**Introduction:**

|  |  |
| --- | --- |
| **Course Logistics** | |
| **Contents** | **Waitage in the final Grade** |
| Video Lectures | 0% |
| quizzes | 0% |
| Homework | 50% |
| Final Exam | 50% |

**Certification: 65% percentage to get pdf certificate**

**What is Mongo DB?**

Mongo DB is a Non Relational DataStore for JSON Objects.

By Non Relational, we mean that they don’t store data as tables like a relational Database does. But it stroes JSON Documents.

All databases(including MongoDB) are represented as

**What is JSON?**

JSON stands for JavaScript Object Notation and is a lightweight data-interchange format, language independent, "self-describing" and easy to understand. It is easy for humans to read and write. It is easy for machines to parse and generate.

**JSON syntax is derived from JavaScript object notation syntax:**

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

## JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

{“name” : “pranith”}

JSON names require double quotes. JavaScript names don't.

## JSON Values

JSON values can be:

* A number (integer or floating point)
* A string (in double quotes)
* A Boolean (true or false)
* An array (in square brackets)
* An object (in curly braces)
* null

## JSON Objects

JSON objects are written inside curly braces.

Just like JavaScript, JSON objects can contain multiple name/values pairs:

{"firstName":"John", "lastName":"Doe"}

## JSON Arrays

JSON arrays are written inside square brackets.

Just like JavaScript, a JSON array can contain multiple objects:

"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},   
    {"firstName":"Peter","lastName":"Jones"}  
]

In the example above, the object "employees" is an array containing three objects. Each object is a record of a person (with a first name and a last name).

Much Like XML Because

* Both JSON and XML is "self describing" (human readable)
* Both JSON and XML is hierarchical (values within values)
* Both JSON and XML can be parsed and used by lots of programming languages
* Both JSON and XML can be fetched with an XMLHttpRequest

Much Unlike XML Because

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

The biggest difference is:

 XML has to be parsed with an XML parser, JSON can be parsed by a standard JavaScript function.

Why JSON?

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* JSON.Parse the JSON string

These JSON Documents are stored in MongoDB. But, what makes it non-relational it that, the data looks like a table(Key is column and value is value within that column) and also can some hirarchy .

{ a : 6,

b:7,

fruit : [“apple”,”pears”]

}

In the above example, fruit items are part of the other object which cant be done in relation DB.

At highest level, MongoDB is simply a JSON Document store and that will be really useful while writing programs because JSON Data Structure looks much closer to data what our programs have(objects) then a relation database(series of tables and columns).

Mongo DB stores Documents is the fundamental fact is what makes it easy for programing.

Mongo DB is Schemaless i.e two documents don’t need to have same schema.

For example DOC1 : {“a” : “1”}

DOC2 : {“a”:”aew”

, “b” : “asdfa”}

Answer Notes

The correct answers are:

* MongoDB is document oriented.
* MongoDB is schemaless.

A brief discussion is probably in order for the schemaless answer. We also sometimes use the term 'dynamic schema.' The point is that, unlike in a relational database, you are not constrained to follow any particular schema. If you wish to change your schema, you are free to do any of the following:

* Begin inserting documents with the new schema.
* Perform a bulk update on the existing documents.
* Begin updating old documents to the new schema one by one at an appropriate event (such as getting read from or written to), as coded in the application.

Contrast this with what happens in a relational database, where the table must typically be taken offline in order to add columns.

As for the other two answers, MongoDB does not support joins as a design decision because they do not scale horizontally, and it does not support SQL because that query language was built around joins and transactions, and tends to assume table structure rather than the flexible document orientation that MongoDB provides.

For years, in order to improve performance on database servers, database administrators have had to buy bigger servers as the database load increases (scaling up) instead of distributing the database across multiple “hosts” as the load increases (scaling out). RDBMS do not typically scale out easily, but the newer NoSQL databases are actually designed to expand easily to take advantage of new nodes and are usually designed with low-cost commodity hardware in mind.

RDBMS have ACID ( <http://en.wikipedia.org/wiki/ACID> ) and supports transactions. Scaling "out" with RDBMS is harder to implement due to these concepts.

NoSQL solutions usually offer record-level atomicity, but cannot guarantee a series of operations will succeed (transaction).

It comes down to: to keep data integrity and support transactions, a multi-server RDBMS would need to have a fast backend communication channel to synchronize all possible transactions and writes, while preventing/handling deadlock.

This is why you usually only see 1 master (writer) and multiple slaves (readers).

Depth of Funtionality

Scalabity and Performance

MongoDb

MemChache

Key Value Stores

RDBMS

**Teaching:**

Let us consider the about graph,

Up in the left hand corner, we have MemChache and other key value stores. They basically have scalabity and good performance but they offer almost no functionaity.

On the other end of the scale, we have RDBMS with really rich functionality.

Now, the question is, can we get a Database which has the functionality of the RDMN but without giving up the scalable . Relational DB will scale up i.e add more hard ware to them, to get higher performance from single node but it is not good at scaling out.

MongoDB tries to retain most or all of the scalibity and performance of DB , while giving you a lot of functionality to work with but not as much we have with RDBMs.

To achieve this,

1. Mongo DB does not supports joins, because joins scales poorly when scaling-out to multiple nodes.
2. Transactions is missed. This is not much bad with MongoDb because the documents are hirarchial and we can’t access these documents automically. So operations that require multiple updates in RDBMS, can be done as signle automic trasaction with a single documents.

## Quiz: MongoDB Relative to Relational

Which features did MongoDB omit in order to retain scalability?

1. Joins
2. Transactions across multiple collections
3. Indexes
4. Secondary Indexes

ANS : MongoDb, deliberately omitted joins because joins poorly support scaling out of DB.

Indexed are supported by MongoDB and needed for performance and does not prevent horizontal scalabity.

Secondary Indexes: are offered but to be honest , it is challenges to work on a system that is horizontally scalable.

Transaction on Multiple Collections. MOngoDB omitted this because, it will be extremely difficult for handling system that extent by scalling out.

**Building An Application with MongoDB.**

**Mongo DB:** MongoDb starts a process on the computer as “MongoD”.

MongoDb processes can be connected by couple of ways

1. Mongo Shell.
2. Mongo Java Driver.

**Mongo Shell:** Mongo Shell is a standalone program that has is written javascript shell language, which connects to MongoDb using TCP(Transmission Control Protocol), this is the way to manupulate data.

As part of this course, we gonna build a Java Application i.e bunch of pages that will inserting posts and showing posts.

This appication is written in Java. This application has appication code, i.e our code.

We gonn use two different framewroks that will make us easy to write web apps.

1. SparkJava.
2. Freemarker.

Now, in order to talk to MongoDB we will use Mongo Driver to make the TCP connection and to manupulate the data.

**Mongo Shell**

To start the Mongo shell, open terminal and go to bin folder of mongo installation and then run mongo.

To select a particular database

Ex: use test.

Here, if the data base is not in use ,then the database is created and we can put collection into it.

JavaScript Object notation is used to express data inside mongo and the shell itself is made of JavaScript.

Lets assume that, there is a collection called “things” and we want to insert something.

Ex: db.things.save({a :1, b:2, c:3})

“db” is the standard keyword inside the shell.

“things” is the collection

“save” is the command to be executed.

Json documents is presented in between the function paranthesis.

To fetch the objects

Ex: db.things.find()

Each MongoDB document has object ID’s and are the unique indentifiers to keep track of the documents and always has index on object identifiers and so we can find documents with the ID.

If we have to filter it with the condition then use the condition in the parenthisis of find.

Ex: db.things.find({a:1})

This will return all the objects, with the value of key “a” is “1”.

To print the docs of the mongoDb in a formatted way use the following Command.

Ex: db.things.find().pretty()

**System Requirements**

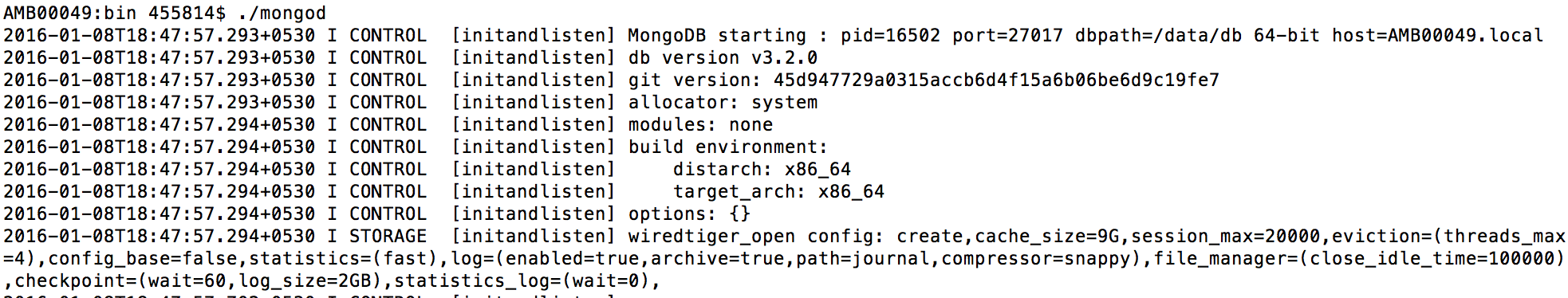
|  |  |
| --- | --- |
| **Operating System(tests done)** | **JDK** |
| **OS X 10.8** | **1.7** |
| **Window 7** |  |
| **Linux** |  |

**Installing MongoDb On Mac**

1. Get the latest download of the mongoDb and Unzip it.
2. After unzip, we have bin folder and there are my files in it.
3. However, the most important of these files are mongod and mongo .
4. “mondod” is the file that actually starts the server
5. “mongo” starts the mongo shell, that allows us to interact with the mongo.
6. MongoDB, by default saves the data onto `/data/db folder. So before starting the Mongo server, we have to create this folders with all the permissions.
7. Run the below commands.

* cd /
* mkdir data
* cd data
* mkdir db
* cd ../
* chmod 777 data
* chmod 777 data/db

\*\*\*\* Note: if you don’t have permissions then run the command in the sudo mode.

1. go to the bin folder in the terminal and run ./mongod. This will print a lot of things. But check the port on which mongo is running and dpath option i.e datastore location. 
2. open a other terminal and run ./mongo to run the shell.
3. Test database is connected. Use the command db.names.insert({a:1, b:2}) to insert into collection “names” with data.
4. “db.names.find()” to display the records in the shell.

By this, all the installment is done.

**Installing MongoDb in Winows.**

1. donwload the latest installation of MongoDb.
2. click on the installer .
3. In the pop-up window, accept the agreement and click next.
4. Slect complete installation and click next and continue still finish.
5. The path of installation is c🡪 program files -> mongoDB
6. Creatr folder(/data/db) in home folder
7. Start the server by running mongod.

**Installing and using Maven**

1. mvn archetype:generate \
2. -DarchetypeGroupId=<archetype-groupId> \
3. -DarchetypeArtifactId=<archetype-artifactId> \
4. -DarchetypeVersion=<archetype-version> \
5. -DgroupId=<my.groupid> \
6. -DartifactId=<my-artifactId>

To execute simple main class in maven, we need a “exec” plugin which is part of maven. Below is the goal to execute the main class.

Mvn clean compile exec:java -Dexec.mainClass=com.mongodb.MavenSample.App

**Intro to Spark Web Application Framework.**

For this course, we are going to use the web application framework “Spark”.

Spark is a very simple web framework. We does’t need any web server or application server. Jetty is embedded into the framework.

All we have to do is define a main method, and invoke “Spark” in the main method. This will invoke the jetty server in the back ground.

Step 1: Create a maven quickstart application.

Step 2: Add the dependency spark depency in the pom.xml file. Jetty is also downloaded along with spark. Please check the “maven Dependencies” in the project explorer. We can find the jetty related jars.

Step 3: Create a class and add the main method.

Step 4: Use Spark.get(Route) static method to invoke the spark application.

**public** **class** App {

**public** **static** **void** main(String[] args) {

Spark.*get*(**new** Route("/") {

@Override

**public** Object handle(Request arg0, Response arg1) {

// **TODO** Auto-generated method stub

**return** "Hi this is my first Example";

}

});

}

}

Explanation: In the above code, we are using static method get of Spark framwork and giving it a route i.e pattern to invoke the handler when the servers are up.

Step 5: Run the main Method.

Step 6: we can see the logs, spark has started and it is listening in the default port “4567” of local host.

Step 7: type the url “http://localhost:4567/” . The page will print the message in the return statement.

**Intro to the freemarker Templating Engine.**

FreeMarker is used to basically takes a template file and replace the place holders with the actualy values.

The place holder pattern is like ${name}.

Step 1: Add freemarker dependency in the pom.xml.

Step 2: add a template file hello.ftl with the following content. In the content ${name} is the placeholder.

<html>

<head>

<title> hello example</title>

</head>

<body> Hello ${name} </body>

</html>

Step 3:

Create a main class and add the following code in the mail method.

**public** **class** FreeMarkerSample {

**public** **static** **void** main(String[] args) {

StringWriter stringWriter = **new** StringWriter();

//Step 1

Configuration configuration = **new** Configuration();

//Step 2

configuration.setClassForTemplateLoading(FreeMarkerSample.**class**, "/");

**try** {

//Step 3

Template helloTemplate = configuration.getTemplate("hello.ftl");

//Step 4

Map<String, Object> map = **new** HashMap<String, Object>();

map.put("name", "Krishna Pranith");

//Step 5

helloTemplate.process(map, stringWriter);

System.***out***.println(stringWriter);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

In the code, at step 2, we are loading the template files. We place the files in the resource folder. So we have taken the class path and “/” indicates that files are root folder.

Step 3: we are getting the template that load. We can get the template using filename.

Step 4: we are creating map, that will used to map the placeholders with data.

Step 5: the process method is used to replace the content in the Stringwriter and print it.

**Spark and Freemaker Together**

**public** **class** App {

**public** **static** **void** main(String[] args) {

**final** Configuration configuration = **new** Configuration();

configuration.setClassForTemplateLoading(App.**class**, "/");

Spark.*get*(**new** Route("/") {

@Override

**public** Object handle(Request arg0, Response arg1) {

StringWriter out = **new** StringWriter();

**try** {

Template template = configuration.getTemplate("hello.ftl");

Map<String, Object> map = **new** HashMap<String, Object>();

map.put("name", "James bond");

template.process(map, out);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** out.toString();

}

});

}

}

**Handling Get Requests in Spark**.

As a Anatomy of Spark, we have jetty server as a web server which is embedded into Spark. When we create a Route, Jetty is automatically started .

Inside, we have a Spark handler i.e whenever a request comes to Jetty, it forwards the requests to Spark handler and this has one or more Routes.

To make the route as get protocal, then we have to use the get static method of spark.

Patterns :

Ex 1 : “/” : Requests that has / as path.

Ex 2 : “/test” : Requests that has /test as path.

Ex 2 : “/echo/:thing” : :thing is a path parameter. “:thing” is a wildcard in this. And can get the value of thing in the code by using request.getParams(“tihing”). This add dynamicity to the code .

**Handling Post Requests in Spark**.

We will come back and update this portion.

**MongoDB is schemaless**

In the Relational Database system, we will have a table and certain columns in tables. And in some time, if we have to add some more information to the table, then we have to expand the table by addding new column to table, using alter command.

In Mongo, we don’t need to do that and different documents has different schemas. That means as the system evolves as Agile Methodology often, and we accomidate in the system.

* db.users.insert("name”:”pranith”)
* db.users.insert("name”:”pranith” : “occupcation” : “business”)
* db.users.find()(2 different documents with different structure, if we want we can update the first one or else no issues, it will be the same. )
* var j = db.users.find(“name”:”pranith”) (this will save the content to j.)
* j (this will print the content)
* j.occupation = “SW”
* db.users.save(j)(This will update the existing document. )

**Json Revisited**

There are only two data structures inside JSON.

1. Arrays : These are list of things, that are represented within [ ] and seperated by ‘,’.
2. Data Dictionaries: which are represented a { }. These are associtive maps. These are bunch of keyword value pairs separated by “:”(colan) and “,” between the pairs.

These two can be combined in a arbitory ways.

Arrays can have dictionaries and dictionaries can have arrays.

The start of any documents is Dictionaries only.

**Json SPEC**

Json is a standard and we look at [www.json.org](http://www.json.org) .

Terminology is same, but instead of calling it as document at the highest level, they are called it as Object({}) .

**Object or document** can have arbitary number of keys and values which are seperated by “:”(colon) and “,”(comma) between each set of them .

**The vaue** can be String, number, object, arrays , true, false, null.

**Arrays** is a sequence of values, that are seperated by comma.

**Introduction to Schema Design:**

In relational, it is easy to finilise the schema design because we try for the 3rd normal form.

But with mongo Db and document oriented design, how do we know to embed a document or keep it a separate collection.?

For example, in a post collection, we can have tags embeded and comments embeded.

Because, most of the cases whenever we access the post, we usually fetch comments and tags with it.

The question for answering the embeded or not, is depended on the way you access the data and some pratical consideration like MongoDb document can’t be more 16mb. So if you gonna have a 16mb of comment, which is extrimely unusual, then we need to have a comments are separate collection.