Introduction to MongoDB

Introduction

What is MongoDB?

MongoDB is an open source document-oriented NoSQL database system. Mongo db is part of NoSQL database system family. Instead of storing data in table, MongoDB stores structured data as JSON like document with dynamic schemas (MongoDB calls the format BSON), making data integration with application faster and easier.

Why use MongoDB?

- **Document-oriented**
  - Documents (objects) map nicely to programming language data types
  - Embedded documents and arrays reduce need for joins
  - Dynamically-typed (schemaless) for easy schema evolution
  - No joins and no multi-document transactions for high performance and easy scalability
- **High performance**
  - No joins and embedding makes reads and writes fast
  - Indexes including indexing of keys from embedded documents and arrays
  - Optional streaming writes (no acknowledgements)
- **High availability**
  - Replicated servers with automatic master failover
- **Easy scalability**
  - Automatic sharding (auto-partitioning of data across servers)
    - Reads and writes are distributed over shards
    - No joins or multi-document transactions make distributed queries easy and fast
  - Eventually-consistent reads can be distributed over replicated servers
- **Rich query language**

How MongoDB reduce Complexity?

- Get rid of migration
    - No create table
    - No alter column
    - No add column
    - No change column
- Get rid of relationship(most of)
    - Many one-to-one and one-to-many relationship is not necessary
    - User :has_one :setting
    - User :has_many :addresses
    - Post :has_many :tags
- Reduce number of database request
    - Pre joined
    - Rich queries
    - Atomic, in-place update
- JSON
    - MongoDB knows JSON
    - Don't have to convert data from/to JSON

Mongo data model

- A Mongo system (see deployment above) holds a set of databases
- A **database** holds a set of collections
- A **collection** holds a set of documents
- A **document** is a set of fields
- A **field** is a key-value pair
- A **key** is a name (string)
- A **value** is a
    - basic type like string, integer, float, timestamp, binary, etc.,
    - a document, or
    - an array of values

Quick Start Guide

Installing MongoDB in windows :

Go to http://www.mongodb.org/downloads,

Ensure your download the proper version of MongoDB for your windows architecture.

After your download finished, extract the downloaded archive file.

Make sure, your archive file contains following files :

Name	Type	Packed ...	Has a...	Size	Ratio	Date
bin	File Folder	0 KB		0 KB	0%	
GNU-AGPL-3.0	0 File	12 KB	No	35 KB	67%	5/9/2012 2:0
README	File	1 KB	No	2 KB	49%	5/9/2012 2:0
THIRD-PARTY-NOTICES	File	4 KB	No	10 KB	68%	5/9/2012 2:0

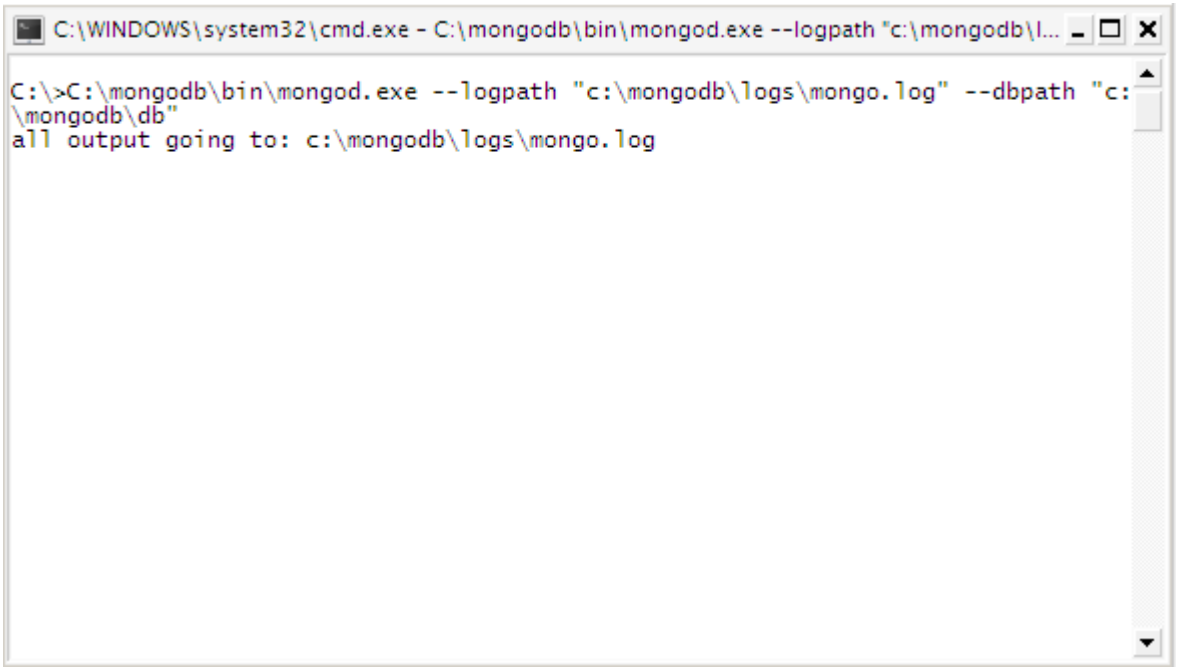
Create folder for logs file and database file, for example :

Name	Size	Type
bin		File Folder
db		File Folder
logs		File Folder
GNU-AGPL-3.0	35 KB	0 File
README	2 KB	File
THIRD-PARTY-NOTICES	10 KB	File

To start MongoDB, start mongod.exe with logpath and dbpath parameters

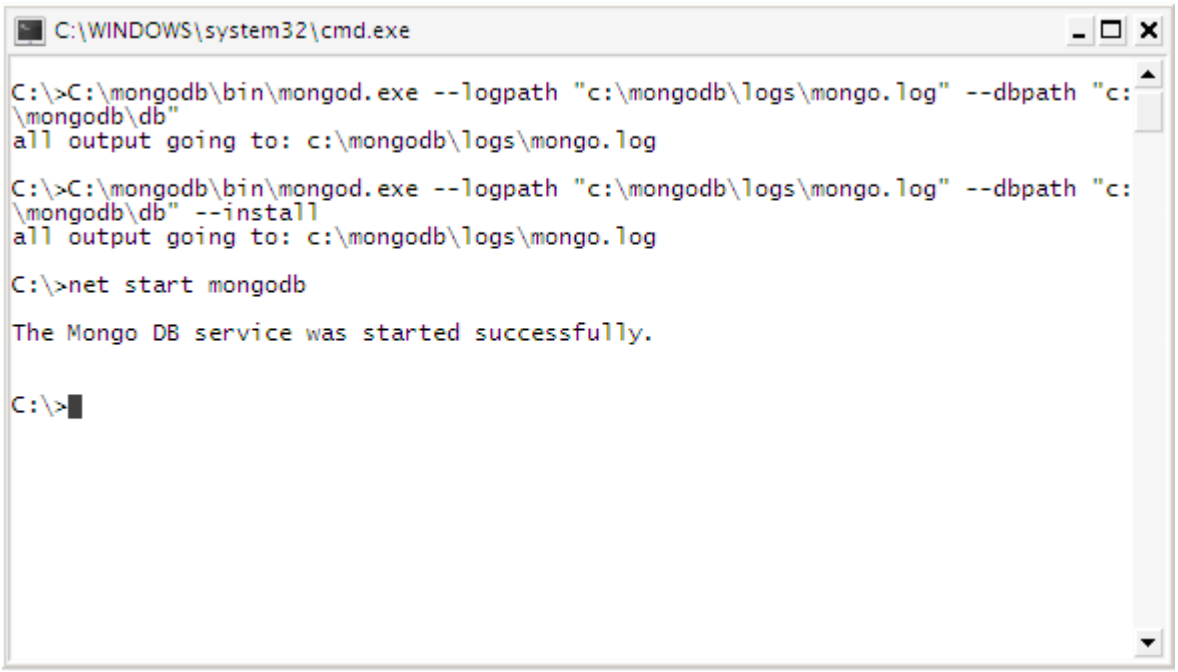
Example :

```
c:\mongodb\bin\mongod.exe --logpath "c:\mongodb\logs\mongo.log" --dbpath "c:\mongodb\db"
```



Or

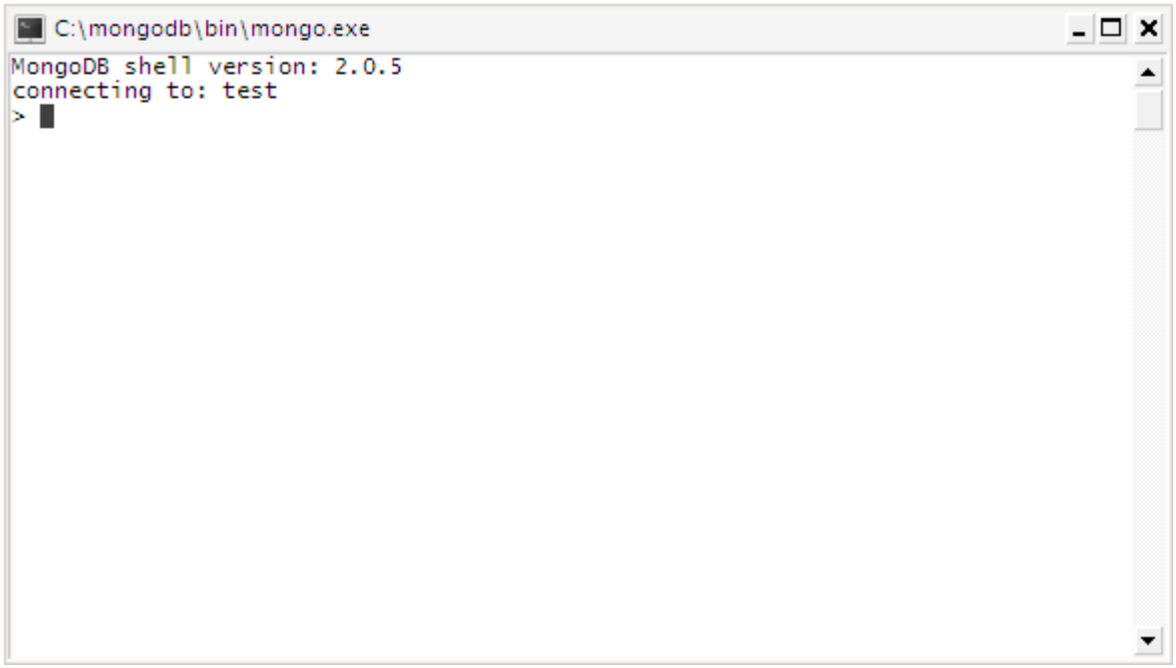
```
c:\mongodb\bin\mongod.exe --logpath "c:\mongodb\logs" --dbpath
"c:\mongodb\db" –install

net start mongo
```



to install and run mongodb as windows service.

MongoDB Query

To test query, run mongo.exe file in bin folder



Create /Drop database and collections

Use [db name]
To see database list, type “show dbs” in Mongo Shell
Database creation running implicitly after collections has been created inside it.

```
Use [db name]
db.dropDatabase();
```

Explicit : `db.createCollection(“collection name”)`
Implicit : `db.[collection name].insert({…})`
Database creates it automatically on the first insert.

MongoDB does not require create query, as soon as you insert something, MongoDB creates the underlying database and collection.

Retrieving data

```
show all item list
db.item.find()

show all item with _id 'm001'
db.item.find({_id:'m001'})

show all item with following _id and show name and _id field only
db.item.find({_id:'m002'}, {name: 1})

show all item without price
db.items.find({}, {price: 0})

show all item with following _id without showing the price
db.item.find({_id:'m002'}, {price: 0})

show only item _id and price field
db.items.find({}, {price: 1})

show only item price field
db.items.find({}, {price: 1, _id: 0})

show all items with price not equals 10000
db.item.find({price: {$ne: 10000}})


show all items with price greater than 10000
db.item.find({price: {$gt: 10000}})

show all items with price lower than 10000
db.item.find({price: {$lt: 10000}})

show all items with price greater or equals than 10000
db.item.find({price: {$gte: 10000}})

show all items with price lower or equals than 10000
db.item.find({price: {$lte: 10000}})

show items with name 'blue' or 10000 price
db.item.find({$or: [{name: 'blue'}, {price: 10000}]})

show all items with minimal
db.item.min({price: 10000})
```

```
show all items with maximal price 10000
db.item.max({price: 10000})

same as find(), but return only the first value instead of item list
db.item.findOne()
```

retrieve data from 2 collections (embed and linking)

```
t = db.item.findOne({name: 'black berry'})
db.trans.find({itemid: t._id} )
```

Insert data

```
db.item.insert({_id: ‘m001’, name: ‘blue berry’, stock: ‘21’, price: ‘5000’})
db.item.save({_id: ‘m001’, name: ‘blue berry’, stock: ‘21’, price: ‘5000’})
// update data if document exists, otherwise insert new document
d = {_id: ‘b001’, name: ‘apple’, quantity: 10}
db.fruit.save(d)
```

Update

Update field increment by given value

```
{ $inc : { field : value } }

Update field set field with given value
{ $set : { field : value } }

Delete given field
{ $unset : { field : 1} }

Add new field with given value
{ $push : { field : value } }
{ $push : { field : value, field2 : value2 } }
```

Add new field with given array, otherwise append the value

```
{ $pushAll : { field : value_array } }

{ $addToSet : { field : value } }
{ $addToSet : { a : { $each : [ 3 , 5 , 6 ] } } }

{ $pop : { field : 1  } }
{ $pop : { field : -1  } }

Remove occurences value from given field if value is an array
{ $pull : { field : value_array } }

Rename the field
{ $rename : { old_field_name : new_field_name } }

Example using parameter above :
db.item.update({_id: 'm001'}, {$inc: {quantity: 1} })

db.item.update({_id: 'm001'}, {$set: {name: 'new name'} })
```

Delete data

```
Removes all document
db.item.remove()
```

Removes document with given value

```
db.item.remove({_id:'m001'})
```

MapReduce

MapReduce Introduction

Map/reduce in MongoDB is useful for batch processing of data and aggregation operations. It is similar in spirit to using something like Hadoop with all input coming from a collection and output going to a collection. Often, in a situation where you would have used GROUP BY in SQL, map/reduce is the right tool in MongoDB.

`map/reduce` is invoked via a database command. Typically the database creates a collection to hold output of the operation. `map` and `reduce` functions are written in JavaScript and execute on the server.

Command syntax:

```
db.runCommand(
 { mapreduce : <collection>,
   map : <mapfunction>,
   reduce : <reducefunction>,
   out : <see output options below>
   [, query : <query filter object>]
   [, sort : <sorts the input objects using this key. Useful for
optimization, like sorting by the emit key for fewer reduces>]
   [, limit : <number of objects to return from collection, not supported
with sharding>]
   [, keeptemp: <true|false>]
   [, finalize : <finalizefunction>]
   [, scope : <object where fields go into javascript global scope >]
   [, jsMode : true]
   [, verbose : true]
 }
);
```

Creating map function

The `map` function references the variable `this` to inspect the current object under consideration. A map function calls `emit(key,value)` any number of times to feed data to the reducer.

```
function() {
    emit( key, value );
  }
```

Ex :

```
function mapf()
{
    emit(this.userid,
         {userid:this.userid, total_time:this.length, count:1, avg_time:0});
}
```

Creating reduce function

When you run a map/reduce, the `reduce` function will receive an array of emitted values and reduce them to a single value. Because the reduce function might be invoked more than once for the same key, the structure of the object returned by the reduce function must be identical to the structure of the `map` function's emitted value.

```
function(key, values) {
    // do something with key and values

    return result;
  }
```

Ex :

```
function reducef(key, values)
{
    var r = {userid:key, total_time:0, count:0, avg_time:0};
    values.forEach(function(v)
                   {
                       r.total_time += v.total_time;
                       r.count += v.count;
                   });
    return r;
}
```

Creating finalize function

```
function(key, reduced) {
    // do something with key and values

    return result;
  }
Ex.
function finalizef(key, reduced)
{
    if (reduced.count > 0)
        reduced.avg_time = reduced.total_time / reduced.count;

    return reduced;
}
```

A `finalize` function may be run after reduction. Such a function is optional and is not necessary for many map/reduce cases. The finalize function takes a key and a value, and returns a finalized value.

Running map reduce in database :

```
db.collection.mapReduce(mapfunction,reducefunction[,options]);
```

Ex.

```
db.session.mapReduce(mapf,reducef, { out : "myoutput" });
db.myoutput.find();
or
db.runCommand(
{ mapreduce:"session",
        map:mapf,
        reduce:reducef,
        query: {ts: {$gt:ISODate('2011-11-03 00:00:00')}},
        out: { reduce: "session_stat" },
        finalize:finalizef
});
db.session_stat.find();
```

Using PHP Driver

Getting MongoDB PHP Driver

Download the correct driver for your environment from http://github.com/mongodb/mongo-php-driver/downloads.

Unzip and add the php_mongo.dll file to your PHP extensions directory (usually the "ext" folder in your PHP installation.)

Add to your php.ini:

```
extension=php_mongo.dll
```

Restart your web server (Apache, IIS, etc.) for the change to take effect

Making a Connection

To connect to the database server, use one of the following:

```
<?php

$connection = new Mongo(); // connects to localhost:27017
$connection = new Mongo( "example.com" ); // connect to a remote host (defaul
t port: 27017)
$connection = new Mongo( "example.com:65432" ); // connect to a remote host a
t a given port

?>
```

Getting a Database

To select a database, use:

```
<?php

$db = $connection->dbname;

?>
```

Getting A Collection

Getting a collection has the same syntax as getting a database:

```
<?php

$db = $connection->baz;
$collection = $db->foobar;

// or, more succinctly
$collection = $connection->baz->foobar;

?>
```

The database does not need to be created in advance, you can create new databases by selecting them.

```
<?php

$db = $connection->mybiglongdbname;
// do some stuff
$db = $connection->mybiglongdbanme;
// now connected to a different database!

?>
```

Inserting a Document

Associative arrays are the basic object that can be saved to a collection in the database. A somewhat random "document" might be:

```
<?php

$doc = array( "name" => "MongoDB",
   "type" => "database",
   "count" => 1,
   "info" => (object)array( "x" => 203, "y" => 102),
   "versions" => array("0.9.7", "0.9.8", "0.9.9")
);

$collection->insert( $doc );

?>
```

Selecting a Document

```
<?php

$obj = $collection->findOne();
var_dump( $obj );

?>
```

Creating An Index

MongoDB supports indexes, and they are very easy to add on a collection. To create an index, you specify the field name and direction: ascending (1) or descending (-1). The following creates an ascending index on the "i" field:

```
<?php

$coll->ensureIndex( array( "i" => 1 ) );  // create index on "i"
$coll->ensureIndex( array( "i" => -
1, "j" => 1 ) );  // index on "i" descending, "j" ascending

?>
```

Example

```
<?php

// connect
$m = new Mongo();

// select a database
$db = $m->comedy;

// select a collection (analogous to a relational database's table)
$collection = $db->cartoons;

// add a record
$obj = array( "title" => "Calvin and Hobbes", "author" => "Bill Watterson" );
$collection->insert($obj);

// add another record, with a different "shape"
$obj = array( "title" => "XKCD", "online" => true );
$collection->insert($obj);
```

```php
// find everything in the collection
$cursor = $collection->find();

// iterate through the results
foreach ($cursor as $obj) {
    echo $obj["title"] . "\n";
}

?>
```

This would output:

```
Calvin and Hobbes
XKCD
```

Another Example

mongo_show.php:

```php
<!-- mongo_show.php -->
<html>
<head></head>
<body>
    <form method="post" action="mongo_proc.php">
        <input type="text" name="username" />
        <input type="submit" value="insert data" />
    </form>
    <table border="1">
    <thead>
        <tr>
            <th>No.</th>
            <th>Username</th>
        </tr>
    </thead>
    <tbody>
        <?php
            $m = new Mongo();
            $db = $m->user;
            $coll = $db->userlist;

            $cur = $coll->find();
            if($cur->count() > 0)foreach($cur as $obj){
        ?>
        <tr>
            <td><?php echo $obj["number"]; ?></td>
            <td><?php echo $obj["name"]; ?></td>
        </tr>
        <?php }else echo "<tr><td colspan='2'>no data</td></tr>"; ?>
```

```
    </tbody<
    </table>
</body>
</html>
```

mongo_proc.php:

```
<?php
    // mongo_proc.php
    $m = new Mongo();
    $db = $m->user;
    $coll = $db->userlist;

    $no = $coll->find()->count() + 1;
    $user = $_POST['username'];

    $coll->save(array('number' => $no, 'name' => $user ));

    header('location:mongo_show.php');
?>
```

References :

http://www.mongodb.org/
http://www.php.net/manual/en/mongo.tutorial.php
http://www.slideshare.net/tiendung/benefits-of-using-mongodb-reduce-complexity-adapt-to-changes
http://blog.mongodb.org/post/28339027694/new-mongodb-desktop-backgrounds