MongoDB

MongoDB tutorial provides basic and advanced concepts of SQL. Our MongoDB tutorial is designed for beginners and professionals.

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++.

What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.

The manufacturing company 10gen has defined MongoDB as:

"MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen

MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

History of MongoDB

The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.

Window azure is a cloud computing platform and infrastructure, created by Microsoft, to build, deploy and manage applications and service through a global network.

MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.

The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010.

MongoDB2.4.9 was the latest and stable version which was released on January 10, 2014.

Purpose of Building MongoDB

The primary purpose of building MongoDB is:

- Scalability
- Performance
- High Availability
- Scaling from single server deployments to large, complex multi-site architectures.
- Key points of MongoDB
- Develop Faster
- Deploy Easier
- Scale Bigger

Example of Document-Oriented Database

MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.

MongoDB stores data as documents, so it is known as document-oriented database.

FirstName = "John", Address = "Detroit",
Spouse = [{**Name**: "Angela"}].
FirstName ="John",
Address = "Wick"

There are two different documents (separated by ".").

Storing data in this manner is called as document-oriented database.

Mongo DB falls into a class of databases that calls Document Oriented Databases. There is also a broad category of database known as No SQL Databases.

Features of MongoDB

These are some important features of MongoDB:

1. Support ad hoc queries

In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing

You can index any field in a document.

3. Replication

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing

It has an automatic load balancing configuration because of data placed in shards.

6. Supports map reduce and aggregation tools.

7. Uses JavaScript instead of Procedures.

8. It is a schema-less database written in C++.

9. Provides high performance.

10. Stores files of any size easily without complicating your stack.

11. Easy to administer in the case of failures.

12. It also supports:

- JSON data model with dynamic schemas
- Auto-sharding for horizontal scalability
- Built in replication for high availability
- Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

NoSQL Databases

We know that MongoDB is a NoSQL Database, so it is very necessary to know about NoSQL Database to understand MongoDB throughly.

What is NoSQL Database

Databases can be divided in 3 types:

1. RDBMS (Relational Database Management System)
2. OLAP (Online Analytical Processing)
3. NoSQL (recently developed database)

NoSQL Database

NoSQL Database is used to refer a non-SQL or non relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

History behind the creation of NoSQL Databases

In the early 1970, Flat File Systems are used. Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards. It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.

Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data. But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed.

Advantages of NoSQL

- It supports query language.
- It provides fast performance.
- It provides horizontal scalability.

MongoDB advantages over RDBMS

In recent days, MongoDB is a new and popularly used database. It is a document based, non relational database provider.

Although it is 100 times faster than the traditional database but it is early to say that it will broadly replace the traditional RDBMS. But it may be very useful in term to gain performance and scalability.

A Relational database has a typical schema design that shows number of tables and the relationship between these tables, while in MongoDB there is no concept of relationship.

MongoDB Advantages

- **MongoDB is schema less**. It is a document database in which one collection holds different documents.
- There may be **difference between number of fields, content and size of the document** from one to other.
- **Structure of a single object is clear** in MongoDB.
- There are **no complex joins** in MongoDB.
- MongoDB provides the **facility of deep query** because it supports a powerful dynamic query on documents.
- It is very **easy to scale**.
- It **uses internal memory for storing working sets** and this is the reason of its fast access.

Distinctive features of MongoDB

- Easy to use
- Light Weight

- Extremely faster than RDBMS

Where MongoDB should be used

- Big and complex data
- Mobile and social infrastructure
- Content management and delivery
- User data management
- Data hub

Performance analysis of MongoDB and RDBMS

- In relational database (RDBMS) tables are using as storing elements, while in MongoDB collection is used.
- In the RDBMS, we have multiple schema and in each schema we create tables to store data while, MongoDB is a document oriented database in which data is written in BSON format which is a JSON like format.
- MongoDB is almost 100 times faster than traditional database systems.

MongoDB Datatypes

- Following is a list of usable data types in MongoDB.

Data Types	Description
String	String is the most commonly used datatype. It is used to store data. A string must be UTF 8 valid in mongodb.
Integer	Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using.
Boolean	This datatype is used to store boolean values. It just shows YES/NO values.
Double	Double datatype stores floating point values.
Min/Max Keys	This datatype compare a value against the lowest and highest bson elements.

Arrays	This datatype is used to store a list or multiple values into a single key.
Object	Object datatype is used for embedded documents.
Null	It is used to store null values.
Symbol	It is generally used for languages that use a specific type.
Date	This datatype stores the current date or time in unix time format. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it.

To install MongoDB on Windows 11, follow these steps:

1. Download MongoDB

- **Visit the MongoDB official download page.**
- **Select the appropriate version (latest stable release).**
- **Choose the MSI installer for Windows.**
- **Click Download.**

2. Install MongoDB

- **Once the download is complete, run the MSI file.**
- **The MongoDB installation wizard will open.**

Installation Process:

- **Welcome Screen: Click Next.**
- **End-User License Agreement: Accept the terms and click Next.**
- **Setup Type:**
  - **Choose Complete for a full installation.**
- **Service Configuration:**
  - **Make sure the option Install MongoDB as a Service is selected.**
  - **Leave the default settings for the service settings (port 27017, run as Network Service User).**
- **Install MongoDB Compass: You can also install MongoDB Compass (GUI for MongoDB), or skip it if you prefer.**
- **Click Install to begin the installation.**

3. Set Up MongoDB Environment Variables

MongoDB binaries are installed at C:\Program Files\MongoDB\Server\<version>\bin by default. To use mongo or mongod from the command prompt, you need to add the MongoDB bin directory to your system's PATH variable.

Steps:

1. Right-click on the **Start** button and select **System**.
2. On the **System** page, click **Advanced system settings** on the right.
3. In the **System Properties** window, click the **Environment Variables** button.
4. Under **System Variables**, scroll down and select **Path**, then click **Edit**.
5. Click **New** and add the following path (adjust if installed elsewhere:)
6. Click **OK** to close all windows.

Data Modeling in MongoDB

In MongoDB, data has a flexible schema. It is totally different from SQL database where you had to determine and declare a table's schema before inserting data. MongoDB collections do not enforce document structure.

The main challenge in data modeling is balancing the need of the application, the performance characteristics of the database engine, and the data retrieval patterns.

Consider the following things while designing the schema in MongoDB

- Always design schema according to user requirements.
- Do join on write operations not on read operations.
- Objects which you want to use together, should be combined into one document. Otherwise they should be separated (make sure that there should not be need of joins).
- Optimize your schema for more frequent use cases.
- Do complex aggregation in the schema.
- You should duplicate the data but in a limit, because disc space is cheaper than compute time.

For example:

let us take an example of a client who needs a database design for his website. His website has the following requirements:

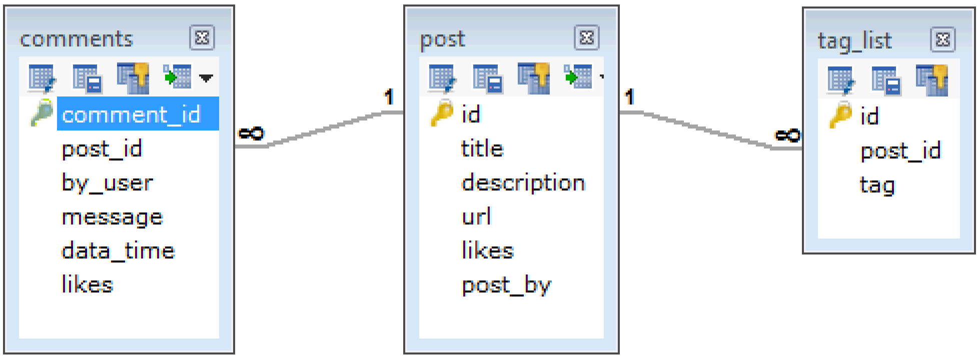
Every post is distinct (contains unique title, description and url).

Every post can have one or more tags.

Every post has the name of its publisher and total number of likes.

Each post can have zero or more comments and the comments must contain user name, message, data-time and likes.

For the above requirement, a minimum of three tables are required in RDBMS.



But in MongoDB, schema design will have one collection post and has the following structure:

```
{
_id: POST_ID
title: TITLE_OF_POST,
description: POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
comments: [
{
user: 'COMMENT_BY',
message: TEXT,
datecreated: DATE_TIME,
like: LIKES
},
{
user: 'COMMENT_BY',
message: TEST,
dateCreated: DATE_TIME,
like: LIKES
}}}
```

MongoDB Create Database

Use Database method:

There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.

It may be look like a weird concept, if you are from traditional SQL background where you need to create a database, table and insert values in the table manually.

Here, in MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.

You also don't need to mention what you want to create, it will be automatically created at the time you save the value into the defined collection.

How and when to create database

If there is no existing database, the following command is used to create a new database.

Syntax:

```
use DATABASE_NAME
```

If the database already exists, it will return the existing database.

Let' take an example to demonstrate how a database is created in MongoDB. In the following example, we are going to create a database "javatpointdb".

See this example

```
>use javatpointdb
Swithched to db javatpointdb
```

To **check the currently selected database**, use the command db:

```
>db
javatpointdb
```

To **check the database list**, use the command show dbs:

```
>show dbs
local 0.078GB
```

Here, your created database "javatpointdb" is not present in the list, **insert at least one document** into it to display database:

```
>db.movie.insert({"name":"javatpoint"})
WriteResult({ "nInserted": 1})
>show dbs
javatpointdb 0.078GB
local 0.078GB
```

MongoDB Drop Database

The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.

Syntax:

```
db.dropDatabase()
```

This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

To **check the database list**, use the command show dbs:

```
>show dbs
javatpointdb 0.078GB
local 0.078GB
```

If you want to **delete the database "javatpointdb"**, use the dropDatabase() command as follows:

```
>use javatpointdb
switched to the db javatpointdb
>db.dropDatabase()
{ "dropped": "javatpointdb", "ok": 1}
```

Now check the list of databases:

```
>show dbs
local 0.078GB
```

MongoDB Create Collection

In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don?t need to create collection. MongoDB creates collection automatically when you insert some documents. It will be explained later. First see how to create collection:

Syntax:

```
db.createCollection(name, options)
```

Here,

Name: is a string type, specifies the name of the collection to be created.

Options: is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

Following is the list of options that can be used.

Field	Type	Description
Capped	Boolean	(Optional) If it is set to true, enables a capped collection. Capped collection is a fixed size colle that automatically overwrites its oldest entries when it reaches its maximum size. If you specif you need to specify size parameter also.
AutoIndexID	Boolean	(Optional) If it is set to true, automatically create index on ID field. Its default value is false.
Size	Number	(Optional) It specifies a maximum size in bytes for a capped collection. Ifcapped is true, the need to specify this field also.
Max	Number	(Optional) It specifies the maximum number of documents allowed in the capped collection.

Let's take an **example to create collection**. In this example, we are going to create a collection name SSSIT.

```
>use test
switched to db test
```

```
>db.createCollection("SSSIT")
{ "ok" : 1 }
```

To **check the created collection**, use the command "show collections".

```
>show collections
```

```
SSSIT
```

How does MongoDB create collection automatically

MongoDB creates collections automatically when you insert some documents. For example: Insert a document named seomount into a collection named SSSIT. The operation will create the collection if the collection does not currently exist.

```
>db.SSSIT.insert({"name" : "seomount"})
>show collections
SSSIT
```

If you want to see the inserted document, use the find() command.

Syntax:

```
db.collection_name.find()
```

MongoDB Drop collection

In MongoDB, db.collection.drop() method is used to drop a collection from a database. It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.

The db.collection.drop() method does not take any argument and produce an error when it is called with an argument. This method removes all the indexes associated with the dropped collection.

Syntax:

```
db.COLLECTION_NAME.drop()
```

MongoDB Drop collection example

Let's take an example to drop collection in MongoDB.

Backward Skip 10sPlay VideoForward Skip 10s

First **check the already existing collections** in your database.

```
>use mydb
```

```
Switched to db mydb
```

```
> show collections
```

```
SSSIT
system.indexes
```

Note: Here we have a collection named SSSIT in our database.

Now **drop the collection** with the name SSSIT:

```
>db.SSSIT.drop()
```

```
True
```

Now **check the collections** in the database:

```
>show collections
```

```
System.indexes
```

Now, there are no existing collections in your database.

MongoDB insert documents

In MongoDB, the **db.collection.insert()** method is used to add or insert new documents into a collection in your database.

Upsert

There are also two methods "db.collection.update()" method and "db.collection.save()" method used for the same purpose. These methods add new documents through an operation called upsert.

Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

Syntax

```
>db.COLLECTION_NAME.insert(document)
```

Let?s take an example to demonstrate how to insert a document into a collection. In this example we insert a document into a collection named javatpoint. This operation will automatically create a collection if the collection does not currently exist.

Example

```
db.javatpoint.insert(
    {
      course: "java",
     details: {
        duration: "6 months",
       Trainer: "Sonoo jaiswal"
     },
     Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
      category: "Programming language"
    }
  )
```

After the successful insertion of the document, the operation will return a WriteResult object with its status.

Output:

```
WriteResult({ "nInserted" : 1 })
```

Here the **nInserted** field specifies the number of documents inserted. If an error is occurred then the **WriteResult** will specify the error information.

Check the inserted documents

If the insertion is successful, you can view the inserted document by the following query.

```
>db.javatpoint.find()
```

You will get the inserted document in return.

Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "java", "details" :
{ "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :
[ {"size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],
 "category" : "Programming language" }
```

Note: Here, the ObjectId value is generated by MongoDB itself. It may differ from the one shown.

To check the data in proper format

```
>db.javatpoint.find().pretty()
```

MongoDB insert multiple documents

If you want to insert multiple documents in a collection, you have to pass an array of documents to the db.collection.insert() method.

Create an array of documents

Define a variable named Allcourses that hold an array of documents to insert.

```
var Allcourses =
   [
     {
      Course: "Java",
       details: { Duration: "6 months", Trainer: "Sonoo Jaiswal" },
      Batch: [ { size: "Medium", qty: 25 } ],
       category: "Programming Language"
    },
     {
      Course: ".Net",
       details: { Duration: "6 months", Trainer: "Prashant Verma" },
      Batch: [ { size: "Small", qty: 5 }, { size: "Medium", qty: 10 }, ],
       category: "Programming Language"
    },
     {
      Course: "Web Designing",
       details: { Duration: "3 months", Trainer: "Rashmi Desai" },
      Batch: [ { size: "Small", qty: 5 }, { size: "Large", qty: 10 } ],
       category: "Programming Language"
    }
   ];
```

Inserts the documents

Pass this Allcourses array to the db.collection.insert() method to perform a bulk insert.

```
> db.javatpoint.insert( Allcourses );
```

After the successful insertion of the documents, this will return a BulkWriteResult object with the status.

```
BulkWriteResult({
   "writeErrors" : [ ],
   "writeConcernErrors" : [ ],
   "nInserted" : 3,
   "nUpserted" : 0,
   "nMatched" : 0,
   "nModified" : 0,
   "nRemoved" : 0,
   "upserted" : [ ]
})
```

Note: Here the nInserted field specifies the number of documents inserted. In the case of any error during the operation, the **BulkWriteResult** will specify that error.

You can check the inserted documents by using the following query:

```
>db.javatpoint.find()
```

MongoDB update documents

In MongoDB, update() method is used to update or modify the existing documents of a collection.

Syntax:

```
db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

Example

Consider an example which has a collection name javatpoint.

Insert the following documents in collection:

```
db.javatpoint.insert(
  {
    course: "java",
    details: {
      duration: "6 months",
      Trainer: "Sonoo jaiswal"
    },
    Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
    category: "Programming language"
  }
)
```

After successful insertion, check the documents by following query:

```
>db.javatpoint.find()
```

Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "java", "details" :
{ "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :
[ {"size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],
 "category" : "Programming language" }
```

Update the existing course "java" into "android":

```
>db.javatpoint.update({'course':'java'},{$set:{'course':'android'}})
```

Check the updated document in the collection:

```
>db.javatpoint.find()
```

Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "android", "details" :
{ "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :
[ {"size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],
 "category" : "Programming language" }
```

db.student.insertMany([{ subject: "Mathematics", age: 18 }, { subject: "Physics", age: 20 }, { subject: "Chemistry", age: 19 }, { subject: "Biology", age: 21 }, { subject: "Computer Science", age: 22 }]);

MongoDB Delete documents

In MongoDB, you can delete documents from a collection using the `deleteOne()` or `deleteMany()` methods, depending on whether you want to delete a single document or multiple documents that match a specified criteria.

Deleting a Single Document

To delete a single document that matches a specific condition, you can use the `deleteOne()` method:

```
Syntax:

db.collection('yourCollection').deleteOne(
    { 'field': 'value' }
);
```

- Replace `'yourCollection'` with the name of your collection.
- `{ 'field': 'value' }` specifies the condition (query) to match the document(s) you want to delete. Only the first matching document found will be deleted.

Example:

```
db.students.deleteOne(
    { 'name': 'Alice' }
);
```

This command deletes the first document in the `students` collection where the `name` field is equal to `'Alice'`.

Deleting Multiple Documents

To delete multiple documents that match a specific condition, you can use the `deleteMany()` method:

```
db.collection('yourCollection').deleteMany({ 'field': 'value' });
```

- Replace `'yourCollection'` with the name of your collection.
- `{ 'field': 'value' }` specifies the condition (query) to match the documents you want to delete. All documents that match this query will be deleted.

Delete Multiple Documents

```
db.students.deleteMany(
    { 'age': { $gte: 18 } }
);
```

This command deletes all documents in the `students` collection where the `age` field is greater than or equal to `18`.

Notes:

- **Query Condition**: Ensure the query condition matches the documents you want to delete. If no documents match the condition, no deletion occurs.
- **Safety**: Deleting documents is a permanent operation. Ensure you have appropriate permissions and carefully review your query conditions before executing the delete operation.
- **Efficiency**: Use `deleteMany()` when you need to delete multiple documents efficiently based on a common criteria.

These methods provide flexibility in deleting documents from MongoDB collections based on specific criteria, helping you manage your data effectively.

Remove documents

In MongoDB, the db.colloction.remove() method is used to delete documents from a collection. The remove() method works on two parameters.

1. Deletion criteria: With the use of its syntax you can remove the documents from the collection.

2. JustOne: It removes only one document when set to true or 1.

Syntax:

```
db.collection_name.remove (DELETION_CRITERIA)
```

Remove all documents

If you want to remove all documents from a collection, pass an empty query document {} to the remove() method. The remove() method does not remove the indexes.

Let's take an example to demonstrate the remove() method. In this example, we remove all documents from the "javatpoint" collection.

```
db.javatpoint.remove({})
```

Remove all documents that match a condition

If you want to remove a document that match a specific condition, call the remove() method with the <query> parameter.

The following example will remove all documents from the javatpoint collection where the type field is equal to programming language.

```
db.javatpoint.remove( { type : "programming language" } )
```

Remove a single document that match a condition

If you want to remove a single document that match a specific condition, call the remove() method with justOne parameter set to true or 1.

The following example will remove a single document from the javatpoint collection where the type field is equal to programming language.

```
db.javatpoint.remove( { type : "programming language" }, 1 )
```

Difference between delete document and remove document.

In MongoDB, the terms "delete document" and "remove document" can often be used interchangeably, but there are some differences based on the specific methods used:

`deleteOne()` and `deleteMany()`

- **Purpose**: These methods are used to delete documents from a collection.
- **Usage**:
  - `deleteOne(filter)`: Deletes a single document (first record) that matches the given filter criteria.
  - `deleteMany(filter)`: Deletes all documents that match the given filter criteria.
- **Return Value**: Both methods return a `DeleteResult` object, which includes the number of documents deleted.
- **Example**:

```
db.collection.deleteOne({ name: "John" });
db.collection.deleteMany({ age: { $lt: 18 } });
```

`remove()`

- **Purpose**: This is an older method that was used to remove documents from a collection.
- **Usage**:
  - `remove(filter, options)`: Removes documents that match the filter criteria. The `options` parameter can include `{ justOne: true }` to remove only one document.
- **Return Value**: The `remove` method returns a document containing the status of the operation, including the number of documents removed.
- **Example**:

```
db.collection.remove({ name: "John" });
db.collection.remove({ age: { $lt: 18 } }, { justOne: true });
```

Key Differences

1. **Modern vs. Deprecated**:
   - `deleteOne()` and `deleteMany()` are the modern methods recommended for use in MongoDB.
   - `remove()` is considered deprecated in favor of `deleteOne()` and `deleteMany()`.
2. **Functionality**:
   - `deleteOne()` explicitly removes only one document.
   - `deleteMany()` explicitly removes multiple documents.
   - `remove()` can remove one or multiple documents based on the `justOne` option.
3. **Return Values**:
   - `deleteOne()` and `deleteMany()` provide a `DeleteResult` object with detailed information.
   - `remove()` provides a less detailed document about the operation's status.

For new projects, it is recommended to use `deleteOne()` and `deleteMany()` to align with current MongoDB best practices and ensure future compatibility.

MongoDB Query documents

In MongoDB, the **db.collection.find()** method is used to retrieve documents from a collection. This method returns a cursor to the retrieved documents.

The db.collection.find() method reads operations in mongoDB shell and retrieves documents containing all their fields.

Syntax:

```
db.COLLECTION_NAME.find({})
```

Select all documents in a collection:

To retrieve all documents from a collection, put the query document ({}) empty. It will be like this:

```
db.COLLECTION_NAME.find()
```

For example: If you have a collection name "canteen" in your database which has some fields like foods, snacks, beverages, price etc. then you should use the following query to select all documents in the collection "canteen".

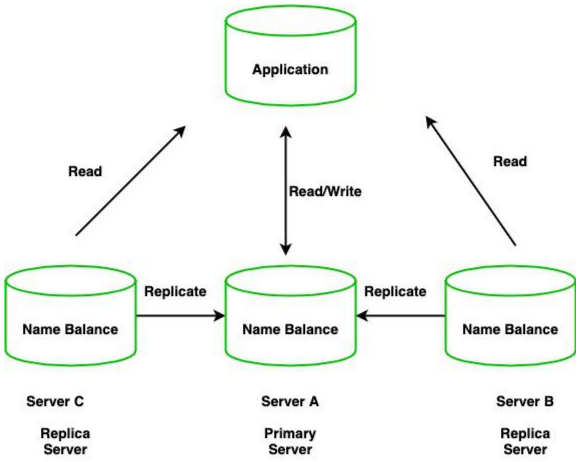
```
db.canteen.find()
```

MongoDB – Replication and Sharding

Replication and **Sharding** are two important features for **scalability and data availability** in MongoDB. Replication enhances data availability by creating duplicate copies of the dataset, whereas sharding helps in horizontal scaling by partitioning the large collection (dataset) into smaller discrete parts called **shards**.

Replication in MongoDB

Replication is the method of duplication of data across multiple servers in **MongoDB**.
For example, we have an application that reads and writes data to a database and says server A has a name and balance which will be copied/replicated to two other servers in two different locations.



Replication **increases redundancy** and **data availability** with multiple copies of data on different database servers. So, it will increase the performance of reading scaling.

The set of servers that maintain the same copy of data is known as **replica servers** or **MongoDB instances**.

Key Features of Replication:

- Replica sets are the clusters of N different nodes that maintain the same copy of the data set.
- The primary server receives all write operations and record all the changes to the data i.e, oplog.
- The secondary members then copy and apply these changes in an asynchronous process.
- All the secondary nodes are connected with the primary nodes. there is one heartbeat signal from the primary nodes. If the primary server goes down an eligible secondary will hold the new primary.

Advantages of Replication

- High Availability of data disasters recovery
- No downtime for maintenance (like backups index rebuilds and compaction)
- Read Scaling (Extra copies to read from)

To create a replica set in MongoDB using the MongoDB shell, follow these steps:

1. **Start `mongod` instances with the `--replSet` option**.
2. **Connect to one of the `mongod` instances using the MongoDB shell**.
3. **Initiate the replica set configuration**.
4. **Add members to the replica set**.

Step-by-Step Guide

Step 1: Start `mongod` Instances with Replica Set Configuration

On each server, start a `mongod` instance specifying the replica set name (e.g., `rs0`):

On Server 1:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db1 --port 27017 --bind_ip localhost,192.168.1.1
```

On Server 2:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db2 --port 27017 --bind_ip localhost,192.168.1.2
```

On Server 3:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db3 --port 27017 --bind_ip localhost,192.168.1.3
```

Step 2: Connect to One of the `mongod` Instances

Connect to the first `mongod` instance using the MongoDB shell:

```
sh
Copy code
mongo --host 192.168.1.1 --port 27017
```

Step 3: Initiate the Replica Set

In the MongoDB shell, run the following command to initiate the replica set:

```
javascript
Copy code
rs.initiate(
  {
    _id: "rs0",
    members: [
      { _id: 0, host: "192.168.1.1:27017" },
      { _id: 1, host: "192.168.1.2:27017" },
      { _id: 2, host: "192.168.1.3:27017" }
    ]
  }
)
```

Step 4: Verify the Replica Set Status

To check the status of the replica set, run:

```
javascript
Copy code
rs.status()
```

Full Example from Command Line

1. **Start `mongod` Instances**:

On Server 1:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db1 --port 27017 --bind_ip localhost,192.168.1.1
```

On Server 2:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db2 --port 27017 --bind_ip localhost,192.168.1.2
```

On Server 3:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db3 --port 27017 --bind_ip localhost,192.168.1.3
```

2. **Connect to the Primary MongoDB Instance**:

```
sh
Copy code
mongo --host 192.168.1.1 --port 27017
```

3. **Initiate the Replica Set**:

```
javascript
Copy code
rs.initiate(
  {
    _id: "rs0",
    members: [
      { _id: 0, host: "192.168.1.1:27017" },
      { _id: 1, host: "192.168.1.2:27017" },
      { _id: 2, host: "192.168.1.3:27017" }
    ]
  }
)
```

4. **Verify the Replica Set Status**:

```
javascript
Copy code
rs.status()
```

This setup will create a MongoDB replica set named `rs0` with three members. The `rs.initiate()` command initializes the replica set, and the `rs.status()` command verifies the status of the replica set members.

4o

Start mongod instances with the --replSet option. how to do this

ChatGPT

To start `mongod` instances with the `--replSet` option, you need to run the `mongod` command with appropriate parameters on each server that will be part of the replica set. Here is how you can do it:

Step-by-Step Instructions

1. Prepare Directories for Data Storage

Ensure you have directories set up for data storage on each server. For example, you can use `/data/db1`, `/data/db2`, and `/data/db3` for three different instances.

2. Start Each `mongod` Instance with the `--replSet` Option

Run the following commands on each server where you want to run the `mongod` instance. Replace `<ip_address>` with the actual IP address of each server.

On Server 1:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db1 --port 27017 --bind_ip
localhost,<ip_address_1>
```

On Server 2:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db2 --port 27017 --bind_ip
localhost,<ip_address_2>
```

On Server 3:

```
sh
Copy code
mongod --replSet rs0 --dbpath /data/db3 --port 27017 --bind_ip
localhost,<ip_address_3>
```

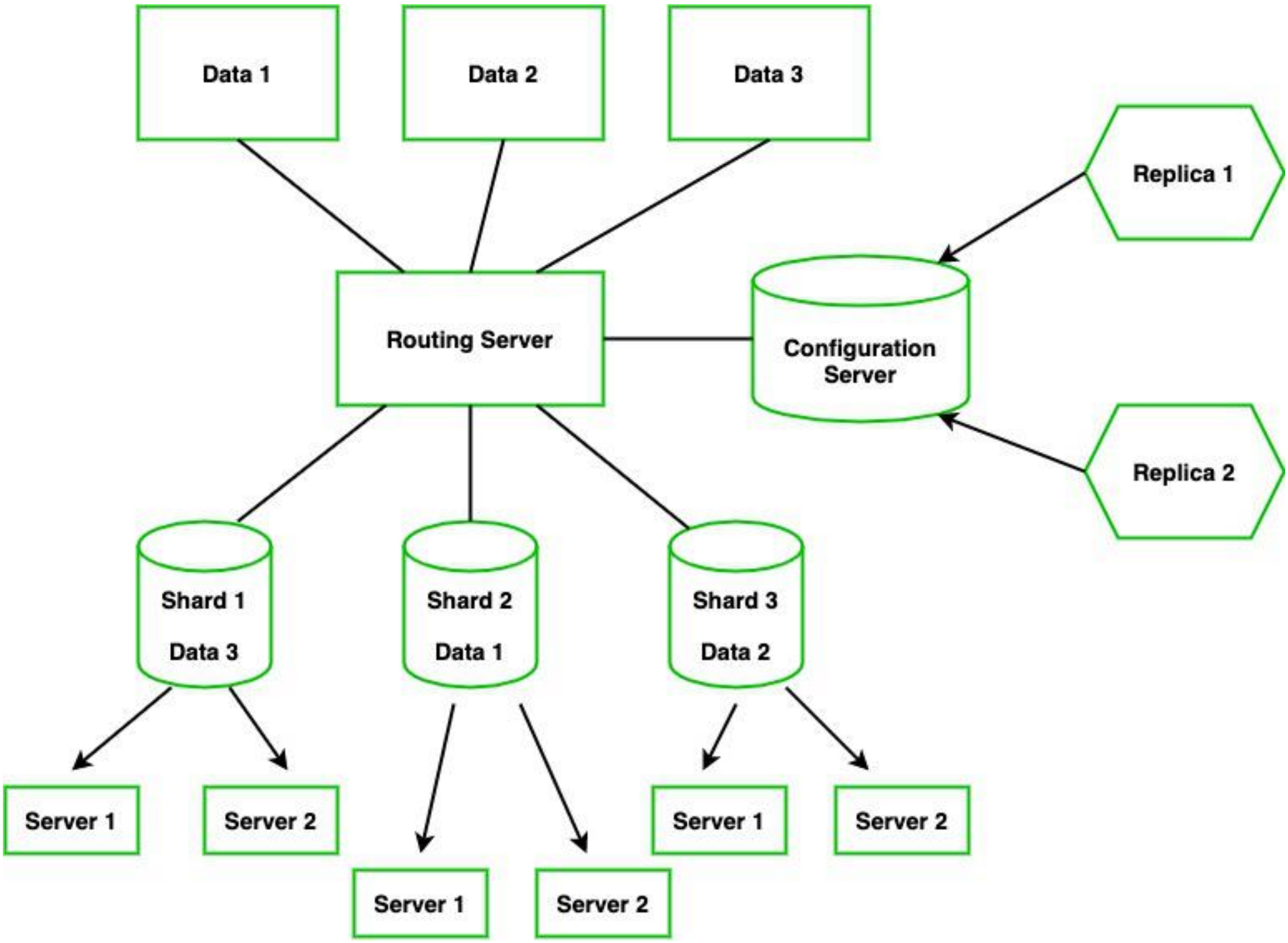
These commands start the `mongod` instances on each server with the `--replSet` option, specifying the replica set name (`rs0`). The `--dbpath` option specifies the data directory, and the `--bind_ip` option binds the server to the specified IP addresses.

Sharding

Sharding is a method for distributing large **collection**(dataset) and allocating it across multiple servers. MongoDB uses sharding to help deployment with very big data sets and high volume operations.
Sharding combines more devices to carry data extension and the needs of read and write operations.

Need for Sharding

Database systems that have big data sets or high throughput requests can not be handled by a single server.
For example, High query flows can drain the CPU limit of the server and large data set stress the I/O capacity of the disk drive.
How does Sharding work?
Sharding determines the problem with horizontal scaling. It breaks the system dataset and store it over multiple servers, adding new servers to increase the volume as needed.



Now, instead of one signal as primary, we have multiple servers called **Shard**. We have different routing servers that will route data to the shard servers.
For example: Let say we have Data 1, Data 2, and Data 3 this will be going to the routing server which will route the data (i.e, Different Data will go to a particular Shard). Each Shard holds some pieces of data.
Here the **configuration server** will hold the metadata and it will configure the routing server to integrate the particular data to a shard however configure server is the MongoDB instance if it goes down then the entire server will go down, So it again has Replica Configure database.
Advantages of Sharding

- Sharding adds more server to a data field automatically adjust data loads across various servers.
- The number of operations each shard manage got reduced.
- It also increases the write capacity by splitting the write load over multiple instances.
- It gives high availability due to the deployment of replica servers for shard and config.
- Total capacity will get increased by adding multiple shards.

In order to create sharded clusters in MongoDB, We need to configure the **shard**, a **config server**, and a **query router**.

To set up sharding in MongoDB using the MongoDB shell, follow these steps:

Step 1: Start `mongod` Instances for Shards

Start a `mongod` instance for each shard. Use the `--shardsvr` option to designate the instances as shard servers.

On Shard 1:

```
sh
Copy code
mongod --shardsvr --replSet shard1 --dbpath /data/shard1 --port 27018 --bind_ip
localhost,192.168.1.1
```

On Shard 2:

```
sh
Copy code
mongod --shardsvr --replSet shard2 --dbpath /data/shard2 --port 27018 --bind_ip
localhost,192.168.1.2
```

On Shard 3:

```
sh
Copy code
```

```
mongod --shardsvr --replSet shard3 --dbpath /data/shard3 --port 27018 --bind_ip
localhost,192.168.1.3
```

Step 2: Start `mongod` Instance for Config Server

Start the `mongod` instance for the config server with the `--configsvr` option.

On Config Server:

```
sh
Copy code
mongod --configsvr --replSet configReplSet --dbpath /data/config --port 27019 --
bind_ip localhost,192.168.1.4
```

Step 3: Start `mongos` Instance

Start the `mongos` instance and specify the config server.

On `mongos` Instance:

```
sh
Copy code
mongos --configdb configReplSet/192.168.1.4:27019 --bind_ip localhost,192.168.1.5 --
port 27017
```

Step 4: Configure the Cluster

Connect to the `mongos` instance using the MongoDB shell and initiate the config server replica set.

Connect to `mongos`:

```
sh
Copy code
mongo --host 192.168.1.5 --port 27017
```

Initialize the Config Server Replica Set:

```
javascript
Copy code
rs.initiate(
  {
    _id: "configReplSet",
    configsvr: true,
    members: [
      { _id: 0, host: "192.168.1.4:27019" }
    ]
  }
)
```

Step 5: Add Shards to the Cluster

Add the shard servers to the cluster.

```
javascript
Copy code
sh.addShard("shard1/192.168.1.1:27018")
sh.addShard("shard2/192.168.1.2:27018")
sh.addShard("shard3/192.168.1.3:27018")
```

Step 6: Enable Sharding on a Database

Enable sharding on a specific database. For example, if your database is named `myDatabase`:

```
javascript
Copy code
sh.enableSharding("myDatabase")
```

Step 7: Shard a Collection

Shard a specific collection within the database based on a shard key. For example, to shard the collection `myCollection` on the `shardKey` field:

```
javascript
Copy code
sh.shardCollection("myDatabase.myCollection", { shardKey: 1 })
```

Full Example

Here is the full set of commands:

1. **Start `mongod` Instances for Shards:**

   On Shard 1:

   ```
   sh
   Copy code
   mongod --shardsvr --replSet shard1 --dbpath /data/shard1 --port 27018 --
   bind_ip localhost,192.168.1.1
   ```

   On Shard 2:

   ```
   sh
   Copy code
   mongod --shardsvr --replSet shard2 --dbpath /data/shard2 --port 27018 --
   bind_ip localhost,192.168.1.2
   ```

   On Shard 3:

   ```
   sh
   Copy code
   mongod --shardsvr --replSet shard3 --dbpath /data/shard3 --port 27018 --
   bind_ip localhost,192.168.1.3
   ```

2. **Start Config Server:**

   ```
   sh
   Copy code
   mongod --configsvr --replSet configReplSet --dbpath /data/config --port 27019
   --bind_ip localhost,192.168.1.4
   ```

3. **Start `mongos` Instance:**

   ```
   sh
   Copy code
   mongos --configdb configReplSet/192.168.1.4:27019 --bind_ip
   localhost,192.168.1.5 --port 27017
   ```

4. **Connect to `mongos` Instance:**

```
sh
Copy code
mongo --host 192.168.1.5 --port 27017
```

5. **Initiate Config Server Replica Set:**

```
javascript
Copy code
rs.initiate(
  {
    _id: "configReplSet",
    configsvr: true,
    members: [
      { _id: 0, host: "192.168.1.4:27019" }
    ]
  }
)
```

6. **Add Shards:**

```
javascript
Copy code
sh.addShard("shard1/192.168.1.1:27018")
sh.addShard("shard2/192.168.1.2:27018")
sh.addShard("shard3/192.168.1.3:27018")
```

7. **Enable Sharding on Database:**

```
javascript
Copy code
sh.enableSharding("myDatabase")
```

8. **Shard a Collection:**

```
javascript
Copy code
sh.shardCollection("myDatabase.myCollection", { shardKey: 1 })
```

This setup will create a sharded MongoDB cluster with three shards, a config server, and a `mongos` instance, and enable sharding for a specified database and collection.

Conclusion

Both Replication and sharding in MongoDB helps in scaling of database. Where replication helps in data availability, sharding is useful to horizontally scale large datasets.

Import data to MongoDB

MongoDB provides the `mongoimport` tool, a powerful utility for importing various file formats such as **JSON**, **CSV**, and **TSV** into MongoDB **databases**.

This tool simplifies the process of loading data from external sources into MongoDB collections and makes it essential for **database administrators**. In this article, We will learn about **how to Import data to MongoDB** with multiple formats in detail.

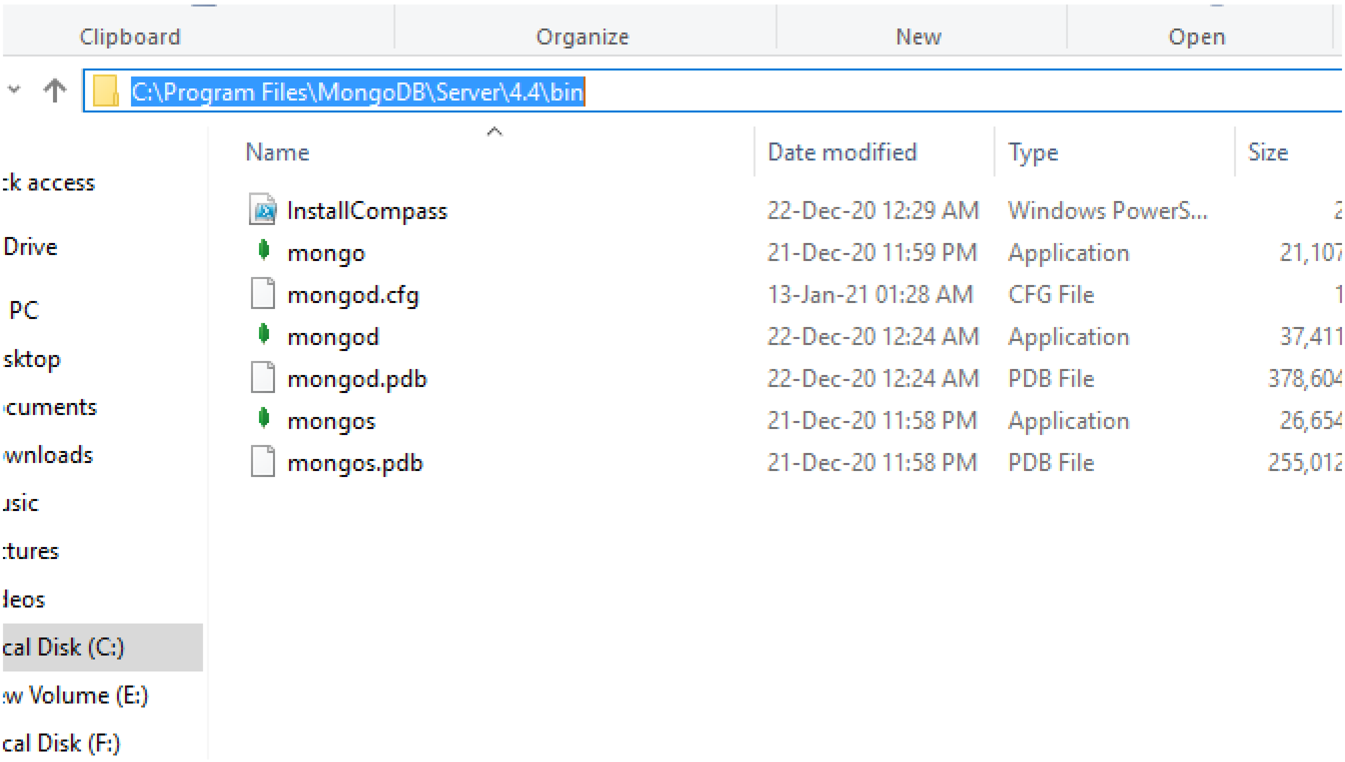
How to import data in MongoDB

Below are the steps to follow for **Import data to MongoDB** as defined below:

Step 1: To use **mongoimport** tool we have to first download the MongoDB [database](#)tools **.zip** file from [**here**](#). Here, we are going to download it for Windows.

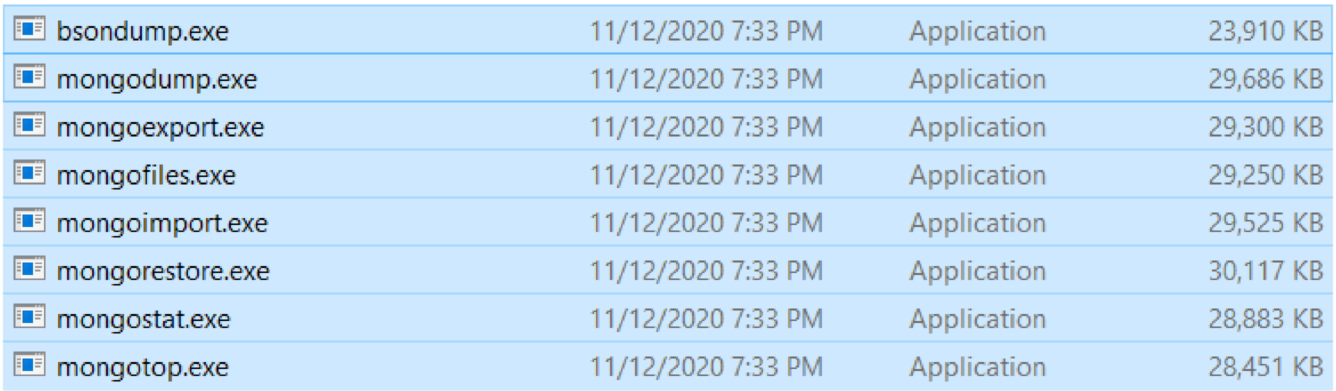
Step 2: After downloading the zip file and **unzip** the downloaded folder.

Step 3: Goto MongoDB bin folder and copy-paste the bin folders all **.exe** files into the MongoDB bin folder**[C:\Program Files\MongoDB\Server\4.4\bin].**



C:\Program Files\MongoDB\Server\4.4\bin

Name	Date modified	Type	Size
InstallCompass	22-Dec-20 12:29 AM	Windows PowerS...	2
mongo	21-Dec-20 11:59 PM	Application	21,107
mongod.cfg	13-Jan-21 01:28 AM	CFG File	1
mongod	22-Dec-20 12:24 AM	Application	37,411
mongod.pdb	22-Dec-20 12:24 AM	PDB File	378,604
mongos	21-Dec-20 11:58 PM	Application	26,654
mongos.pdb	21-Dec-20 11:58 PM	PDB File	255,012



bsondump.exe	11/12/2020 7:33 PM	Application	23,910 KB
mongodump.exe	11/12/2020 7:33 PM	Application	29,686 KB
mongoexport.exe	11/12/2020 7:33 PM	Application	29,300 KB
mongofiles.exe	11/12/2020 7:33 PM	Application	29,250 KB
mongoimport.exe	11/12/2020 7:33 PM	Application	29,525 KB
mongorestore.exe	11/12/2020 7:33 PM	Application	30,117 KB
mongostat.exe	11/12/2020 7:33 PM	Application	28,883 KB
mongotop.exe	11/12/2020 7:33 PM	Application	28,451 KB

And now we are ready to use **mongoimport** tool.

1. Import JSON file in MongoDB

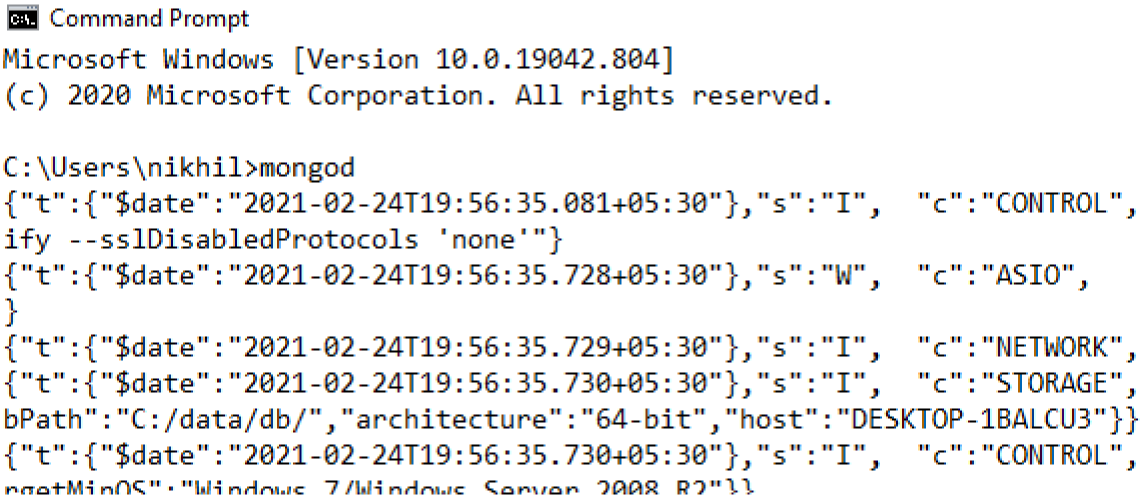
In MongoDB, we can import **JSON** files using mongoimport tool.

Syntax:

mongoimport –jsonArray –db database_name –collection collection_name –file file_location

To import **JSON** file you need to follow the following steps:

Step 1: Open a command prompt and give command mongod to connect with MongoDB server and don't close this cmd to stay connected to the server.

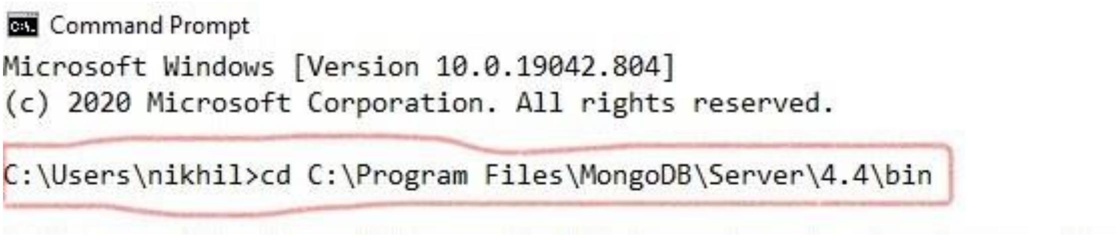


```
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\nikhil>mongod
{"t":{"$date":"2021-02-24T19:56:35.081+05:30"},"s":"I",  "c":"CONTROL",
ify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2021-02-24T19:56:35.728+05:30"},"s":"W",  "c":"ASIO",
}
{"t":{"$date":"2021-02-24T19:56:35.729+05:30"},"s":"I",  "c":"NETWORK",
{"t":{"$date":"2021-02-24T19:56:35.730+05:30"},"s":"I",  "c":"STORAGE",
bPath":"C:/data/db/","architecture":"64-bit","host":"DESKTOP-1BALCU3"}}
{"t":{"$date":"2021-02-24T19:56:35.730+05:30"},"s":"I",  "c":"CONTROL",
rgetMinOS":"Windows 7/Windows Server 2008 R2"}}
```

Step 2: Open another command prompt and run the mongo shell. Using the mongo command.

Step 3: Open one more command prompt window and direct it to bin folder**[C:\Program Files\MongoDB\Server\4.4\bin]** and now you are ready to import files in mongoDB database.

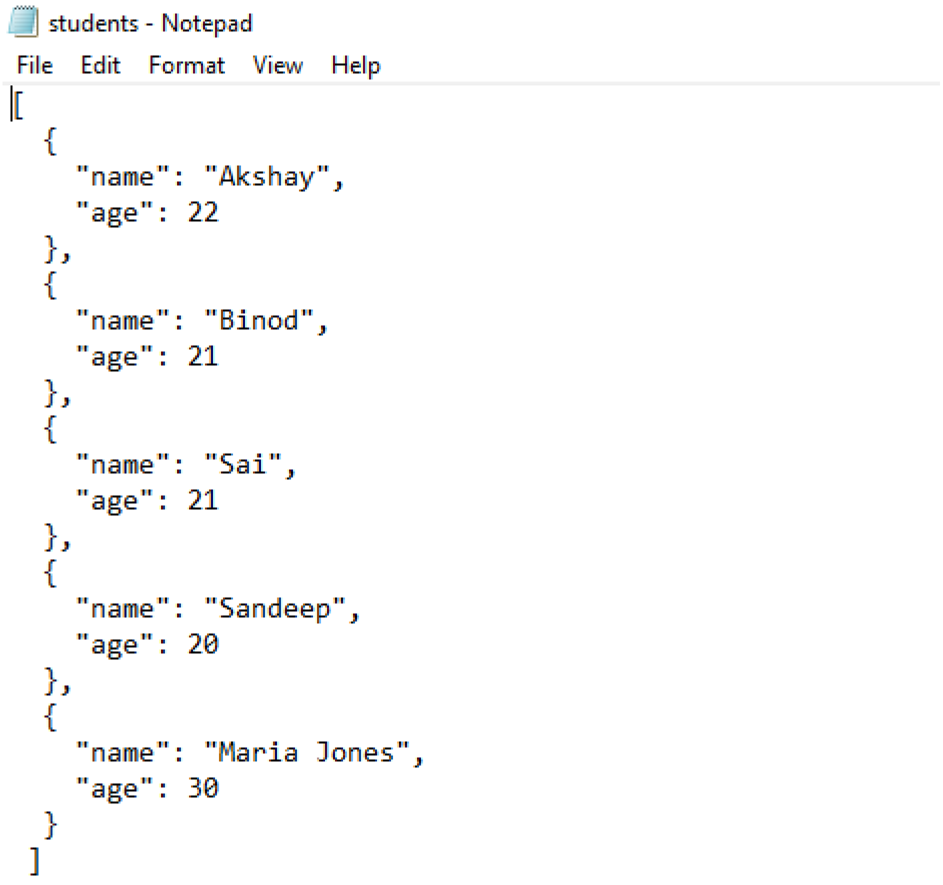


```
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\nikhil>cd C:\Program Files\MongoDB\Server\4.4\bin
```

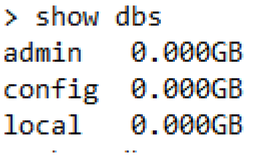
Example:

In this example, we have a JSON file, which we are going to import:



```
[
  {
    "name": "Akshay",
    "age": 22
  },
  {
    "name": "Binod",
    "age": 21
  },
  {
    "name": "Sai",
    "age": 21
  },
  {
    "name": "Sandeep",
    "age": 20
  },
  {
    "name": "Maria Jones",
    "age": 30
  }
 ]
```

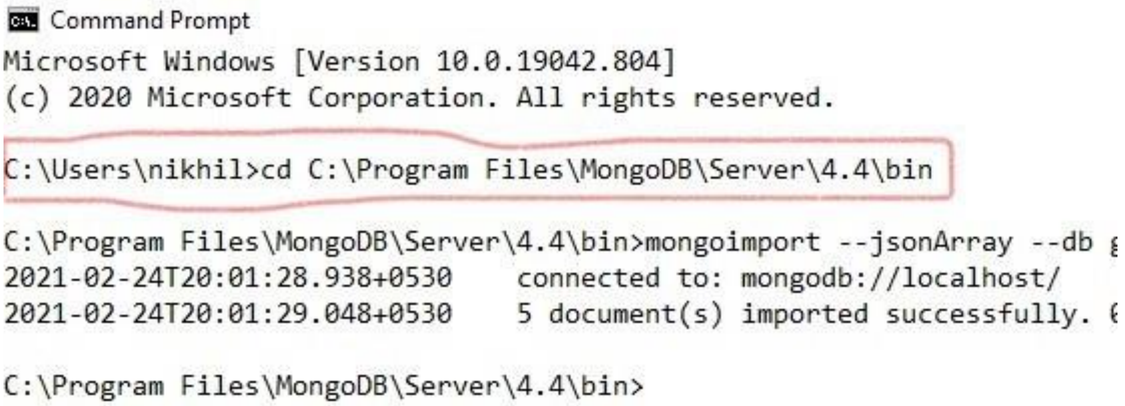
Before import data:



```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

After creating mongod server open another command prompt and direct it to bin folder[C:\Program Files\MongoDB\Server\4.4\bin] and now you can import files in the MongoDB. Here, in this example, we are going to import the JSON file in the student collection of gfg database:

```
mongoimport --jsonArray --db gfg --collection student --file
H:\students.json
```



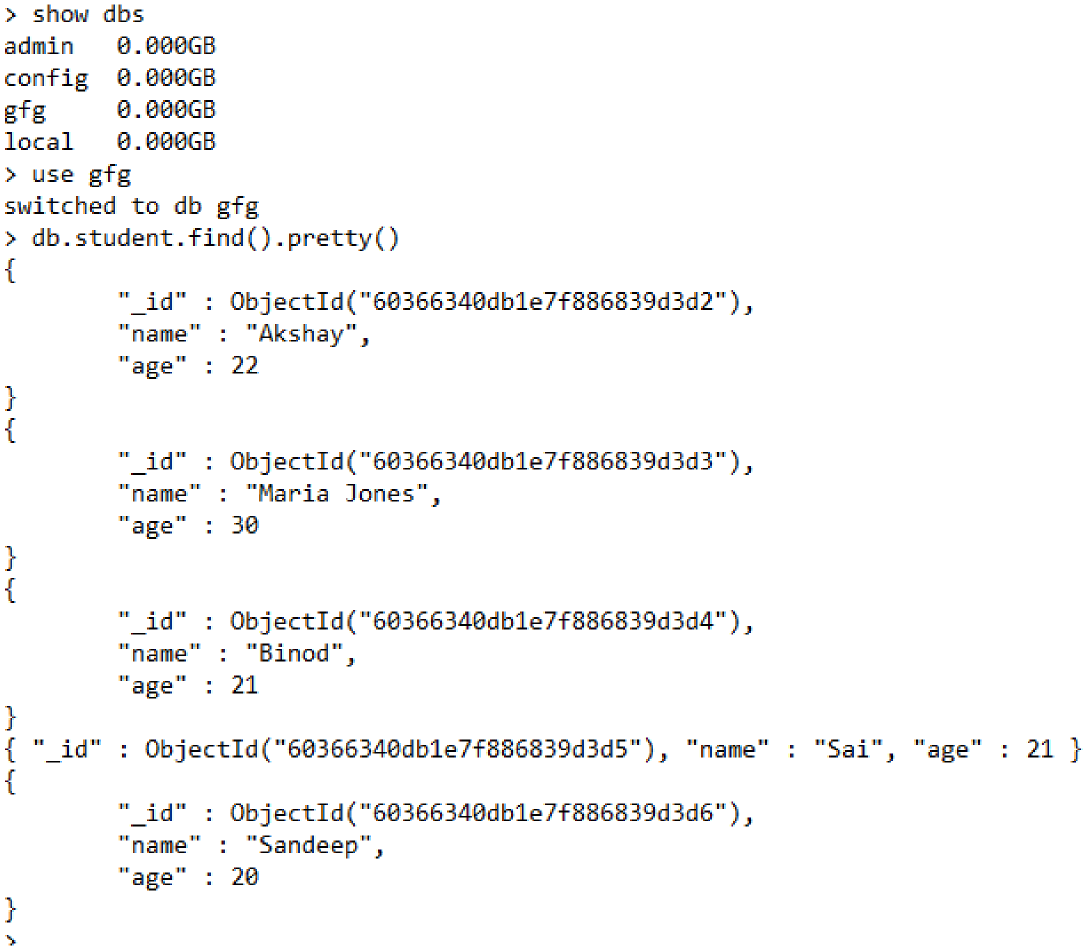
```
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\nikhil>cd C:\Program Files\MongoDB\Server\4.4\bin

C:\Program Files\MongoDB\Server\4.4\bin>mongoimport --jsonArray --db
2021-02-24T20:01:28.938+0530    connected to: mongodb://localhost/
2021-02-24T20:01:29.048+0530    5 document(s) imported successfully.

C:\Program Files\MongoDB\Server\4.4\bin>
```

Now, open the mongo shell window and check for the imported data:



```
> show dbs
admin   0.000GB
config  0.000GB
gfg     0.000GB
local   0.000GB
> use gfg
switched to db gfg
> db.student.find().pretty()
{
        "_id" : ObjectId("60366340db1e7f886839d3d2"),
        "name" : "Akshay",
        "age" : 22
}
{
        "_id" : ObjectId("60366340db1e7f886839d3d3"),
        "name" : "Maria Jones",
        "age" : 30
}
{
        "_id" : ObjectId("60366340db1e7f886839d3d4"),
        "name" : "Binod",
        "age" : 21
}
{ "_id" : ObjectId("60366340db1e7f886839d3d5"), "name" : "Sai", "age" : 21 }
{
        "_id" : ObjectId("60366340db1e7f886839d3d6"),
        "name" : "Sandeep",
        "age" : 20
}
>
```

Note: If the collection name is not specified then the collection name is created based on the first name of the file. Using these steps you can also import TSV files, simply using –type tsv.

2. Import CSV file in MongoDB

In MongoDB, we can also import csv file into two ways:

- **With header row**
- **Without header row**

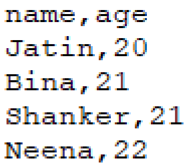
1. With header row: We can import data with header row with the help of –header that shows the name of the fields will use the first line of the CSV file.

Syntax:

mongoimport –db database_name –collection collection_name –type csv –file file_location –header

Example:

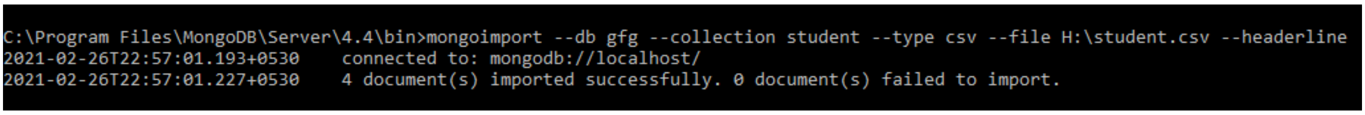
We have the following CSV file called student.csv:



```
name,age
Jatin,20
Bina,21
Shanker,21
Neena,22
```

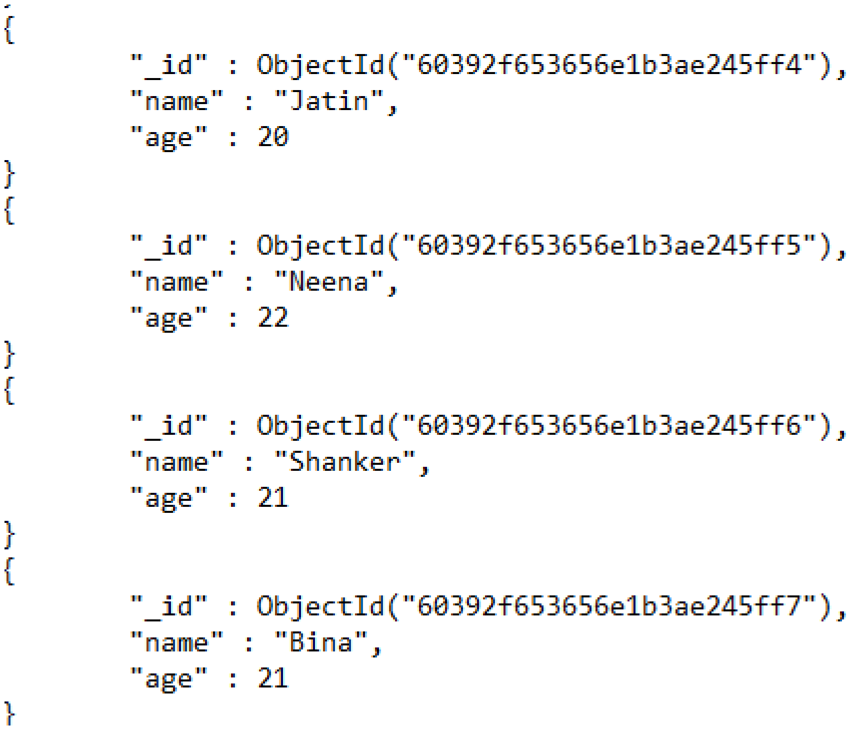
Import data from CSV file:

mongoimport –db gfg –collection student –type csv –file H:\students.csv –header



```
C:\Program Files\MongoDB\Server\4.4\bin>mongoimport --db gfg --collection student --type csv --file H:\student.csv --headerline
2021-02-26T22:57:01.193+0530    connected to: mongodb://localhost/
2021-02-26T22:57:01.227+0530    4 document(s) imported successfully. 0 document(s) failed to import.
```

After import data:



```
{
        "_id" : ObjectId("60392f653656e1b3ae245ff4"),
        "name" : "Jatin",
        "age" : 20
}
{
        "_id" : ObjectId("60392f653656e1b3ae245ff5"),
        "name" : "Neena",
        "age" : 22
}
{
        "_id" : ObjectId("60392f653656e1b3ae245ff6"),
        "name" : "Shanker",
        "age" : 21
}
{
        "_id" : ObjectId("60392f653656e1b3ae245ff7"),
        "name" : "Bina",
        "age" : 21
}
```

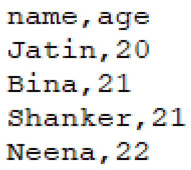
2. Without header row: We can import data without header row by excluding –header. But in the place of the header, we have to put **–fields** that show the name of the field that we want to give. Field names are separated by a comma.

Syntax:

mongoimport –db database_name –collection collection_name –type csv –fields field_names –file file_location

Example:

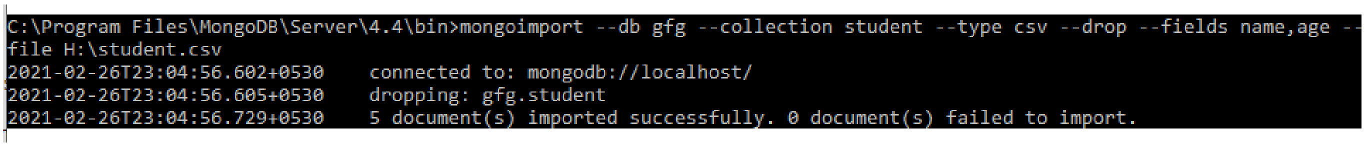
We have the following CSV file called student.csv:



```
name,age
Jatin,20
Bina,21
Shanker,21
Neena,22
```

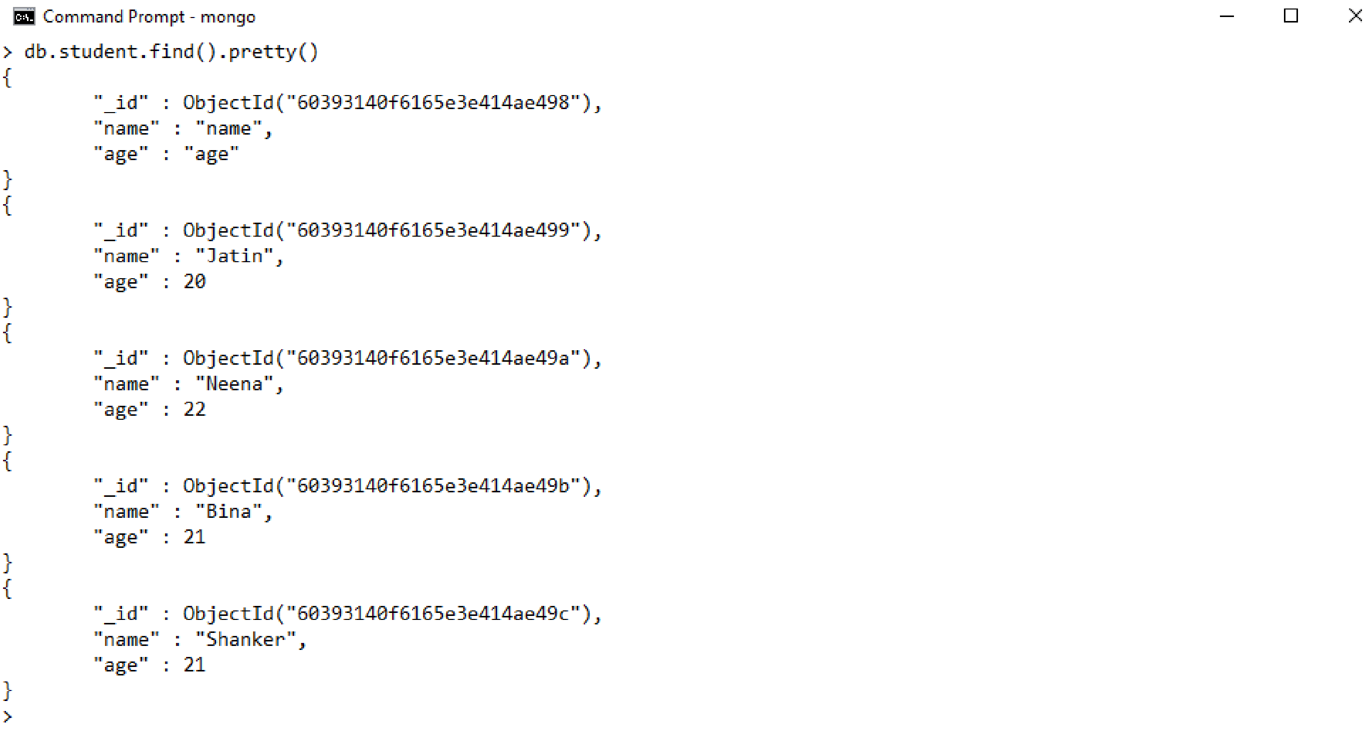
Import data from CSV file:

mongoimport –db gfg –collection student –type csv –fields name,age –file H:\students.csv



```
C:\Program Files\MongoDB\Server\4.4\bin>mongoimport --db gfg --collection student --type csv --drop --fields name,age --
file H:\student.csv
2021-02-26T23:04:56.602+0530    connected to: mongodb://localhost/
2021-02-26T23:04:56.605+0530    dropping: gfg.student
2021-02-26T23:04:56.729+0530    5 document(s) imported successfully. 0 document(s) failed to import.
```

After import data:



```
> db.student.find().pretty()
{
        "_id" : ObjectId("60393140f6165e3e414ae498"),
        "name" : "name",
        "age" : "age"
}
{
        "_id" : ObjectId("60393140f6165e3e414ae499"),
        "name" : "Jatin",
        "age" : 20
}
{
        "_id" : ObjectId("60393140f6165e3e414ae49a"),
        "name" : "Neena",
        "age" : 22
}
{
        "_id" : ObjectId("60393140f6165e3e414ae49b"),
        "name" : "Bina",
        "age" : 21
}
{
        "_id" : ObjectId("60393140f6165e3e414ae49c"),
        "name" : "Shanker",
        "age" : 21
}
>
```

Conclusion

The `mongoimport` tool is essential for efficiently importing JSON, CSV, and TSV files into MongoDB collections. By following these steps, you can seamlessly integrate external data into your MongoDB databases, ensuring robust data management and application scalability.

Working with Binary data in MongoDB (with examples)

Introduction

Binary data can encompass a variety of data types, such as images, files, and blobs that are not natively supported by JSON document structures. However, MongoDB offers a solution in the form of Binary JSON (BSON), an extension of the JSON format which allows for the encoding of binary data and other types not natively supported in JSON.

Understanding BSON

BSON extends the JSON model to provide additional data types, such as the ‘binData’ type, which is specifically designed to hold binary data. Binary data is represented in BSON as a base64-encoded string along with a subtype byte that describes the type of data that the binary represents. Subtypes can signal generic binary data, UUIDs, MD5 hashes, and user-defined binary types.

Basic CRUD Operations with Binary Data

CRUD operations with binary data are similar to working with any other BSON data type within MongoDB. The following are basic code examples highlighting operations you can perform with binary data:

Inserting Binary Data

The MongoDB Drivers provide a way to create a Binary object by directly inserting binary data:

```
// MongoDB JavaScript Shell
const fs = require('fs');

// Read binary data from a file
const binaryData = fs.readFileSync('/path/to/your/file');

// Save binary data to MongoDB
db.collection('binaries').insertOne({
  file_data: new BinData(0, binaryData.toString('base64'))
});
```