# MongoDB: An introduction

**MongoDB** stands out as a leading **NoSQL**database, offering an open-source, document-oriented approach that diverges from traditional relational databases. Unlike **SQL**databases, MongoDB stores data in BSON format, akin to JSON, allowing for more flexible data storage and retrieval. In this article, We will get a in **detailed** knowledge about **MongoDB.**

# What is MongoDB?

* **MongoDB** the most popular**NoSQL**database, is an**open-source document-oriented** database. The term ‘NoSQL’ means ‘**non-relational**‘.
* It means that MongoDB isn’t based on the **table-like relational database**structure but provides an altogether different mechanism for the **storage**and **retrieval**of data. This format of storage is called BSON ( similar to **JSON**format).

**A simple MongoDB document Structure:**

**{  
 title: 'mongodb',  
 by: 'Sandeep Singh',  
 url: 'http://sandeepmangodbtutorial.com',  
 type: 'NoSQL'  
}**

* SQL databases store data in tabular format. This data is stored in a predefined data model which is not very much flexible for today’s real-world highly growing applications.
* Modern applications are more networked, social and interactive than ever. Applications are storing more and more data and are accessing it at higher rates.
* Relational Database Management System(RDBMS) is not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable. If the database runs on a single server, then it will reach a scaling limit.
* NoSQL databases are more scalable and provide superior performance. MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

**RDBMS vs MongoDB**

* RDBMS has a typical schema design that shows number of tables and the relationship between these tables whereas MongoDB is document-oriented. There is no concept of schema or relationship.
* Complex transactions are not supported in MongoDB because complex join operations are not available.
* MongoDB allows a highly flexible and scalable document structure. For example, one data document of a collection in MongoDB can have two fields whereas the other document in the same collection can have four.
* MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques.
* There are a few terms that are related in both databases. What’s called Table in RDBMS is called a Collection in MongoDB. Similarly, a Row is called a Document and a Column is called a Field. MongoDB provides a default ‘\_id’ (if not provided explicitly) which is a 12-byte hexadecimal number that assures the uniqueness of every document. It is similar to the Primary key in RDBMS.

## **MongoDB database features**

* **Document Oriented:** MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk etc.
* **Indexing:** Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.
* **Scalability:** MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database.
* **Replication and High Availability:** MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.
* **Aggregation:** Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

## **Where do we use MongoDB?**

MongoDB is preferred over RDBMS in the following scenarios:

* **Big Data**: If we have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding our database.
* **Unstable Schema**: Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not effect old documents and will be very easy.
* **Distributed data** Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

## **Language Support by MongoDB**

MongoDB currently provides official driver support for all popular programming languages like C, C++, Rust, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go and Erlang.

## **Conclusion**

In conclusion, MongoDB’s document-oriented structure, scalable architecture through sharding, and robust features like indexing and aggregation make it a preferred choice for modern applications handling large volumes of data. With its ability to manage distributed data effectively and support a wide array of programming languages, MongoDB continues to empower developers to build scalable and efficient applications.

# DataTypes in MongoDB

**MongoDB** uses **BSON** (Binary **JSON**) to store documents and offers robust support for a diverse range of data types which enables **flexible**and efficient **data management**.

In this article, We will learn about the **data types in MongoDB**by understanding the various data types in detail.

## MongoDB data types

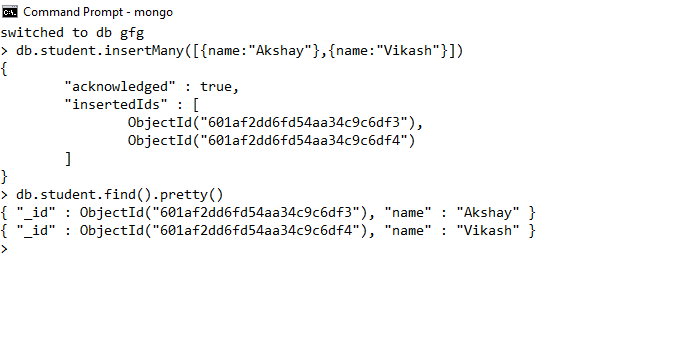
In **MongoDB**, the documents are stored in **BSON**, which is the binary encoded format of **JSON** and using BSON we can make remote procedure calls in MongoDB. BSON data format supports various data types.

Below are the enlisted MongoDB data types:

### **1. 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 data types to the string format of the language to**UTF-8**while serializing and de-serializing BSON. The string must be a valid UTF-8.

**Example:**In the following example we are storing the name of the student in the student collection:

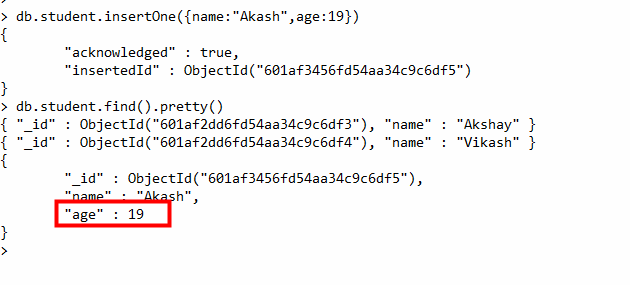


Here, the data type of the value of the name field is a string.

### **2. 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.

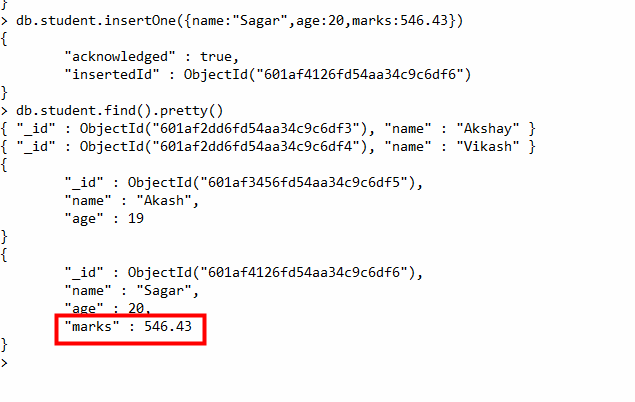
**Example:**In the following example we are storing the age of the student in the student collection:



### **3. Double**

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

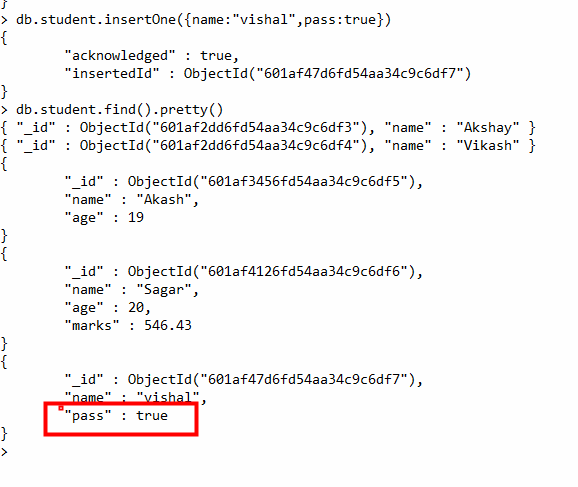
**Example:** In the following example we are storing the marks of the student in the student collection:



### **4. Boolean**

The boolean data type is used to store either true or false.

**Example:**In the following example we are storing the final result of the student as pass or fail in boolean values.



### **5. Null**

The null data type is used to store the null value.

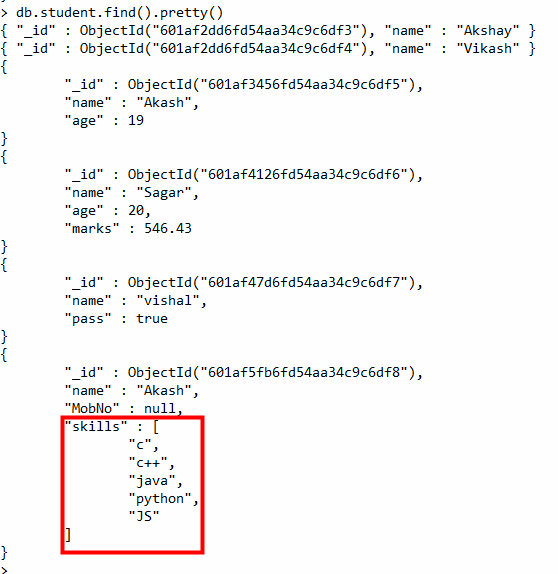
**Example:**In the following example,  the student does not have a mobile number so the number field contains the value null.



### **6. 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([]).

**Example:**In the following example, we are storing the technical skills of the student as an array.



### **7. 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.

**Example:** In the following example, we are storing all the information about a book in an embedded document.



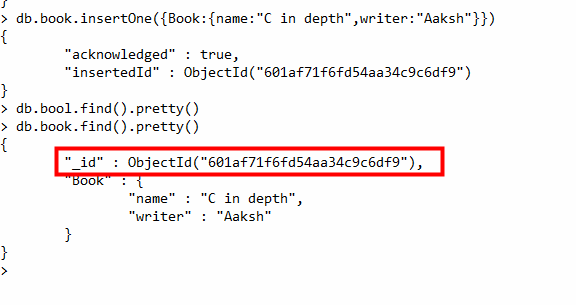
### **8. Object Id**

Whenever we create a new document in the collection MongoDB automatically creates a unique [object id](https://www.geeksforgeeks.org/what-is-objectid-in-mongodb/) 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.

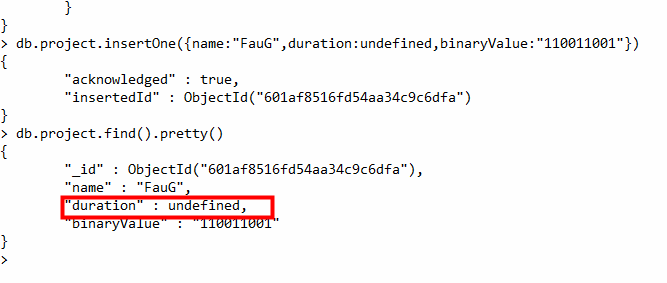
**Example:**In the following example, when we insert a new document it creates a new unique object id for it.



### **9. Undefined**

This data type stores the undefined values.

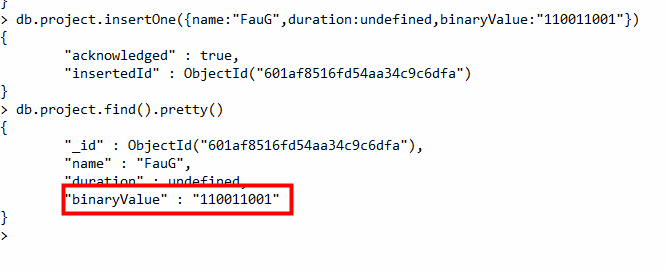
**Example:**In the following example the type of the duration of the project is undefined.



### **10. Binary Data**

This datatype is used to store binary data.

**Example:**In the following example the value stored in the binaryValue field is of binary type.

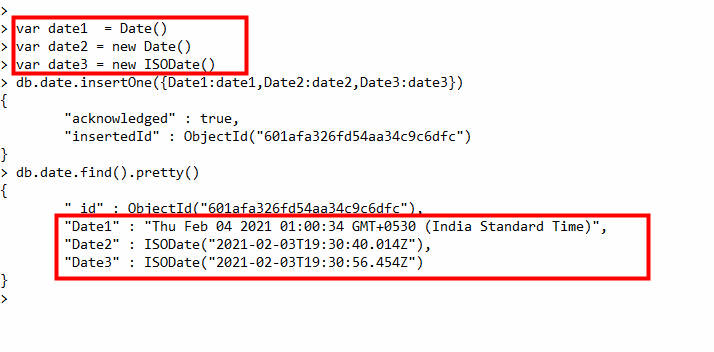


### **11. Date**

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.

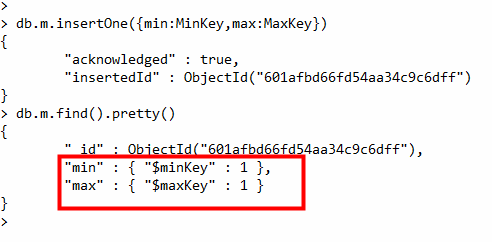
**Example:**In the following example we are using all the above method of the date:



### **12. Min & Max key**

Min key compares the value of the lowest BSON element and Max key compares the value against the highest BSON element. Both are internal data types.

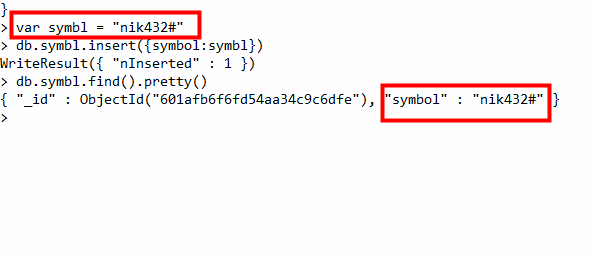
**Example:**



### **13. Symbol**

This data type similar to the string data type. It is generally not supported by a mongo shell, but if the shell gets a symbol from the database, then it converts this type into a string type.

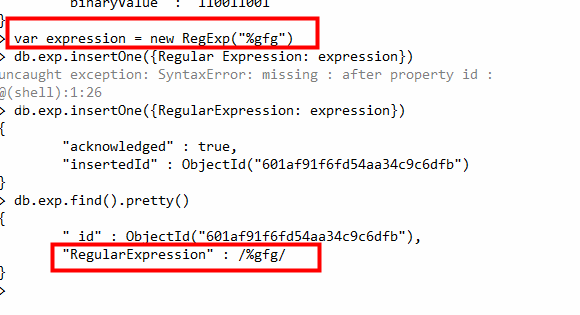
**Example:**



### **14. Regular Expression**

This datatype is used to store regular expressions.

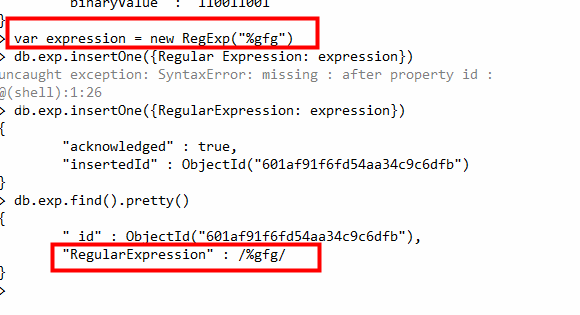
**Example:** In the following example we are storing the regular expression gfg:



### **15. JavaScript**

This datatype is used to store JavaScript code into the document without the scope.

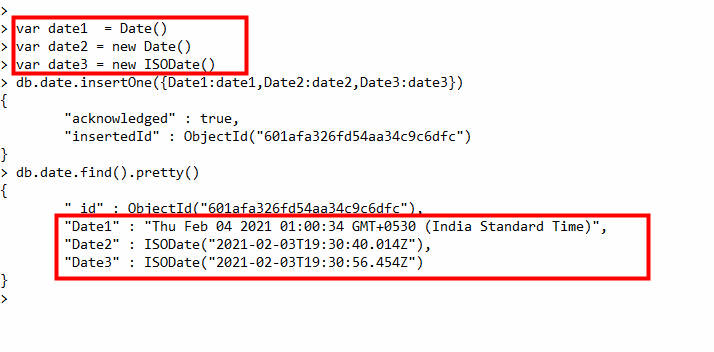
**Example:**In this example, we are using the JavaScript syntax in the shell:



### **16. JavaScript with Scope**

**T**his MongoDB data type store JavaScript data with a scope. This data type is deprecated in MongoDB 4.4.

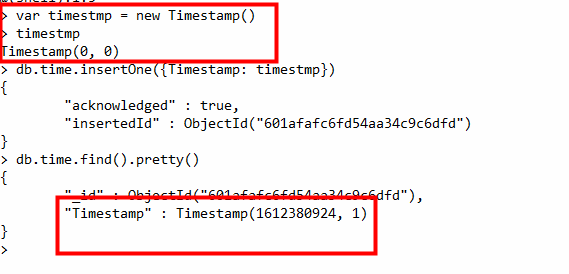
**Example:**In this example, we are using the **JavaScript**syntax in the shell:



### **17. 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.

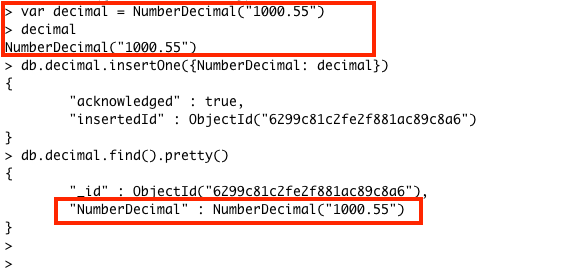
**Example:**



### **18. Decimal**

This MongoDB data type store 128-bit decimal-based floating-point value. This data type was introduced in MongoDB version 3.4

**Example:**



## **Conclusion**

# Administering Database Users

A database administrator, or DBA, is responsible for maintaining, securing, and operating databases and also ensures that data is correctly stored and retrieved.

In addition, DBAs often work with developers to design and implement new features and troubleshoot any issues. A DBA must have a strong understanding of both technical and business needs.

The role of DBA is becoming increasingly important in today’s information-driven business environment. Thoroughout the world, more and more organizations depend on data to discover analytical insights on market conditions, new business models, and cost-cutting measures. The global cloud computing market is also expected to expand as companies move their business operations to the cloud. Consequently, the need for qualified DBAs will only continue to grow.

The specific responsibilities of a database administrator vary depending on the size and needs of the organization they work for. However, most DBA duties will include developing and maintaining [databases](https://www.oracle.com/in/database/what-is-database/), ensuring data security, tuning performance, backing up data, and providing training and support to users. DBAs may also be responsible for designing databases and overseeing their construction in larger organizations.

## How has the role of a DBA evolved with cloud computing?

The role of a database administrator has evolved significantly with the advent of cloud computing. Rather than being responsible for managing on-premises hardware and software, DBAs now need to be able to work with cloud-based platforms. This requires a different set of skills and knowledge and a different approach to work.

DBAs need to be able to work with different types of databases, such as MySQL, MongoDB, and Cassandra. They also need to be familiar with cloud-based tools and platforms, such as Amazon Web Services (AWS) and Microsoft Azure.

One of the most significant changes is that DBAs are no longer responsible for managing the underlying infrastructure. With cloud computing, this is all managed by the provider. As a result, DBAs now perform more strategic tasks, such as data analytics, user experience design, and cybersecurity. DBAs often work directly with users and business leaders on developing new ways to use data and software to automate processes, reduce costs, and stay competitive.

This requires a new set of skills from DBAs. In the past, having strong technical skills was the most important requirement. There is less need for these skills with cloud computing. Instead, DBAs need to communicate and collaborate with users to understand their needs and business environment. They also need to work with other teams, such as DevOps, to help deliver software that will solve business problems.

Overall, the traditional role of a DBA is changing significantly thanks to cloud computing. DBAs need to be able to adapt to these changes to be successful in their roles.

## Understanding Access Control Lists

**Access Control Lists (ACLs) are a collection of permits and deny conditions, called rules, that provide security by blocking unauthorized users and allowing authorized users to access specific resources.**

ACLs can also provide traffic flow control, restrict contents of routing updates, and decide which types of traffic are forwarded or blocked. Normally ACLs reside in a firewall router or in a router connecting two internal networks.

You can set up ACLs to control traffic at Layer 2, Layer 3, or Layer 4. MAC ACLs operate on Layer 2. IP ACLs operate on Layers 3 and 4.

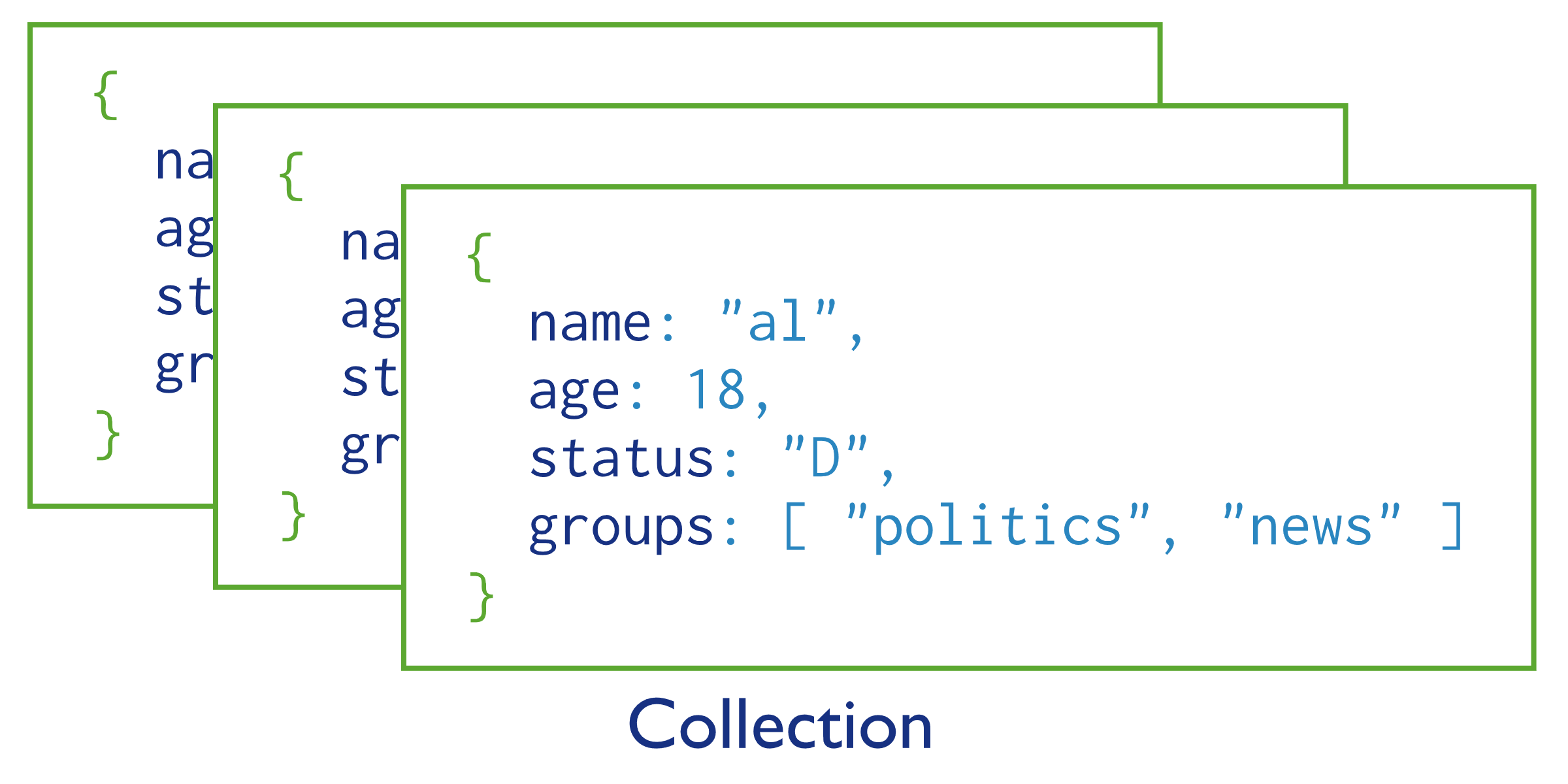
# Managing Databases and Collections in MongoDB

MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents.

You can manage databases and collections on the Atlas cluster from the Atlas UI, mongosh, or MongoDB Compass. This page describes how to manage databases and collections on the Atlas cluster from the Atlas UI. For self-managed deployments, you can use mongosh or MongoDB Compass to manage databases and collections. Select the client that you want to use to manage databases and collections.

MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment. To learn more, see MongoDB Compass.

## **Databases**

To select a database to use, complete the following steps:

**1** Start MongoDB Compass and connect to your cluster.To learn more, see Connect to MongoDB.

**2** Select *Databases* from the left navigation.

The **Databases** tab opens to list the existing databases for your MongoDB deployment.

### Create a Database

#### **Open the**Databases**tab.**

#### **Click the**Create database**button.**

#### **Enter database and first collection names in the**Create Database**dialog.**

#### **Click**Create Database**to create the database and its first collection.**

#### **Collections**

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

### Create a Collection

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

#### **Click the name of the database where you to want to create a collection in the left navigation.**

#### **Click the**+**icon next to the database name.**

#### **Enter the name of the collection in the**Create Collection**dialog.**

#### **Click**Create Collection**to create the collection.**

### Explicit Creation

#### **Click the name of the database where you to want to create a collection in the left navigation.**

#### **Click the**Create collection**button.**

#### **Enter the name of the collection and optionally, configure additional preferences.**

#### **Click**Create Collection**to create the collection.**

MongoDB Compass provides the following additional preferences that you can configure for your collection:

1. **Create a Capped Collection**
2. **Create a Clustered Collection**
3. **Create a Collection with Collation**
4. **Create a Collection with Encrypted Field**
5. **Create a Time Series Collection**

### Document Validation

By default, a collection does not require its documents to have the same schema; i.e. 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.

## **Databases**

**In MongoDB, databases hold one or more collections of documents.**

**To select a database to use, in [mongosh](https://www.mongodb.com/docs/mongodb-shell/" \l "mongodb-binary-bin.mongosh" \t "_self), issue the use <db> statement, as in the following example:**

|  |
| --- |
| **use myDB** |

**Create a 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 operation in [mongosh:](https://www.mongodb.com/docs/mongodb-shell/" \l "mongodb-binary-bin.mongosh" \t "_self)**

|  |
| --- |
| **use myNewDB** |
|  |
| **db.myNewCollection1.insertOne( { x: 1 } )** |

The [insertOne()](https://www.mongodb.com/docs/upcoming/reference/method/db.collection.insertOne/" \l "mongodb-method-db.collection.insertOne) operation creates both the database myNewDB and the collection myNewCollection1 if they do not already exist. Be sure that both the database and collection names follow MongoDB [Naming Restrictions.](https://www.mongodb.com/docs/upcoming/reference/limits/#std-label-restrictions-on-db-names)

## **Collections**

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

### **Create a Collection**

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()](https://www.mongodb.com/docs/upcoming/reference/method/db.collection.insertOne/" \l "mongodb-method-db.collection.insertOne) and the [createIndex()](https://www.mongodb.com/docs/upcoming/reference/method/db.collection.createIndex/" \l "mongodb-method-db.collection.createIndex) operations create their respective collection if they do not already exist. Be sure that the collection name follows MongoDB [Naming Restrictions.](https://www.mongodb.com/docs/upcoming/reference/limits/#std-label-restrictions-on-db-names)

### **Explicit Creation**

MongoDB provides the [db.createCollection()](https://www.mongodb.com/docs/upcoming/reference/method/db.createCollection/" \l "mongodb-method-db.createCollection) method to explicitly create a collection with various options, such as setting the maximum size or the documentation validation rules. If you are not specifying these options, you do not need to explicitly create the collection since MongoDB creates new collections when you first store data for the collections.

**Mongodb connect to node js , (Crud operations.)**

const { MongoClient } = require('mongodb');

// or as an es module:

// import { MongoClient } from 'mongodb'

// Connection URL

const url = 'mongodb://localhost:27017';

const client = new MongoClient(url);

// Database Name

const dbName = 'myProject';

async function main() {

// Use connect method to connect to the server

await client.connect();

console.log('Connected successfully to server');

const db = client.db(dbName);

const collection = db.collection('documents');

// the following code examples can be pasted here...

return 'done.';

}

main()

.then(console.log)

.catch(console.error)

.finally(() => client.close());

he **insertMany** command returns an object with information about the insert operations.

### **Find All Documents**

Add a query that returns all the documents.

const findResult = await collection.find({}).toArray();

console.log('Found documents =>', findResult);

This query returns all the documents in the **documents** collection. If you add this below the insertMany example, you'll see the documents you've inserted.

### **Find Documents with a Query Filter**

Add a query filter to find only documents which meet the query criteria.

const filteredDocs = await collection.find({ a: 3 }).toArray();

console.log('Found documents filtered by { a: 3 } =>', filteredDocs);

Only the documents which match 'a' : 3 should be returned.

### **Update a document**

The following operation updates a document in the **documents** collection.

const updateResult = await collection.updateOne({ a: 3 }, { $set: { b: 1 } });

console.log('Updated documents =>', updateResult);

The method updates the first document where the field **a** is equal to **3** by adding a new field **b** to the document set to **1**. updateResult contains information about whether there was a matching document to update or not.

### Remove a document

Remove the document where the field **a** is equal to **3**.

const deleteResult = await collection.deleteMany({ a: 3 });

console.log('Deleted documents =>', deleteResult);

### **Index a Collection**

[**Indexes**](https://www.mongodb.com/docs/manual/indexes/) can improve your application's performance. The following function creates an index on the **a** field in the **documents** collection.

const indexName = await collection.createIndex({ a: 1 });

console.log('index name =', indexName);