Mongodb Document

Created by Vineet Semwal
(vineetsemwal82@gmail.com)

Mongodb

1) Document Oriented database

2) Not a rdbms (no concept of join, if you want details of other document then other document will be embedded in the parent, example is provided later)

3) Nosql

4) High Performance and scalable

5) Javascript is used as query language

6) Mongodb supports many languages, like for java, driver is there , there is driver for other languages too

7) Mongodb keeps replicas(more than one system keeping same copy) for high availability which means if one is down, others are available, Secondary replicas keep copy of primary one , Writes(changes to database) are only done in primary replica, Secondary replicas are used for reads only, Secondary replicas never take the role of primary replica which means no writes on them

8) Sharding is a method for distributing data across multiple machines, MongoDB uses sharding to support very large datasets for loadbalancing

9) Default port on which mongodb runs is 27017

10) mongodb uses bson for storing json like documents

Database
Database contains collections(same as table)

Collection(similar to table)
1)Contains documents
2)Schema is not enforced

Document (similar to row)
1) Dynamic Schema
2) Documents objects in the same collection can have different fields
(read it in this way two rows can have different columns in the same table)
3) _id field is kept to uniquely identify the document object ,

Mongodb methods

help : see all commands and methods

db.help() : help on db methods

db.collection.help(): help on collection methods

use databasename: create and switch to this database
Foreg. Use mydb , will create database mydb and we will be switched to this database, if database already exists then it will not be created

show dbs : show database names

db.dropDatabase() : drop the selected database

db.stats(): see stats of the current database

db.createUser() : Create user for the database and assign role
Example

```
db.createUser({user:'scooby', pwd: 'scooby123' , roles:[{role: "userAdmin" ,
db: 'mydb'}]});
```

This creates user scooby with password scooby123 , role assigned is admin which means user has administrative privileges (read,write,drop etc)

Since mongodb uses javascript as query language array is defined with [], objects with {}

show users : show users in current database

db.createCollection("collectionname"): Create collection with a specified name, options can be provided as second argument ,
Options are like 1)Document 2)capped 3) autoIndexId
Example db.createCollection("employees");

db.getCollectionNames() : find all collections in database

db.collectionName.drop(): Drop a collection from database
db.employees.drop();

db.collectionName.insert()

Insert a document into the collection
If the collection doesn’t exist before, it is created

Example

```
db.employees.insert({ eid:1, ename:'mohan'  })
```

if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
_id is 24 bytes hexadecimal number unique for every document in a collection.

```
db.employees.insert({_id:ObjectId('123456789abc123456789abc') ,
  eid:3,  ename:'mohan'  });
```

db.collectionName.find()

fetches data from the collection
db.employees.find() will fetch all the employees from the collection
This is shortcut of db.employees.find({})

db.employees.find({eid:’e-1’}) will fetch only employee with eid e-1

If you want result to be displayed in more readable format
db.employees.find().pretty()

Comparison operators

Greater Than

See below example of using comparison operator

Find employees with age greater than 21

```
db.employees.find({ age: { $gt:21 } } )
db.employees.find({ age: {$lt:21} });
```

Similarly use below operators

$gte: Greater than equal to

$lt: Less than

$lte: Less than equal to

Logical Query operators

Or

db.collectionName.find({ $or:[{“key”:value}, {“key”:value}])

Example

Find employees whose id is either e1 or e2

db.employees.find({ $or:[{_id:"e1" } ,{_id: "e2" }] })

And

db.collectionName.find({ $and:[{key:value}, {key:value}])

This in short can be done in this way

db.collectionName.find({key1:value1 ,key2:value2})

and is default/implicit in the above query

Example: find employees whose name is scooby and age is 21

db.employees.find({ $and:[{ename:'scooby'} ,{age:21}] })

This in short can be written in this way

db.employees.find({ename:'scooby', age:23})

Not

Find employees where age is not greater than 21

db.employees.find({ age: { $not : { $gt :21} }});

Delete

Delete employee with id e1

```
db.employees.deleteMany({_id:'e1'});
```

Delete all employees

```
db.employees.deleteMany({});
```

Update method

db.collectionName.update(criteria, {document which will get updated})

Example:

```
db.employees.update( {_id: "e1" } , {}  )
```

this will update to an empty document

To update data back

```
db.employees.update({_id :"e1"} , { ename : "scooby" } )
```

Update or a delete operation result gives two things in information

1) Acknowledgement
2) Number of rows affected

Save method

replaces existing document with the new document, if the document with the same _id doesn’t exist before it inserts a new document

```
db.employees.save({ _id :"e1" ,  ename: "scooby" });
```

This will **update** a new employee if employee document exists with _id e1

This will **insert** a new employee if employee document does NOT exist with _id e1

Since mongodb is not a rdbms, there is no concept of join however you can embed one document inside another, address document will be stored/embedded inside employee document

```
@Document("employees")
public class Employee implements Serializable {
 private  Address address;
 public Address getAddress(){
  return address;
  }
public void setAddress(Address address){
  This.address=address;
}
```

If an array is kept as an field in the above document and that array is small size then it is ok but it can grow and become bigger than document size itself
Foreg. SchoolClass is kept as document and then Student[] (array) is kept as field of the document then this array will grow bigger than the document itself

mongodump and mongorestore

This is used to create a dump from original database and then using this dump for setting up another database , this feature is often used in testing

As we have seen from java side when connecting to a database from java side for development only two things are required
1) ip (for local machine localhost) 2) portnumber (27017 by default) so the url constructed in end is mongodb://localhost:27017
where mongodb is the protocol

mongoexport is a command that produces a JSON or CSV export of data stored in mongodb database
mongoexport --collection collection_name --db databasename --out events.json
If out is skipped in the above command then the result will be displayed in the standard output