INTRODUCTION TO MONGO DB(NO SQL DATA BASE)

Overview

- What is NO SQL Data base?
- Types of NO SQL Data base.
- What is Mongo DB?
- Why Mongo DB?
- Mongo DB Architecture.
- Document (JSON) Structure.
- Differences between XML and JSON.
- Different Methods.
- When to use Mongo DB?

WHAT IS NO SQL DATA BASE

- It’s Not No SQL it’s NOT ONLY SQL.
- It’s not even a replacement to RDBMS.

As compared to the good olden days we are saving more and more data.

Connection between the data is growing in which we require an architecture that takes advantage of these two key issues.

TYPES OF NO SQL DATA BASE

- Key Value pair

Dynamo DB

Azure Table Storage (ATS)

- Document Based

Mongo Db

AmazonSimple DB

Couch DB

```
(#key,#value)
(Name, Tom)
(Age,25)
(Role, Student)
(University, CU)
```

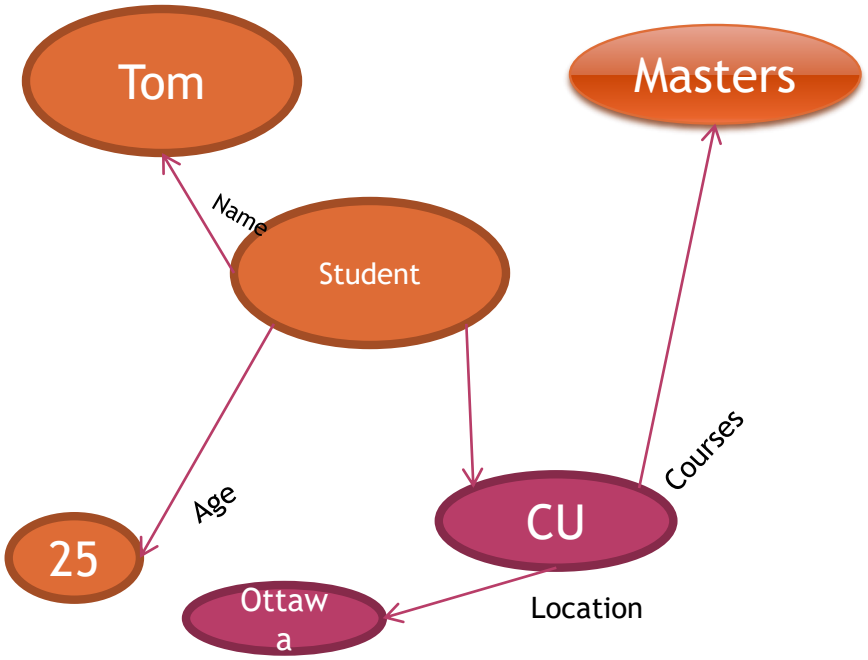
```
[
  {
    "Name": "Tom",
    "Age": 30,
    "Role": "Student",
    "University": "CU",

  }
]
```

TYPES OF NO SQL DATA BASE

Graph database

- Neo4j
- Infogrid



Column Oriented database

Row Id	Columns	
1	Name	Tom
	Age	25
	Role	Studen t

Bigtable(Google)

H Base

WHAT IS MONGO DB

MongoDB is a cross-platform, document oriented database that provides

- High performance.
- High availability.
- Easy scalability.

MongoDB works on concept of collection and document.

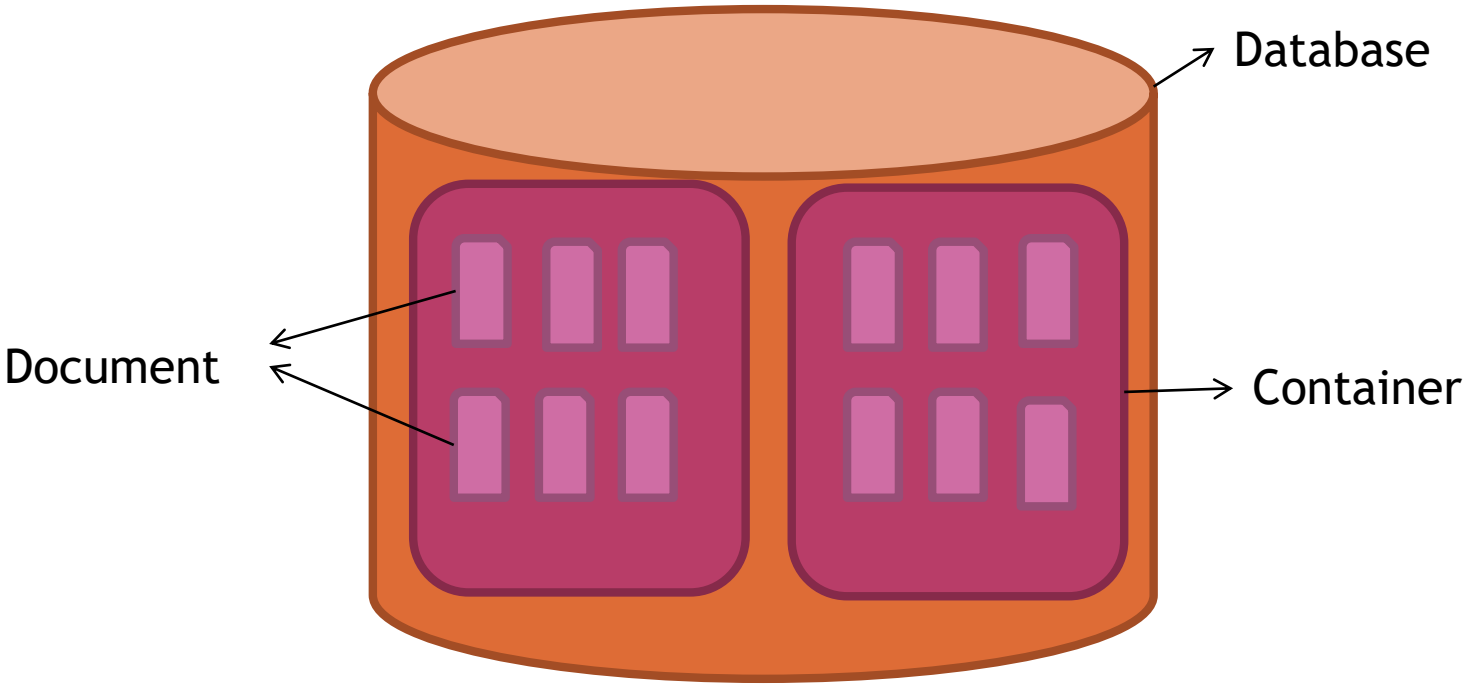
WHY MONGO DB?

- All the modern applications deals with huge data.
- Development with ease is possible with mongo DB.
- Flexibility in deployment.
- Rich Queries.
- Older database systems may not be compatible with the design.

And it's a document oriented storage:- Data is stored in the form of JSON Style.

MONGO DB ARCHITECTURE

Architecture : -



DOCUMENT(JSON) STRUCTURE

- The document has simple structure and very easy to understand the content
- JSON is smaller, faster and lightweight compared to XML.
- For data delivery between servers and browsers, JSON is a better choice
- Easy in parsing, processing, validating in all languages
- JSON can be mapped more easily into object oriented system.

```
[
{
    "Name": "Tom",
    "Age": 30,
    "Role": "Student",
    "University": "CU",
  }
  {
    "Name": “Sam",
    "Age": 32,
    "Role": "Student",
    "University": “OU",
  }
]
```

DIFFERENCE BETWEEN XML AND JSON

XML	JSON
It is a markup language.	It is a way of representing objects.
This is more verbose than JSON.	This format uses less words.
It is used to describe the structured data.	It is used to describe unstructured data which include arrays.
JavaScript functions like *eval*(), *parse*() doesn't work here.	When *eval* method is applied to JSON it returns the described object.
Example: <car> <company>Volkswagen</company> <name>Vento</name> <price>800000</price> </car>	{ "company": Volkswagen, "name": "Vento", "price": 800000

WHY JSON?

- JSON is faster and easier than XML when you are using it in AJAX web applications:
- Steps involved in exchanging data from web server to browser involves:

Using XML

1. Fetch an XML document from web server.
2. Use the XML DOM to loop through the document.
3. Extract values and store in variables.
4. It also involves type conversions.

Using JSON

1. Fetch a JSON string.
2. Parse the JSON string using eval() or parse() JavaScript functions.

THE INSERT() METHOD

- To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save( )** method.

- The basic syntax of **insert()** command is as follows –

“db.COLLECTION_NAME.insert(document)”

- Example: -

```
db.StudentRecord.insert (

{
    "Name": "Tom",
    "Age": 30,
    "Role": "Student",
    "University": "CU",
     },
   {
    "Name": “Sam",
    "Age": 22,
    "Role": "Student",
    "University": “OU",
   }

)
```

THE FIND() METHOD

- To query data from MongoDB collection, you need to use MongoDB's **find()** method.
- The basic syntax of **find()** method is as follows –

  "db.COLLECTION_NAME.find()"
- find() method will display all the documents in a non-structured way.
- To display the results in a formatted way, you can use **pretty()** method.

  "db.mycol.find().pretty() "

- Example: -

```
db.StudentRecord.find(
).pretty()
```

THE REMOVE() METHOD

- MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.
- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.
- Syntax
- **db.COLLECTION_NAME.remove(DELETION_CRITTERIA)**

Remove based on DELETION_CRITERIA

```
db.StudentRecord.remove(
    {"Name": "Tom})
```

Remove Only One:-
Removes first record

```
db.StudentRecord.remove(
DELETION_CRITERIA,1)
```

Remove all Records

```
db.StudentRecord.remove()
```

WHEN TO USE MONGO DB?

When your requirements has these properties :

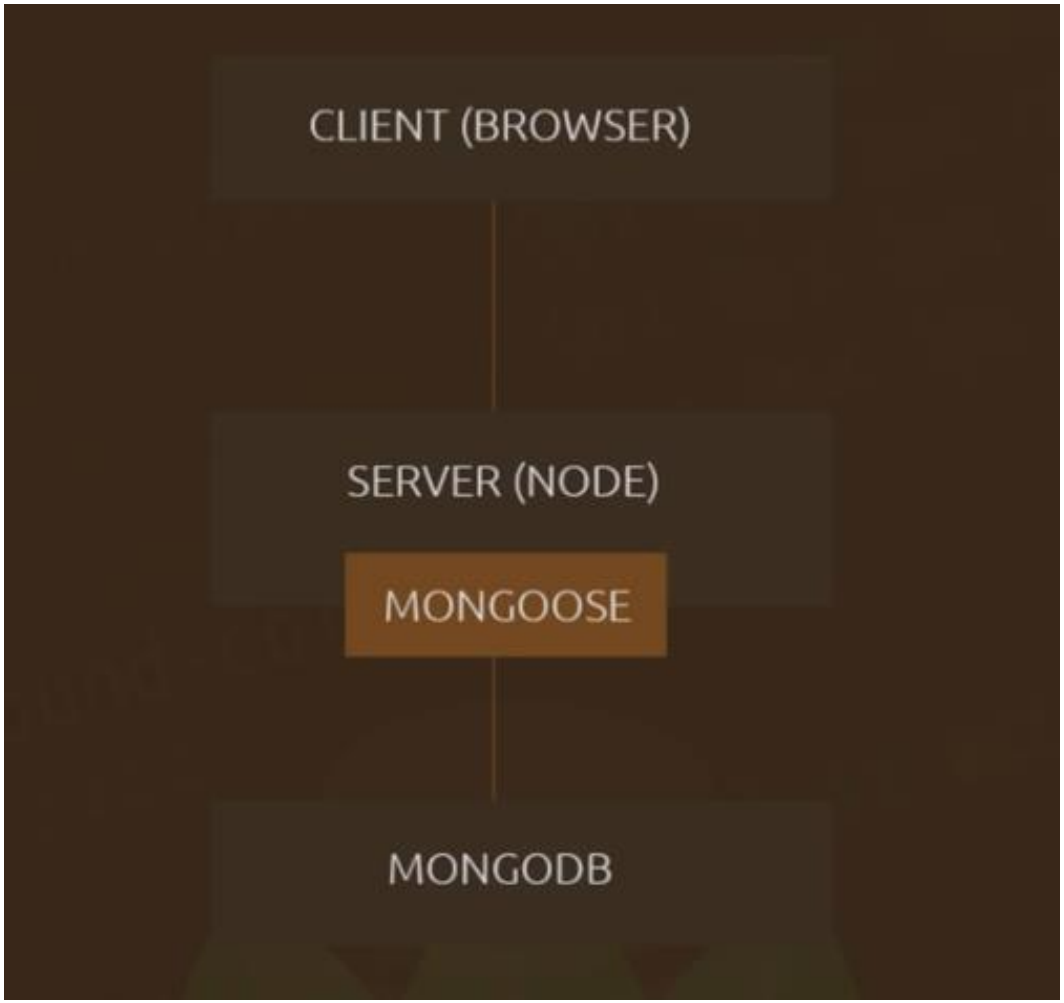
- You absolutely must store unstructured data. Say things coming from 3rd-party API you don't control, logs whose format may change any minute, user-entered metadata, but you want indexes on a subset of it.
- You need to handle more reads/writes than single server can deal with and master-slave architecture won't work for you.

- You change your schema very often on a large dataset.

MONGO IN NODEJS

- NO SQL DB
- Instead of storing the data in table we store it in documents of collection of objects.
- Easier to communicate
- Use it anytime we want to store data in our application
- Mongo is the M in MEAN

MONGO IN NODEJS



MONGOOSE

- Package that can be installed in node js , which makes communication with mongodb easier.
- STEPS:
  - Install MongoDB and Mongoose
  - Use basic CRUD operations
  - Mocha – Testing enviornment

- Download MongoDB:
  - MongoDB.com
  - Check Os
  - It doesn't know where to store data

  (dir) – create /data/db folder in c drive.
- Prerequisite
  - Javascript
  - Install nodejs

CONNECTING TO MONGODB

- Just because we have installed mongoose it doesn't automatically know to connect to MONGODB.
- It doesn’t even know that a DB exist.
- We have to explicitly test mongoose to connect.
- How?

CREATING A CONNECTION

- Test/connection.js
- Connection file contains all connections.

connection.js

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

// ES6 Promises
mongoose.Promise = global.Promise;

// Connect to db before tests run
before(function(done){

    // Connect to mongodb
    mongoose.connect('mongodb://localhost/testaroo');
    mongoose.connection.once('open', function(){
        console.log('Connection has been made, now make fireworks...');
        done();
    }).on('error', function(error){
        console.log('Connection error:', error);
    });

});

// Drop the characters collection before each test
beforeEach(function(done){
    // Drop the collection
    mongoose.connection.collections.mariochars.drop(function(){
        done();
    });
});
```

COLLECTION AND MODEL

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  // Create a Schema and a Model
5
6  const MarioCharSchema = new Schema({
7      name: String,
8      weight: Number
9  });
10
11 const MarioChar = mongoose.model('mariochar', MarioCharSchema);
12
13 module.exports = MarioChar;
14
```

MOCHA

- Testing library
- Application to check the working.
- Checks in all the stages.

- Npm install mocha –save
- Create test files.

CREATING TEST

```
Const assert=require('assert');
Describe('some demo test', function () {
It ('adds two number together' , function ()
   {
  assert(2+3 ===5); } ); });
```

SAVE DATA

- We have just used mongoose to create it, but it is not in our DB yet.
- So what we have to do is save the char variable.
- When we create this new instance mongoose gives us a lot of methods to work/interact with DB.

SAVE DATA

saving_test.js

```
const assert = require('assert');
const MarioChar = require('../models/mariochar');

// Describe our tests
describe('Saving records', function(){

  // Create tests
  it('Saves a record to the database', function(done){

    const char = new MarioChar({
      name: 'Mario'
    });

    char.save().then(function(){
      assert(!char.isNew);
      done();
    });

  });

});
```

- .save is an asynchronous method.
- .then is default promise library.