

UNIT-5



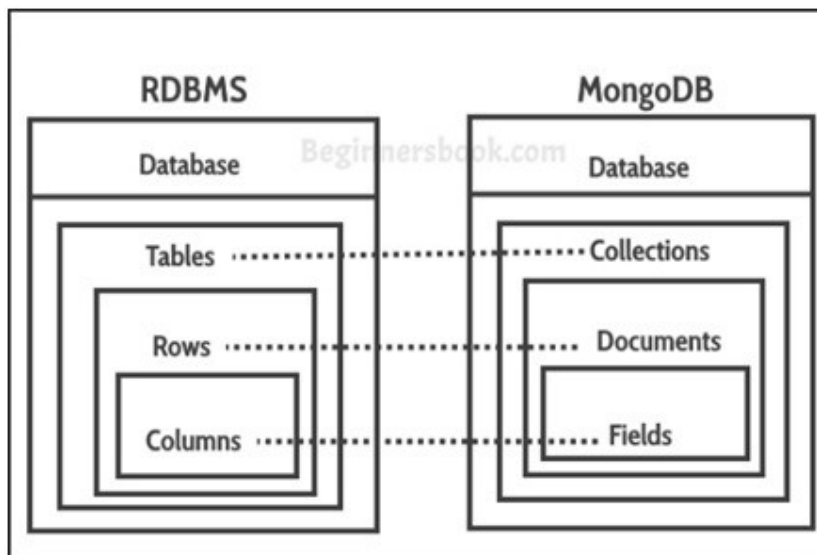
1. Why NoSql ?

- The concept of NoSQL (Not Only SQL) consists of technologies that provide storage and retrieval without the tightly constrained models of traditional SQL relational databases.
- The **motivation** behind NoSQL is mainly simplified designs, horizontal scaling, and finer control of the availability of data.
- NoSQL breaks away from the traditional structure of relational databases and allows developers to implement models in ways that more closely fit the data flow needs of their systems. This allows NoSQL databases to be implemented in ways that traditional relational databases could never be structured.
- There are several different NoSql technologies such as HBase's column structure, Redis's key/value structure, and Neo4j's graph structure to implement the backend storage for web applications and services.

2. Understanding MongoDB :

- MongoDB is a NoSQL database based on a document model where data objects are stored as separate documents inside a collection.
- The **motivation** of the MongoDB language is to implement a data store that provides high performance, high availability, and automatic scaling.

Structure of MongoDB:



2.1) Understanding Collections:

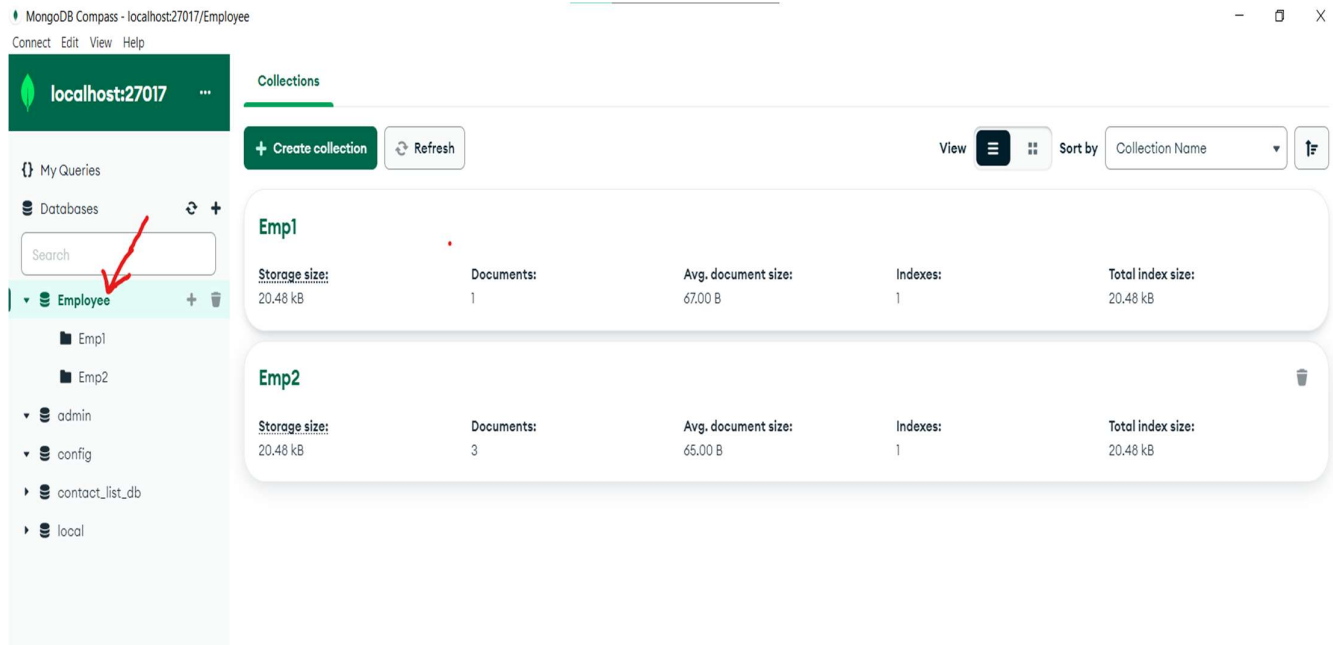
- MongoDB groups data together through collections. A collection is simply a grouping of documents that have the same or a similar purpose.
- A collection acts similarly to a table in a traditional SQL database, with one major difference. In MongoDB, a collection is not enforced by a strict schema; instead, documents in a collection can have a slightly different structure from one another as needed. This reduces the need to break items in a document into several different tables, which is often done in SQL implementations.

2.2) Understanding Documents:

→ A *document* is a representation of a single entity of data in the MongoDB database.

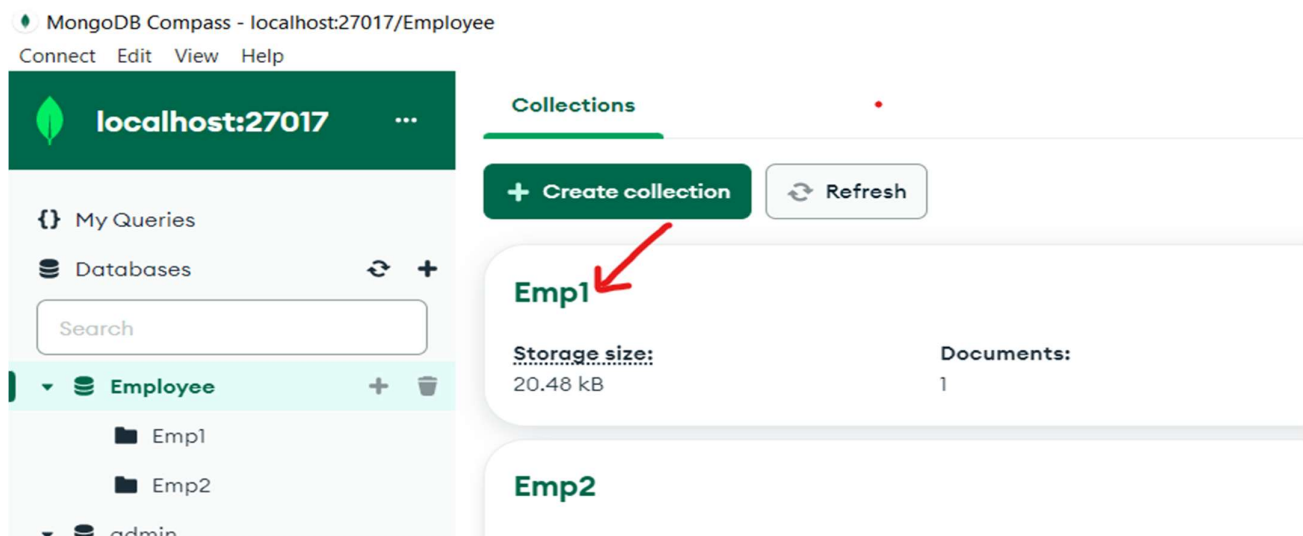
→ A collection is made up of one or more related objects. However, in MongoDB, documents can contain embedded subdocuments, thus providing a much closer inherent data model to your applications.

→ The records in MongoDB that represent documents are stored as BSON, which is a lightweight binary form of JSON, with `field:value` pairs corresponding to JavaScript `property:value` pairs. These `field:value` pairs define the values stored in the document.

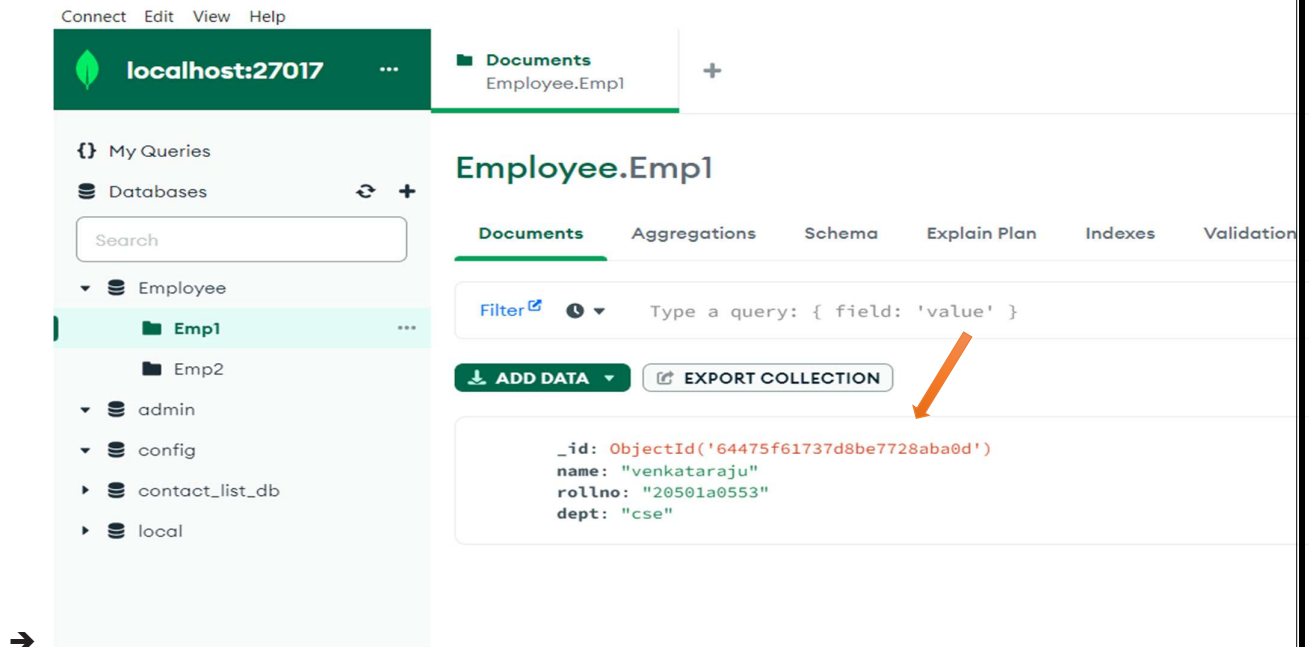


→ Here Employee, admin, config, contact_list_db, local are the databases in MongoDB compass.

→ The Employee database consist of Emp1 and Emp2 collections.



→ Here Emp1 collection consist of One Document.



→

- That the document structure contains fields/properties that are strings, integers, arrays, and objects, just like a JavaScript object.
- The field names cannot contain null characters, . (dots), or \$ (dollar signs). Also, the `_id` field name is reserved for the Object ID. The `_id` field is a unique ID for the system that is made up of the following parts:
 - A 4-byte value representing the seconds since the last epoch
 - A 3-byte machine identifier
 - A 2-byte process ID
 - A 3-byte counter, starting with a random value.
- The maximum size of a document in MongoDB is **16MB**, which prevents queries that result in an excessive amount of RAM being used or intensive hits to the file system.
- Although you may never come close, you still need to keep the maximum document size in mind when designing some complex data types that contain file data.

3. MongoDB Data Types:

- The BSON data format provides several different types that are used when storing the JavaScript objects to binary form. These types match the JavaScript type as closely as possible.
- It is important to understand these types because you can actually query MongoDB to find objects that have a specific property that has a value of a certain type.
- For example, you can look for documents in a database whose timestamp value is a `String` object or query for ones whose timestamp is a `Date` object.
- MongoDB assigns each of the data types an integer ID number from **1 to 255** that is used when querying by type. Table 11.1 shows a list of the data types that MongoDB supports along with the number MongoDB uses to identify them.

Table 11.1 MongoDB data types and corresponding ID number

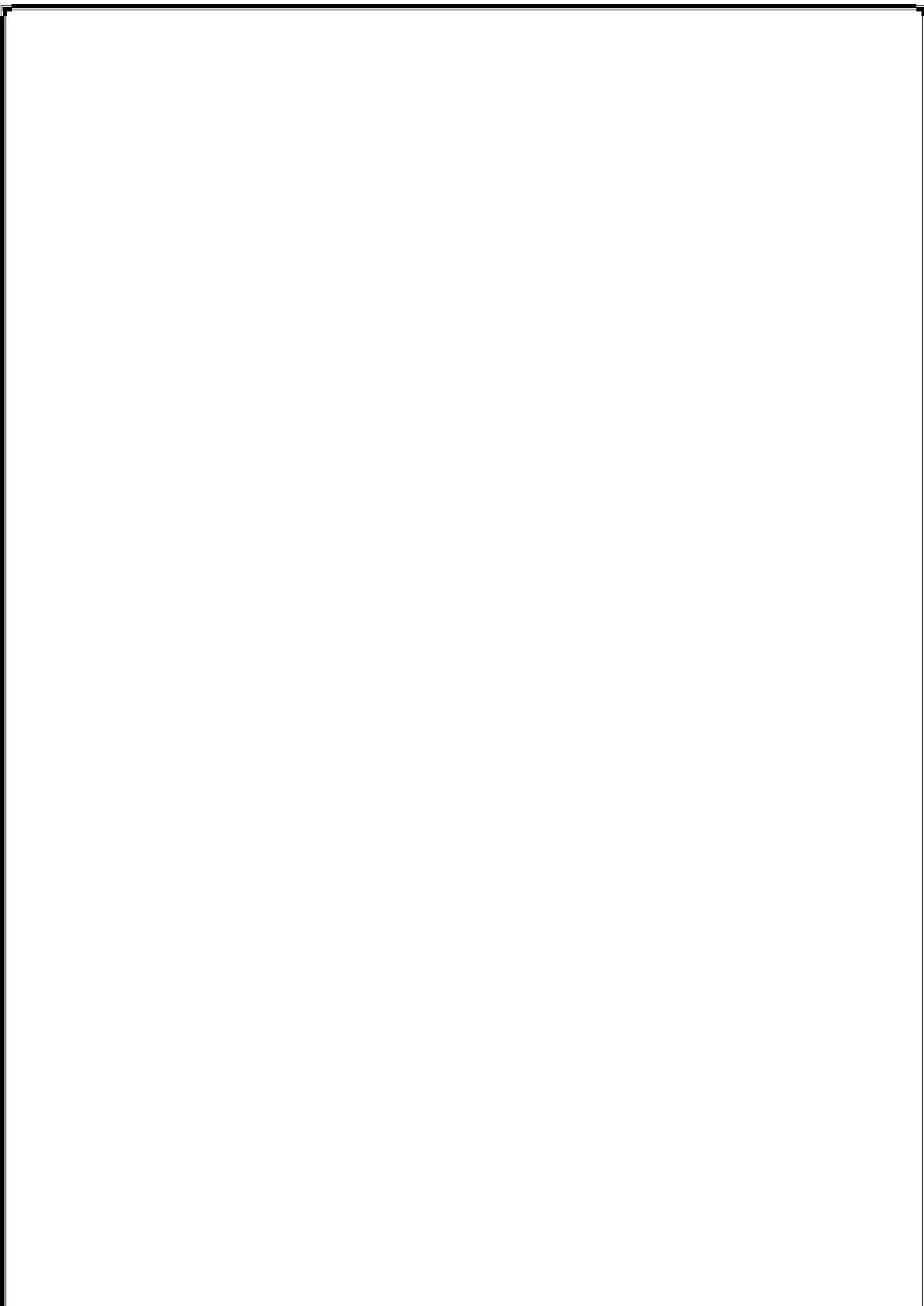
Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13

→ JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Decimal128	19
Min key	-1
Max key	127

→ Another thing to be aware of when working with different data types in MongoDB is the order in which they are compared. When comparing values of different BSON types, MongoDB uses the following comparison order from lowest to highest:

1. Min Key (internal type)
2. Null
3. Numbers (32-bit integer, 64-bit integer, Double)
4. String
5. Object
6. Array
7. Binary Data
8. Object ID
9. Boolean
10. Date, Timestamp
11. Regular Expression
12. Max Key (internal type)





UNIT – 5

GETTING STARTED WITH MONGODB:

- **Building the MongoDB Environment**
- **Installing MongoDB**
- **Starting MongoDB**
- **Stopping MongoDB**
- **Accessing MongoDB from the Shell Client**

BUILDING THE MONGODB ENVIRONMENT:

- To get started with MongoDB, the first task is to install it on your development system.
- Once installed on your development system, you can play around with the functionality, learn the MongoDB shell, and prepare for “Getting Started with MongoDB and Node.js,” in which you begin integrating MongoDB into your Node.js applications.
- The following sections cover installation, starting and stopping the database engine, and accessing the shell client.
- Once you can do those things you are ready to begin using MongoDB in your environment.

INSTALLING MONGODB:

- The first step in getting MongoDB implemented into your Node.js environment is installing the MongoDB server.
- There is a version of MongoDB for each of the major platforms, including Linux, Windows, Solaris, and OS X.
- There is also an enterprise version available for the Red Hat, SuSE, Ubuntu, and Amazon Linux distributions.
- The enterprise version of MongoDB is subscription-based and provides enhanced security, management, and integration support.
- For the purpose of the learning MongoDB, the standard edition of MongoDB is perfect.
- Before continuing, go to the MongoDB website at <http://docs.mongodb.org/manual/installation/>.
- Follow the links and instructions to download and install MongoDB in your environment:

As part of the installation and setup process, perform the following steps:

1. Download and extract the MongoDB files.
2. Add the <mongo_install_location>/bin to your system path.
3. Create a data files directory: <mongo_data_location>/data/db.
4. Start MongoDB using the following command from the console prompt:

```
mongod -dbpath <mongo_data_location>/data/db
```

STARTING MONGODB:

- Once you have installed MongoDB, you need to be able to start and stop the database engine.
- The database engine starts by executing the mongod (mongod.exe on Windows) executable in the <mongo_install_location>/bin location.
- This executable starts MongoDB and begins listening for database requests on the configured port.
- The mongod executable accepts several different parameters that provide methods of controlling its behavior.
- For example, you can configure the IP address and port MongoDB listens on as well as logging and authentication.
- Table 12.1 provides a list of some of the most commonly used parameters.
- Here is an example of starting MongoDB with a port and dbpath parameters:

```
mongod -port 28008 -dbpath <mongo_data_location>/data/db
```

Table 12.1 mongod command-line parameters

Parameter	Description
--help, -h	Returns basic help and usage text.
--version	Returns the version of MongoDB.
--config <filename>, -f <filename>	Specifies a configuration file that contains runtime-configurations.
--verbose, -v	Increases the amount of internal reporting sent to the console and written to the log file specified by --logpath.
--quiet	Reduces the amount of reporting sent to the console and log file.
--port <port>	Specifies a TCP port for mongod to listen for client connections. Default: 27017.

Parameter	Description
<code>--bind_ip <ip address></code>	Specifies the IP address on which mongod will bind to and listen for connections. Default: All Interfaces
<code>--maxConns <number></code>	Specifies the maximum number of simultaneous connections that mongod will accept. Max: 20000
<code>--logpath <path></code>	Specifies a path for the log file. On restart, the log file is overwritten unless you also specify <code>--logappend</code> .
<code>--auth</code>	Enables database authentication for users connecting from remote hosts.
<code>--dbpath <path></code>	Specifies a directory for the mongod instance to store its data.
<code>--nohttpinterface</code>	Disables the HTTP interface.
<code>--nojournal</code>	Disables the durability journaling.
<code>--noprealloc</code>	Disables the preallocation of data files, which shortens the startup time but can cause significant performance penalties during normal operations.
<code>--repair</code>	Runs a repair routine on all databases.

STOPPING MONGODB:

- Each platform has different methods of stopping the mongod executable once it has started.
- However, one of the best methods is to stop it from the shell client because that cleanly shuts down the current operations and forces the mongod to exit.
- To stop the MongoDB database from the shell client, use the following commands to switch to the admin database and then shut down the database engine:
 - use admin
 - `db.shutdownServer()`

ACCESSING MONGODB FROM THE SHELL CLIENT:

- Once you have installed, configured, and started MongoDB, you can access it through the MongoDB shell.
- The MongoDB shell is an interactive shell provided with MongoDB that allows you to access, configure, and administer MongoDB databases, users, and much more.
- You use the shell for everything from setting up user accounts to creating databases to querying the contents of the database.
- The following sections take you through some of the most common administration tasks that you perform in the MongoDB shell.
- Specifically, you need to be able to create user accounts, databases, and collections to follow the examples in the rest of the chapter.
- Also you should be able to perform at least rudimentary queries on documents to help you troubleshoot any problems with accessing data.
- To start the MongoDB shell, first make sure that mongod is running, and then run the mongod command, then execute the mongo command from the console prompt.
- The shell should start up as shown in Figure 12.1.
- Once you have accessed the MongoDB shell, you can administer all aspects of MongoDB.
- There are a couple of things to keep in mind when using MongoDB.
 - First is that it is based on JavaScript and most of its syntax is available.
 - Second, the shell provides direct access to the database and collections on the server so changes you make directly impact the data on the server.

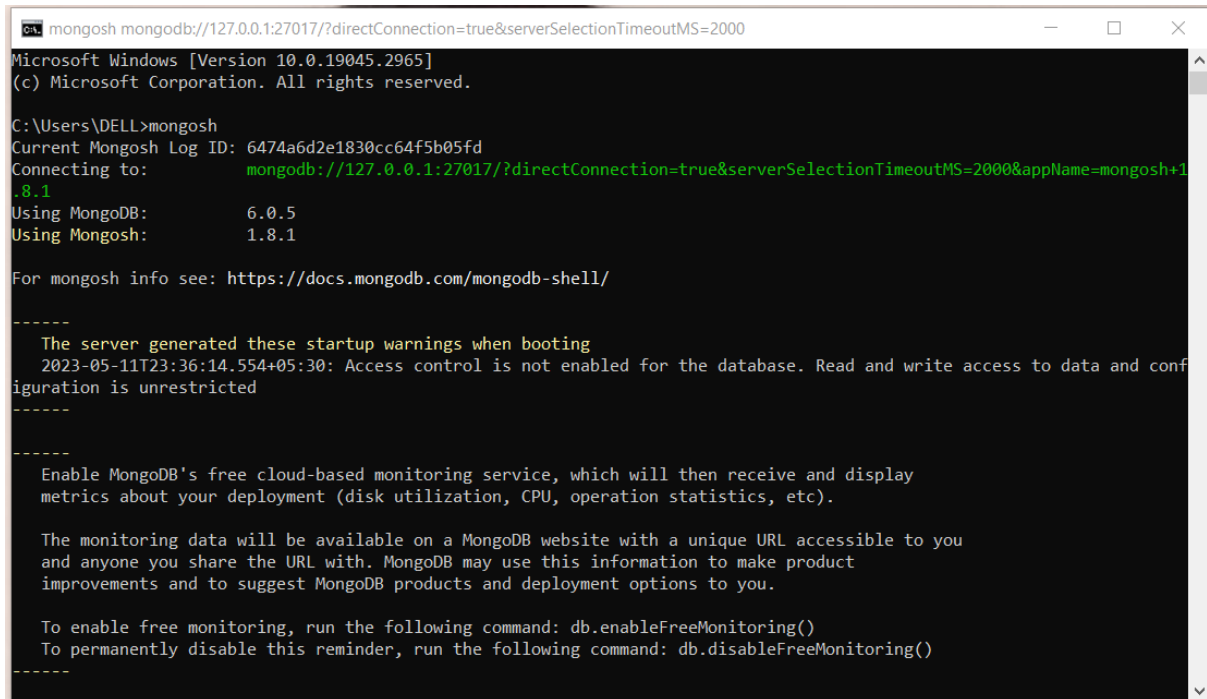
```
MongoDB shell version: 2.4.8
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
> _
```

Figure 12.1 Starting the MongoDB console shell

MongoDB shell commands and methods :

1. Starting the MongoDB Shell:

- Open a terminal or command prompt.
- Type ``mongosh`` and press Enter to start the MongoDB shell.



```
cmd mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>mongosh
Current Mongosh Log ID: 6474a6d2e1830cc64f5b05fd
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.1
Using MongoDB:      6.0.5
Using Mongosh:      1.8.1

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

-----
  The server generated these startup warnings when booting
  2023-05-11T23:36:14.554+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

-----
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
-----
```

2. Database Operations:

- ``use database_name``: Switch to a specific database or create a new one.
- ``show dbs``: List all available databases.
- ``db.dropDatabase()``: Delete the current database.



```
test> show dbs
admin    40.00 KiB
config   72.00 KiB
ipl      72.00 KiB
local    72.00 KiB
student  80.00 KiB
test     56.00 KiB
todos    72.00 KiB
test>

test> use mydatabase
switched to db mydatabase
mydatabase>

mydatabase> db.dropDatabase()
{ ok: 1, dropped: 'mydatabase' }
mydatabase>

mydatabase>
```

3. Collection Operations:

- `db.createCollection('collection_name')`: Create a new collection.
- `show collections`: List all collections in the current database.
- `db.collection_name.drop()`: Delete a collection.

```
mydatabase> db.createCollection("Indian_Team")
{ ok: 1 }
mydatabase> show collections
Indian_Team
Indian_Team
student
mydatabase> db.Indian_Team.drop()
true
mydatabase> _
```

4. Document Operations:

- `db.collection_name.insertOne(document)`: Insert a single document into a collection.
- `db.collection_name.insertMany(documents)`: Insert multiple documents into a collection.

```
mydatabase> db.Indian_Team.insertOne({name:"Gill",age:24,type:"Batter"})
{
  acknowledged: true,
  insertedId: ObjectId("6474ab03bd4104ab7e3a20e5")
}
mydatabase> db.Indian_Team.insertMany([{name:"Kohli",age:35,type:"Batter"},{name:"Bumrah",age:28,type:"Bowler"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6474ab11bd4104ab7e3a20e6"),
    '1': ObjectId("6474ab11bd4104ab7e3a20e7")
  }
}
```

- `db.collection_name.find(query)`: Retrieve documents based on a query.

```
mydatabase> db.Indian_Team.find()
[
  {
    _id: ObjectId("6474ab03bd4104ab7e3a20e5"),
    name: 'Gill',
    age: 24,
    type: 'Batter'
  },
  {
    _id: ObjectId("6474ab11bd4104ab7e3a20e6"),
    name: 'Kohli',
    age: 35,
    type: 'Batter'
  },
  {
    _id: ObjectId("6474ab11bd4104ab7e3a20e7"),
    name: 'Bumrah',
    age: 28,
    type: 'Bowler'
  }
]
```

- `db.collection_name.updateOne(filter, update)`: Update a single document that matches the filter.
- `db.collection_name.updateMany(filter, update)`: Update multiple documents that match the filter.

```
mydatabase> db.Indian_Team.updateOne({age:24},{set:{name:"Shubmann Gill"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mydatabase> db.Indian_Team.updateMany({},{$set:{name:"Shubmann Gill"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 2,
  upsertedCount: 0
}
mydatabase> _
```

- `db.collection_name.deleteOne(filter)`: Delete a single document that matches the filter.
- `db.collection_name.deleteMany(filter)`: Delete multiple documents that match the filter.

```
mydatabase> db.Indian_Team.deleteOne({age:24})
{ acknowledged: true, deletedCount: 1 }
mydatabase> db.Indian_Team.deleteMany({age:24})
{ acknowledged: true, deletedCount: 0 }
mydatabase>
```

5. Query Operators:

- Comparison operators: `$eq`, `$ne`, `$gt`, `$lt`, `$gte`, `$lte`.
- Logical operators: `$and`, `$or`, `$not`, `$nor`.
- Element operators: `$exists`, `$type`.
- Array operators: `$in`, `$nin`, `$all`, `$elemMatch`.

6. Aggregation Pipeline:

- Aggregation stages: `$match`, `$project`, `$group`, `$sort`, `$limit`, `$skip`, `$unwind`, `$lookup`, `$addField`, `$facet`.
- Aggregation operators: `$sum`, `$avg`, `$min`, `$max`, `$first`, `$last`, `$push`, `$addToSet`, `$size`, `$regexMatch`.

7. Indexing:

- `db.collection_name.createIndex(keys, options)`: Create an index on specified keys.
- `db.collection_name.explain().find(query)`: Display the execution plan of a query.

8. Miscellaneous:

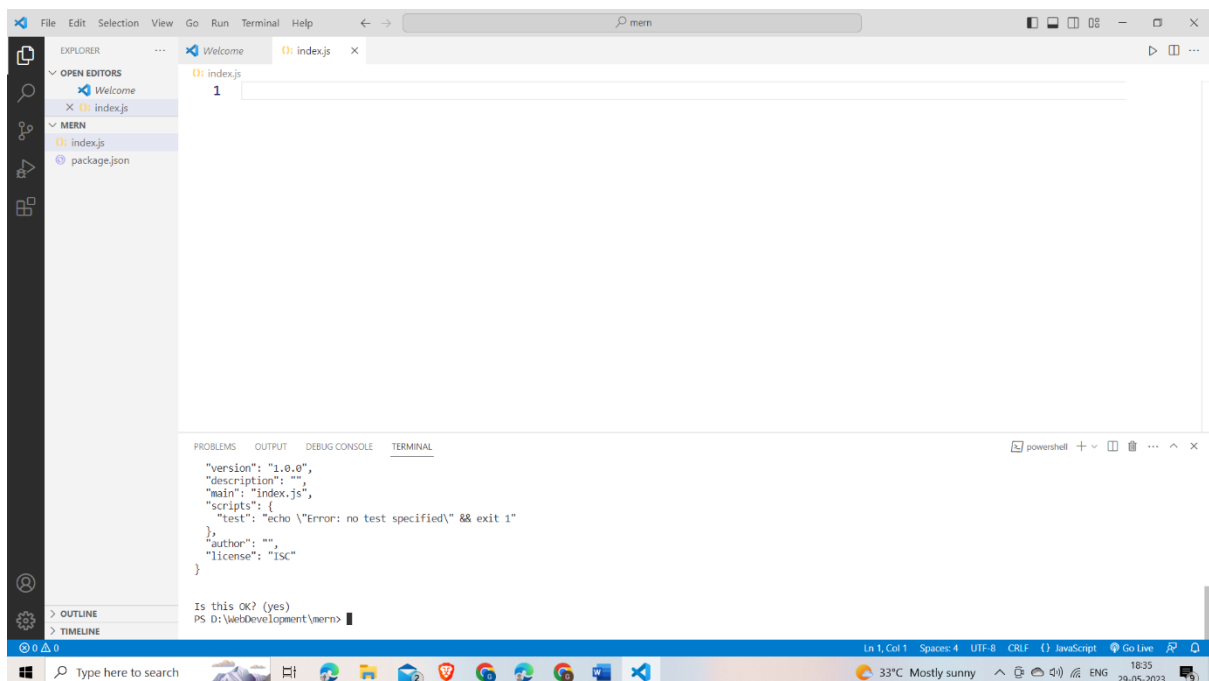
- ``db.collection_name.count(query)``: Count the number of documents that match a query.
- ``db.collection_name.find().sort(sort_criteria)``: Sort documents based on specified criteria.
- ``db.collection_name.distinct(field)``: Retrieve distinct values of a field in a collection.

Adding MongoDB Driver to Node.js

To use MongoDB with Node.js, you'll need to add the MongoDB driver to your Node.js project. Here's a step-by-step guide on how to add the MongoDB driver to Node.js:

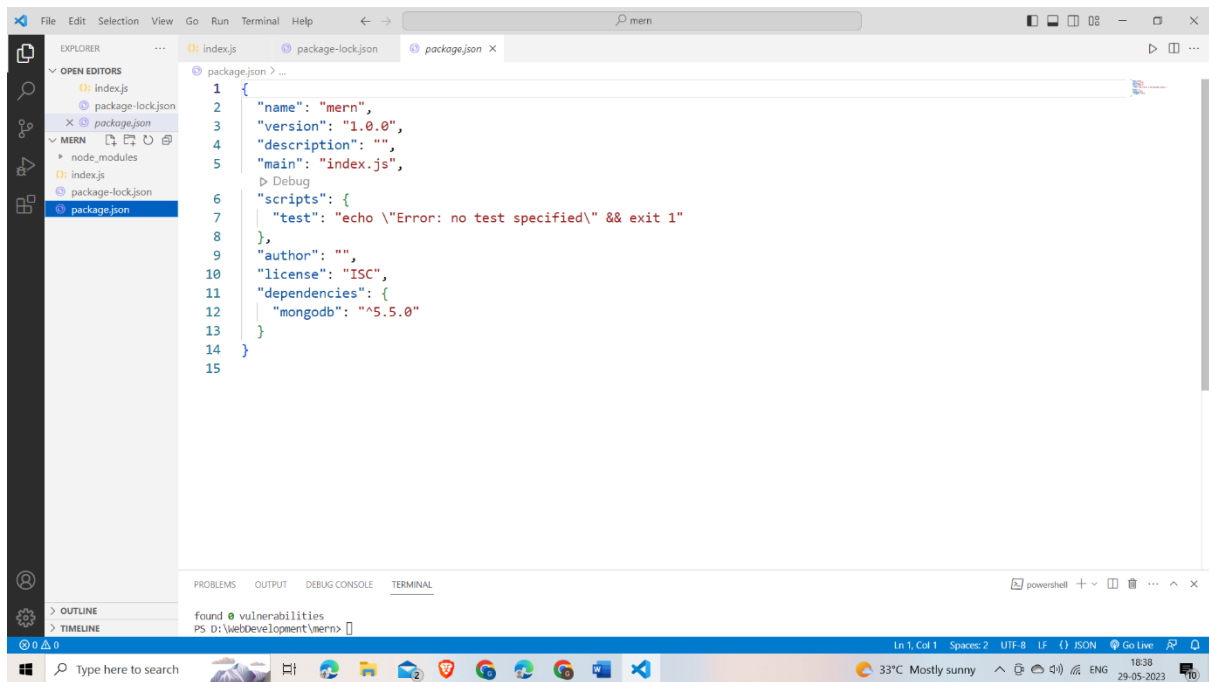
1. Initialize your Node.js project:

- Open your project directory in a terminal or command prompt.
- Run ``npm init`` and follow the prompts to initialize a new Node.js project. This will create a ``package.json`` file.



2. Install the MongoDB driver:

- Run ``npm install mongodb`` to install the MongoDB driver package from the npm registry.
- This command will download the MongoDB driver and add it as a dependency in your ``package.json`` file.



3. Import the MongoDB driver in your Node.js code:

- In your JavaScript file, import the MongoDB client using the following line:

```
'''
```

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

```
'''
```

4. Connect to a MongoDB database:

- To connect to a MongoDB database, use the `MongoClient` object and its `connect()` method. Here's an example:

```
const uri = 'mongodb://localhost:27017'; // Replace with your MongoDB connection string
```

```
const client = new MongoClient(uri);
```

```
async function connectToDatabase() {
```

```
  try {
```

```
    await client.connect();
```

```
    console.log('Connected to the database');
```

```

    // Perform database operations here

} catch (error) {

    console.error('Error connecting to the database', error);

} finally {

    // Close the connection when done

    await client.close();

}

}

connectToDatabase();

```

The screenshot shows a VS Code editor with a file explorer on the left showing a project named 'mern' with files 'index.js', 'package-lock.json', and 'package.json'. The main editor displays the following JavaScript code in 'index.js':

```

1  const { MongoClient } = require('mongodb');
2  const uri = 'mongodb://127.0.0.1:27017'; // Replace with your MongoDB connection string
3  const client = new MongoClient(uri);
4
5  async function connectToDatabase() {
6    try {
7      await client.connect();
8      console.log('Connected to the database');
9      // Perform database operations here
10   } catch (error) {
11     console.error('Error connecting to the database', error);
12   } finally {
13     // Close the connection when done
14     await client.close();
15   }
16 }
17 connectToDatabase();

```

Below the code editor, the 'TERMINAL' panel shows the output of running the command:

```

[Running] node "d:\WebDevelopment\mern\index.js"
Connected to the database

[Done] exited with code=0 in 0.463 seconds

```

The status bar at the bottom indicates the file is 'Ln 2, Col 33', uses 'Spaces: 4', 'UTF-8' encoding, and 'CRLF' line endings. It also shows the system status as '33°C Sunny' and the date '29-05-2023'.

5. Perform MongoDB operations:

- Once connected to the database, you can use the MongoDB driver to perform operations such as inserting, updating, querying, and deleting documents. Refer to the MongoDB Node.js driver documentation for more details and examples.

Remember to replace the `uri` variable with your actual MongoDB connection string, which should include the hostname, port number, and any necessary authentication details.

By following these steps, you'll be able to add the MongoDB driver to your Node.js project and start interacting with a MongoDB database.

Getting Started with MongoDB and Node.js

You can use several modules to access MongoDB from your Node.js applications. The MongoDB group adopted the MongoDB Node.js driver as the standard method. This driver provides all the functionality and is similar to the native commands available in the MongoDB shell client.

Adding the MongoDB Driver to Node.js

The first step in implementing MongoDB access from your Node.js applications is to add the MongoDB driver to your application project. The MongoDB Node.js driver is the officially supported native Node.js driver for MongoDB. It has by far the best implementation and is sponsored by MongoDB.

Adding the MongoDB Node.js driver to your project is a simple npm command. From your project root directory, execute the following command using a console prompt: **npm install mongodb**

A `node_modules` directory is created if it is not already there, and the `mongodb` driver module is installed under it. Once that is done, your Node.js application files can use the **`require('mongodb')`** command to access the `mongodb` module functionality.

Connecting to MongoDB from Node.js

Once you have installed the `mongodb` module using the npm command, you can begin accessing MongoDB from your Node.js applications by opening up a connection to the MongoDB server. The connection acts as your interface to create, update, and access data in the MongoDB database.

Accessing MongoDB is best done through the `MongoClient` class in the `mongodb` module.

This class provides two main methods to create connections to MongoDB. One is to create an instance of the `MongoClient` object and then use that object to create and manage the

MongoDB connection. The other method uses a connection string to connect. Either of these options works well.

Connecting to MongoDB from Node.js Using the MongoClient Object

Using a MongoClient object to connect to MongoDB involves creating an instance of the client, opening a connection to the database, authenticating to the database if necessary, and then handling logout and closure as needed.

To connect to MongoDB via a MongoClient object, first create an instance of the MongoClient object using the following syntax:

```
var client = new MongoClient();
```

After you have created the MongoClient, you still need to open a connection to the MongoDB server database using the connect(url, options, callback) method. The url is composed of several components listed in Table 13.2. The following syntax is used for these options:

```
mongodb://[username:password@]host[:port][/][database][?options]]
```

For example, to connect to a MongoDB database named MyDB on a host named MyDBServer on port 8088, you would use the following URL:

```
client.connect('mongodb://MyDBServer:8088/MyDB');
```

Table 13.2 MongoClient connection url components

Option	Description
mongodb://	Specifies that this string is using a MongoDB connection format.
username	(Optional) Specifies the user name to use when authenticating.
password	(Optional) Specifies the password to use when authenticating.
host	Specifies the host name or address of the MongoDB server. You can specify multiple host:port combinations to connect to multiple MongoDB servers by separating them by a comma. For example: mongodb://host1:270017,host2:27017,host3:27017/testDB
port	Specifies the port to use when connecting to the MongoDB server. Default is 27017.
database	Specifies the database name to connect to. Default is admin.
options	Specifies the key/value pairs of options to use when connecting. These same options can be specified in the dbOpt and serverOpt parameters.

In addition to the connection url information, you can also provide an options object that specifies how the MongoClient object creates and manages the connection to MongoDB. This options object is the second parameter to the connect() method.

For example, the following code shows connecting to MongoDB with a reconnect interval of 500 and a connection timeout of 1000 milliseconds:

```
client.connect ('mongodb://MyDBServer:8088/MyDB',{
  connectTimeoutMS: 1000,reconnectInterval: 500 },

  function(err, db){ . . . });
```

Table 13.3 lists the most important settings in the options object that you can set when defining the MongoClient object. The callback method is called back with an error as the first parameter if the connection fails or with a MongoClient object as the second parameter if the connection is successful.

Table 13.3 Options used to create the server object for the MongoClient

Option	Description
readPreference	Specifies which read preference to use when reading objects from a replica set. Setting the read preference allows you to optimize read operations. For example, read only from secondary servers to free up primary. <ul style="list-style-type: none"> ReadPreference.PRIMARY ReadPreference.PRIMARY_PREFERRED ReadPreference.SECONDARY ReadPreference.SECONDARY_PREFERRED ReadPreference.NEAREST
ssl	A Boolean that, when true, specifies that the connection uses SSL. The mongod also needs to be configured with SSL. If you are using ssl, you can also specify the sslCA, sslCert, sslKey, and sslPass options to set the SSL certificate authority, certificate, key, and password.
poolSize	Specifies the number of connections to use in the connection pool for the server. Default is 5, meaning there can be up to five connections to the database shared by the MongoClient.
ReconnectInterval	Specifies the amount of time in milliseconds that the server waits between retries.
auto_reconnect	A Boolean that, when true, specifies whether the client will try to re-create the connection when an error is encountered.
readConcern	Sets the read concern for the collection.
w	Sets the write concern. (See Table 13.1.)
wTimeout	Sets the timeout value of the write concern.
reconnectTries	Sets the number of times the server attempts to reconnect.
nodelay	A Boolean that specifies a no-delay socket.
keepAlive	Specifies the keepalive amount for the socket.
connectionTimeout	Specifies the amount of time in milliseconds for the connection to wait before timing out.
socketTimeout	Specifies the amount of time in milliseconds for a socket send to wait before timing out.

The callback function accepts an error as the first parameter, and a Db object instance as the second parameter. If an error occurs, the Db object instance will be null; otherwise, you can use it to access the database because the connection will already be created and authenticated.

While in the callback function, you can access the MongoDB database using the Db object passed in as the second parameter. When you are finished with the connection, call close() on the Db object to close the connection.

.Listing 13.1 db_connect_url.js: Connecting to MongoDB

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/SampleDB');

const db = mongoose.connection;

db.on('error',console.error.bind(console,'Error in Connection'));

db.once('open',function(){

    console.log("Database connection is Successful");

});
```

Output

Database connection is Successful