UNIT III MONGO DB

Syllabus :

Understanding NoSQL and MongoDB – Building MongoDB Environment – User accounts – Access control – Administering databases – Managing collections – Connecting to MongoDB from Node.js – simple applications

Understanding NoSQL

- Agenda
  - Introduction
  - SQL
  - NoSQL
  - Key features of NoSQL
  - Categories of NoSQL

Understanding NoSQL

Good web applications must **store and retrieve data with accuracy, speed, and reliability.** Therefore, the data storage mechanism is done with the help of a **database**.

A database is a **collection of data, usually stored in electronic form.**

Database types

- **SQL(Structured Query Language)**
- **NO SQL(Not Only / Non Structured Query Language)**

SQL

SQL (Structured Query Language) is a computer based structured, formatted database language designed for managing data in Relational Database Management Systems (RDBMS).

NoSQL

- It is a type of DBMS that is designed to handle and store large volumes of unstructured and semi-structured data.
- The term NoSQL originally referred to "non-SQL" or "non-relational" databases, but the term has since evolved to mean "not only SQL," as NoSQL databases have expanded to include a wide range of different database architectures and data models.
- MongoDB is one of the most popular and well supported NoSQL databases currently available

Key Features of NoSQL

- **Dynamic schema:** No fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.
- **Horizontal scalability:** NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
- **Document-based:** Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in semi-structured format, such as JSON or BSON.
- **Key-value-based:** Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
- **Distributed and high availability:** NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
- **Flexibility:** NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
- **Performance:** NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

NoSQL databases are generally classified into four main categories:

- **Document databases:** These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.

  Examples – Mongo DB, Couch DB

- **Key-value stores:** These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.

  Examples – Redis, Coherence

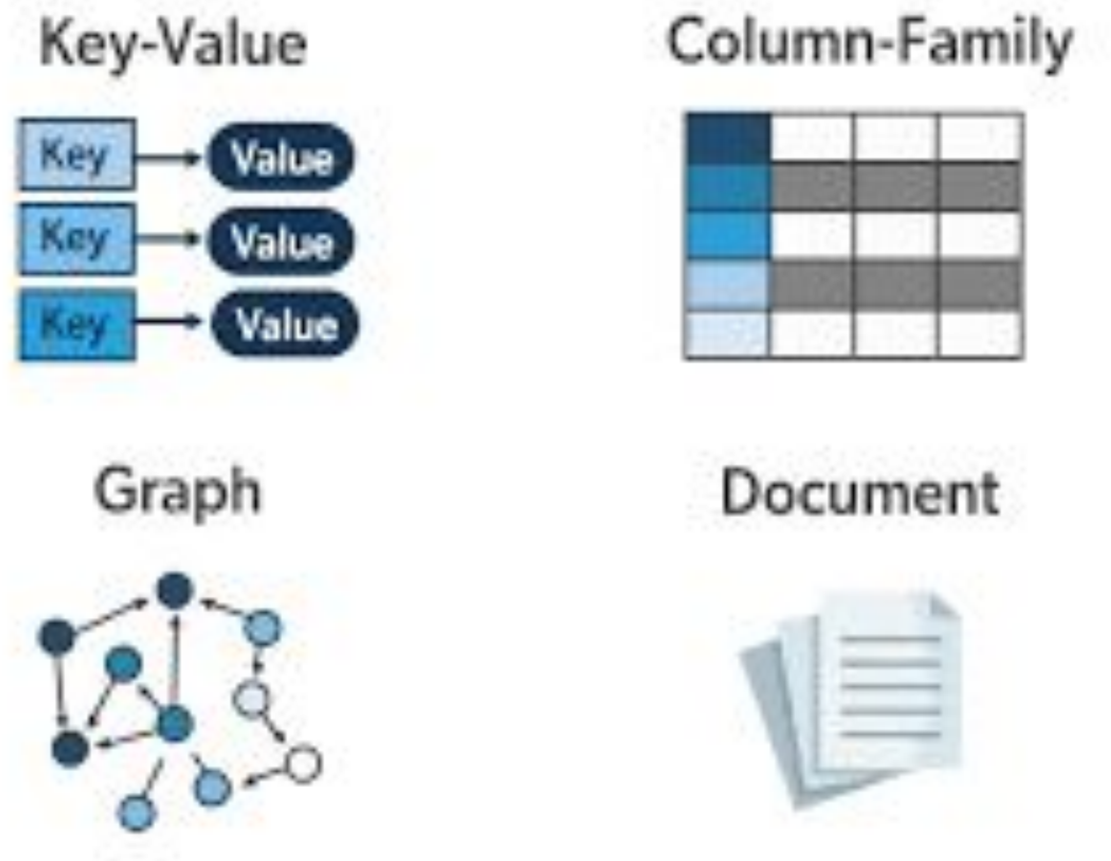
- **Column-family stores:** These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.

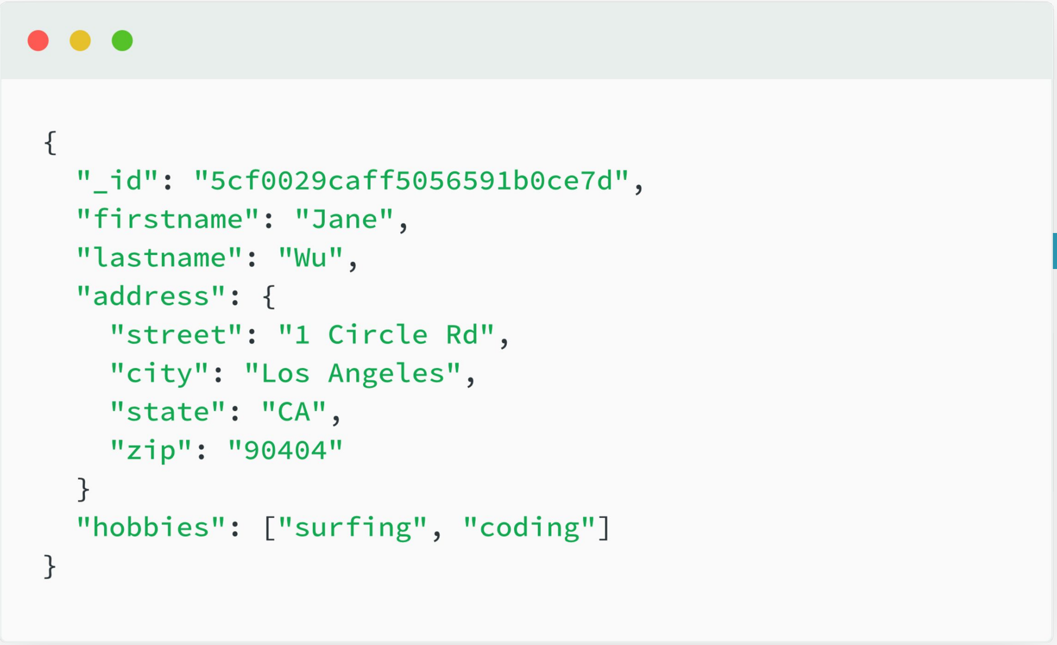
  Examples – Hbase, Big Table

- **Graph databases:** These databases store data as nodes and edges, and are designed to handle complex relationships between data.

  Examples – Amazon Neptune

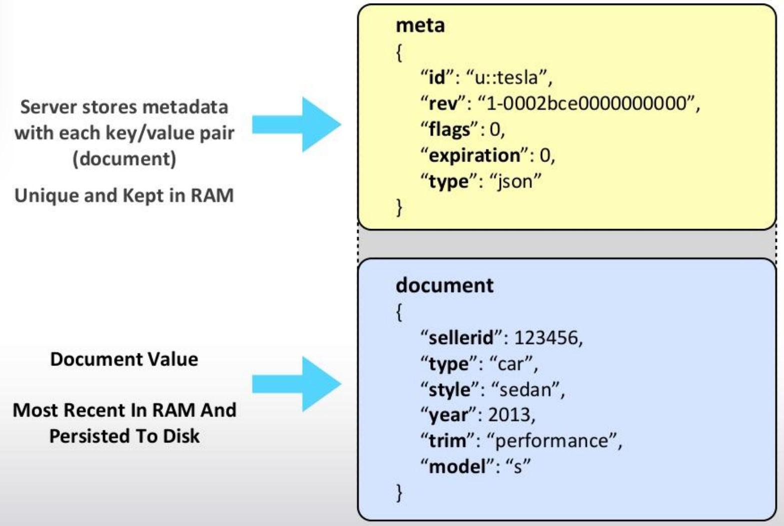
NoSQL


Key-Value
Key
Value
Key
Value
Key
Value
Column-Family
Graph
Document



```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  }
  "hobbies": ["surfing", "coding"]
}
```

Couchbase Data



Server stores metadata with each key/value pair (document)

Unique and Kept in RAM

meta

```
{
  "id": "u::tesla",
  "rev": "1-0002bce0000000000",
  "flags": 0,
  "expiration": 0,
  "type": "json"
}
```

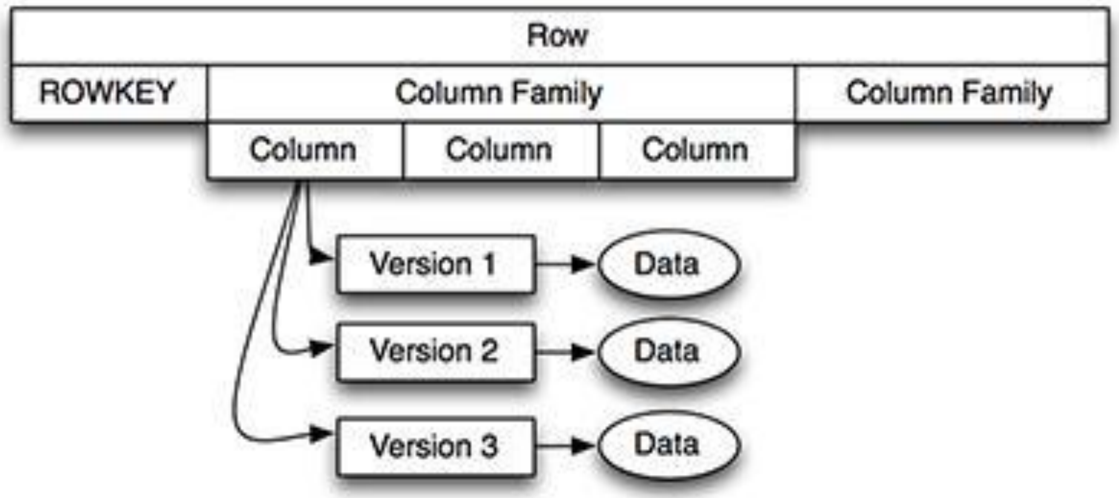
Document Value

Most Recent In RAM And Persisted To Disk

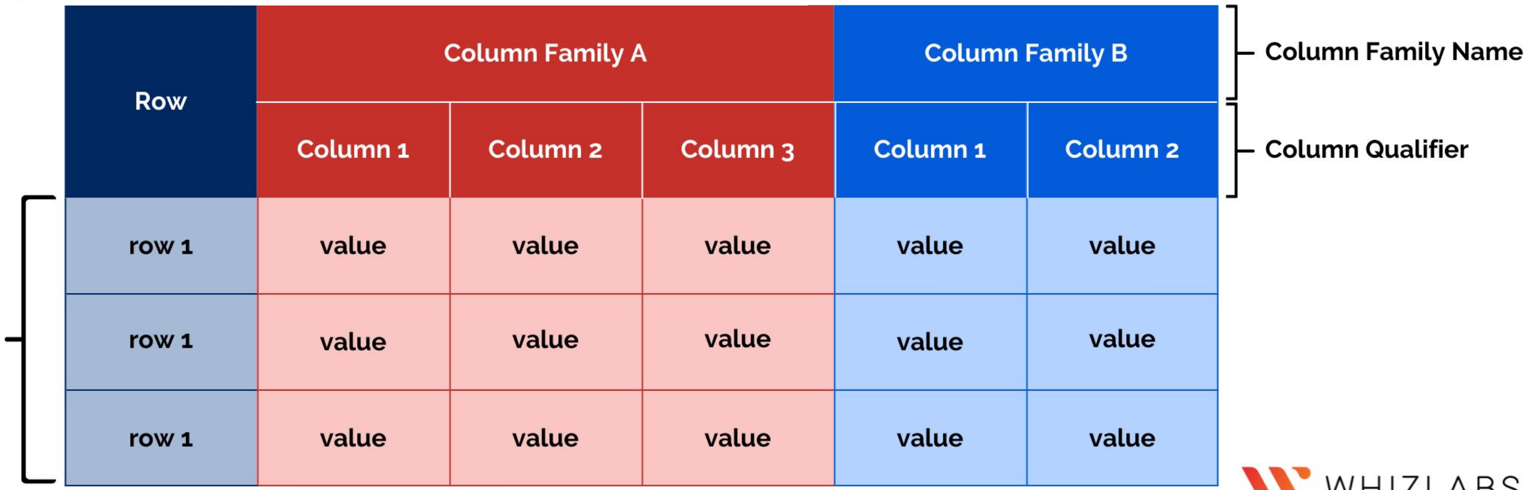
document

```
{
  "sellerid": 123456,
  "type": "car",
  "style": "sedan",
  "year": 2013,
  "trim": "performance",
  "model": "s"
}
```

HBASE

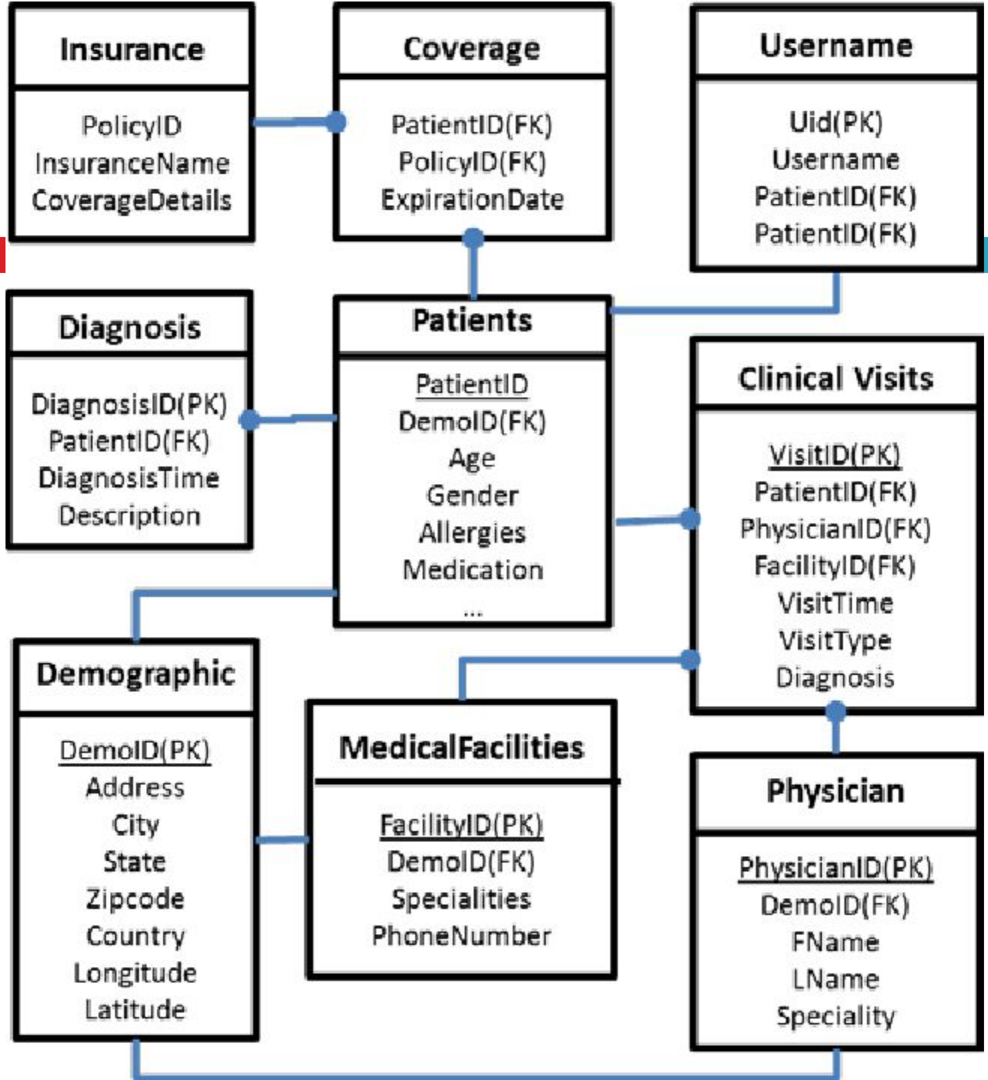


BIG TABLE

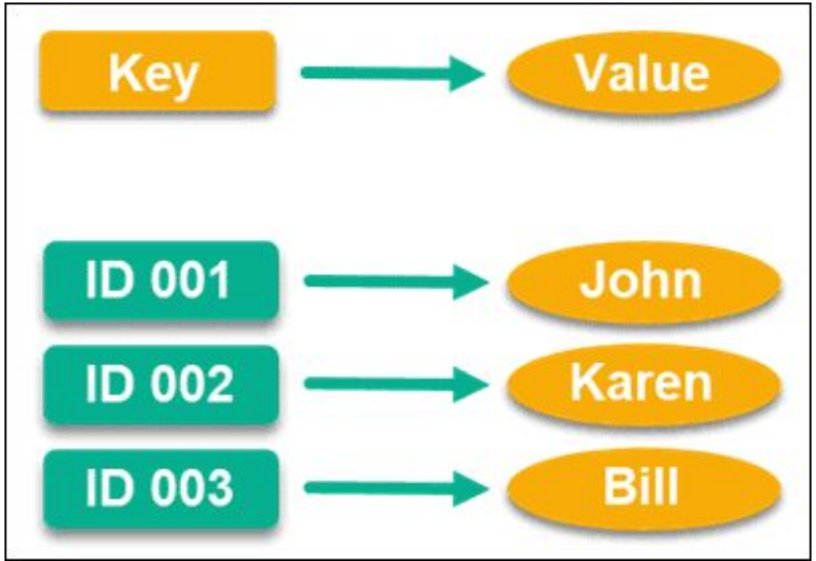

Column Family Name
Column Qualifier
Row Key

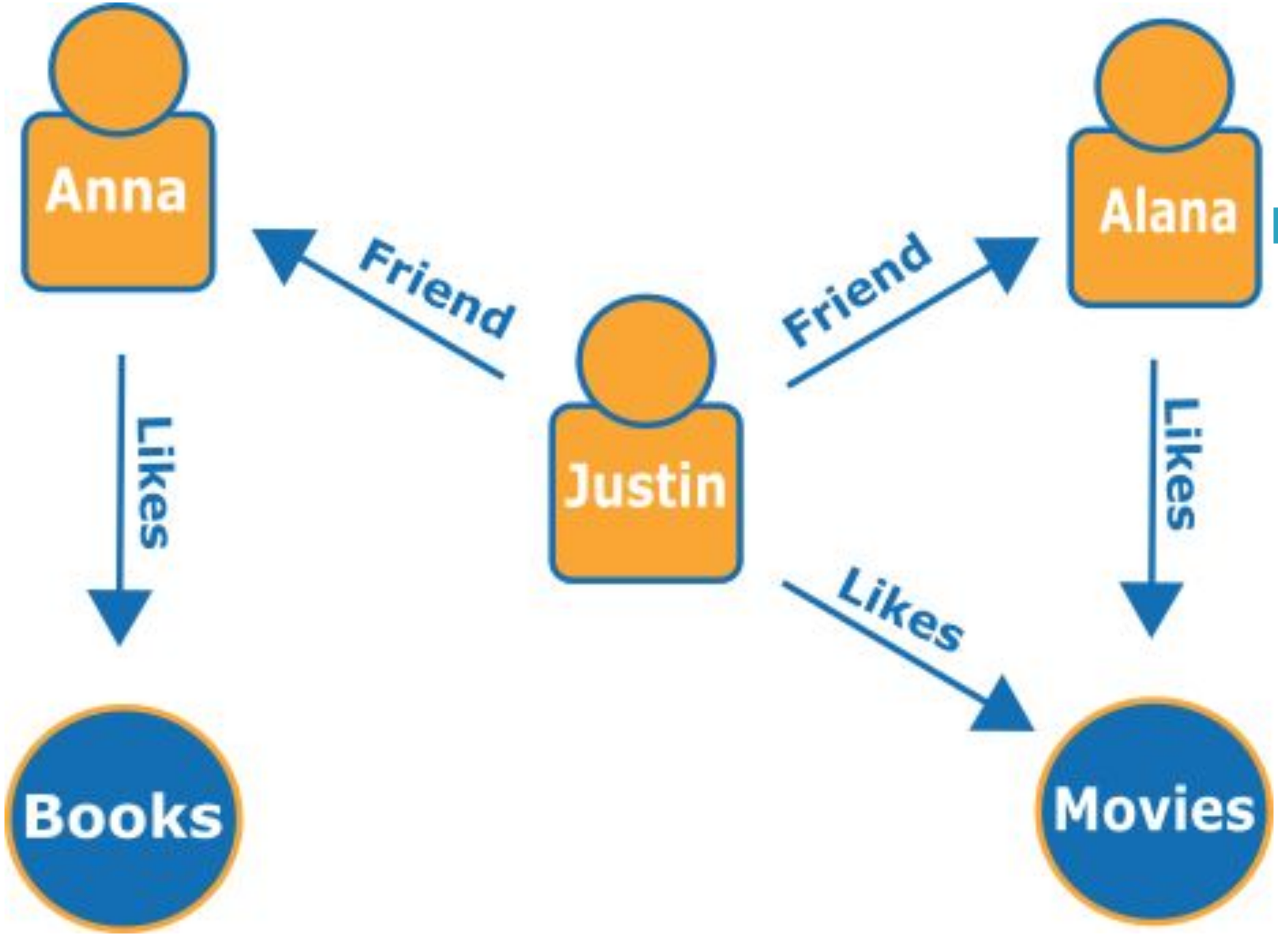
Row	Column Family A			Column Family B	
	Column 1	Column 2	Column 3	Column 1	Column 2
row 1	value	value	value	value	value
row 1	value	value	value	value	value
row 1	value	value	value	value	value

Coherence



Redis





Amazon Neptune

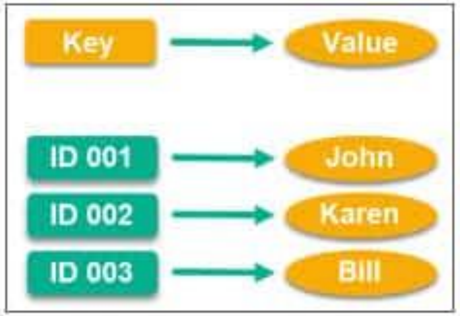
SQL Databases

Table

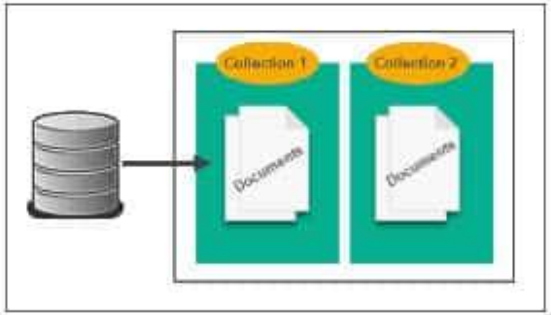
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

NoSQL Databases

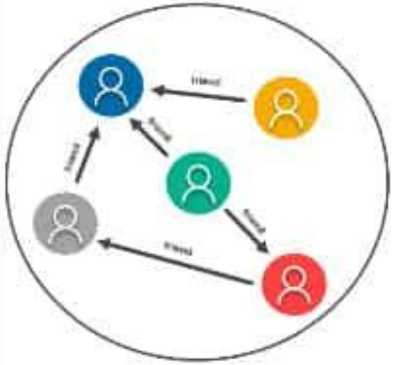
Key-value



Document



Graph



Wide-column

Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

Understanding MongoDB

- Agenda
  - Why use MongoDB?
  - Where to use MongoDB?
  - RDBMS vs MongoDB.
  - MongoDB datatypes.

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.

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

- MongoDB is a document database. It stores data in a type of JSON format called BSON.
- Records in a MongoDB database are called documents, and the field values may include numbers, strings, booleans, arrays, or even nested documents.

Example

Sample Document:

```
{
    name: "New Project",
    version: 1,
    languages: ["JavaScript", "HTML", "CSS"],
    admin: {name: "Brad", password: "****"}
}
```

Notice that the document structure contains fields/properties that are strings, integers, arrays, and objects, just like a JavaScript object.

Why Use MongoDB?

- Document Oriented Storage.
- Index on any attribute
- Replication and high availability
- Rich queries
- Fast in-place updates

Where to Use Mongo DB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Internet of Things

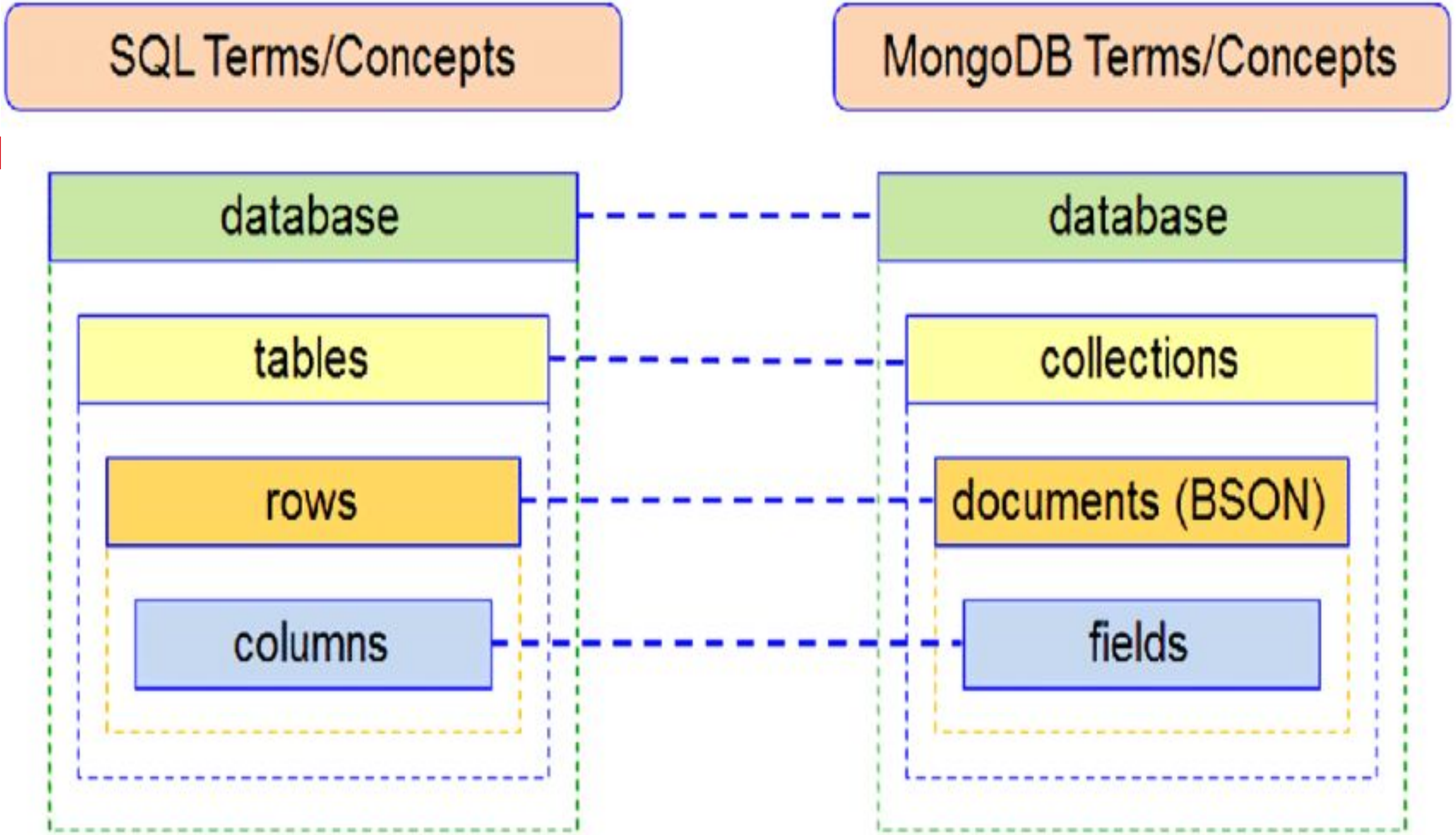
Mobile applications

Real time analysis

Personalization

Catalog management

Content management


SQL Terms/Concepts
MongoDB Terms/Concepts
database
database
tables
collections
rows
documents (BSON)
columns
fields

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.
- Complex transactions are not supported in MongoDB because complex join operations are not available.
- MongoDB allows a highly flexible and scalable document structure.
- MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques.

MongoDB supports many data types. Some of them are −

- **String** − This is the most commonly used data type to store the data.
- **Integer** − This type is used to store a numerical value
- **Boolean** − This type is used to store a Boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest JSON elements.
- **Arrays** − This type is used to multiple values into one key.
- **Date** − This data type is used to store the current date
- **Object ID** − This data type is used to store the document's ID.

Activity

- Exit Slip!

Building MongoDB Environment

- Objective – Learn to install MongoDB

Building MongoDB Environment on Windows

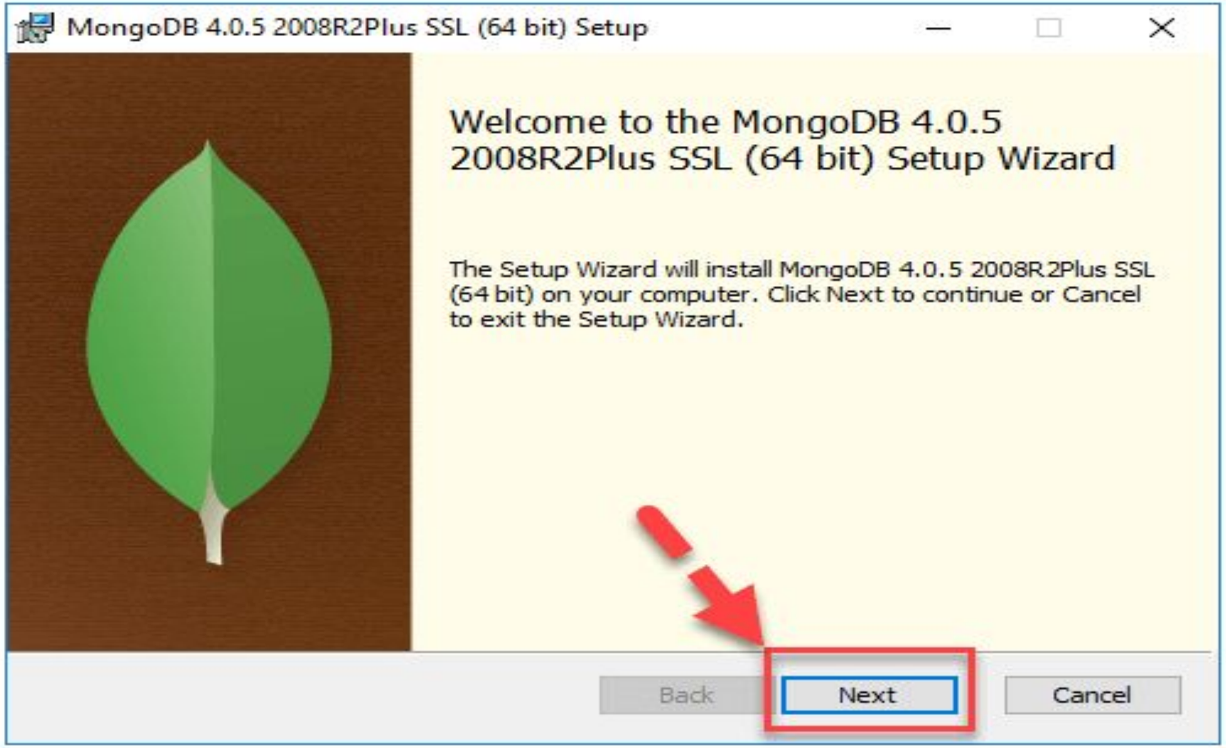
Step 1: Download MongoDB Community Server,

Go to link and

Download MongoDB Community Server.

We will install the 64-bit version for Windows.

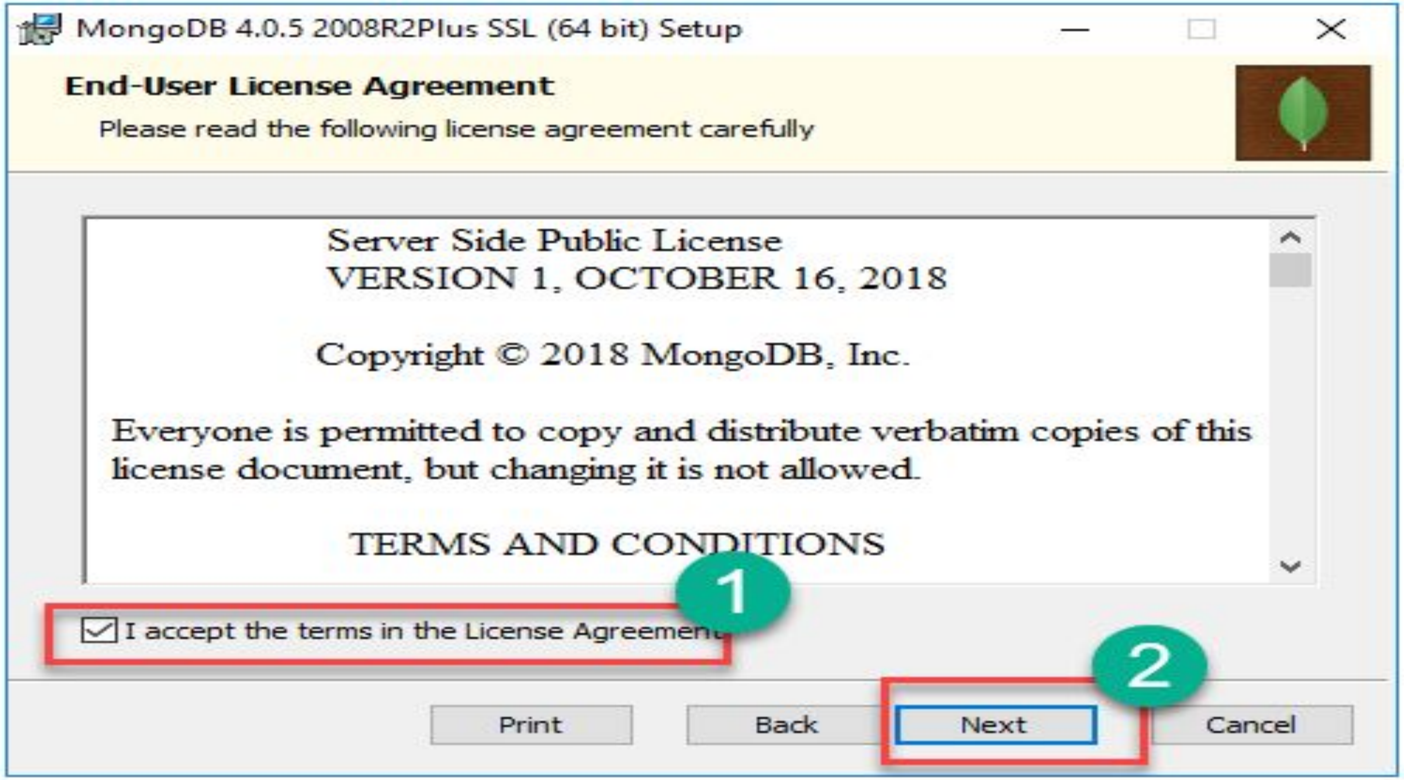
Step 2 : Click on Setup. Once download is complete open the msi file. Click Next in the start up screen


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 4.0.5 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step 3) Accept the End-User License Agreement
Click Next


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

End-User License Agreement

Please read the following license agreement carefully

Server Side Public License
VERSION 1, OCTOBER 16, 2018

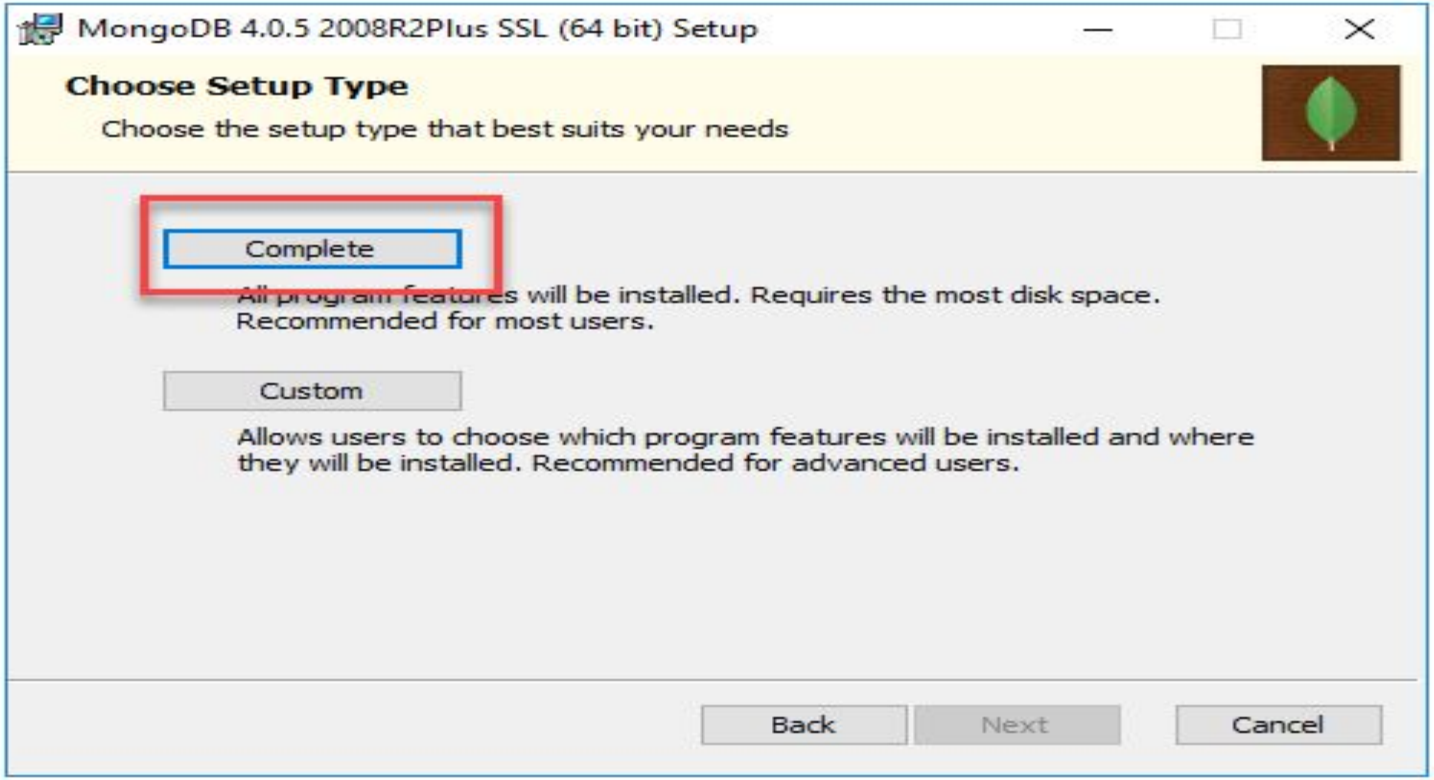
Copyright © 2018 MongoDB, Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

TERMS AND CONDITIONS

[✓] I accept the terms in the License Agreement

Step 4) Click on the "complete" button to install all of the components



MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

Complete

All program features will be installed. Requires the most disk space. Recommended for most users.

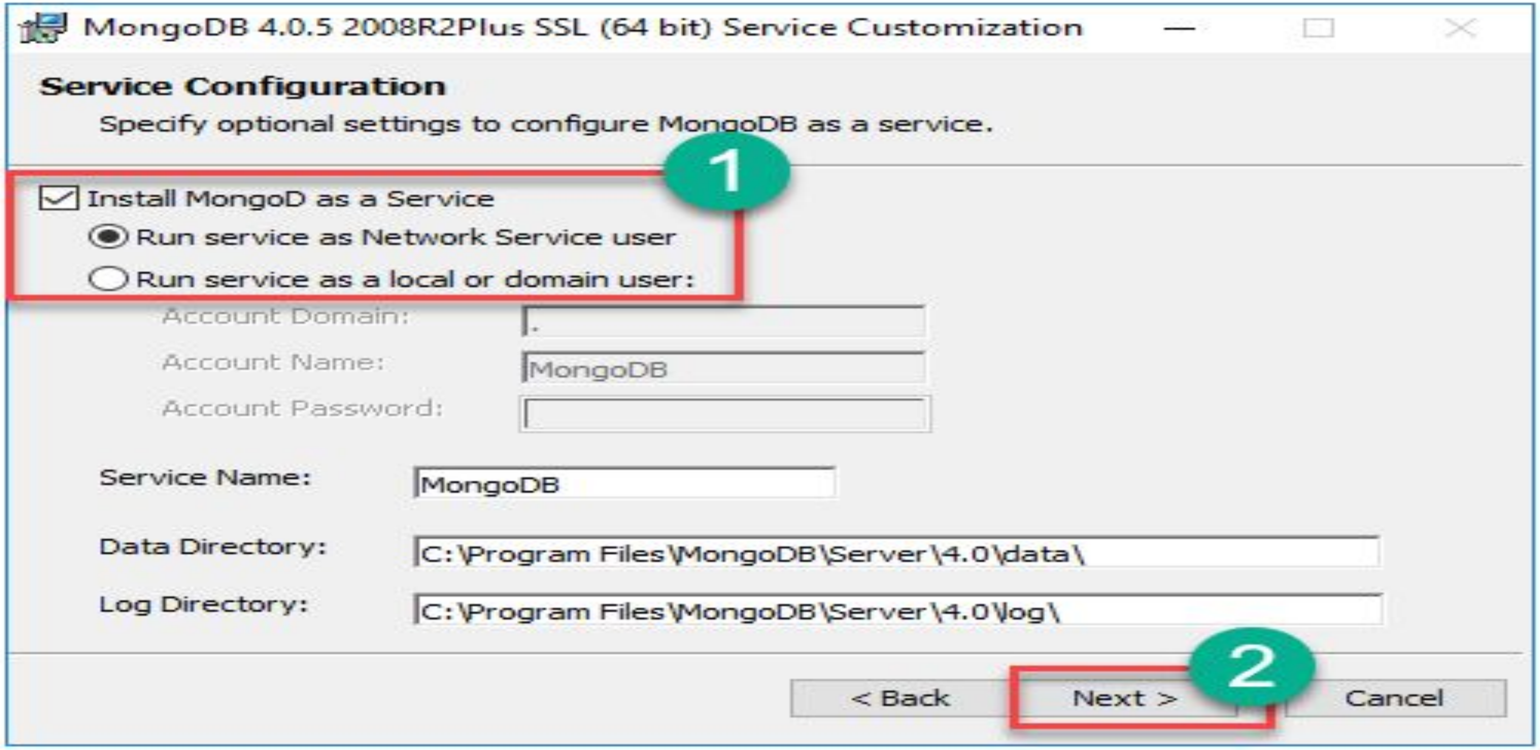
Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

Back Next Cancel

Step 5) Service Configuration

Select “Run service as Network Service user”. Click Next



MongoDB 4.0.5 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

[✓] Install MongoD as a Service

[● Run service as Network Service user]

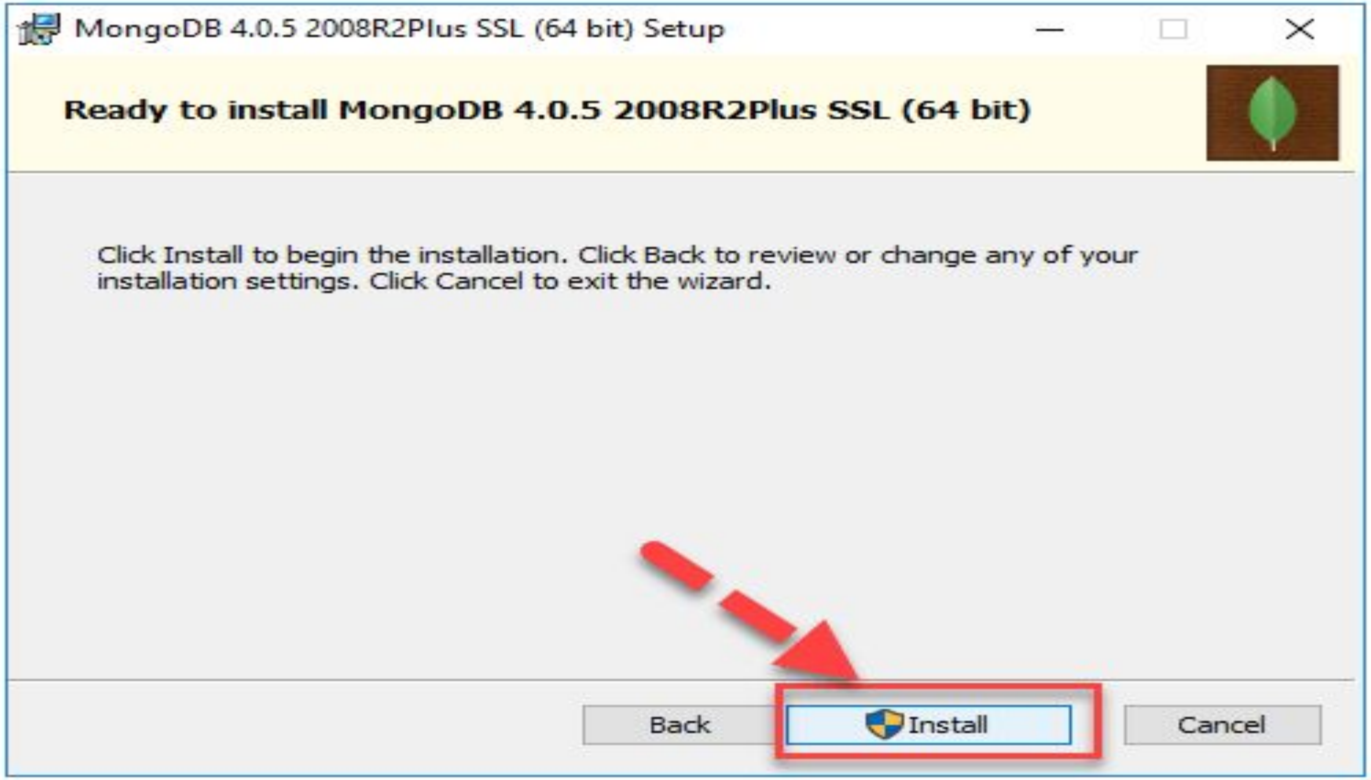
○ Run service as a local or domain user:

Account Domain:	.
Account Name:	MongoDB
Account Password:	

Service Name:	MongoDB
Data Directory:	C:\Program Files\MongoDB\Server\4.0\data\
Log Directory:	C:\Program Files\MongoDB\Server\4.0\log\

Step 6) Start installation process

Click on the Install button to start the installation.

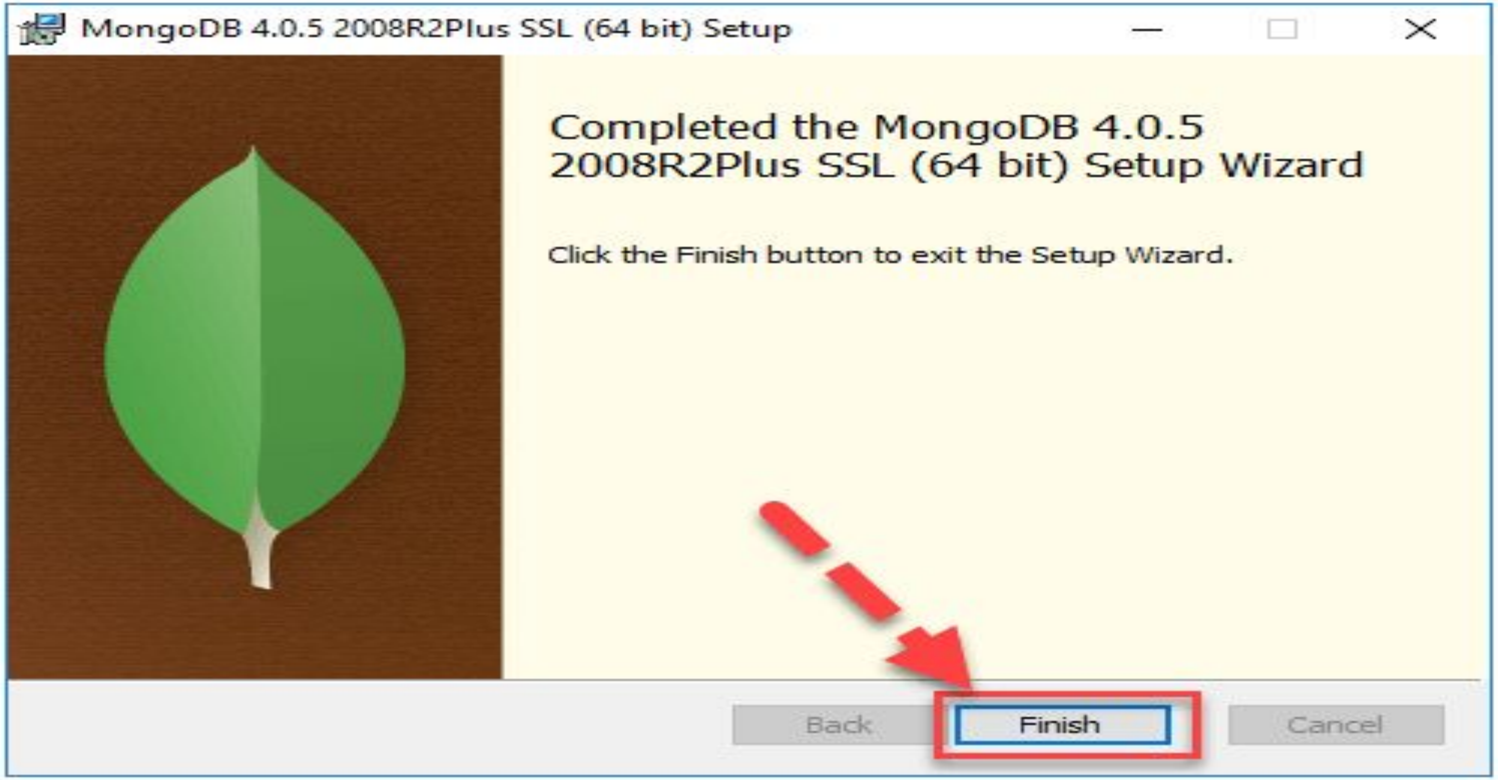


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 4.0.5 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 7) Click Next once completed,   Installation begins. Click Next once completed
Step 8) Click on the Finish button, Final step, Once complete the installation, Click on the Finish button

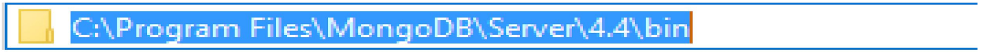

MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Completed the MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup Wizard

Click the Finish button to exit the Setup Wizard.

To set environment variable

Step 1: go to the location where MongoDB installed in copy the bin path



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

Step 2: Now, to create an environment variable open system properties << Environment Variable << System variable << path << Edit Environment variable and paste the copied link to your environment system and click Ok:

Step 3: After cmd type **mongod**

Type **mongod –version** command to know the version

mongoshell

MongoDB Shell commands

- **help <option>:** The option argument allows you to specify an area where you want help.
- **use <database>:** Database operations are processed on the current database handle.
- **db.help:** Displays help options for the database methods.
- **show <option>:** Shows a list based on the option argument. The value of option can be:
  - **dbs:** Displays a list of databases.
  - **collections:** Displays a list of collections for the current database.
  - **profile:** Displays the five most recent system. profile entries taking more than 1 millisecond.
  - **databases:** Displays a list of all available databases.
  - **roles:** Displays a list of all roles for the current database, both built-in anduser-defined.
  - **users:** Displays a list of all users for that database.
- **exit:** Exits the database.

MongoDB Shell Methods

- **load(script):** Loads and runs a JavaScript file inside the shell. This is a great way to script operations for the database.
- **UUID(string):** Converts a 32-byte hex string into a BSON UUID.
- **db.auth(username, password):** Authenticates you to the current database.

Administering Databases(CRUD operations)

1. Displaying a List of Databases
   - show dbs
2. Changing the Current Database
   - use fullstack
   - db – To check current Database

3. Creating Databases

Your created database (fullstack) is not present in list. To display database, you need create collection and need to insert at least one document into it.

- db.createCollection(“student”)
- show collections – To see existing Collections

4. Deleting Databases
   - db.student.drop() – To drop collection

5. db.dropdatabase() – to delete Default Database

6. Managing Collections

a) Displaying a List of Collections in a Database
   - show collections

b) Creating Collections
   - db.createCollection(“student“)

c) Deleting Collections
   - db.student.drop() – To drop collection

7. Adding Documents to a Collection

- db.student.insertOne({'name’ : 'raju’}) – To add single document
- db.student.insertMany([{"name":"raju"}, {"name":"rao"}]) – To add Many documents

8. Finding Documents in a Collection

- db.student.find() – To view documents in student collection
- db.student.findOne() – To view First Document in Student Collection

9. Deleting Documents in a Collection

- coll.remove({'name' : 'raju'}) – To remove a Single Document
- coll.remove() – To remove a All Documents

Deleting Databases

- db.student.drop() – To drop collection
- db.dropdatabase() – to delete Default Database

10. Updating Documents in a Collection

- db.student.update({'name':raju'},{$set:{name ':'ram'}}) – To update document

11. Quit the mongo shell

- exit

Example

```
db.empDetails.insertOne(
 {
First_Name: “xyzvudutha",
Last_Name: “abc”,
email: “abc.5021@sxcce.edu.in",
 phone: "9848022338" }
)
```

```
db.posts.insertOne
({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  }
])
```

Database Update Operators

MongoDB provides different types of field update operators to update the values of the fields of the documents that matches the specified condition. The following table contains the field update operators:

Operator	Description
$currentDate	This operator is used to set the value of a field to current date, either as a Date or a Timestamp.
$inc	This operator is used to increment the value of the field by the specified amount.
$min	This operator is used only to update the field if the specified value is less than the existing field value
$max	This operator is used only to update the field if the specified value is greater than the existing field value.
$mul	This operator is used to multiply the value of the field by the specified amount.
$rename	This operator is used to rename a field.
$set	The $set operator replaces the value of a field with the specified value.

Update using $rename

- The $rename operator updates the name of a field and has the following form:

{$rename: { <field1>: <newName1>, <field2>: <newName2>, ... } }

Example :

```
db.posts.updateOne( { title: "Post Title 1" }, { $rename: { Category: Meet } } )
```

*****UpdateOne() or Updatemany() can use******

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

Examples of $Set

```
db.products.insertOne(
{
_id: 100,
quantity: 250,
instock: true,
reorder: false,
details: { model: "14QQ", make: "Clothes
Corp" },
tags: [ "apparel", "clothing" ],
ratings: [ { by: "Customer007", rating: 4 } ]
}
)
```

```
db.products.updateOne(
{ _id: 100 },
{ $set:
{
quantity: 500,
details: { model: "2600", make: "Fashionaires"
},
tags: [ "coats", "outerwear", "clothing" ]
}
}
)
```

$inc – to add 5000 to Salary as Annual Increment
db.employee.updateOne({“EmpNo": “1134"}, {$inc: {Salary: 5000}})

$min – to set Book Price as 500. It will be updated only if the existing price is more than 500
db.book.updateOne({“BookId": “1"}, {$min: {price: 500}})

$max – to set Book Price as 600. It will be updated only if the existing price is less than 500
db.book.updateOne({“BookId": “1"}, {$min: {price: 600}})

$mul – to multiply the value of field by a number
db.book.updateOne({“BookId": “1"}, {$mul: {price: NumberDecimal(“100"), Quantity: 2}})

$rename – The rename operator changes the name of a field.
db.book.updateOne({}, {$rename: {“bookTitle”, “BookTitle”}})

Administering User Accounts & Configuration of Access Control

Create Administrator User for Any Database

Creating a user administrator in MongoDB is done by using the createUser method.

```
db.createUser(
{ user: “admin",
pwd: “123",
roles: [ ]
} )
```

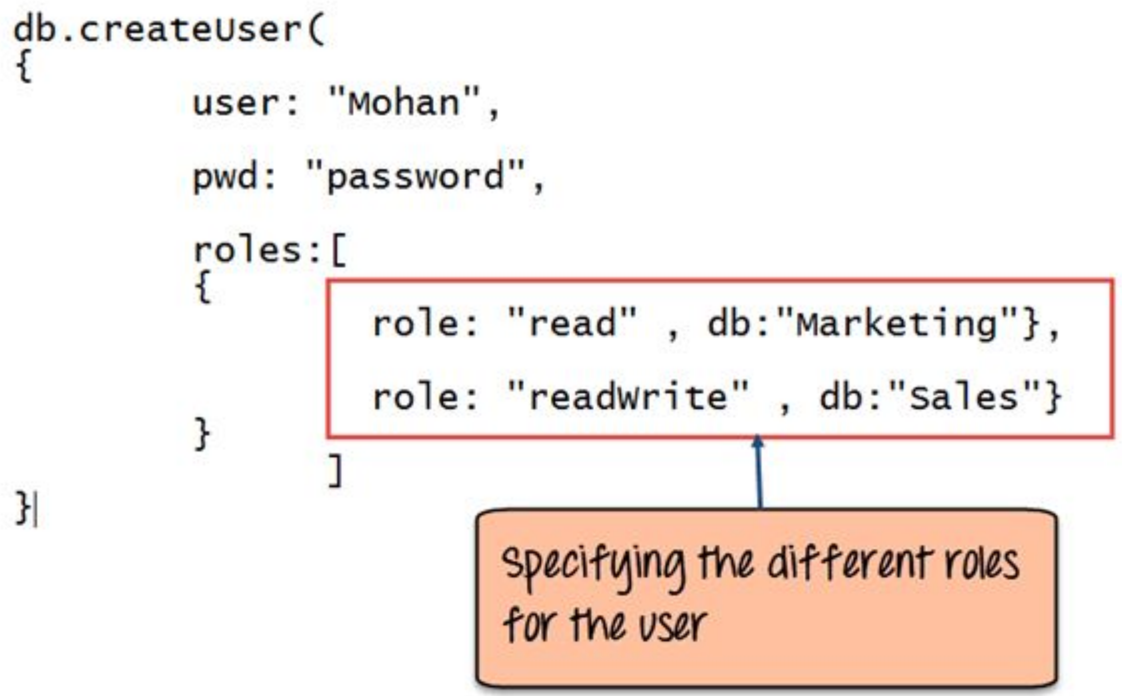
Create User for Single Database

To create a user who will manage a single database, we can use the same command as mentioned above but we need to use the “userAdmin” option only.

```
db.createUser(
{
user: “admin",
pwd: “123",
roles: [{role: “admin", db:”fdp"}]
} )
```

Managing users

For example, there is a the “read role” which only allows read only access to databases and then there is the “readwrite” role which provides read and write access to the database , which means that the user can issue the insert, delete and update commands on collections in that database.



```
db.createUser(
{
        user: "Mohan",

        pwd: "password",

        roles:[
        {
                role: "read" , db:"Marketing"},

                role: "readWrite" , db:"Sales"}
        }
                ]
}
```

```
db.createUser(
  {
    user: "reportUser256",
    pwd: passwordPrompt(),
    roles: [ { role: "readWrite", db: "reporting" } ],
})
```

For authentication go to mongo.cfg
add

```
security :
authentication :enable
```

Services mongo restart as admin

```
db.auth( “username”, “password” ) //to check user authentication
db.changeUserPassword(“username”, “new password”)
db.dropUser("reportUser1")
db.dropAllUser()
```

Assignment

1. Start Mongocompass
2. Create FDP Database
3. Create faculty Collection
4. Insert faculty Data (Id No and Name)
   a) Single Record – Your Details
   b) Two or More Records – Your friends Details
5. Show Collection List
6. Show Database List
7. Find First Document
8. Find Document by Criteria
9. Update faculty Data by Criteria
10. Remove a Document by Criteria
11. Remove All Documents
12. Drop faculty Collection
13. Drop FDP Database

Array Update Operators

Name	**Description**
$	Acts as a placeholder to update the first element that matches the query condition.
$[]	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
$[<identifier>]	Acts as a placeholder to update all elements that match the arrayFilters condition for the documents that match the query condition.
$addToSet	Adds elements to an array only if they do not already exist in the set.
$pop	Removes the first or last item of an array.
$pull	Removes all array elements that match a specified query.
$push	Adds an item to an array.
$pullAll	Removes all matching values from an array.

Update Operator Modifiers

Name	Description
$each	Modifies the $push and $addToSet operators to append multiple items for array updates.
$position	Modifies the $push operator to specify the position in the array to add elements.
$slice	Modifies the $push operator to limit the size of updated arrays.
$sort	Modifies the $push operator to reorder documents stored in an array.

Cursor

- The **Cursor** is a **MongoDB Collection** of the document which is returned upon the find method execution.
- It is just like a pointer which is pointing upon a specific index value.
- All the documents which are returned are saved in a virtual cursor.

- Example
  - db.student().find()
  - db.student().find().count() - count how many records are there.
  - db.student().find().limit(2) - fetch limited records from a cursor.
  - db.student().find().pretty() - return all the documents in the form of JSON

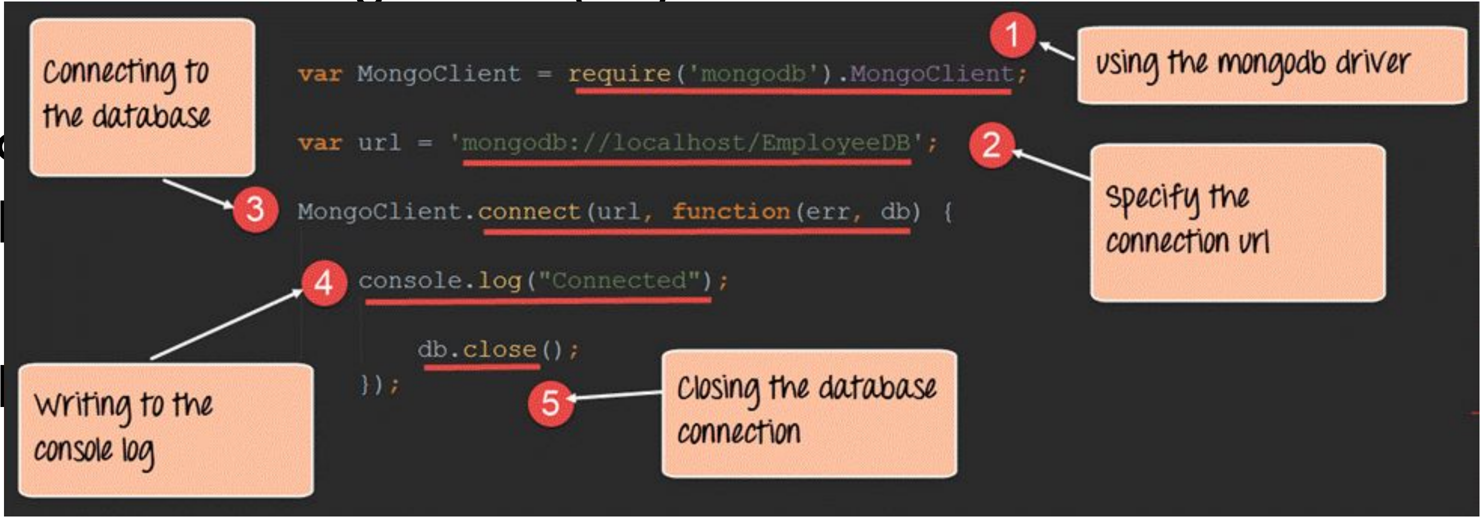
Connecting to Mongo DB from Node.js

<u>Node Js to mongo connection:</u>

- Node js can act as server for user interfaces.
- So, the node can be used to connect to database server, like: mongodb
- The module we need to install is : mongodb
- **npm install mongodb** Options are g(global), i(interactive).
- It contains a module to connect to database server.

Example-1

```
const {MongoClient}=require('mongodb');
const url = 'mongodb://127.0.0.1:27017';
const client=new MongoClient(url)
try{
   client.
console.l
}catch(e
console.l
```



```
var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    console.log("Connected");

        db.close();
    });
```

1
using the mongodb driver
2
Specify the connection url
Connecting to the database
3
4
Writing to the console log
5
Closing the database connection

Example-2
Insert ,Update , Read operations

maindb.js

```
const {MongoClient}=require('mongodb');
const url = 'mongodb://127.0.0.1:27017';
const client=new MongoClient(url);
async function dbConnect()    {
    let result=await client.connect();
    db=result.db("sravanthi“);
    return db.collection(“students");
    }
module.exports=dbConnect;
```

Readopr.js

```
const dbConnect=require('./maindb')
async function getData()    {
    let result=await dbConnect();
   result =await result.find().toArray();
    console.log(result);
    }
     getData();
```

Insertopr.js

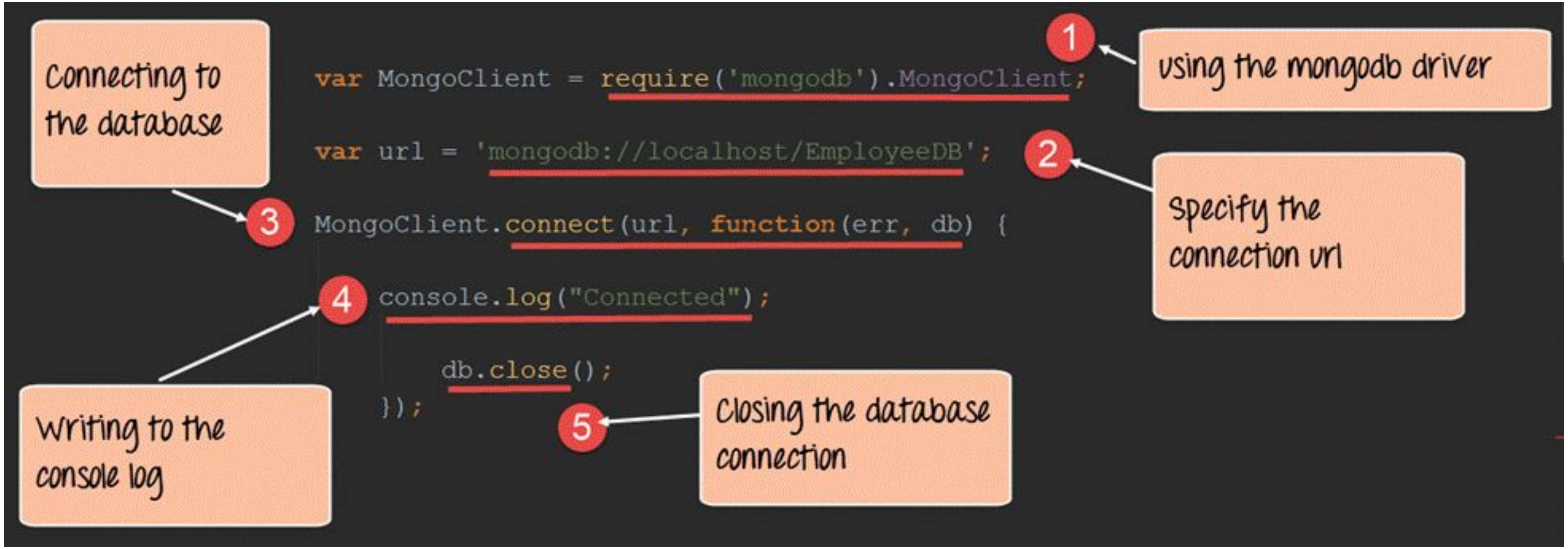
```
const dbConnect=require('./maindb');
const insertd=async ()=>{
const db= await dbConnect();
 const result=db.insertOne({name:"daddy"});


}
insertd();
```

Updateopr.js

```
const
dbConnect=require('./maindb')
const updateData=async ()=>{
    let data=await dbConnect();
    let result=data.updateOne(
{name:'duggu'},
{$set:{name:'sankarsh'}});
}
updateData();
```

Example-3
Delete data in mongodb

```
const {MongoClient}=require('mongodb');
const url = 'mongodb://127.0.0.1:27017';
const client=new MongoClient(url);
async function DeleteData()     {
    let result=await client.connect();
    db=result.db("sravanthi");
    collection =db.collection("kids");
    let data=await collection.deleteOne(data)
   console.log(data);
}

DeleteData();
```



```
var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

  console.log("Connected");

    db.close();
});
```

Connecting to the database
1
using the mongodb driver
2
specify the connection url
3
4
Writing to the console log
5
Closing the database connection

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.

- 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.

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');

- 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){ . . . });

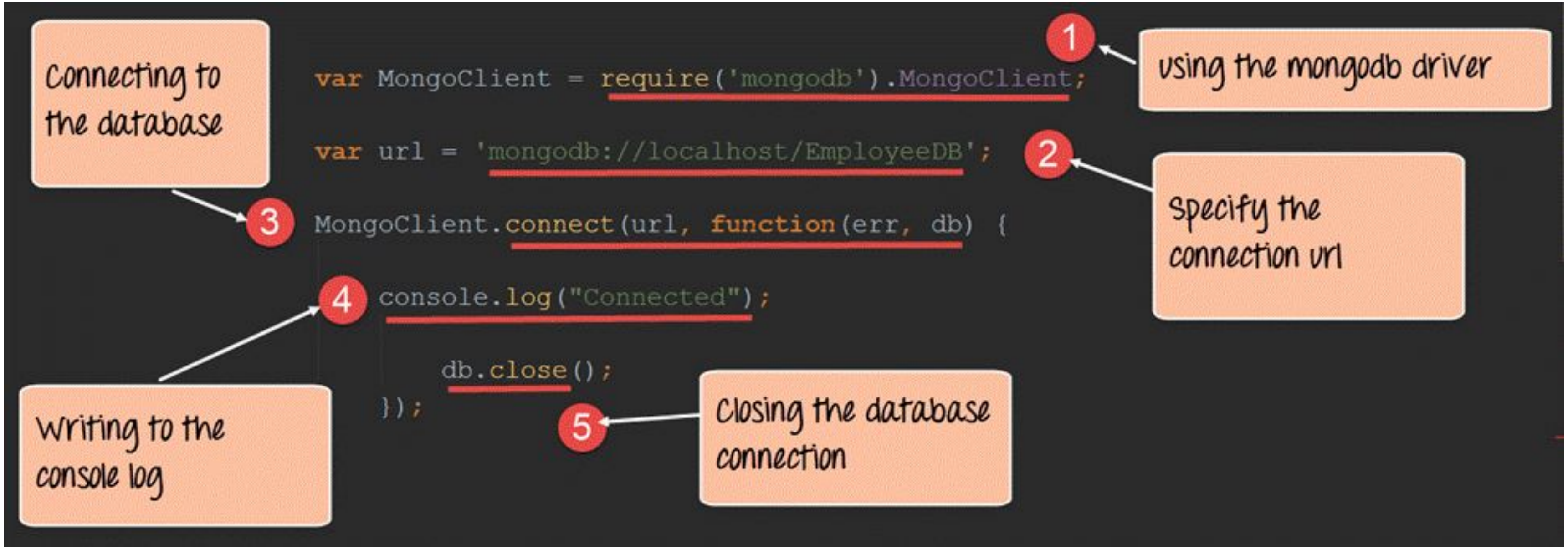
- we can perform the normal operations of reading data from a database as well as inserting, deleting, and updating records in a MongoDB database.
- let's assume that we have the below MongoDB data in place.
- Database name: EmployeeDB
- Collection name: Employee

Documents

```
{
    {Employeeid : 1, Employee Name : Guru99},
    {Employeeid : 2, Employee Name : Joe},
    {Employeeid : 3, Employee Name : Martin},
}
```

- **Installing the NPM Modules**
- You need a driver to access Mongo from within a Node application. There are a number of Mongo drivers available, but MongoDB is among the most popular. To install the MongoDB module, run the below command
- **npm install mongodb**

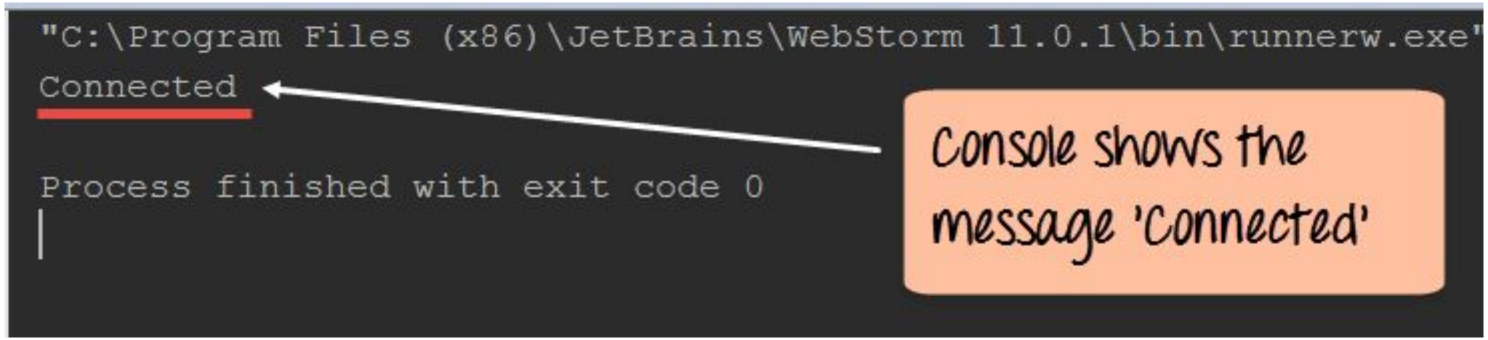
Creating and closing a connection to a MongoDB database.



```
var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    console.log("Connected");

        db.close();
    });
```

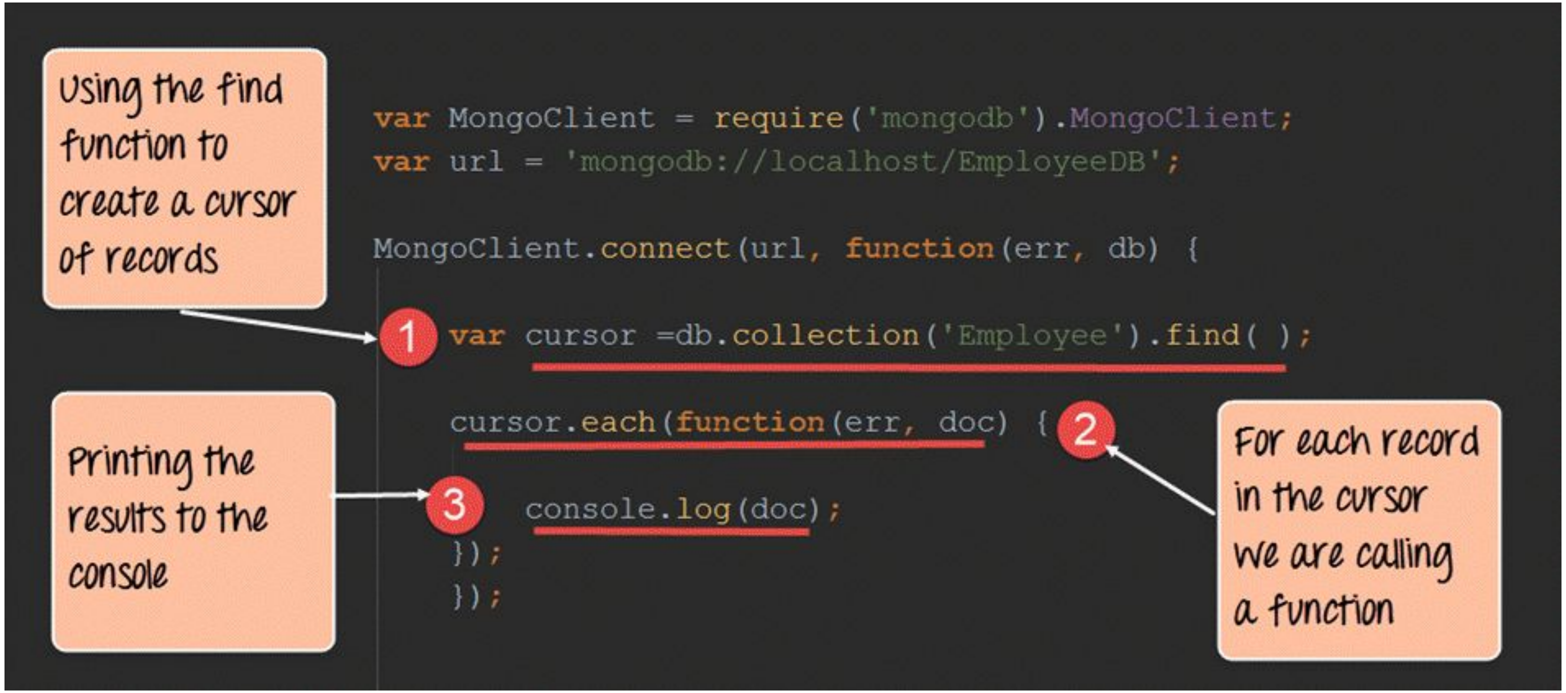
1
using the mongodb driver
2
specify the connection url
3
Connecting to the database
4
Writing to the console log
5
Closing the database connection

- The **first step is to include the mongoose module,** which is done through the require function.
- Next, we **specify the connection string to the database.** In the connect string, there are 3 key values which are passed.
- The **first is ‘mongodb**’ which specifies that we are connecting to a **mongoDB database**.
- The **next is ‘localhost**’ which means we are connecting to a **database on the local machine.**
- The **next is ‘EmployeeDB’** which is the **name of the database** defined in our MongoDB database.
- The next step is to actually **connect to our database**. The **connect function takes in our URL and has the facility to specify a callback function.** It will be called when the connection is opened to the database. This gives us the opportunity **to know if the database connection was successful or not.**
- In the function, we are writing the string **“Connection established” to the console** to indicate that a successful connection was created.
- Finally, we are closing the connection using the **db.close statement.**



```
"C:\Program Files (x86)\JetBrains\WebStorm 11.0.1\bin\runnerw.exe"
Connected

Process finished with exit code 0
```

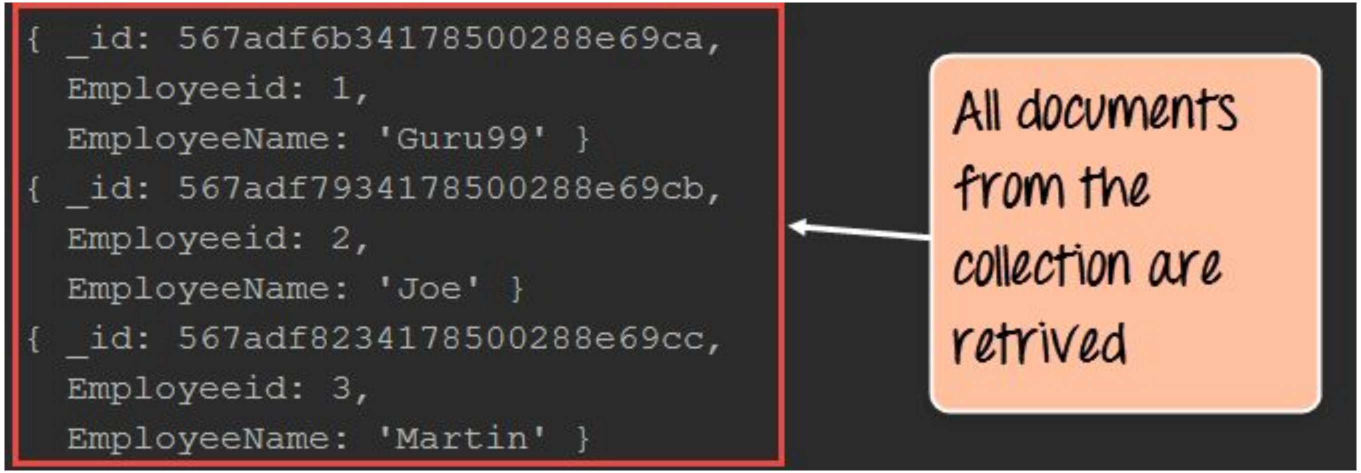
Querying for data in a MongoDB database



```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    var cursor =db.collection('Employee').find( );

    cursor.each(function(err, doc) {

        console.log(doc);
    });
    });
```

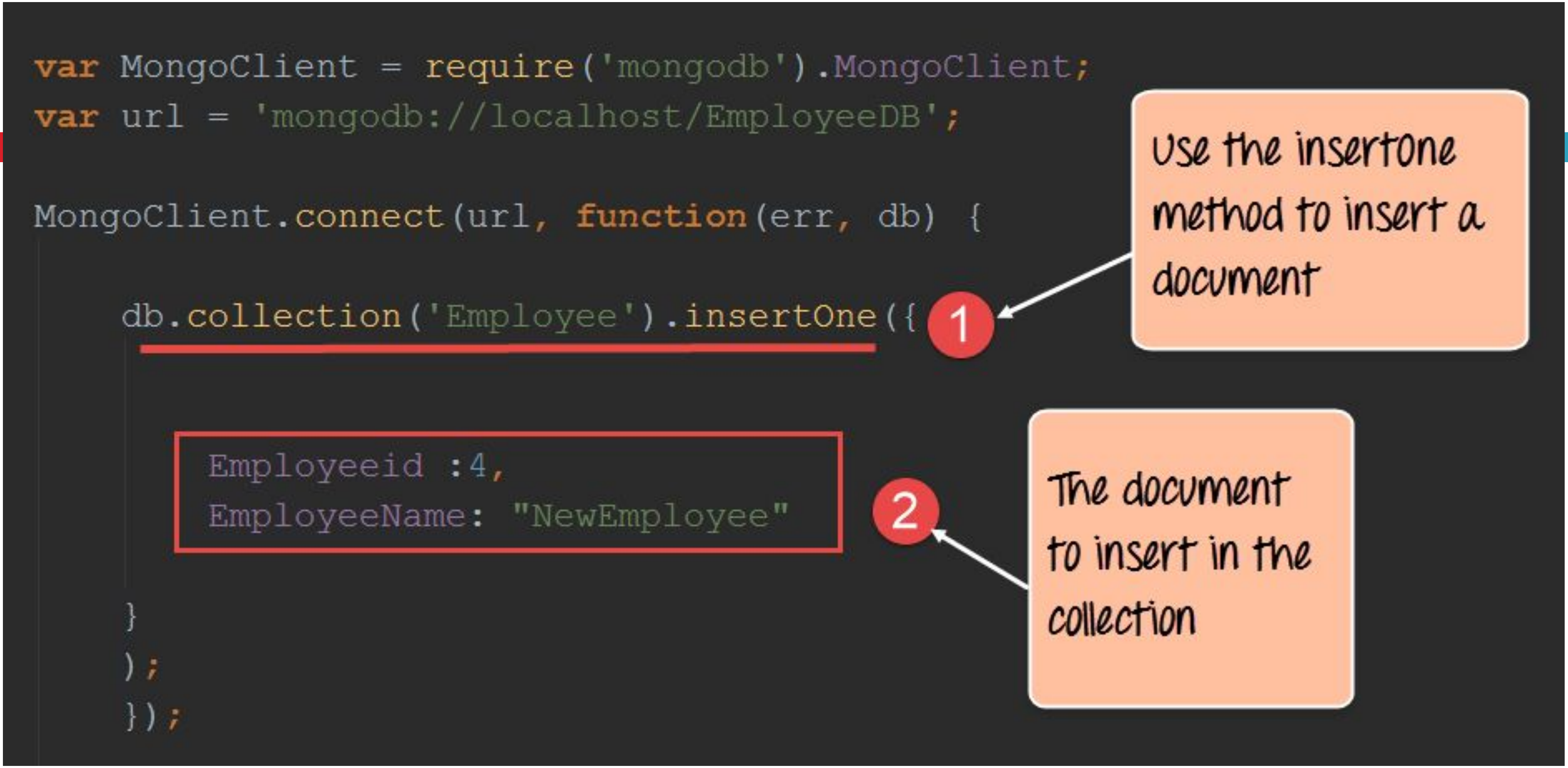
Using the find function to create a cursor of records
1
2
For each record in the cursor we are calling a function
3
Printing the results to the console

Output



```
{ _id: 567adf6b34178500288e69ca,
  Employeeid: 1,
  EmployeeName: 'Guru99' }
{ _id: 567adf7934178500288e69cb,
  Employeeid: 2,
  EmployeeName: 'Joe' }
{ _id: 567adf8234178500288e69cc,
  Employeeid: 3,
  EmployeeName: 'Martin' }
```

- **Inserting documents in a collection** – Documents can be inserted into a collection using the insertOne method provided by the MongoDB library.

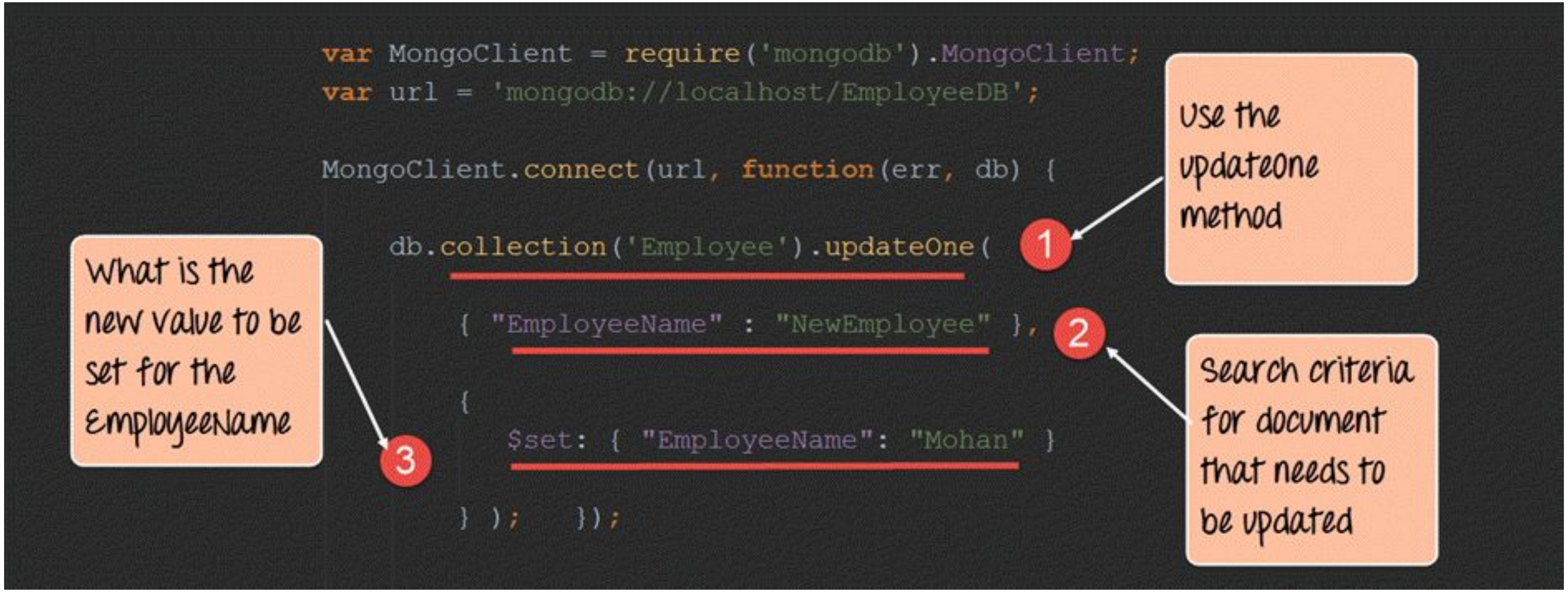


```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    db.collection('Employee').insertOne({


        Employeeid :4,
        EmployeeName: "NewEmployee"

    }
    );
    });
```

1
Use the insertOne method to insert a document
2
The document to insert in the collection

- To check that the data has been properly inserted in the database, you need to execute the following commands in MongoDB
- Use EmployeeDB
- db.Employee.find({Employeeid :4 })

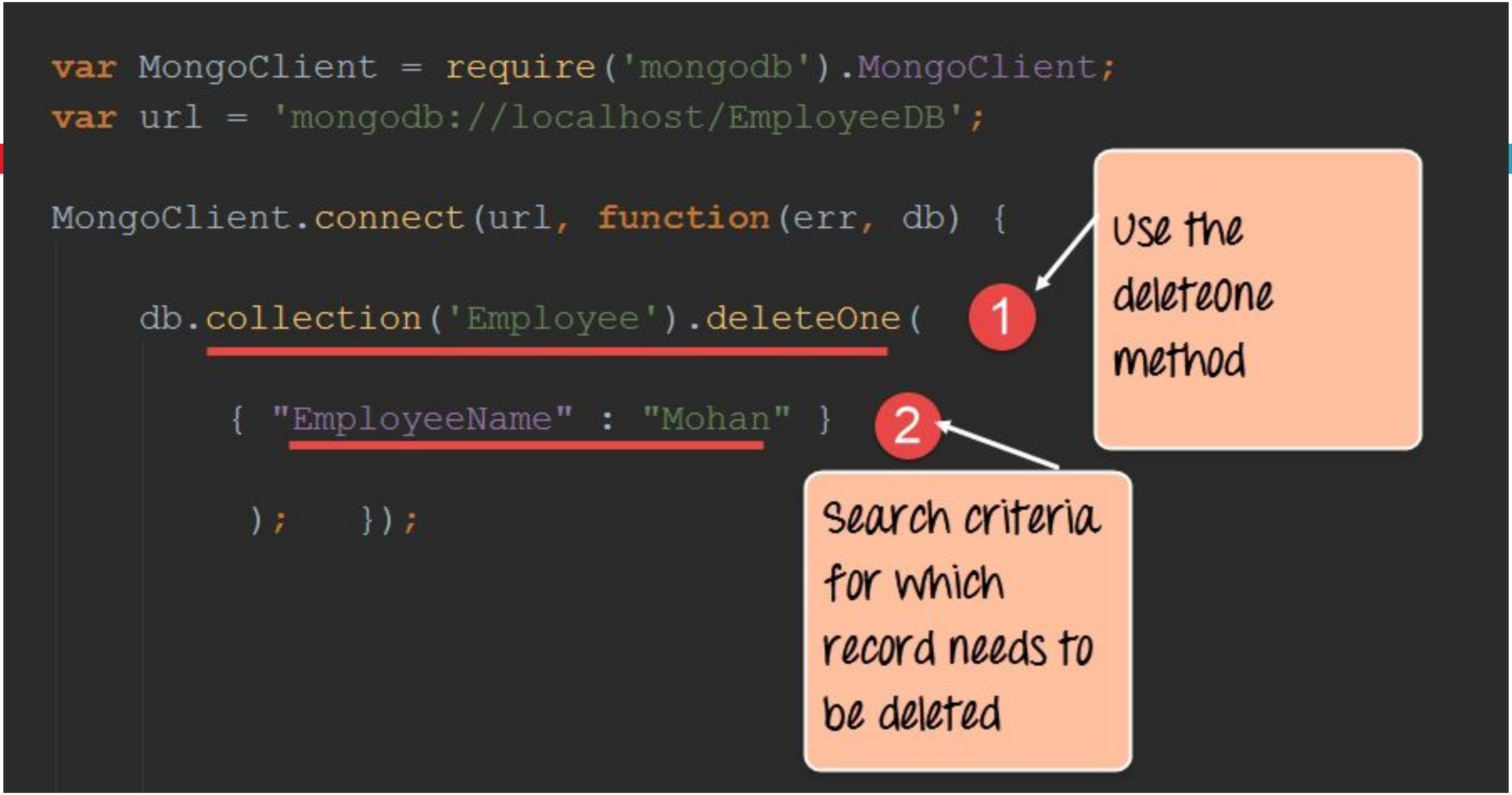
- **Updating documents in a collection** – Documents can be updated in a collection using the updateOne method provided by the MongoDB library.



```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    db.collection('Employee').updateOne(

        { "EmployeeName" : "NewEmployee" },

        {
          $set: { "EmployeeName": "Mohan" }

        } );    });
```

1
Use the updateOne method
2
Search criteria for document that needs to be updated
3
What is the new value to be set for the EmployeeName

- **Deleting documents in a collection** – Documents can be deleted in a collection using the “deleteOne” method provided by the MongoDB library.



```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    db.collection('Employee').deleteOne(

        { "EmployeeName" : "Mohan" }

        );    });
```