FULL STACK DEVELOPMENT USING OPEN-SOURCE TECHNOLOGY

By: Mrs. Nidhi Divecha (ME CMPN)

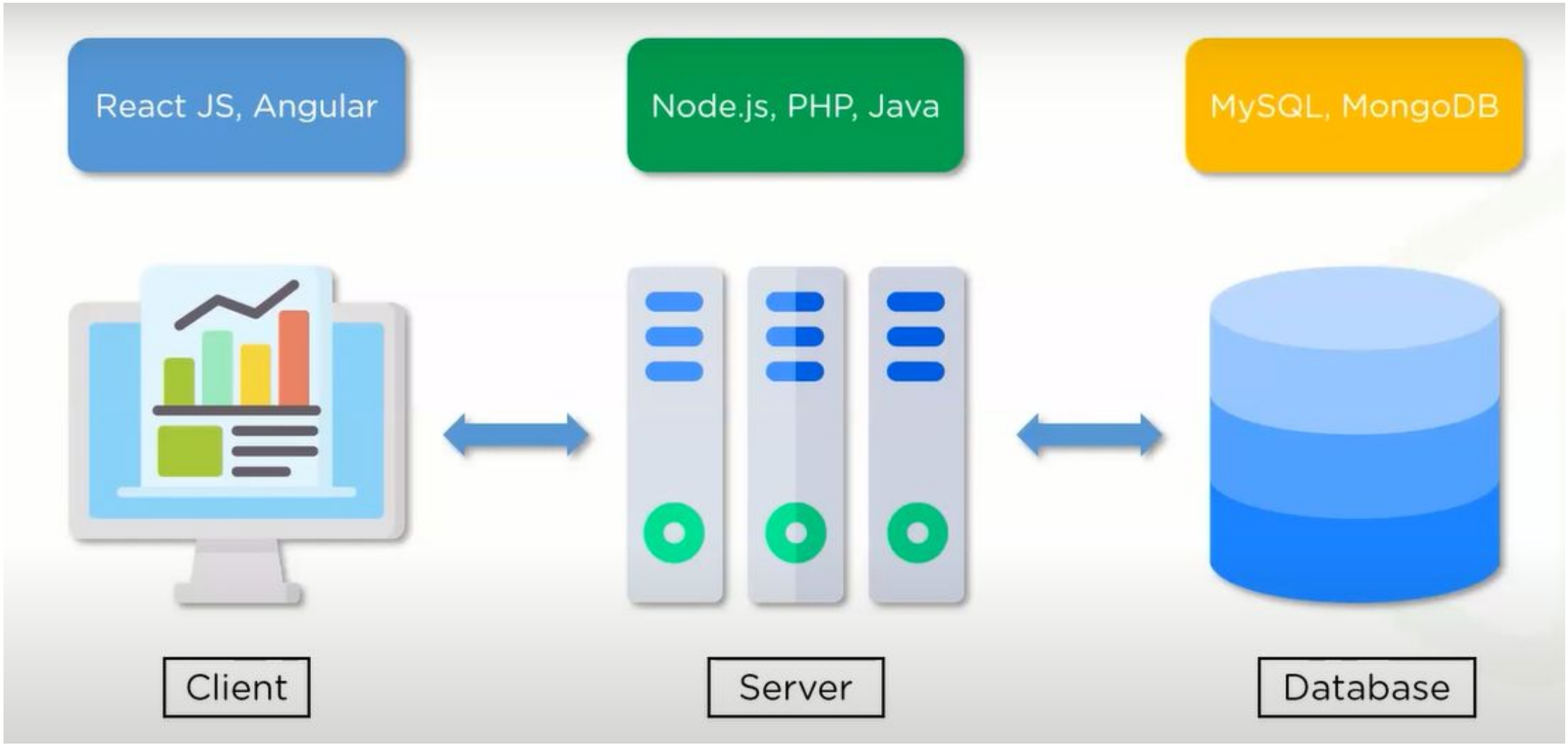
(UNIT 3)

INTRODUCTION TO NODE.JS

- Node.js is an **open source server environment**
- Node.js is **free**
- Node.js **runs on various platforms** (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js is used for creating **server-side web applications.**
- Node.js was **developed by Ryan Dahl in 2009.**

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database


React JS, Angular
Node.js, PHP, Java
MySQL, MongoDB
Client
Server
Database

WHY NODE.JS??

- Node.js is really fast (Being built on Google Chromes V8 JavaScript engine, its library is extremely fast in code execution)
- Whatever functionalities required for an application can be easily imported from npm.
- Easy for web developers to start using node.js in their projects as it is a JavaScript library.
- Better synchronization of code between server and client due to same code base.

HELLO WORLD EXAMPLE

- Install/build Node.js
- Open any editor and start typing JavaScript
- When you are done, open cmd/terminal and type this:

‘node YOUR_LIFE.js’

Here is a simple example which prints ‘hello world’

```
var sys = require(“sys”);
setTimeout(function(){
sys.puts(“world”);}, 3000};
sys.puts(“hello”);
```

- //it prints ‘hello’ first and waits for 3 seconds and then prints ‘world’

NODE.JS MODULES

- Modules ae like JavaScript libraries that can be used in a node.js application to include a set of functions.
- Each modules can be bundled under a single package.
- NPM(Node Package Manager) is used to install, update, uninstall and configure Node.js platform modules/packages very easily.

Installing Third-Party modules

- NPM install
- npm install <options> <package unique name>
- E.g. npm install express --save

NODE.JS “REQUIRE”

- The modules/packages available can be imported in .js file using “require” function.
- Eg – require(“http”);
- This helps in loading “http” package in current.js file.

MODULES EXPORT AND IMPORT

- A module in node.js can be exported , by which it can be imported in other files.
- The “require” function helps in importing modules.

- Importing

var <varName> = require (<module name>);

- Exporting

module.exports{<module to be exported>}

NODE.JS HTTP MODULE

The Built-in HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
- To include the HTTP module, use the require() method:

```
var http = require('http');
```

Node.js as a Web Server

- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- Use the createServer() method to create an HTTP server:

Example

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

NODE.JS FILE SYSTEM MODULE

- Node.js as a File Server
- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the require() method:

```
var fs = require('fs');
```

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Read Files

- The fs.readFile() method is used to read files on your computer.
- Create a Node.js file that reads the HTML file, and return the content:

Example

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Create Files

- The File System module has methods for creating new files:
- fs.appendFile()
- fs.open()
- fs.writeFile()
- The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

Example

- Create a new file using the appendFile() method:
- ```
  var fs = require('fs');

  fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
    if (err) throw err;
    console.log('Saved!');
  });
  ```


  ```

Update Files

- The File System module has methods for updating files:
- fs.appendFile()
- fs.writeFile()
- The fs.appendFile() method appends the specified content at the end of the specified file:

Example

- Append "This is my text." to the end of the file "mynewfile1.txt":
-

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

Delete Files

- To delete a file with the File System module, use the fs.unlink() method.
- The fs.unlink() method deletes the specified file:

Example

- Delete "mynewfile2.txt":
-

```
var fs = require('fs');

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename Files

- To rename a file with the File System module, use the fs.rename() method.
- The fs.rename() method renames the specified file:

Example

- Rename "mynewfile1.txt" to "myrenamedfile.txt":
-

```
var fs = require('fs');

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

NODE.JS URL MODULE

The Built-in URL Module

- The URL module splits up a web address into readable parts.
- To include the URL module, use the require() method:

```
var url = require('url’);
```

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Example

- Split a web address into readable parts:

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Node.js File Server

Create two html files and save them in the same folder as your node.js files.

summer.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

winter.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

demo_fileserver.js:

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

- Initiate demo_fileserver.js:

```
C:\Users\Your Name>node demo_fileserver.js
```

NODE.JS NPM

- NPM is a package manager for Node.js packages, or modules.
- The NPM program is installed on your computer when you install Node.js

What is a Package?

- A package in Node.js contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.

Download a Package

- Open the command line interface and tell NPM to download the package you want.
- Download "upper-case":

```
C:\Users\Your Name>npm install upper-case
```

- NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.
- My project now has a folder structure like this:

```
C:\Users\My Name\node_modules\upper-case
```

- Once the package is installed, it is ready to use.

```
var uc = require('upper-case');
```

- Create a Node.js file that will convert the output "Hello World!" into upper-case letters:

Example

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("Hello World!"));
  res.end();
}).listen(8080);
```

- Save the code above in a file called "demo_uppercase.js", and initiate the file:
- Initiate demo_uppercase:

```
Users\Your Name>node demo_uppercase.js
```

NODE.JS EVENTS

- Node.js is perfect for event-driven applications.
- Every action on a computer is an event. Like when a connection is made or a file is opened.
- Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

Example:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

Events Module

- Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.
- To include the built-in Events module use the require() method.
- In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
```

The EventEmitter Object

- In the example below we have created a function that will be executed when a "scream" event is fired.
- To fire an event, use the emit() method.

Example

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

//Create an event handler:
var myEventHandler = function () {
  console.log('I hear a scream!');
}

//Assign the event handler to an event:
eventEmitter.on('scream', myEventHandler);

//Fire the 'scream' event:
eventEmitter.emit('scream');
```

NODE.JS UPLOAD FILES

The Formidable Module

- There is a very good module for working with file uploads, called "Formidable".
- The Formidable module can be downloaded and installed using NPM:
  - `C:\Users\Your Name>npm install formidable`
- After you have downloaded the Formidable module, you can include the module in any application:

```
var formidable = require('formidable');
```

Upload Files

Step 1: Create an Upload Form

- Create a Node.js file that writes an HTML form, with an upload field:

Example

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
  res.write('<input type="file" name="filetoupload"><br>');
  res.write('<input type="submit">');
  res.write('</form>');
  return res.end();
}).listen(8080);
```

Step 2: Parse the Uploaded File

- Include the Formidable module to be able to parse the uploaded file once it reaches the server.
- When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

Example:

The file will be uploaded, and placed on a temporary folder:

```
var http = require('http');
var formidable = require('formidable');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Step 3: Save the File

- When a file is successfully uploaded to the server, it is placed on a temporary folder.
- The path to this directory can be found in the "files" object, passed as the third argument in the parse() method's callback function.
- To move the file to the folder of your choice, use the File System module, and rename the file:

Example

- Include the fs module, and move the file to the current folder:

```
var http = require('http');
var formidable = require('formidable');
var fs = require('fs');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.path;
      var newpath = 'C:/Users/Your Name/' + files.filetoupload.name;
      fs.rename(oldpath, newpath, function (err) {
        if (err) throw err;
        res.write('File uploaded and moved!');
        res.end();
      });
 });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

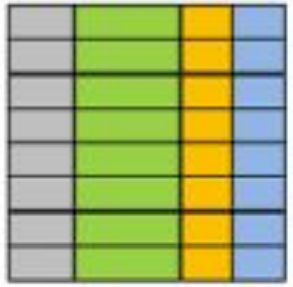
NOSQL DATABASES

INTRODUCTION TO NOSQL

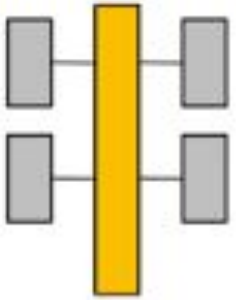
- **NoSQL database** stands for "Not Only SQL" or "Not SQL."
- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins and is easy to scale.
- NoSQL databases are used in real-time web applications and big data and their use are increasing over time.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- Traditional RDBMS uses SQL syntax to store and retrieve data for further insights.
- **NoSQL databases** were created to overcome the **limitations of relational databases.**

SQL Database

Relational

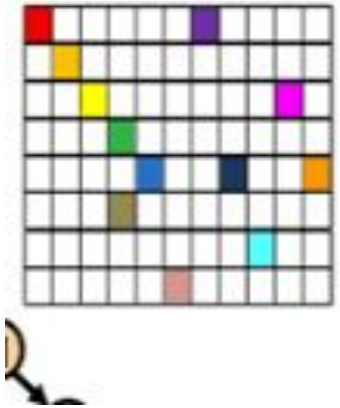


Analytical (OLAP)

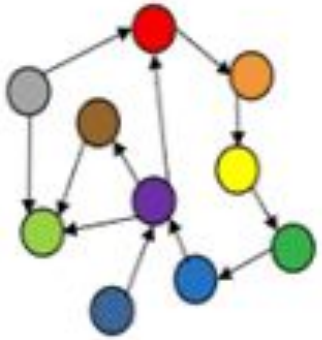


NoSQL Database

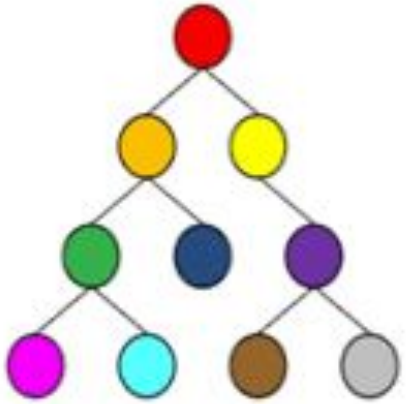
Column-Family



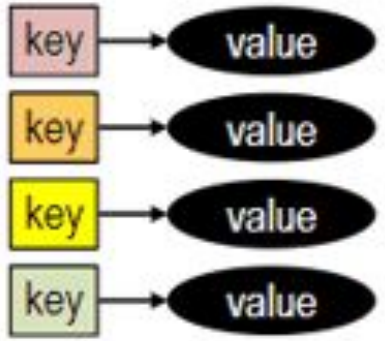
Graph



Document

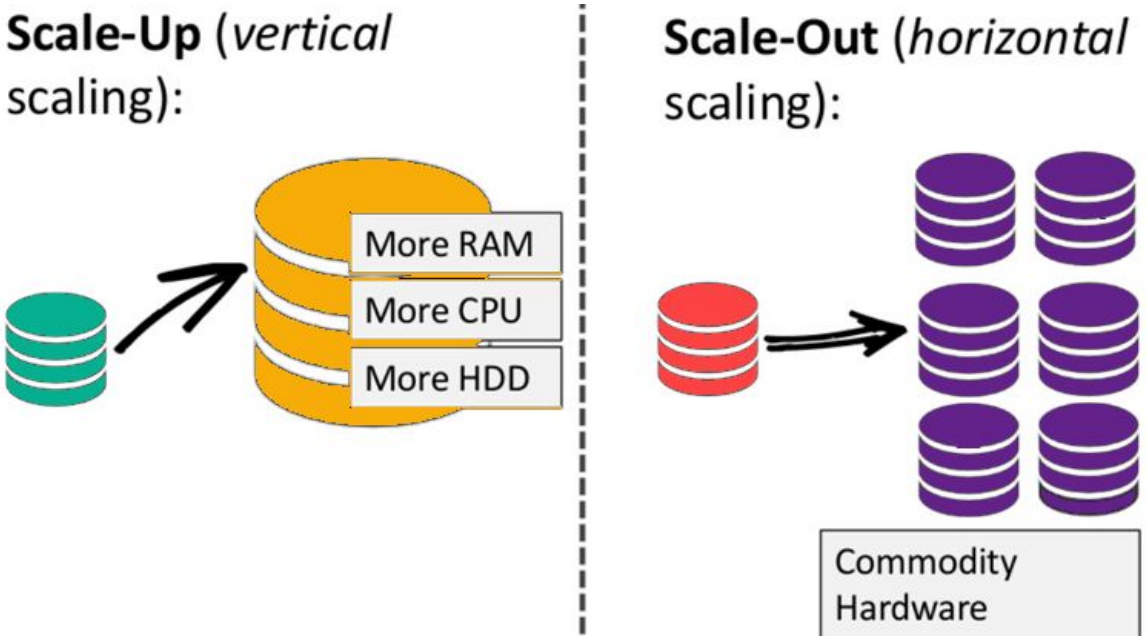


Key-Value



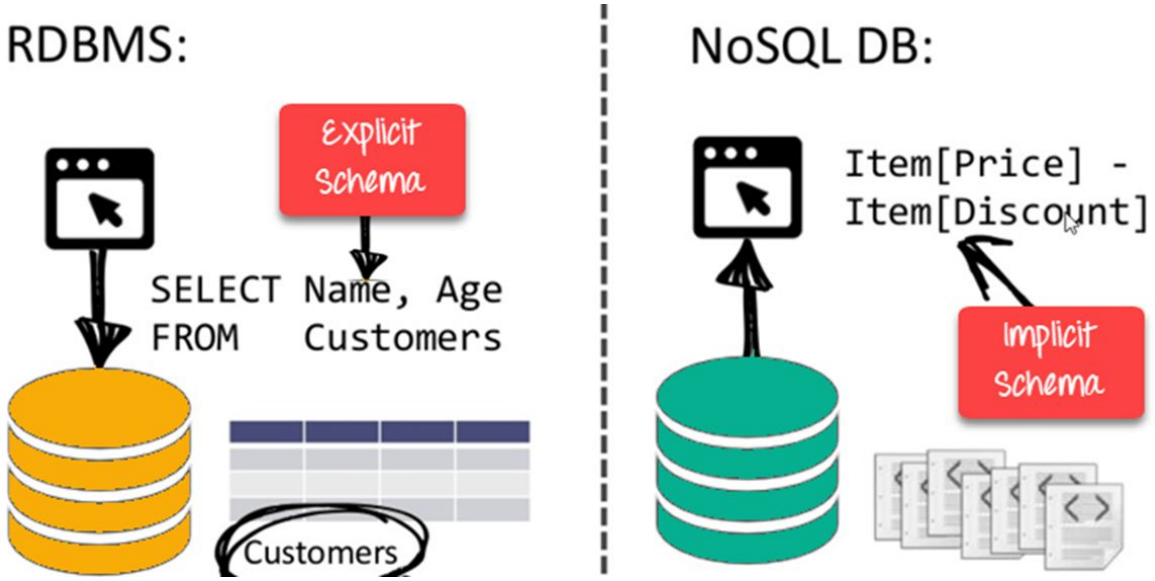
Parameter	SQL	NOSQL
Definition	SQL databases are primarily called RDBMS or Relational Databases	NoSQL databases are primarily called as Non-relational or distributed database
Design for	Traditional RDBMS uses SQL syntax and queries to analyze and get the data for further insights. They are used for OLAP systems.	NoSQL database system consists of various kind of database technologies. These databases were developed in response to the demands presented for the development of the modern application.
Query Language	Structured query language (SQL)	No declarative query language
Type	SQL databases are table-based databases	NoSQL databases can be document based, key-value pairs, graph databases
Schema	SQL databases have a predefined schema	NoSQL databases use dynamic schema for unstructured data.
Ability to scale	SQL databases are vertically scalable	NoSQL databases are horizontally scalable
Examples	Oracle, Postgres, and MS-SQL.	MongoDB, Neo4j, Cassandra, Hbase.

Best suited for	An ideal choice for the complex query intensive environment.	It is not good fit complex queries.
Hierarchical data storage	SQL databases are not suitable for hierarchical data storage.	More suitable for the hierarchical data store as it supports key-value pair method.
Consistency	It should be configured for strong consistency.	It depends on DBMS as some offers strong consistency like MongoDB, whereas others offer only offers eventual consistency, like Cassandra.
Best Used for	RDBMS database is the right option for solving ACID problems.	NoSQL is a best used for solving data availability problems
Hardware	Specialized DB hardware (Oracle Exadata, etc.)	Commodity hardware
Storage Type	Highly Available Storage (SAN, RAID, etc.)	Commodity drives storage (standard HDDs, JBOD)



WHY NOSQL?

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out.“
- NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.



FEATURES OF NOSQL

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

Schema-free

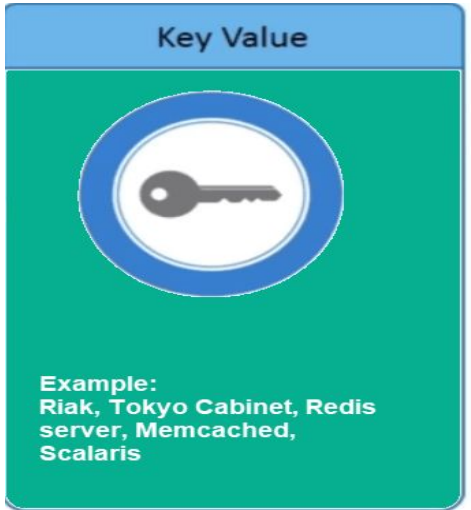
- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data.
- Faster performance
- Horizontally scalable

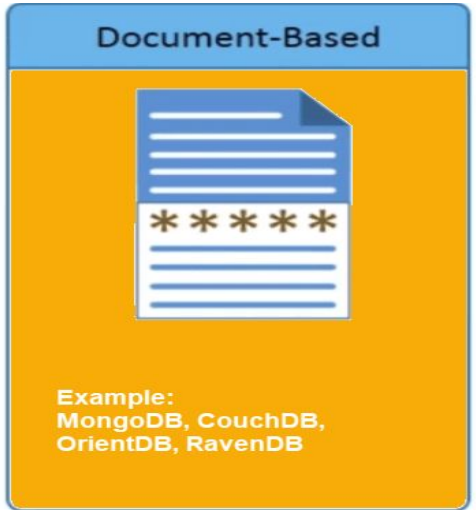
TYPES OF NOSQL DATABASES

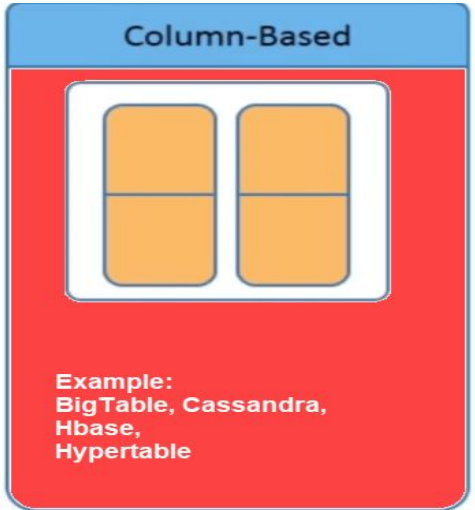
- **NoSQL Databases** are mainly categorized into four types.
- None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

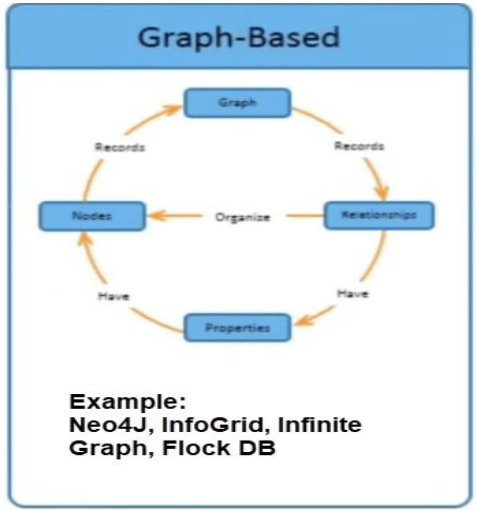
Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented









DOCUMENT-ORIENTED:

- Document-Oriented NoSQL DB stores and retrieves data as a **key value pair but the value part is stored as a document.**
- The document is stored in **JSON or XML formats**.
- Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object.
- You do not require to define schema which make it flexible.
- The document type is mostly used for blogging platforms, real-time analytics & e-commerce applications.
- Can perform nested queries and can do much more with your data when compared to key-value store.
- **SimpleDB, CouchDB, MongoDB, Riak,** are popular Document originated DBMS systems.

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Relational Vs. Document

MONGODB

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

- Document for Student in MongoDB will be like:

```
{
    name : "Studytonight",
    rollno : 1,
    subjects : ["C Language", "C++", "Core Java"]
}
```

- Relational Database Management System(RDBMS) is not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable.
- If the database runs on a single server, then it will reach a scaling limit.
- NoSQL databases are more scalable and provide superior performance.
- MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

FEATURES OF MONGODB

Document Oriented: MongoDB is Document oriented Database. There are different documents to store different types of data. Each document has unique system generated key.

Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.

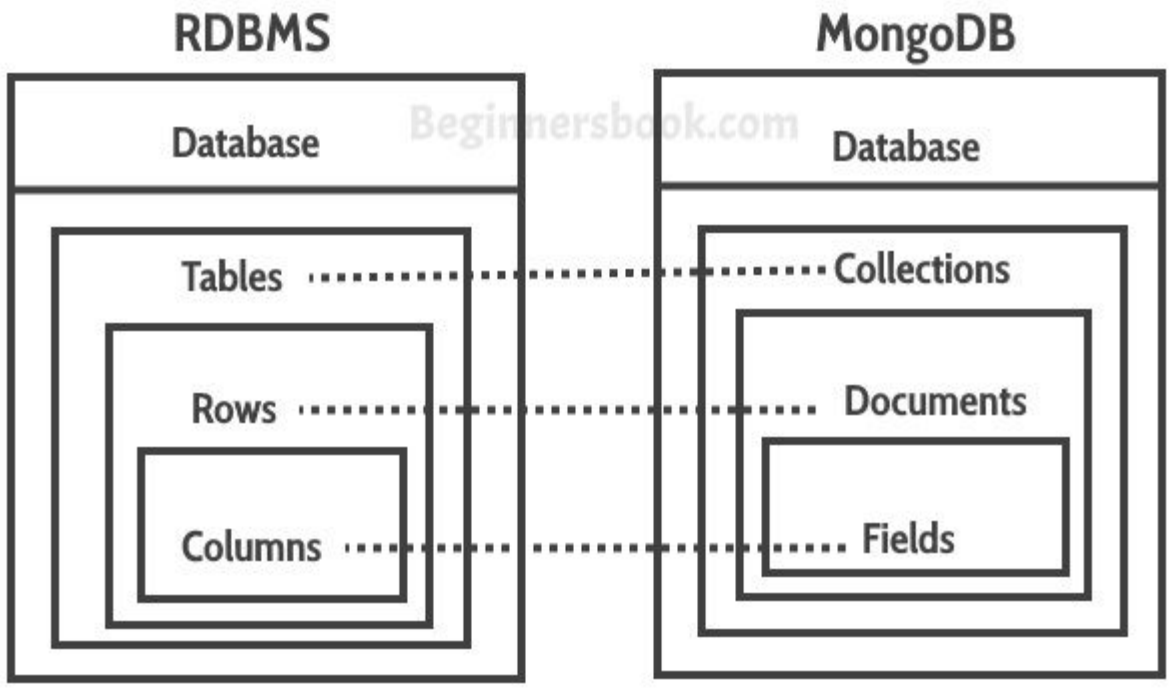
Scalability: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that resides across many physical servers.

Replication and High Availability: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.

Aggregation: Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

Sharding: For large set of data, we need Sharding mechanism. It is the type of database partitioning that separates very large databases into smaller, faster, more easily managed parts called data shards.

MAPPING RELATIONAL DATABASES TO MONGODB



Collections in MongoDB is equivalent to the tables in RDBMS.
Documents in MongoDB is equivalent to the rows in RDBMS.
Fields in MongoDB is equivalent to the columns in RDBMS.

Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB


```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```


MONGODB DATATYPES

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.

- **1. String:** This is the most used MongoDB data types, BSON strings are UTF-8.
- Drivers for each programming language convert from the string format of the language to UTF-8 while serializing and de-serializing BSON.
- **Example:** db.TestCollection.insert({"string data type" : "This is a sample message."})

```
> db.TestCollection.insert({"string data type" : "This is a sample message."})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d08724bce0d9292a5121a78"), "Integer example" : 62 }
{ "_id" : ObjectId("5d087412ce0d9292a5121a79"), "Nationality Indian" : true }
{ "_id" : ObjectId("5d0874dcce0d9292a5121a7a"), "double data type" : 3.1415 }
{ "_id" : ObjectId("5d087547ce0d9292a5121a7b"), "string data type" : "This is
 sample message." }
>
```

- **2. Integer:** In MongoDB, the integer data type is used to store an integer value. We can store integer data type in two forms 32 -bit signed integer and 64 – bit signed integer.
- **Example:** In the following example we are storing the age of the student in the student collection:

```
> db.student.insertOne({name:"Akash",age:19})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af3456fd54aa34c9c6df5")
}
> db.student.find().pretty()
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df3"), "name" : "Akshay" }
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df4"), "name" : "Vikash" }
{
        "_id" : ObjectId("601af3456fd54aa34c9c6df5"),
        "name" : "Akash",
        "age" : 19
}
```

- **3. Double:** The double data type is used to store the floating-point values.
- **Example:** In the following example we are storing the marks of the student in the student collection:

```
}
> db.student.insertOne({name:"Sagar",age:20,marks:546.43})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af4126fd54aa34c9c6df6")
}
> db.student.find().pretty()
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df3"), "name" : "Akshay" }
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df4"), "name" : "Vikash" }
{
        "_id" : ObjectId("601af3456fd54aa34c9c6df5"),
        "name" : "Akash",
        "age" : 19
}
{
        "_id" : ObjectId("601af4126fd54aa34c9c6df6"),
        "name" : "Sagar",
        "age" : 20,
        "marks" : 546.43
}
>
```

- **4. Boolean:** The boolean data type is used to store either true or false.
- **Example:** In the following example we are storing the final result of the student as pass or fail in boolean values.

```
> db.student.insertOne({name:"vishal",pass:true})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af47d6fd54aa34c9c6df7")
}
> db.student.find().pretty()
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df3"), "name" : "Akshay" }
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df4"), "name" : "Vikash" }
{
        "_id" : ObjectId("601af3456fd54aa34c9c6df5"),
        "name" : "Akash",
        "age" : 19
}
{
        "_id" : ObjectId("601af4126fd54aa34c9c6df6"),
        "name" : "Sagar",
        "age" : 20,
        "marks" : 546.43
}
{
        "_id" : ObjectId("601af47d6fd54aa34c9c6df7"),
        "name" : "vishal",
        "pass" : true
}
>
```

- **5. Null:** The null data type is used to store the null value.
- **Example:** In the following example, the student does not have a mobile number so the number field contains the value null.

```
> db.student.insertOne({name:"Akash",MobNo:null,skills:["c","c++","java","python","JS"]})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af5fb6fd54aa34c9c6df8")
}
> db.student.find().pretty()
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df3"), "name" : "Akshay" }
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df4"), "name" : "Vikash" }
{
        "_id" : ObjectId("601af3456fd54aa34c9c6df5"),
        "name" : "Akash",
        "age" : 19
}
{
        "_id" : ObjectId("601af4126fd54aa34c9c6df6"),
        "name" : "Sagar",
        "age" : 20,
        "marks" : 546.43
}
{
        "_id" : ObjectId("601af47d6fd54aa34c9c6df7"),
        "name" : "vishal",
        "pass" : true
}
{
        "_id" : ObjectId("601af5fb6fd54aa34c9c6df8"),
        "name" : "Akash",
        "MobNo" : null,
        "skills" : [
                "c",
                "c++",
                "java",
                "python",
                "JS"
        ]
}
>
```

- **6. Array:** The Array is the set of values. It can store the same or different data types values in it. In MongoDB, the array is created using square brackets([]).
- **Example:** In the following example, we are storing the technical skills of the student as an array.

```
> db.student.find().pretty()
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df3"), "name" : "Akshay" }
{ "_id" : ObjectId("601af2dd6fd54aa34c9c6df4"), "name" : "Vikash" }
{
        "_id" : ObjectId("601af3456fd54aa34c9c6df5"),
        "name" : "Akash",
        "age" : 19
}
{
        "_id" : ObjectId("601af4126fd54aa34c9c6df6"),
        "name" : "Sagar",
        "age" : 20,
        "marks" : 546.43
}
{
        "_id" : ObjectId("601af47d6fd54aa34c9c6df7"),
        "name" : "vishal",
        "pass" : true
}
{
        "_id" : ObjectId("601af5fb6fd54aa34c9c6df8"),
        "name" : "Akash",
        "MobNo" : null,
        "skills" : [
                "c",
                "c++",
                "java",
                "python",
                "JS"
        ]
}
>
```

- **7. Object:** Object data type stores embedded documents. Embedded documents are also known as nested documents. Embedded document or nested documents are those types of documents which contain a document inside another document.
- **Example:** In the following example, we are storing all the information about a book in an embedded document.

```
> db.book.insertOne({Book:{name:"C in depth",writer:"Aaksh"}})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af71f6fd54aa34c9c6df9")
}
> db.bool.find().pretty()
> db.book.find().pretty()
{
        "_id" : ObjectId("601af71f6fd54aa34c9c6df9"),
        "Book" : {
                "name" : "C in depth",
                "writer" : "Aaksh"
        }
}
>
```

- **8. Object Id:** This data type in MongoDB stores the unique key Id of documents stored. There is an_id field in MongoDB for each document. The data which is stored in Id is in hexadecimal format. The size of ObjectId is 12 bytes which are divided into four parts as follows.

Part name	Size(bytes)
Timestamp	4
Machine Id	3
Process Id	2
Counter	3

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

```
> db.book.insertOne({Book:{name:"C in depth",writer:"Aaksh"}})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af71f6fd54aa34c9c6df9")
}
> db.bool.find().pretty()
> db.book.find().pretty()
{
        "_id" : ObjectId("601af71f6fd54aa34c9c6df9"),
        "Book" : {
                "name" : "C in depth",
                "writer" : "Aaksh"
        }
}
>
```

- **10. Binary Data:** This datatype is used to store binary data.
- **Example:** In the following example the value stored in the binaryValue field is of binary type.

```
> db.project.insertOne({name:"FauG",duration:undefined,binaryValue:"110011001"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601af8516fd54aa34c9c6dfa")
}
> db.project.find().pretty()
{
        "_id" : ObjectId("601af8516fd54aa34c9c6dfa"),
        "name" : "FauG",
        "duration" : undefined,
        "binaryValue" : "110011001"
}
>
```

- **11. Date:** Date data type stores current date or time. There are various methods to return date. It can be either as a string or as a date object. In the below table, we have discussed the methods for the date.

Date Method	Description
Date()	It returns the current date in string format.
New Date()	Returns a date object. Uses the ISODate() wrapper.
ISODate()	It also returns a date object. Uses the ISODate() wrapper.

```
>
> var date1  = Date()
> var date2 = new Date()
> var date3 = new ISODate()
> db.date.insertOne({Date1:date1,Date2:date2,Date3:date3})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601afa326fd54aa34c9c6dfc")
}
> db.date.find().pretty()
{
        "_id" : ObjectId("601afa326fd54aa34c9c6dfc"),
        "Date1" : "Thu Feb 04 2021 01:00:34 GMT+0530 (India Standard Time)",
        "Date2" : ISODate("2021-02-03T19:30:40.014Z"),
        "Date3" : ISODate("2021-02-03T19:30:56.454Z")
}
>
```

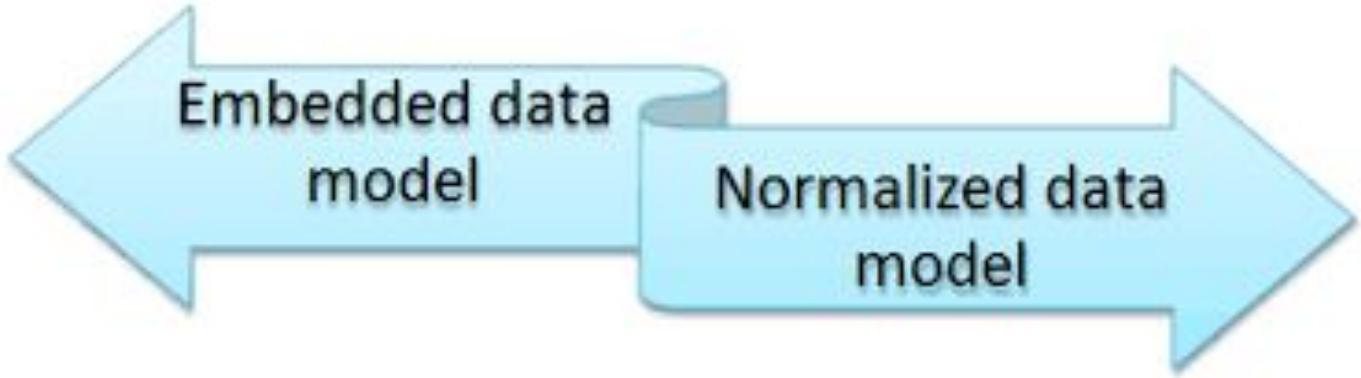
- **12. Min & Max key:** Min key compares the value of the lowest BSON element and Max key compares the value against the highest BSON element. Both are internal data types.

```
>
> db.m.insertOne({min:MinKey,max:MaxKey})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601afbd66fd54aa34c9c6dff")
}
> db.m.find().pretty()
{
        "_id" : ObjectId("601afbd66fd54aa34c9c6dff"),
        "min" : { "$minKey" : 1 },
        "max" : { "$maxKey" : 1 }
}
>
```

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

```
}
> var symbl = "nik432#"
> db.symbl.insert({symbol:symbl})
WriteResult({ "nInserted" : 1 })
> db.symbl.find().pretty()
{ "_id" : ObjectId("601afb6f6fd54aa34c9c6dfe"), "symbol" : "nik432#" }
>
```

DATA MODELLING IN MONGODB



- MongoDB provides two types of data models: — **Embedded data model and Normalized data model.**
- Based on the requirement, you can use either of the models while preparing your document.

Embedded Data Model

- In this model, you can have (embed) all the related data in a single document, it is also known as **de-normalized data model.**
- For example, assume we are getting the details of employees in three different documents namely, Personal_details, Contact and, Address, you can embed all the three documents in a single one as shown below −

```
{
    _id: ,
    Emp_ID: "10025AE336"
    Personal_details:{
        First_Name: "Radhika",
        Last_Name: "Sharma",
        Date_Of_Birth: "1995-09-26"
    },
    Contact: {
        e-mail: "radhika_sharma.123@gmail.com",
        phone: "9848022338"
    },
    Address: {
        city: "Hyderabad",
        Area: "Madapur",
        State: "Telangana"
    }
}
```

Normalized Data Model

- In this model, you can refer the sub documents in the original document, using references. For example, you can re-write the above document in the normalized model as:

Employee:

```
{
    _id: <ObjectId101>,
    Emp_ID: "10025AE336"
}
```

Personal_details:

```
{
    _id: <ObjectId102>,
    empDocID: " ObjectId101",
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
}
```

Contact:

```
{
    _id: <ObjectId103>,
    empDocID: " ObjectId101",
    e-mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
}
```

Address:

```
{
    _id: <ObjectId104>,
    empDocID: " ObjectId101",
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana"
}
```

anki — mongo — 80×55

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
>
```

anki — mongo — 80×55

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertMany([
... {
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... },
...
... {
... name : "Rohit",
... age : 21,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... }
...
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5e540d3192e6dfa3fc48ddaf"),
                ObjectId("5e540d3192e6dfa3fc48ddb0")
        ]
}
>
```

CRUD OPERATIONS

- In Mongodb the CRUD operation refers to the **creating, reading, updating, and deleting** of documents.

Create

- Create operations add new documents to a collection.
- There are two ways to add new documents to collections:

db.collection.insertOne()

db.collection.insertMany()

anki — mongo — 80×55

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```

Read

- Read operations retrieve documents from a collection. If there is no collection that currently exists, it will automatically create a new one.
- Here is the method to retrieve information:

  db.collection.find()

- In this example, we are retrieving the details of students from the student collection using db.collection.find() method.

Update

- Update operations modify documents from a collection.
- The three ways to add new documents to collections are:

db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

Delete

- Delete operations delete documents from a collection.
- There are two ways to add new documents to collections:

db.collection.deleteOne()

db.collection.deleteMany()