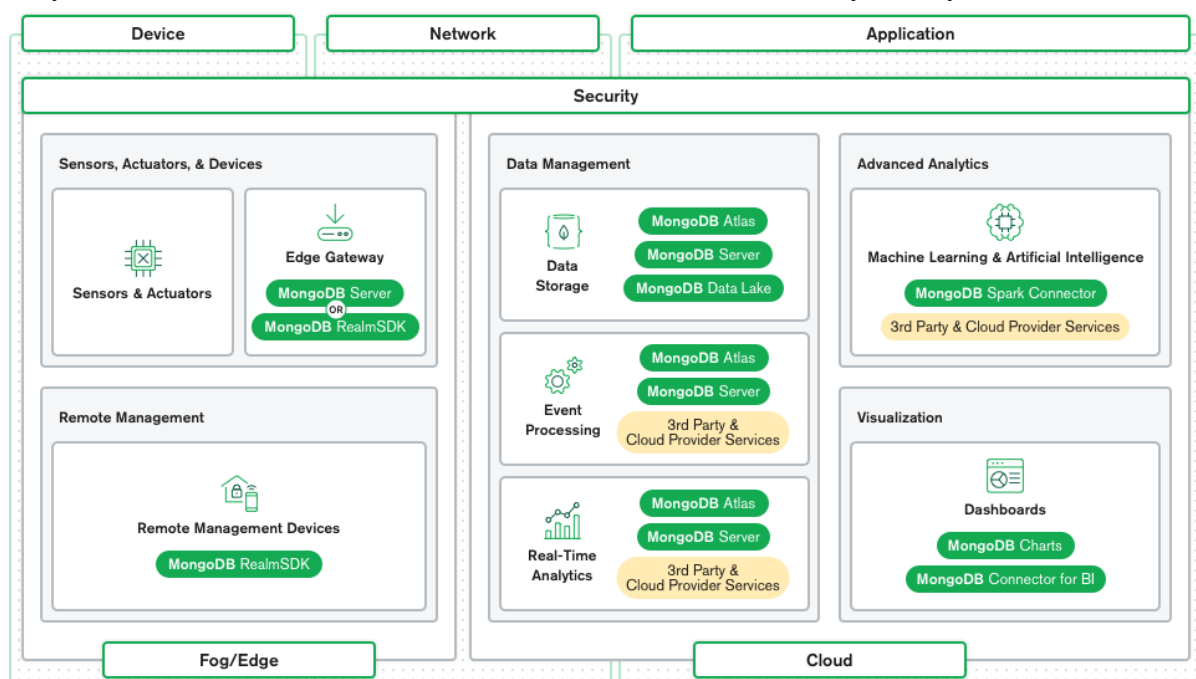


Fig 24: IoT architecture in MongoDB Atlas

At the Perception layer, you can use MongoDB Server and MongoDB RealmSDK to store data and provide an interface for the mobile devices of your users.

Once the data leaves the device and heads into the Network layer, MongoDB Atlas can provide you with many ways to configure your servers to achieve proximity with your devices. You can even deploy MongoDB on the edge with MongoDB data locality and workload isolation features enabled on your edge cluster. You can use MongoDB 5.0 Time-Series native support to store your data in collections that are perfectly suited for IoT applications since they are optimized to gather measurements over time from various sources. You might also need to plan for bad network connectivity. Atlas App Services can help you with this by providing you with offline-first syncing capabilities.

For your application layer, MongoDB also provides you with connectors for platforms such as Spark for your machine learning and data analysis needs. You can also connect to your BI tools with the MongoDB Connector for BI, or use MongoDB Charts directly to create dashboards and have a visual representation of your data.

12.3 MongoDB in Big Data Analytics

One of the significant challenges of big data analytics is handling unstructured data. Traditional databases are not designed to handle unstructured data, which is becoming increasingly common in big data applications. MongoDB is a document-based NoSQL database that can handle unstructured data efficiently. It allows developers to store data in a flexible document format, making it easier to store, process, and analyze unstructured data. MongoDB's document-based approach is particularly useful for big data analytics applications that deal with a variety of data formats, such as log files, social media data, and sensor data. Big data analytics requires high performance, and MongoDB is known for its high-speed data processing capabilities. It has a flexible data model and supports horizontal scaling, which allows it to handle large volumes of data and high traffic loads. MongoDB's performance is enhanced by its in-memory caching and indexing, which makes it possible to perform queries at sub-millisecond response times.

Big data analytics often requires an agile approach, where the database schema and queries need to be changed frequently to adapt to changing business requirements. MongoDB's flexible schema design and dynamic queries make it an ideal database for agile analytics. It allows developers to make changes to the database schema and queries without downtime, making it easier to adapt to changing business requirements. Big data analytics requires distributed data processing to handle large data volumes efficiently. MongoDB supports distributed data processing, making it easy to scale out and process data across multiple servers. The database also supports sharding, which allows developers to partition data across multiple servers and scale the database horizontally. Big data analytics often requires real-time processing of data, which requires a database that can handle high throughput and low latency. MongoDB supports real-time analytics, making it an ideal choice for big data applications that require real-time data processing. It has a powerful aggregation framework that can process data in real-time, making it possible to generate real-time insights and make quick decisions based on the data. MongoDB also supports change

streams, which allow developers to receive real-time notifications when data changes in the database.

Advantages of MongoDB in Big data analytics:

- Unstructured Data Handling
- High Performance
- Agile Analytics
- Distributed Data Processing
- Real-time Analytics

12.4 E-Commerce

In today's competitive digital ecommerce landscape, retailers are fighting for the attention of shoppers who are tech-savvy, mobile, and receiving ads for hundreds of deals and promotions across multiple platforms every day. Today's consumers expect personalized digital shopping experiences that offer dynamic and relevant recommendations, too. Modern retail applications will be defined by their ability to drive better customer experiences, surface insights, and take action directly within the application on live operational data, in real time. As analytics "shift left" to the source of the data, application developers take on the task of delivering next generation data analytics to their organizations.

Application-driven analytics can be leveraged to meet two broad use cases: In-App Analytics and Real-Time Business Visibility. Real-time business visibility enables data sitting in operational systems and application-generated data to be combined in real time to make just-in-time strategic decisions. This data can also be leveraged by business intelligence tools (e.g., Tableau, PowerBI) to provide input for centralized analytics systems accessing data from Enterprise data warehouses. The major difference with Enterprise data warehouses is that data generated through applications often take several hops (and time) before making its way into a data warehouse before it is queried.

For instance, building an aggregated view of customers based on key metrics such as demographics, spending habits, and geography can help marketing teams assess the effectiveness of promotion campaigns, analyze their impact on revenues and warehouse in real-time (e.g. to terminate a campaign earlier if items become out of stock). Retailers can go a step further and predict the best time to run a flash sale by monitoring trends in the sales pipeline and spotting the main revenue streams for the business (e.g. online vs in-store, sub-brands, age groups).

12.5 Operational Intelligence

MongoDB is a NoSQL database that is commonly used in operational intelligence due to its ability to handle large and complex data sets. In this case study, we will explore how a company used MongoDB to improve their operational intelligence.

The company in question was a large retail chain with thousands of stores across the world. They needed to be able to track sales, inventory, and customer data in real-time to make informed decisions and stay ahead of the competition. They decided to implement a MongoDB-based operational intelligence system to handle this massive amount of data. The first step in this process was to set up a MongoDB cluster

with multiple nodes to ensure high availability and scalability. The company also used MongoDB's sharding capabilities to distribute data across multiple nodes and ensure that queries could be executed quickly and efficiently.

Once the cluster was set up, the company began collecting data from their point-of-sale systems, inventory management systems, and customer databases. They used MongoDB's document model to store this data, which allowed them to easily add new fields and data types as needed. They also took advantage of MongoDB's flexible schema to store data in a way that made sense for their specific use case. To analyze the data, the company used a combination of MongoDB's built-in aggregation framework and custom scripts written in Python. They were able to quickly generate reports and dashboards that showed sales trends, inventory levels, and customer demographics in real-time. This allowed them to make informed decisions about everything from inventory management to marketing campaigns.

The company also used MongoDB's full-text search capabilities to quickly find and analyze specific pieces of data. For example, they could quickly search for all transactions that involved a specific product or customer, allowing them to identify trends and make adjustments to their strategy as needed. Overall, the MongoDB-based operational intelligence system was a huge success for the retail chain. It allowed them to quickly and easily collect, store, and analyze massive amounts of data in real-time, giving them a competitive edge in a crowded market. The flexibility and scalability of MongoDB were key factors in the success of this project, and the company continues to rely on MongoDB for their operational intelligence needs today.

Advantages of MongoDB in operational intelligence:

- **High scalability:** MongoDB's sharding capabilities allow for horizontal scaling across multiple nodes, making it easy to handle large volumes of data and high query loads. This also ensures high availability of the system, which is critical in operational intelligence.
- **Real-time processing:** MongoDB's document model and built-in aggregation framework allow for real-time processing and analysis of data. This is crucial in operational intelligence, where decisions need to be made quickly based on the latest data.
- **Full-text search:** MongoDB's full-text search capabilities allow for quick and efficient searching of large data sets. This is useful in operational intelligence for finding specific pieces of data and identifying trends.
- **Developer-friendly:** MongoDB's JSON-based document model and support for multiple programming languages make it easy for developers to work with and integrate with existing systems. This allows for faster development and deployment of operational intelligence systems.
- **Cost-effective:** MongoDB's open-source license and ability to run on commodity hardware make it a cost-effective solution for operational intelligence. This allows companies to save on hardware and software costs while still getting the benefits of a powerful database system.

CONCLUSION

MongoDB is a powerful and versatile NoSQL database that is designed to handle large volumes of semi-structured and unstructured data. Its flexible data model, dynamic schema, and powerful aggregation framework make it an ideal choice for a wide range of applications, including real-time analytics, social media, and e-commerce. MongoDB's scalability and performance are second to none, making it an ideal database for big data analytics applications. Its ability to handle distributed data processing, sharding, and horizontal scaling, make it possible to handle large volumes of data and high traffic loads.

Moreover, MongoDB's ease of use, comprehensive documentation, and active developer community make it a popular choice for developers and businesses alike. Its support for multiple programming languages and platforms, including JavaScript, Python, and Java, makes it easy to integrate with existing technology stacks. MongoDB also provides a cloud-based solution called MongoDB Atlas, which offers an easy way to manage and scale MongoDB databases in the cloud. With MongoDB Atlas, businesses can easily deploy and manage their MongoDB databases on a global scale, with built-in security, scalability, and availability.

Overall, MongoDB is a powerful and flexible NoSQL database that is well-suited to a wide range of applications, including big data analytics, real-time analytics, social media, and e-commerce. Its scalability, performance, ease of use, and cloud-based solution make it a popular choice for businesses and developers alike. As data volumes continue to grow, MongoDB is likely to play an increasingly important role in modern data-driven applications.

REFERENCES

- [1] D. Ramesh, E. Khosla, and S. N. Bhukya, "Inclusion of e-commerce workflow with NoSQL DBMS: MongoDB document store," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, IEEE, Dec. 2016, pp. 1–5. doi: 10.1109/ICCIC.2016.7919652.
- [2] M. Eyada, W. Saber, M. El Genidy, and F. Amer, "Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments," *IEEE Access*, vol. 8, pp. 110656–110668, Apr. 2020.
- [3] K. S. M and T. Student, "A Study on Mongoddb Database," 2015. [Online]. Available: www.ijrsrd.com
- [4] K. Tabet, R. Mokadem, and M. R. Laouar, "Towards a New Data Replication Strategy in MongoDB Systems," in *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences*, New York, NY, USA: ACM, Jul. 2018, pp. 1–7. doi: 10.1145/3213187.3287609.
- [5] V. V Mhetre and N. R. Ranade, "Data Modeling Using MongoDB." [Online]. Available: www.asianssr.org
- [6] S. G. Edward and N. Sabharwal, "MongoDB Architecture," in *Practical MongoDB*, Berkeley, CA: Apress, 2015, pp. 95–157. doi: 10.1007/978-1-4842-0647-8_7.
- [7] A. Kanade and S. Kanade, "A Study of MongoDB Data Models and A Novel Hybrid Data Modeling Approach," in *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, IEEE, Jun. 2021, pp. 1554–1562. doi: 10.1109/ICOEI51242.2021.9452962.
- [8] Institute of Electrical and Electronics Engineers, *A Review on Various Aspects of MongoDB Databases*.
- [9] MongoDB. "MongoDB Documentation." www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/docs/manual/.
- [10] MongoDB. "MongoDB Architecture." www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/basics/database-architecture.
- [11] MongoDB. "Data Modeling Explained." www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/basics/data-modeling.
- [12] MongoDB. "MongoDB CMS." www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/scale/mongodb-cms.
- [13] Broadhead, Genevieve. "Three Ways Retailers Use Search beyond the Ecommerce Store." Www.mongodb.com, MongoDB, 29 Mar. 2023, www.mongodb.com/blog/post/three-ways-retailers-use-search-beyond-ecommerce-store.
- [14] MongoDB. "MongoDB Indexing." www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/docs/manual/indexes/.

- [15] MongoDB. “*MongoDB Aggrigation.*” www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/docs/manual/aggregation/.
- [16] O'Reilly Media, Inc, *MongoDB Applied Design Patterns*, Rick Copeland , Released March 2013, ISBN: 9781449340049
- [17] MongoDB: The Definitive Guide, Second Edition, by Kristina Chodorow, Copyright © 2013 Kristina Chodorow. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472
- [18] MongoDB. “*MongoDB v/s MYSQL.*” www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/compare/mongodb-mysql.
- [19] MongoDB. “*MongoDB v/s Cassandra.*” www.mongodb.com, MongoDB, 7 Aug. 2022, www.mongodb.com/compare/cassandra-vs-mongodb.