Web Programming

YJ – Aug 2015

MongoDB

- MongoDB is an open-source document database, and leading NoSQL database. MongoDB is written in c++.
- In object-oriented development, we're encouraged to approach code development through logical models, so that we can more readily conceptualise it in our mind. When we do this, we're better able discern the logical operations used to interact with it and information that it would contain at different times.
- What if you could store the programmatic models almost exactly like you model them? What if you could store them as they are instead of in a series of rows in tables? By learning about MongoDB, you're going to be able to do just that!

MongoDB Concepts

- 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.

MongoDB Concepts

Below given 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)

MongoDB Sample

Below given example shows the document structure of a blog site which is simply a comma separated key value pair.

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
      },
   {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
   }
   ]
}
```

Flexible Schema

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.

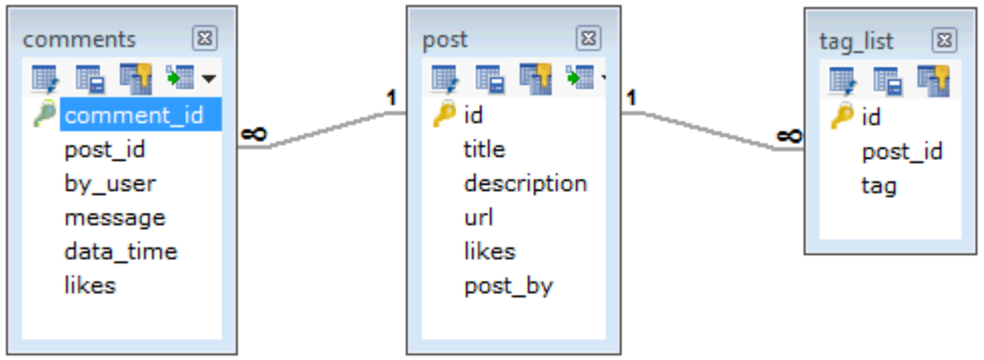
Some considerations while designing schema in MongoDB:

1. Design your schema according to user requirements.
2. Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
3. Duplicate the data (but limited) because disk space is cheap as compare to compute time.
4. Do joins while write, not on read.
5. Optimize your schema for most frequent use cases.

Flexible Schema

Suppose a client needs a database design for his blog website and see the differences between RDBMS and MongoDB schema design.

RDBMS schema



MongoDB schema

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

So while showing the data, in RDBMS you need to join three tables and in mongodb data will be shown from one collection only.

Install MongoDB

Mac

Step 1: Install Homebrew package manager:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Step: 2: Update the Hombrew packages database

```
$ sudo brew update
```

Step 3: Install MongoDB

```
$ sudo brew install mongodb
```

Step 4: Run MongoDB

```
$ mongod

$ mongo
```

Install MongoDB

Windows

Step 1: downloaded 32-bit version MongoDB .msi file from

https://www.mongodb.org/downloads?_ga=1.135497713.1929057503.1447386197#production

Assume that you have installed MongoDB to C:\mongodb

Step: 2: Create this folder using the following commands from a **Command Prompt**:

```
$ md \data\db
```

Step 3: Run MongoDB

```
$ C:\mongodb\bin\mongod.exe

$ C:\mongodb\bin\mongo.exe
```

Test MongoDB

1. Copy the content on right and paste it in your mongo shell
2. To confirm that the database and accompanying records have been created, type in the following command:

```
db.nettuts.find();
```

3. Try basic selector. Say that we want to find all actors that are female.

```
db.nettuts.find({gender:  'f'});
```

```
db.nettuts.insert({
    first: 'matthew',
    last: 'setter',
    dob: '21/04/1978',
    gender: 'm',
    hair_colour: 'brown',
    occupation: 'developer',
    nationality: 'australian'
});
        db.nettuts.insert({
    first: 'arnold',
    last: 'schwarzenegger',
    dob: '03/06/1925',
    gender: 'm',
    hair_colour: 'brown',
    occupation: 'actor',
    nationality: 'american'
});
db.nettuts.insert({
    first: 'jamie lee',
    last: 'curtis',
    dob: '22/11/1958',
    gender: 'f',
    hair_colour: 'brown',
    occupation: 'actor',
    nationality: 'american'
});
db.nettuts.insert({
    first: 'michael',
    last: 'caine',
    dob: '14/03/1933',
    gender: 'm',
    hair_colour: 'brown',
    occupation: 'actor',
    nationality: 'english'
});
db.nettuts.insert({
    first: 'judi',
    last: 'dench',
    dob: '09/12/1934',
    gender: 'f',
    hair_colour: 'white',
    occupation: 'actress',
    nationality: 'english'
});
```

Create/Drop Database

Create database

```
> use mydb
```

Check database

```
> show dbs
```

To display database you need to insert at least one document into it.

```
> db.movie.insert({"name":"tutorials point"})
> show dbs
```

Drop database

```
> use mydb
> db.dropDatabase()
```

Create/Drop Collection

Create collection

```
> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )
```

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a collection fixed size collecction 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.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. If **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.

Check collection

```
> show collections
```

In mongodb you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.tutorialspoint.insert({"name" : "tutorialspoint"})
>show collections
```

Drop collection

```
> use mydb
> db.mycollection.drop()
```

Insert Document

Insert Document

```
>db.COLLECTION_NAME.insert(document)
```

If the collection doesn't exist in the database, then MongoDB will create this collection and then insert document into it.
If we don't specify the _id parameter, then MongoDB assigns an unique ObjectId for this document.

```
>db.mycol.insert({
_id: ObjectId(7df78ad8902c),
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
})
```

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

Retrieve Document

Retrieve document

```
>db.COLLECTION_NAME.find()
>db.COLLECTION_NAME.findOne()
>db.COLLECTION_NAME.find().pretty()
```

```
>db.mycol.find().pretty()
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "tutorials point",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100"
}
```

Where Clause Equivalents

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

Retrieve Document

AND Clause Equivalents

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

OR Clause Equivalents

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

AND/OR combination Clause Equivalents

```
db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"}, {"title": "MongoDB Overview"}]}).pretty()
```

Equivalent sql where clause is **'where likes>10
AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

Update Document

mongoDB

Update document

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

```
{ "_id" : ObjectId(5983548781331adf45ec5),
"title":"MongoDB Overview"}
```

```
>db.mycol.update({'title':'MongoDB
Overview'},{$set:{'title':'New MongoDB
Tutorial'}})
```

By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'}, {$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

The save() method replaces the existing document with the new document passed in save() method

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

```
>db.mycol.save(
{
"_id" : ObjectId(5983548781331adf45ec7),
"title":"Tutorials Point New Topic",
"by":"Tutorials Point"
}
)
```

Projection

In mongodb projection meaning is selecting only necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

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

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

```
{ "_id" : ObjectId(5983548781331adf45ec5),
"title":"MongoDB  Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6),
"title":"NoSQL  Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Tutorials  Point Overview"}
```

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB  Overview"}
{"title":"NoSQL  Overview"}
{"title":"Tutorials  Point Overview"}
```

Sort

To sort documents in MongoDB, you need to use **sort()** method. **sort()**method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
```

Advantage Features

❖ Indexes

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require the mongodb to process a large volume of data.

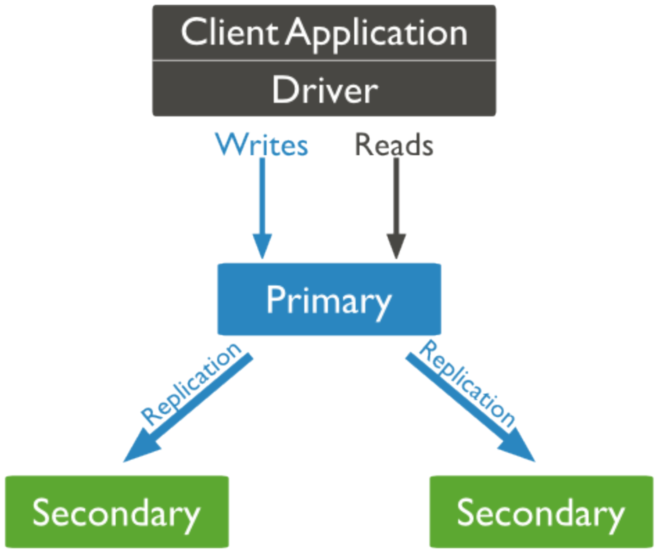
```
>db.records.createIndex( { userid: 1 } )
```

❖ Replication

Replication is the process of synchronizing data across multiple servers.

Replication provides redundancy and increases data availability with multiple copies of data on different database servers, replication protects a database from the loss of a single server.

Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.



Advantage Features

❖ Sharding

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth.

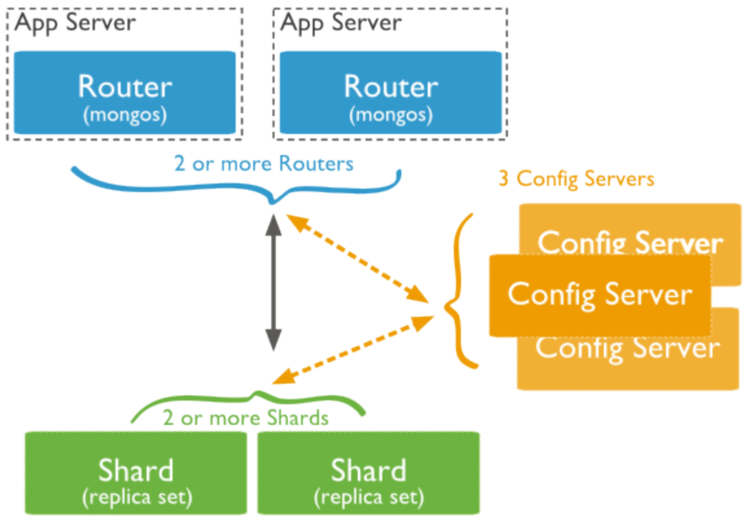
As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.

Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Shards: Shards are used to store data. They provide high availability and data consistency. In production environment each shard is a separate replica set.

Config Servers: Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. In production environment sharded clusters have exactly 3 config servers.

Query Routers: Query Routers are basically mongos instances. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally a sharded cluster have many query routers.



Advantage Features

❖ Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In sql count(*) and with group by is an equivalent of mongodb aggregation.

```
{
   _id: ObjectId(7df78ad8902c)
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
_id: ObjectId(7df78ad8902d)
title: 'NoSQL Overview',
description: 'No sql database is very fast',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 10
},
{
_id: ObjectId(7df78ad8902e)
title: 'Neo4j Overview',
description: 'Neo4j is no sql database',
by_user: 'Neo4j',
url: 'http://www.neo4j.com',
tags: ['neo4j', 'database', 'NoSQL'],
likes: 750
},
```

Now from the right collection if you want to display a list that how many tutorials are written by each user then you will use aggregate() method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
"result" : [
{
"_id" : "tutorials point",
"num_tutorial" : 2
},
{
"_id" : "Neo4j",
"num_tutorial" : 1
}
],
"ok" : 1
}
```

Sql equivalent query for the above use case will be
select by_user, count(*) from mycol group by by_user

Advantage Features

❖ Aggregation

There is a list available aggregation expressions.

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

MongoDB in PHP

To use mongodb with php you need to use mongodb php driver.

Download the driver from (https://s3.amazonaws.com/drivers.mongodb.org/php/index.html).

Unzip the archive and put php_mongo.dll in your PHP extension directory ("ext" by default) and add the following line to your php.ini file:

```
extension = php_mongo.dll
```

Connect to mongoDB

```php
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
?>
```

MongoDB in PHP

Create a collection

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";

// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->createCollection("mycol");
echo "Collection created succsessfully";
?>
```

Insert a document

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully”;
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully”;
$document = array(
   "title" => "MongoDB",
   "description" => "database",
   "likes" => 100,
   "url" => "http://www.tutorialspoint.com/mongodb/",
   "by", "tutorials point"
);
$collection->insert($document);
echo "Document inserted successfully";
?>
```

MongoDB in PHP

Find all documents

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";

// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";

$cursor = $collection->find();
// iterate cursor to display title of documents

foreach ($cursor as $document) {
   echo $document["title"] . "\n";
}
?>
```

Update a document

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";
// now update the document
$collection->update(array("title"=>"MongoDB"),
   array('$set'=>array("title"=>"MongoDB Tutorial")));
echo "Document updated successfully";
// now display the updated document
$cursor = $collection->find();
// iterate cursor to display title of documents
echo "Updated document";
foreach ($cursor as $document) {
   echo $document["title"] . "\n";
}
?>
```

MongoDB in PHP

Delete a document

Remaining mongodb methods findOne(), save(), limit(), skip(), sort() etc works same as explained in above slides.

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully”;
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully”;
// now remove the document
$collection->remove(array("title"=>"MongoDB Tutorial"),false);
echo "Documents deleted successfully”;
// now display the available documents
$cursor = $collection->find();
// iterate cursor to display title of documents
echo "Updated document”;
foreach ($cursor as $document) {
   echo $document["title"] . "\n";
}
?>
```