Getting “MEAN”

- A Practical Workshop

MongoDB

 # MongoDB :

- MongoDB is an **open-source document database** and leading **NoSQL database**. MongoDB is written in C++.

- MongoDB is a **cross-platform**, **document oriented database** that provides, **high performance,** high availability, and easy scalability.

- MongoDB works on concept of **collection** and **document**.

 # Database :

- Database is a **physical container for collections**.

- Each database gets its own set of files on the file system.

- A single MongoDB server typically has multiple databases.

 # Collection :

- Collection is a group of MongoDB documents.

- It is the equivalent of an **RDBMS table**.

- A collection exists within a single database. Collections do not enforce a schema.

- Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

 # Document

- A document is a set of **key-value pairs**.

- Documents have **dynamic schema**.

- **Dynamic schema :** means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of **RDBMS terminology with MongoDB** :

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Advantages of MongoDB over RDBMS

- **Schema less**: MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

- **Structure** of a single object is clear.

- No complex joins.

- **Deep query-ability**. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.

- **Ease of scale-out**: MongoDB is easy to scale.

Why Use MongoDB?

- Document Oriented Storage: Data is stored in the form of JSON style documents.

- Index on any attribute

- Replication and high availability

- Rich queries

- Fast in-place updates

- Professional support by MongoDB

Where to Use MongoDB?

- Big Data

- Content Management and Delivery

- Mobile and Social Infrastructure

- User Data Management

- Data Hub

Installation

Installation :

Step 1 :

You can find all the steps of installation on this below link :

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

Step 2 : Setup the MongoDB environment :

MongoDB requires a data directory to store all data. MongoDB's default data directory is \data\db. Create this folder using the commands from command prompt :

$ md \data\db

You can specify an alternative path for data files using the –**dbpath** option to **mongod.exe**
Example :

C:\mongodb\bin\mongod.exe –dbpath d:\test\mongodb\data

Note : If your path includes spaces, enclose the entire path in double quotes, for example :

C:\mongodb\bin\mongod.exe –dbpath "d:\test\mongo db\data"

Step 3 : Connect to MongoDB

To connect to mongodb throught the mongo.exe shell, open another command prompt.

C:\mongodb\bin\mongo.exe

Commands

 # Helpful command:

- MongoDB Help

To get a list of commands, type **db.help()** in MongoDB client. This will give you a list of commands.

- MongoDB Statistics

To get stats about MongoDB server, type the command **db.stats()** in MongoDB client.

This will show the database name, number of collection and documents in the database.

Create And Drop DB

 # Create Database :

The **use** Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of use DATABASE statement is as follows:

use DATABASE_NAME

Example :

If you want to create a database with name <testdb>, then use DATABASE statement would be as follows:

>use testdb

switched to db testdb

To check your currently selected database, use the command **db**

>db

testdb

If you want to check your databases list, use the command show dbs.

> show dbs

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

Note : In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

 # Drop Database :

The **dropDatabase()** Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax :

Basic syntax of dropDatabase() command is as follows:

db.dropDatabase()

This will delete the selected database. If you have not selected any database, then it will delete default **'test'** database.

Example :

First, check the list of available databases by using the command, **show dbs**.

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

>

If you want to delete new database **<mydb>,** then dropDatabase() command would be as follows:

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

Now check list of databases.

>show dbs

local 0.78125GB

test 0.23012GB

Collections

#1 Create Collection :

The **createCollection()** Method:

MongoDB db.createCollection(name, options) is used to create collection.

Syntax

Basic syntax of createCollection() command is as follows:

db.createCollection(name, options)

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Note : Options parameter is optional, so you need to specify only the name of the collection.

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	Boolean	(Optional) If true, automatically create index on _id field. Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

Following is the list of options you can use:

#2. Drop Collection :

The **drop()** Method

MongoDB's db.collection.drop() is used to drop a collection from the database.

Syntax

Basic syntax of drop() command is as follows:

db.COLLECTION_NAME.drop()

Example

First, check the available collections into your database mydb.

>use mydb

switched to db mydb

>show collections

users

students

persons

Now drop the collection with the name mycollection.

>db.users.drop()

true

Again check the list of collections into database.

>show collections

students

persons

Note : drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

Datatypes

MongoDB supports many datatypes. Some of them are:

- **String:** This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.

- **Integer:** This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

- **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 BSON elements.

- **Arrays:** This type is used to store arrays or list or multiple values into one key.

- **Timestamp**: This can be handy for recording when a document has been modified or added.

- **Null:** This type is used to store a Null value.

- **Symbol:** This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

- **Date:** This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

- **Object ID:** This datatype is used to store the document’s ID.

Binary data: This datatype is used to store binary data.

- **Code:** This datatype is used to store JavaScript code into the document.

- **Regular expression:** This datatype is used to store regular expression.

Documents

#1. The insert() Method
To insert data into MongoDB collection, you need to use MongoDB's insert() or save()method

Syntax : **>db.COLLECTION_NAME.insert(document)**

Example :

```
>db.mycol.insert({
    “_id”: ObjectId(57da4c69a8bb6d54a9d8b147),
    “title”: 'MongoDB Example',
    “description”: 'MongoDB is no sql database',
    “likes”: 100
})
```

Explanation :

Here mycol is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

Note : In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

#2. The find() Method
To query data from MongoDB collection, you need to use MongoDB's find()method.

Syntax: **>db.COLLECTION_NAME.find()**

#3. The pretty() Method
To display the results in a formatted way, you can use pretty() method.

Syntax : **>db.mycol.find().pretty()**

Example :

```
>db.mycol.find().pretty()
{
    "_id": ObjectId(57da4c69a8bb6d54a9d8b147),
```

```
        "title": "MongoDB Overview",
        "description": "MongoDB is no sql database",
    }
```

Note : Apart from find() method, there is findOne() method, that returns only one document.

#4. The update() Method

The update() method updates the values in the existing document.

Syntax : > **db.COLLECTION_NAME.update (SELECTIOIN_CRITERIA, UPDATED_DATA)**

Example :

Consider the mycol collection has the following data.

{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"Mean Workshop"}
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"ng works "}
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"meaningfull success"}

Following example will set the new title 'New MongoDB Update' of the documents whose title is 'Mean Workshop'.

```
>db.mycol.update({'title':'Mean Workshop'},{$set:
{'title':'New MongoDB Update'}})
```

#5. The remove() Method

The remove() method is used to remove a document from the collection.remove() method accepts two parameters.

i. **deletion criteria:** (Optional) deletion criteria according to documents will be removed.

ii. **justOne:** (Optional) if set to true or 1, then remove only one document.

Syntax :

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Example :
Consider the mycol collection has the following data.
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"Mean Workshop"}
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"ng works "}
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"meaningfull success"}

Following example will remove all the documents whose title is 'ng works'.
>db.mycol.remove({'title':'ng works'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"Mean Workshop"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"meaningfull success"}

Remove Only One :
If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)

Remove All Documents :
If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.
>db.mycol.remove({})
>db.mycol.find()

Where clause

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations :

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({ “by”:”sharif malik” }).pretty()	Where by = “sharif malik”
Less than	{<key>: {$lt:<value>}}	db.mycol.find({ “age” {$lt:30}}). pretty()	Where age < 30
Less Than Equals	{<key> {$lte :<value>}}	db.mycol.find({"age": {$lte:30}}).pretty()	Where age <= 30
Greater than	{<key>:{$gt : <value>}}	db.mycol.find({"age": {$gt:20}}).pretty()	Where age > 20
Greater than equals	{<key>:{$gte :<value>}}	db.mycol.find({"likes" :{$gte:20}}).pretty()	Where age >= 20
Not equals	{<key>: {$ne:<value>}}	db.mycol.find({"age": {$ne:24}}).pretty()	Where age != 24

#AND in MongoDB

In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND −

```
>db.mycol.find({key1:value1, key2:value2}).pretty()
```

#OR in MongoDB

To query documents based on the OR condition, you need to use $or keyword.
Syntax of OR :

```
>db.mycol.find(
      {
            $or: [
            {key1: value1}, {key2:value2}
            ]
      }).pretty()
```

Projection

MongoDB Projection :

Projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 15 fields and you need to show only 3, then select only 3 fields from them.

The find() Method

MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you can execute find() method, then it displays all fields of a document.

To limit this, you need to set a list of fields with **value 1 or 0**.
1 is used to show the field while **0 is used to hide** the fields.

Syntax : **>db.COLLECTION_NAME.find({},{KEY:1})**

Example :
Consider the collection mycol has the following data

**{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147),
"title":"Mean Workshop"}**
{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147), "title":"ng works"}
**{ "_id" : ObjectId(57da4c69a8bb6d54a9d8b147),
"title":"meaningfull success"}**

Following example will display the title of the document while querying the document.

>db.mycol.find({},{"title":1,_id:0})
{"title":"mean workshop"}
{"title":"ng works"}
{"title":"meaningfull success"}

Note : _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

Object Id

Object Id :

- MongoDB uses ObjectIds as the default value of _id field of each document, which is generated while the creation of any document.

- The complex combination of ObjectId makes all the _id fields unique.

- An **ObjectId** is a 12-byte BSON type having the following structure:

- The first 4 bytes representing the seconds since the unix epoch

- The next 3 bytes are the machine identifier

- The next 2 bytes consists of process id

- The last 3 bytes are a random counter value

 # Creating New ObjectId

To generate a new ObjectId use the following code:

```
>myObjectId = ObjectId()
```

The above statement returned the following uniquely generated id:

```
> ObjectId("57da4c69a8bb6d54a9d8b147")
```

Instead of MongoDB generating the ObjectId, you can also provide a 12-byte id:

```
>myObjectId =ObjectId("57da4ca5a8bb6d54a9d8b148")
```

 # Creating Timestamp of a Document

Since the _id ObjectId by default stores the 4-byte timestamp, in most cases you do not need to store the creation time of any document. You can fetch the creation time of a document using **getTimestamp()** method:

Example :

```
> myObjectId.getTimestamp()
```

This will return the creation time of this document in ISO date format.

```
> ISODate("2016-09-15T07:23:21Z")
```

 # Converting ObjectId to String:

In some cases, you may need the value of ObjectId in a string format. To convert the ObjectId in string, use the following code:

```
>myObjectId.str
```

The above code will return the string format of the Guid:

```
>57da4c69a8bb6d54a9d8b147
```

Mongoose

What is Mongoose and how it works ?

- Mongoose was built specifically as a Mongo DB **Object-Document Modeler** (ODM) for Node applications.

- One of the key principles is that you can manage your data model from within your application.

- You don’t have to mess around directly with databases or external frameworks or relational mappers; you can just define your data model in the comfort of your application.

Why model the data?

We had already talked about how Mongo DB is a document store, rather than a traditional table-based database using rows and columns.

This allows Mongo DB great freedom and flexibility, but sometimes we want—that is, we need—structure to our data.

Example :

```
{
  "firstname" : "Simon",
  "surname" : "Holmes",
  _id : ObjectId("52279effc62ca8b0c1000007")
}
```

Example MongoDB document

```
{
  firstname : String,
  surname : String
}
```

Corresponding Mongoose schema

Explanation : the schema bears a very strong resemblance to the data itself. The schema defines the name for each data path, and the data type it will contain. In this example we’ve simply declared the paths firstname and surname as strings.

About the _id path :

You may have noticed that we haven't declared the id path in the schema. id is the unique identifier—the primary key if you like—for each document.

- MongoDB automatically creates this path when each document is created and assigns it a unique ObjectId value.

- The value is designed to always be unique by combining the time since the Unix epoch with machine and process identifiers and a counter.

Conclusion :

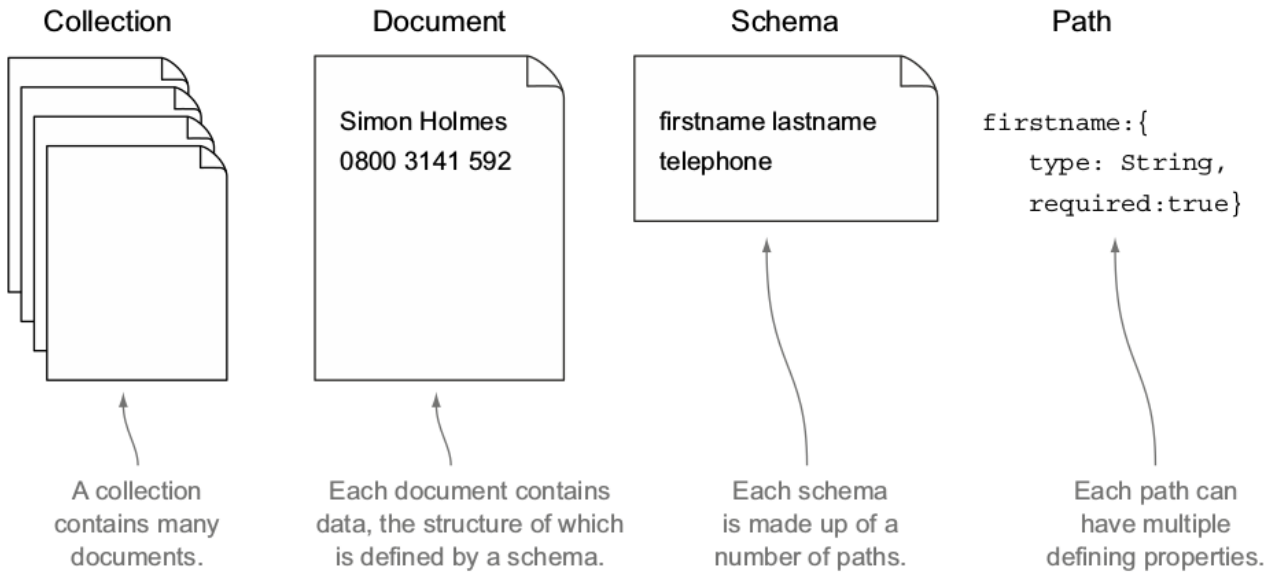
First off, let's get some naming conventions out of the way:

- In Mongo DB **each entry in a database** is called a **document**.

- In Mongo DB a **collection of documents** is called a **collection** (think "table" if you're used to relational databases).

- In Mongoose the definition of a **document** is called a **schema**.

- Each individual data entity defined in a **schema** is called a **path**.



Relationships among collections, documents, schemas, and paths in MongoDB and Mongoose.

One final definition is for **models**. A **model** is the compiled version of a **schema**.

Note : All data interactions using Mongoose go through the model.

Allowed Schema Types

#Allowed schema types

The schema type is the property that defines the data type for a given path. It’s required for all paths. If the only property of a path is the type, then the shorthand definition can be used. There are eight schema types that you can use:

String : Any string, UTF-8 encoded

Number : Mongoose doesn’t support long or double numbers, but it can be extended to do so using Mongoose plugins; the default support is enough for most cases

Date : Typically returned from MongoDB as an ISODate object

Boolean : True or false

Buffer : For binary information such as images

Mixed : Any data type

Array : Can either be an array of the same data type, or an array of nested subdocuments

ObjectId : For a unique ID in a path other than _id; typically used to reference _id paths in other documents

First Application

 # Defining simple Mongoose schemas :

We've just discussed that a Mongoose schema is essentially a JavaScript object, which we define from within the application. Let's start by setting up and including the file so that it's done and out of the way, leaving us to concentrate on the schema.

As you'd expect we're going to define the schema in the model folder.

Lets take an example of Users model.

Example 1 : You can download the source code from github

(http://github.com/virtualsharif/gettingMEAN/mongo/examples/example1)

// Step 1 : Mongoose import

You need Mongoose to define a Mongoose schema, naturally, so enter the following line into app.js (model directory)

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

// Step 2 :Mongoose connection to MongoDB URI

```
mongoose.connect('mongodb://localhost:27017/mongoose-test', function (error) {
    if (error) {
        console.log(error);
    }
});
```

// Step 3 : Mongoose Schema definition

Mongoose gives you a constructor function for defining new schemas, which you typically assign to a variable so that you can access it later. It looks like the following line:

```
var UserSchema = new mongoose.Schema({
    firstName: String,
    lastName: String,
    emailId: String
});
```

// Step 4 : Mongoose Model definition : (COMPILING A MODEL FROM A SCHEMA)

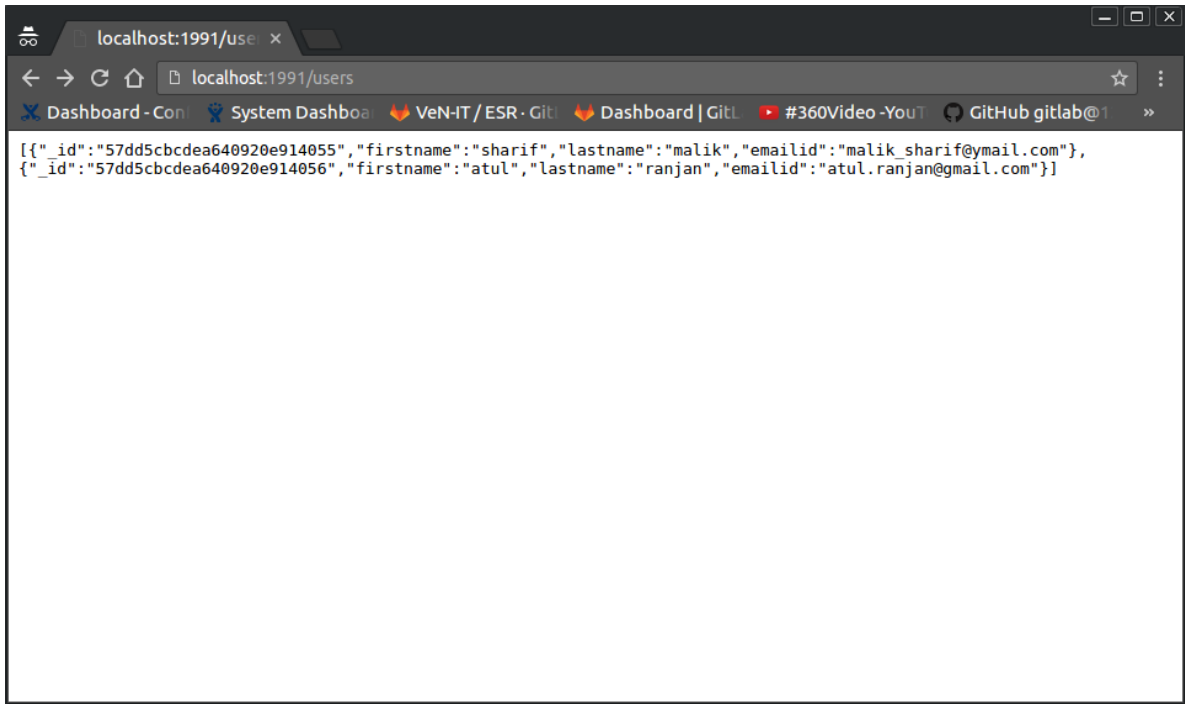
Anything with the word “compiling” in it tends to sound a bit complicated. In reality, compiling a Mongoose model from a schema is a really simple one-line task. You just need to ensure that the schema is complete before you invoke the model command.

```
var user = mongoose.model('user', UserSchema);
```

Run the example :

node app.js

To see the output :
Open the browser : localhost:1991/users



Check out the example 2 : i.e. there you can some enhancement to the example1.
Even the code is modularize.

You can download it from (http://github.com/virtualsharif/gettingmean/mongodb/example/example2)

There are 4 files :

app.js – main file which is require to run.

config.js – which consists database properties eg. Url, database name and so on.

userModel.js – which consists the schema for the user model which will be used to relate with the mongodb.

mongodb.txt – some db statements which can be used initially to setup database.

To run the application :

node app.js

To check the different routes , use POSTMAN chrome extension.(for help checkout the express documentation).