
# MongoDB Connectivity

**Prof.Pavan A C and**
**Prof.Shruthi L**

Department of Computer Science and Engineering

## Node JS and MongoDB Connectivity

- Node.js is a powerful JavaScript runtime that allows server-side execution of JavaScript code.

- Connecting Node.js to MongoDB enables seamless communication between the application and the database.

- Use the MongoDB Node.js driver to connect, perform CRUD operations, and handle errors effectively.

## Node JS and MongoDB Connectivity

- Node.js uses a promise to manipulate MongoDB databases
- Alternately, use more sophisticated third party module like ‘mongoose’ that provides Object Data Modeling capabilities

**Creating a Database**

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database.

- MongoDB will create the database if it does not exist, and make a connection to it.

## Node JS and MongoDB Connectivity

```
//Create Database

var MongoClient=require('mongodb').MongoClient;
var url="mongodb://127.0.0.1:27017/ndb";
MongoClient.connect(url,function(err,db)
{
    if(err) throw err;
    console.log("Database created");
    db.close();
});
```

## Creating a Collection

- To create a collection in MongoDB, use the Collection() method
- In MongoDB, a collection is not created until it gets content

## Insert a single document Into Collection

- To insert document into a collection, use the insertOne() method
- A document in MongoDB is the same as a record in MySQL
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.
- It also takes a callback function where you can work with any errors, or the result of the insertion
- To insert multiple documents at once, use insertMany() method

# NODE JS

## Node JS and MongoDB Connectivity

```
//Create Collection

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017";

let client;

MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db("ndb");
        return dbo.createCollection("employee");
    })
.then((res) => {
        console.log("Collection created");
        // close the connection
        client.close();
    })
    .catch((err) => {
        console.error("An error occurred:",
err);
    });
```

**Insert the documents to collection:**

- In MongoDB use the insertOne() and insertMany() methods to add documents to the collection
- Just like the SELECT statement is used to find data in a table in a MySQL database.

**insertOne() and insertMany():**

- The insert() method Inserts a single document into a collection.
- The insertMany() method add multiple documents to existing collection.

# NODE JS

## Node JS and MongoDB Connectivity

```
//Insert One

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017";

let client;

MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db("ndb");
        return dbo.createCollection("employee");
    })
    .then((res) => {
        console.log("Collection created");
```

# Node JS and MongoDB Connectivity

```
// Insert example data
        const data = { name: "John Doe", phone: "1234567890" };
        const collection = client.db("ndb2").collection("employee");
        return collection.insertOne(data);
    })
    .then((result) => {
        console.log("1 document inserted");
        // close the connection
        client.close();
    })
    .catch((err) => {
        console.error("An error occurred:", err);
    });
```

## Node JS and MongoDB Connectivity

**Select the documents from collection**:

- In MongoDB use the find() and findOne() methods to find data in a collection.
- Just like the SELECT statement is used to find data in a table in a MySQL database.

**Find:**

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object.

## Node JS and MongoDB Connectivity

```
//Find

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017";

let client;

MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db(“ndb");
```

# Node JS and MongoDB Connectivity

```
dbo.collection("employee").find({}, { projection: {_id:0,name: 1,
phone :1} }).toArray(function(err, result)
        {
            if (err) throw err;
            console.log(result);
            client.close();

        })

    })
    .then((result) => {
        console.log(" Document fetched");
       client.close();
    })
    .catch((err) => {
        console.error("An error occurred:", err);
    });
```

**Update the documents in collection**:

- In MongoDB use the updateOne() and UpdateMany to update data in a collection.

**UpdateOne():**

- Updates a single document within the collection based on the filter.

**UpdateMany()**

- Updates all documents that match the specified filter for a collection.

## Node JS and MongoDB Connectivity

```
//Update

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017";

let client;
MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db("ndb");
        var myquery = { name:"Ajay" };
  var newvalues = { $set: {name: "Aarav", phone:1234567890} };
  dbo.collection("employee").updateOne(myquery, newvalues,
function(err, result)
   {
    if (err) throw err;
    console.log(result);
    client.close();
  })
})
                                        .then((result) => {
                                            console.log(" Document updated");
                                        })
                                        .catch((err) => {
                                            console.error("An error occurred:", err);
                                        })
```

**Delete the documents in collection**:

- To delete a record, we use the deleteOne() method or deleteMany()

deleteOne()- Removes a single document from a collection.

deleteMany- Removes all documents that match the `filter` from a collection

**Drop collection**:

- To delete a table, or collection as it is called in MongoDB, we use the drop() method.
- The drop() method takes a callback function containing the error object and the result parameter which returns true if the collection was dropped successfully, otherwise it returns false.

## Node JS and MongoDB Connectivity

```
//Delete document

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017/";

let client;

MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db("ndb2");
        const myquery = { name: 'John Doe' };
        return
dbo.collection("students").deleteOne(myquery);
    })
    .then((obj) => {
        console.log("1 document deleted");
       client.close();
})
    .catch((err) => {
        console.error("An error occurred:",
err);
    });
```

# Node JS and MongoDB Connectivity

```
//Drop collection

const { MongoClient } = require('mongodb');
const url = "mongodb://127.0.0.1:27017/";

let client;

MongoClient.connect(url)
    .then((connectedClient) => {
        client = connectedClient;
        const dbo = client.db("ndb1");
        return dbo.collection("students").drop();
    })
    .then((delOK) => {
        if (delOK) {
            console.log("Collection deleted");
        }
        client.close();
    })
    .catch((err) => {
        console.error("An error occurred:",
err);
    });
```


# THANK YOU

**Prof.Pavan A C and**
**Prof.Shruthi L**

Department of Computer Science and Engineering