Mongo DB and Node JS

Introduction to Mongo DB- creating DB, collection – CRUD operations - Accessing MongoDB from Node.js. – Accessing online Mongo DB from Node JS.

Online Course

- https://university.mongodb.com/courses/catalog
- Coursera

2

MongoDB

- MongoDB is a open source, cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- MongoDB works on concept of collection and document.
- Database is a physical container for collections, Collection is a group of MongoDB documents and a document is a set of key-value pairs.
- MongoDB uses dynamic schemas, meaning that you can create records without first defining the structure, such as the fields or the types of their values.
- You can change the structure of records (which we call documents) simply by adding new fields or deleting existing ones.
- This data model give you the ability to represent hierarchical relationships, to store arrays, and other more complex structures easily

Node.js MongoDB

- Node.js can be used in database applications. Node js can work with both relational (such as Oracle and MS SQL Server, MySQL, IBM DB2) and non-relational databases (such as MongoDB, DocumentDB, Hbase, Neo4j).
- Over the years, NoSQL database as MongoDB becomes quite popular as databases for storing data.
- In relational database you need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL you don't have to worry about that, you can insert, update data on the fly.
- MongoDB is an open-source, document database designed with both scalability and developer agility in mind.

4

- Instead of storing data in rows and columns as one would with a relational database, MongoDB stores JSON documents in collections with dynamic schemas.
- MongoDB is that it supports dynamic schema which means one documentof a collection can have 4 fields while the other document has only 3 fields.This is not possible in relational database.
- MongoDB's document data model makes it easy for you to store and combine data of any structure, without giving up sophisticated validation rules, flexible data access, and rich indexing functionality.
- MongoDB Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can scale up and down when you need to, with no downtime, full security, and high performance.

5

Mapping relational database to 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

Relational databases usually work with structured data, while non-relational databases usually work with semi-structured data (i.e. XML, JSON)

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"
}
```

```
{
    name: "Chaitanya",          ← Field: Value
    age: 30,                    ← Field: Value
    website: "beginnersbook.com", ← Field: Value
    hobbies: ["teaching", "watching tv"] ← Field: Value
}
```

} Document

Why use MongoDB instead of MySQL?

- Organizations of all sizes are adopting MongoDB because it enables them to build applications faster.
- Development is simplified as MongoDB documents map naturally to modern, object-oriented programming languages.
- For example, schema changes that took days of weeks in The Weather Channel's MySQL databases could be made in just hours with MongoDB.
- MongoDB is a general purpose database that is used for a variety of use cases. The most common use cases for MongoDB include Internet of Things, Mobile, Real-Time Analytics, Personalization, Catalog, and Content Management.

8

Differences in Query Languages

- Both MySQL and MongoDB have a rich query language

MySQL	MongoDB

`INSERT INTO users (user_id, age, status)`
`VALUES ('bcd001', 45, 'A')`
	`db.users.insert({`
	`  user_id: 'bcd001',`
	`  age: 45,`
	`  status: 'A'`
	`})`

`SELECT * FROM users`	`db.users.find()`

`UPDATE users SET status = 'C'`
`WHERE age > 25`
	`db.users.update(`
	`  { age: { $gt: 25 } },`
	`  { $set: { status: 'C' } },`
	`  { multi: true }`
	`)`

9

Installation Procedure in Windows

1. Download the latest release of MongoDB from http://www.mongodb.org/downloads
2. In this, select community server, then windows 64bit software msi format. (depends on your system)
3. It will be installed automatically in C:\Program Files\MongoDB
4. You should manually create a folder like **C:\data\db** to store MongoDB files.
5. You should open two command prompt for mongo server and mongo client.

10

Run the command to act as a mongo server

C:\Program Files\MongoDB\Server\4.2\bin>mongod

Command Prompt - mongod

```
2020-10-12T17:01:37.728+0530 I  CONTROL  [initandlisten] **          Remote systems will be unable to connect to this se
rver.
2020-10-12T17:01:37.728+0530 I  CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specif
y which IP
2020-10-12T17:01:37.729+0530 I  CONTROL  [initandlisten] **          addresses it should serve responses from, or with -
-bind_ip_all to
2020-10-12T17:01:37.729+0530 I  CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired
, start the
2020-10-12T17:01:37.730+0530 I  CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this war
ning.
2020-10-12T17:01:37.731+0530 I  CONTROL  [initandlisten]
2020-10-12T17:01:37.759+0530 I  SHARDING [initandlisten] Marking collection local.system.replset as collection version:
<unsharded>
2020-10-12T17:01:37.815+0530 I  STORAGE  [initandlisten] Flow Control is enabled on this deployment.
2020-10-12T17:01:37.816+0530 I  SHARDING [initandlisten] Marking collection admin.system.roles as collection version: <u
nsharded>
2020-10-12T17:01:37.817+0530 I  SHARDING [initandlisten] Marking collection admin.system.version as collection version:
<unsharded>
2020-10-12T17:01:37.894+0530 I  SHARDING [initandlisten] Marking collection local.startup_log as collection version: <un
sharded>
2020-10-12T17:01:47.792+0530 I  FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory '
C:/data/db/diagnostic.data'
2020-10-12T17:01:47.795+0530 I  SHARDING [LogicalSessionCacheRefresh] Marking collection config.system.sessions as colle
ction version: <unsharded>
2020-10-12T17:01:47.795+0530 I  SHARDING [LogicalSessionCacheReap] Marking collection config.transactions as collection
version: <unsharded>
2020-10-12T17:01:48.008+0530 I  SHARDING [ftdc] Marking collection local.oplog.rs as collection version: <unsharded>
2020-10-12T17:01:48.079+0530 I  NETWORK  [listener] Listening on 127.0.0.1
2020-10-12T17:01:48.079+0530 I  NETWORK  [listener] waiting for connections on port 27017
```

Run the command to act as a mongo client

C:\Program Files\MongoDB\Server\3.2\bin>mongo

Command Prompt - mongo

```
MongoDB shell version v4.2.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("7f07a908-c2da-4a52-9eab-a23eff0c404b") }
MongoDB server version: 4.2.3
Server has startup warnings:
2020-10-11T11:57:11.246+0530 I  CONTROL  [initandlisten]
2020-10-11T11:57:11.246+0530 I  CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-10-11T11:57:11.246+0530 I  CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2020-10-11T11:57:11.246+0530 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

MongoDB- Creating a Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.
- MongoDB will create the database if it does not exist, and make a connection to it.
- **use DATABASE_NAME** is used to create database in mongo client command prompt.

Command Prompt - mongo

```
shellHelper@src/mongo/shell/utils.js:790:15
@(shellhelp2):1:1
> use test1
switched to db test1
```

- > show dbs → See the list of databases
- > db.dropDatabase() → delete current database

Creating a Collection

- A **collection** in MongoDB is the same as a **table** in MySQL.
- To create a collection in MongoDB, use the createCollection() method.
- MongoDB waits until you have inserted a document before it actually creates the collection.
- In Mongo Client command prompt,

> **db.createCollection("stud1");**

{ "ok" : 1 }

> **show collections;** □ ***display all collections (tables)***

> **db.emp1_collection.drop()** □ ***delete a collection*** returns true

```
> db.createCollection("stud1")
{ "ok" : 1 }
> show collections
stud1
> db.createCollection("Course")
{ "ok" : 1 }
> show collections
Course
stud1
> db.stud1.insert({name:"jaya", school:"PSBBM"})
WriteResult({ "nInserted" : 1 })
```

Insert Into Collection

- A **document** in MongoDB is the same as a **record** in MySQL.
- To insert a record, or *document* as it is called in MongoDB, into a collection, we use the insertOne() method.
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.

>**db.stud1.insert**({name:"jaya",school:"PSBBM"})

WriteResult({ "nInserted" : 1 })

```
> db.createCollection("stud1")
{ "ok" : 1 }
> show collections
stud1
> db.createCollection("Course")
{ "ok" : 1 }
> show collections
Course
stud1
> db.stud1.insert({name:"jaya", school:"PSBBM"})
WriteResult({ "nInserted" : 1 })
```

- To insert multiple documents in a single query, you can pass an array of documents in insert() command

>db.emp_collection.insert([{name:"jaya",school:"PSBBM"},{name:"Rakshitha",school:"VV"}**]);**

```
> db.stud1.insert([{name:"jaya", school:"PSBBM"},{name:"Rakshitha", school:"VV"}]);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

Query Document - Display values

- >db.COLLECTION_NAME.find() → **find()** method display all the documents in a non-structured way.
- In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
- _id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows –
- _id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

Command Prompt - mongo

```
> use test1
switched to db test1
> show collections
Course
stud1
> db.stud1.find()
{ "_id" : ObjectId("5f83335273fa7608ad824343"), "name" : "jaya", "school" : "PSBBM" }
{ "_id" : ObjectId("5f83344e73fa7608ad824344"), "name" : "jaya", "school" : "PSBBM" }
{ "_id" : ObjectId("5f83344e73fa7608ad824345"), "name" : "Rakshitha", "school" : "VV" }
>
```

To improve the readability, we can format the output in JSON format with this command:

- db.emp_collection.find().pretty(); (or)
- db.emp_collection.find().forEach(printjson);

Command Prompt - mongo

```
> db.stud1.find().forEach(printjson)
{
        "_id" : ObjectId("5f83335273fa7608ad824343"),
        "name" : "jaya",
        "school" : "PSBBM"
}
{
        "_id" : ObjectId("5f83344e73fa7608ad824344"),
        "name" : "jaya",
        "school" : "PSBBM"
}
{
        "_id" : ObjectId("5f83344e73fa7608ad824345"),
        "name" : "Rakshitha",
        "school" : "VV"
}
>
```

Query Document based on the criteria

- **Equality Criteria:** I want to fetch the data of School "VV" from stud1. The command for this should be:

Command Prompt - mongo

```
> db.stud1.find({school:"VV"}).pretty()
{
        "_id" : ObjectId("5f83344e73fa7608ad824345"),
        "name" : "Rakshitha",
        "school" : "VV"
}
>
```

- **Greater Than Criteria:**
- I would like to fetch the details of students having age > 32 then the query should be:
- db.students.find({"age":{$gt:32}}).pretty()
- **Less than Criteria:**
- db.students.find({"StudentId":{$lt:3000}}).pretty()
- **Not Equals Criteria:**
- db.students.find({"StudentId":{$ne:1002}}).pretty()
- **Greater than equals Criteria:**
- db.collection_name.find({"field_name":{$gte:criteria_value}}).pretty()
- **Less than equals Criteria:**
- db.collection_name.find({"field_name":{$lte:criteria_value}}).pretty()

Updating Document

- db.collection_name.update(criteria, update_data)

> **db.emp_collection.update(**{school:"SENSE"}, {$set:{school:"SELECT"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

```
> db.emp_collection.update({school:"SENSE"},{$set:{school:"SELECT"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.emp_collection.find().pretty();
{
        "_id" : ObjectId("5badbce1b701835b49de0e79"),
        "name" : "marees",
        "school" : "SCOPE"
}
{
        "_id" : ObjectId("5badbce1b701835b49de0e7a"),
        "name" : "eswari",
        "school" : "SELECT"
}
{
        "_id" : ObjectId("5badc420b701835b49de0e7b"),
        "name" : "marees",
        "school" : "SITE"
}
```

- By default the update method updates a single document. In the above example we had only one document matching with the criteria, however if there were more then also only one document would have been updated. To enable update() method to update multiple documents you have to set "multi" parameter of this method to true as shown below.

➢ **db.emp_collection.update({name:"maghes"}, {$set:{name:"magheswari"}},{multi:true})**

- WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

Delete Document from a Collection

- **db.collection_name.remove(delete_criteria)**

```
> db.emp_collection.remove({name:"magesh"});
WriteResult({ "nRemoved" : 0 })
> db.emp_collection.remove({name:"mageshwari"});
WriteResult({ "nRemoved" : 2 })
>
```

- When there are more than one documents present in collection that matches the criteria then all those documents will be deleted if you run the remove command. However there is a way to limit the deletion to only one document so that even if there are more documents matching the deletion criteria, only one document will be deleted.
- **db.collection_name.remove(delete_criteria, justOne)**
- Here justOne is a Boolean parameter that takes only 1 and 0, if you give 1 then it will limit the document deletion to only 1 document.

db.collection_name.remove({}) → Remove all Documents

Projection – select particulars

- to get the selected fields of the documents rather than all fields.
- Value 1 means show that field and 0 means do not show that field. When we set a field to 1 in Projection other fields are automatically set to 0, except _id, so to avoid the _id we need to specifically set it to 0 in projection. The vice versa is also true when we set few fields to 0, other fields set to 1 automatically.

```
> db.emp_collection.find({},{name:1});
{ "_id" : ObjectId("5badbce1b701835b49de0e7a"), "name" : "eswari" }
{ "_id" : ObjectId("5bae072bb701835b49de0e7c"), "name" : "marees" }
{ "_id" : ObjectId("5bae074db701835b49de0e7d"), "name" : "marees" }
{ "_id" : ObjectId("5bae0769b701835b49de0e7e"), "name" : "venkat" }
> db.emp_collection.find({},{_id:0,name:1});
{ "name" : "eswari" }
{ "name" : "marees" }
{ "name" : "marees" }
{ "name" : "venkat" }
>
```

- **Sorting Documents using sort() method**
- db.collection_name.find().sort({field_key:1 or -1})
- 1 is for ascending order and -1 is for descending order. The default value is 1.

```
> db.emp_collection.find().sort({school:1}).pretty();
{
        "_id" : ObjectId("5bae074db701835b49de0e7d"),
        "name" : "marees",
        "school" : "SCOPE"
}
{
        "_id" : ObjectId("5bae0769b701835b49de0e7e"),
        "name" : "venkat",
        "school" : "SCOPE"
}
{
        "_id" : ObjectId("5badbce1b701835b49de0e7a"),
        "name" : "eswari",
        "school" : "SELECT"
}
{
        "_id" : ObjectId("5bae072bb701835b49de0e7c"),
        "name" : "marees",
        "school" : "SITE"
}
>
```

- To display the *school* field of all the *emp_collection* in **ascending order** and display it.

```
> db.emp_collection.find({},{school:1,_id:0}).sort({school:1}).pretty();
{ "school" : "SCOPE" }
{ "school" : "SCOPE" }
{ "school" : "SELECT" }
{ "school" : "SITE" }
```

- **The default is ascending order**
- > db.emp1_collection.find({}, {school: 1, _id:0}).sort({})